Improved Approaches to Minimize the Makespan on Single-Machine Scheduling with Periodic Preventive Maintenance Activities

Hongming Zhou, Ya-Chih Tsai, Ful-Chiang Wu, Shenquan Huang, and Fuh-Der Chou

1College of Mechanical and Electrical Engineering, Wenzhou University, Wenzhou 325035, Zhejiang, China
2Department of Hotel Management, Vanung University, Jung-Li, Taoyuan, Taiwan

Correspondence should be addressed to Shenquan Huang; hshenquan@163.com and Fuh-Der Chou; fdchou@tpts7.seed.net.tw

Received 22 August 2019; Accepted 21 March 2020; Published 22 April 2020

Academic Editor: Purushothaman Damodaran

Copyright © 2020 Hongming Zhou et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This paper addresses a single-machine scheduling problem with periodic preventive maintenance activities that are predetermined so that the machine is not available all the time, and jobs have to be processed between two consecutive maintenance periods. We propose a mixed integer programming (MIP) model and two heuristics to minimize the makespan. With more constraints in our model, the model is more efficient than the recent model of Perez-Gonzalez and Framinan, and our model could solve problems with up to fifty jobs. Two heuristic algorithms, namely, H (MW) and H (LB*), are also proposed, in which two bin-packing policies of the minimum waste and minimum lower bound are used, respectively. Furthermore, we also proposed an improvement procedure. The results showed that the heuristic H (MW) outperformed other heuristics of the paper, indicating that the bin-packing policy of the minimum waste is more effective than well-known ones such as full batch and best fit. Additionally, all the heuristic algorithms addressed in this paper combined with the improvement procedure could achieve a similar and high quality of solutions with a very tiny increase in computational expense.

1. Introduction

The manufacturing processes constituted by a series of automated machines for processing jobs or dangerous tasks are common practices in many production factories today. Those automated machines are usually expensive, and for this capital-intensive nature of operations, machine malfunction or inappropriate machine idle will cause large costs and losses; thus, many companies provided maintenance policies based on reliability theory to consider unit replacement problems [1–3] or adopted preventive maintenance to reduce the chances of machine malfunction. During the maintenance or unit replacement, machines are not available for processing any jobs, resulting in decreased production time. Additionally, inappropriate production scheduling without considering maintenance requirements will lead to machines waiting for auxiliary resources such as spare parts, tools, and repair staff. Therefore, the importance of cooperation between periodic maintenance and production scheduling has increasingly been received by researchers and decision makers.

Preventive maintenance (PM) is one of the factors causing machines to be unavailable for processing jobs, and it can significantly affect the performance of job scheduling. The majority of the studies in the literature assumed that each maintenance activity is either assigned in advance, i.e., its starting and completion times are known, [4–16] or maintenance activities and jobs are jointly scheduled in the planning horizon [12, 17–26]. In the abovementioned studies, they assumed that the maintenance time (maintenance duration) is a constant; however, in some maintenance activities such as cleaning, refilling, and recharging,
the maintenance time is a nondecreasing function of its starting time, i.e., the later the maintenance activity begins, the longer its duration. Kubzin and Strusevich [27] were the first pioneers who considered the scheduling problems with variable maintenance duration, and they considered the objective of minimizing the makespan. They showed that the two-machine open shop problem is polynomially solvable, while the 2-machine flowshop problem is NP-hard, for which an efficient algorithm is provided. Since then, various scheduling problems with variable maintenance durations in different machine environments have been considered in the literature [23, 28, 29]. Additionally, more realistic scheduling problems involving deterioration jobs and maintenance activity simultaneously have received considerable attention. Deterioration jobs indicate that the processing times of jobs are not constant, depending on their starting time or position, namely, time-dependent deterioration or position-dependent deterioration effects, respectively. The readers may refer to the survey by Gawiejnowicz [30] and Janiak and Rudek [31] for the problem with deterioration jobs. Regarding this kind of studies, most studies considered the maintenance duration as a constant, including Ji et al. [32], Lee and Wu [33], and Yang et al. [34], whereas Yang [35] further investigated the unrelated parallel-machine problems with deterioration effects and deteriorating multimaintenance activities, in which the maintenance duration is not constant, depending on its running time. The author developed mathematical programming models to find the optimal maintenance frequencies, the optimal maintenance positions, and the optimal job sequences such that the objective of the total completion time is minimized.

This paper pursues the stream of research initiated by Ji et al. [9], who were the first pioneers to study a single-machine problem with periodic maintenance activity. For the problem, the planning horizon comprises availability periods (T time unit) followed by nonavailability periods (t time unit) as shown in Figure 1. More precisely, all jobs are available at time zero, and the processing time of job j is a constant \( p_j \) for \( j = 1, 2, \ldots, n \). A single machine is also available at time zero; however, the machine should be stopped to perform maintenance activities regularly. The time of each maintenance activity is \( t \), and the availability period of the machine between consecutive maintenance activities is \( T \) time unit, which is also referred to as a batch in the literature. To achieve a high utilization rate of the machine, we consider the objective of minimizing the makespan, and the problem is denoted here as 1\|PM, t, T\|C_{\text{max}} by the well-known three-field notation (Graham et al. [36]).

The 1\|PM, t, T\|C_{\text{max}} problem is strongly NP-hard [4], and this problem is also similar to the classical bin-packing problem; thus, bin-packing policies such as first-fit and best-fit are encountered in some two-phase heuristics for solving this kind of problems with periodic maintenance activities [9, 17, 20, 21, 37, 38]. Those heuristic algorithms sorted the jobs first and were then followed by bin-packing policies to assign the ordered jobs into batches. Perez-Gonzalez and Framinan [16] gave a complete list for the existing two-phase heuristic algorithms in the literature. They also developed a mixed integer programming (MIP) formulation and two-phase heuristic algorithms with local search to find optimal or near-optimal solutions for the problems.

Inspired by the similarity between the considered problem and bin-packing problem, we found that a useful constraint on the minimum number of batches (or bins) could make our MIP model more efficient compared with the PM-MILP model of Perez-Gonzalez and Framinan [16]. Additionally, we proposed two heuristic algorithms where two different bin-packing policies are used to guide which jobs are assigned to bins. One of the bin-packing policies is the minimum waste time that is motivated by the property of full batch, i.e., a batch that is filled as full as possible is a benefit for the objective of minimizing makespan [9]. The other is the minimum lower bound. Moreover, the two heuristic algorithms were improved by job-exchange and insertion moves. The computational results showed that the proposed heuristic algorithms are more efficient. The remainder of this paper is organized as follows. In Section 2, we present the MIP model of Perez-Gonzalez and Framinan [16]. In Section 3, our MIP model is addressed, and comparisons between the two models are also given. In Section 4, different lower bounds are discussed. The two proposed heuristic algorithms are introduced in Section 5. Numerical experiments are conducted in Section 6, and conclusions are given in Section 7.

2. The Formulation of Perez-Gonzalez and Framinan [16]

The following are the notations and variables of the MIP model addressed by Perez-Gonzalez and Framinan [16]:

**Notations:**
- \( n \), the number of jobs;
- \( p_j \), the processing time of job \( j, j = 1, \ldots, n \);
- \( T \), the availability period (size of batches);
- \( M \), a large integer value (\( M \geq T \)).

**Variables:**
- \( b_i \), \( b_i = 1 \) if batch \( i \) is occupied, and otherwise \( b_i = 0 \) and \( i = 1, \ldots, n \);
- \( x_{ij} \), \( x_{ij} = 1 \) if job \( j \) is assigned to batch \( i \), and otherwise \( x_{ij} = 0, i = 1, \ldots, n, j = 1, \ldots, n \).
- \( \text{Maxslack} \) is a continuous variable that computes the maximum slack of the batches.
- \( \delta, \delta_i = 1 \) if the maximum slack is given by batch \( i \).

Using the above variables, the MIP model of Perez-Gonzalez and Framinan [16] denoted by the PF model later in this paper is described as follows:

\[
\text{Min } T \times \sum_{i=1}^{n} b_i - \text{Maxslack},
\]
subject to
\[ \sum_{j=1}^{n} p_j x_{ij} \leq T b_i, \quad \forall i = 1, \ldots, n, \] (2)

\[ \sum_{j=1}^{n} x_{ij} = 1, \quad \forall j = 1, \ldots, n, \] (3)

\[ \text{max slack} \leq M (1 - \delta_i) + T b_i - \sum_{j=1}^{n} p_j x_{ij}, \quad \forall i = 1, \ldots, n, \] (4)

\[ \sum_{j=1}^{n} x_{ij} \geq b_i, \quad \forall i = 1, \ldots, n, \] (5)

\[ \sum_{i=1}^{n} \delta_i = 1, \] (6)

\[ \delta_i \leq b_i, \quad \forall i = 1, \ldots, n, \] (7)

\[ x_{ij}, b_i, \delta_i \in \{0, 1\}, \quad \forall i = 1, \ldots, n; j = 1, \ldots, n, \] (8)

\[ \text{max slack} \geq 0. \] (9)

In the PF model, maintenance time \( t \) is neglected because the time is constant and does not impact the optimal schedule [9]; for the explanations of constraints (2)–(9), the readers can refer to the study of Perez-Gonzalez and Framinan [16]. In addition, Perez-Gonzalez and Framinan [16] also addressed another model named BP-MILP in their study which adopts the model of Martello et al. [39] for solving the bin-packing problem. The main difference between BP-MILP and PF is that one solution where a batch with the greatest slack value among all optimal solutions obtained by BP-MILP model is an optimal solution for the PF model.

3. The Proposed MIP Model

The BP-MILP model of Perez-Gonzalez and Framinan [16] combined the model provided by Martello et al. [39] and the following equation to obtain solutions for the \(| nr - pm | C_{\text{max}} \) problem:

\[ C_{\text{max}} = T \cdot \left( \sum_{i=1}^{n} b_i - 1 \right) + \sum_{j=1}^{n} p_j x_{kj}, \] (10)

where \( \sum_{i=1}^{n} b_i \) is the minimum number of batches, and \( \sum_{j=1}^{n} p_j x_{kj} \) indicates the lowest load at batch \( k \). However, the BP-MILP model cannot guarantee obtaining the lowest load at batch \( k \). Thus, Perez-Gonzalez and Framinan [16] provided another model (PF) mentioned above to obtain optimal solution as mentioned above. Through BP-MILP and PF models, it shows again that the bin-packing problem and \(| nr - pm | C_{\text{max}} \) Problem are very similar and the solution of bin-packing problem can be used to obtain better solutions for the \(| nr - pm | C_{\text{max}} \) problem. For bin-packing problems, the minimum number of batches is an important factor influencing the schedule results, and it could serve as a constraint or a decision variable. However, constraints on the minimum number of batches are not involved in the PF model of Perez-Gonzalez and Framinan [16]. Thus, we used the formula \( \sum_{j=1}^{n} p_j / T \) to estimate the minimum number of batches, where \( x \) indicates the smallest integer larger than or equal to \( x \) and added the constraint of \( i \cdot x_{ij} \leq N \) in our model to make sure that the number of batches is minimum.

In our model, the notations and variables are almost the same as the PF model of Perez-Gonzalez and Framinan [16], other than the following:

Notation:

\( t \), the maintenance time

Variables:

\( N \), the number of batches: \( \left\lceil \sum_{j=1}^{n} p_j / T \right\rceil \leq N \leq n \).

\[ \text{Min} \ (T + t) \cdot (N - 1) + (T - \text{max slack}), \] (11)

subject to

\[ \sum_{i=1}^{n} x_{ij} = 1, \quad \forall j = 1, \ldots, n, \] (12)

\[ \sum_{j=1}^{n} p_j x_{ij} \leq T, \quad \forall i = 1, \ldots, n, \] (13)

\[ i \cdot x_{ij} \leq N, \quad \forall i = 1, \ldots, n; j = 1, \ldots, n, \] (14)

\[ \text{max slack} \geq T - \sum_{j=1}^{n} p_j \cdot x_{ij}, \quad \forall i = 1, \ldots, N, \] (15)

\[ x_{ij} \in \{0, 1\}, \quad \forall i = 1, \ldots, n; j = 1, \ldots, n, \] (16)

\( N \), max slack are area positive integers. (17)

Objective (11) is to minimize the makespan, which is inspired by the bin-packing problem addressed in the studies of Ji et al. [9] and Perez-Gonzalez and Framinan [13]. That is, the optimal solution is in the form of having a minimum number of batches plus the lowest load. Constraints (12) and
(13) are the same as constraints (3) and (2). Constraint (14) enforces that jobs be assigned in the preceding available batches, which minimizes the number of batches. Constraint (15) is used to find the maximum slack time.

The variables used in our model are the same as those of the PF model, but maintenance time $t$ is involved in our objective function; thus, the objective of the PF model is changed to $\min (T + t) \times \text{Low}_0 \left( \frac{\sum_{j=1}^{n} p_j}{T} - 1 \right) + T - \text{max slack}$ in this paper for fair comparison later, even though the value of $t$ does not influence the results.

Table 1 shows the differences in the number of constraints and the number of variables between the abovementioned two models. Table 1 reveals that the number of binary variables in our model is less than that in the PF model; more obviously, our model has many more constraints than the PF model, which could theoretically cut down the solution search space so that our model is much more efficient for finding optimal solutions, which has been proven by our computational results. However, too many binary variables remain in the two models to be solved even for modest problem sizes; for example, when $n$ is equal to 100, there are 10200 and 10000 binary variables for the PF model and our model, respectively, to be solved. More specifically, such mathematical programming approaches, including the two models addressed in this paper, suffer from the inherent drawback, i.e., it is tough to solve problems of large sizes in a reasonable CPU time for NP-hard problems.

### 4. The Calculation of Lower Bound

For an NP-hard scheduling problem, lower bounds used to measure the performance of heuristic algorithms are commonly encountered in the literature because optimal solutions are hard to obtain. Additionally, the lower bound plays a key role in developing implicit enumeration methods such as branch-and-bound algorithms or serves as an index in developing heuristic algorithms. Therefore, the precise estimation of lower bounds is an important issue.

For a $1|\text{pm} \rightarrow \text{pm}||\text{max}$ problem, Low et al. [20] used a lower bound to evaluate the performance of heuristic algorithms. The lower bound is denoted as $LB_{\text{Low}}$ here and estimated by

$$LB_{\text{Low}} = \sum_{j=1}^{n} p_j + \left( \frac{\sum_{j=1}^{n} p_j}{T} - 1 \right) \cdot t,$$

(18)

where $[x]$ indicates the smallest integer larger than or equal to $x$. The first term is the total processing time, and the second term is the minimum total maintenance time. In another study, Low et al. [21] also used lower bound as a performance index for single-machine scheduling problems with flexible and periodic maintenance where the maintenance activity should be implemented during the prespecified intervals. Their lower bound ($LB_{\text{Low}}'$) is estimated as follows:

$$LB_{\text{Low}}' = \max \left\{ \sum_{j=1}^{n} p_j + (n/k - 1) \cdot t, \sum_{j=1}^{n} p_j + \left( \frac{\sum_{j=1}^{n} p_j}{T} - 1 \right) \cdot t \right\},$$

(19)

where $k$ is the maximum number of jobs processed in the machine’s available interval. From formulas (18) and (19), it is evident that formula (19) dominates formula (18), and the estimation for the minimum number of batches, i.e., $x$, is an important factor for the quality of lower bound as usually recognized by many researchers in numerous works. In addition, we briefed three observations from numerous studies in the literature as follows:

Property 1. With respect to schedule makespan as a measure of performance, it is necessary to consider the schedules in which the number of batches is minimum [9].

Property 2. For the makespan objective, switching jobs within a batch does not influence the makespan.

Property 3. Among the schedules with minimum number of batches, it is necessary to consider the optimal schedules in which the last batch has the minimum load, i.e., the total processing time of jobs in the last batch is minimized [16].

Based on the abovementioned findings, we introduced three estimations for the minimum number of batches ($B_{\text{min}}$): (1) $B_{\text{min}}^1 = \sum_{j=1}^{n} p_j/T$, which is the same as that of $LB_{\text{Low}}$ [20]; (2) $B_{\text{min}}^2 = \left( \sum_{j=1}^{n} p_j + W_{\text{max}} \right)/T$, where $W_{\text{max}}$ is the maximum waste and will be explained in the following; and (3) $B_{\text{min}}^3$ is obtained by the integer programming model for bin-packing problems provided by Martello et al. [39].

To obtain the value of $W_{\text{max}}$ and $B_{\text{min}}^2$, we developed a maximum waste ($\text{Max}_W$) method. The $\text{Max}_W$ method first assumes that $v$ is in the range of $[1, T/2]$, i.e., $1 \leq v \leq (T/2)$, and let $N_A$ and $N_B$ be the number of jobs in the sets $A$ and $B$, respectively.

$\text{Max}_W$ method:

Step 1: Let $v = 1$, $W_{\text{max}} = 0$.

Step 2: Place the jobs in set $A$ if the processing time of job $j$ is greater than or equal to $(T - v)$, i.e., $p_j \geq (T - v)$.

Step 3: Place the jobs in set $B$ if the processing time of job $j$ is less than or equal to $v$, i.e., $p_j \leq v$.

Step 4: Calculate the waste value of $W_v$ by the formula of $W_v = \max(0, N_A \times T - \sum_{j \in A} p_j + \sum_{j \in B} p_j)$, if $W_v > W_{\text{max}}'$, $W_{\text{max}} = W_v$.

Step 5: If $v < T/2$, $v = v + 1$, go to Step 2; otherwise, go to Step 6.

Step 6: If $B_{\text{min}}^1 = \left( \sum_{j=1}^{n} p_j + W_{\text{max}} \right)/T$, stop.

The $\text{Max}_W$ method is aimed at creating full batches as possible to avoid undesirable capacity waste of batches while there exist jobs to form a full batch to accommodate.

| Table 1: Analysis between PF and our models. |
|---------------------------------------------|
| Analysis items    | PF model | Our model |
| The number of constraints | $5n + 2$ | $n^2 + 3n + 1$ |
| The number of integer variables | 1 | 2 |
| Binary variables | $n^2 + 2n$ | $n^2$ |
Regarding the estimation of the lower bound, we modified formula (19) based on the properties as follows:

$$LB^* = \max \left\{ (T + t) \cdot (B_{\text{min}} - 1) + \min_i \sum_{j=1}^{n} P_j + (B_{\text{min}} - 1) \cdot t \right\},$$

(20)

where $B_{\text{min}}$ is the estimated number of batches. Consider an example with 6 jobs, where $T = 10$, $t = 3$, $p_1 = 9$, $p_2 = 8$, $p_3 = p_4 = 2$, $p_5 = 5$, and $p_6 = 4$. Using the formula of $B_{\text{min}}^1 = \sum_{i=1}^n p_j / T$, $B_{\text{min}}^2 = 30/10 = 3$. For $B_{\text{min}}^2$, we first used the Max_W method to obtain $W_{\text{max}}$, the steps are illustrated in detail in Table 2, and $W_{\text{max}} = \max(W_i) = 1$. Finally, $B_{\text{min}}^2 = 30 + 1/10 = 4$ based on the formula $B_{\text{min}}^2 = (\sum_{j=1}^n P_j + W_{\text{max}}) / T$. Regarding $B_{\text{min}}^3$, the result is 4, obtained by the integer programming model of Martello et al. [39].

Based on the obtained $B_{\text{min}}$, the lower bound of $LB^*$ ($B_{\text{min}}^1 = \max((10 + 3) \cdot (3 - 1) + 2, 30 + (3 - 1) \cdot 3) = \max\{28, 36\} = 36$, and $LB^* (B_{\text{min}}^2) = LB^* (B_{\text{min}}^3) = \max((10 + 3) \cdot (4 - 1) + 2, 30 + (4 - 1) \cdot 3) = \max\{41, 39\} = 41$, which is also equal to the optimal solution.

Because $B_{\text{min}}^2$ is dominated by $B_{\text{min}}^1$ and the Max_W method is much more efficient than the integer programming model of Martello et al. [39], we adopted $B_{\text{min}}^2$ to obtain the estimated number of batches in this paper.

### 5. The Proposed Heuristic Algorithms

According to the research work by Ji et al. [9], the machine’s availability period can be treated as a bin with capacity $T$, and then inspired by bin-packing problems, many researchers developed different heuristic algorithms with two phases. First, the jobs are sorted by a certain dispatching rule, and the bin-packing policy is then adopted to fill the bin as much as possible, based on the study of Ji et al. [9]. Perez-Gonzalez and Framanin [16] provided a survey for this kind of two-phase heuristic algorithms in the literature; additionally, they also developed a two-phase heuristic algorithm with a local search technique to solve the problem.

Based on the studies of Perez-Gonzalez and Framanin [16] and Ji et al. [9], the LPT rule for generating job sequences is better than other dispatching rules in the first phase; thus, in this paper, the jobs first were sorted in nonincreasing order of their processing time, i.e., $p_{[1]} \geq p_{[2]} \geq \cdots \geq p_{[n]}$, and then we developed two bin-packing policies of the minimum waste and the minimum lower bound that are different from the well-known ones such as full batch and best fit which are prevalent encountered in the literature for solving the scheduling problem with machine maintenance. Moreover, these proposed heuristic algorithms were improved by an improvement procedure.

Note that the initial job-sequence (JS) is given by the LPT rule, i.e., $p_{[1]} \geq p_{[2]} \geq \cdots \geq p_{[n]}$, and bin-packing policies including full batch, best fit, the minimum waste, and the minimum lower bound are described as follows:

| $v$ | $N_A$ | $N_B$ | $W_v$ |
|-----|-------|-------|-------|
| 1   | 1     | 0     | $\max[0, 1 \times 10 - (9 + 0)] = 1$ |
| 2   | 2     | 2     | $\max[0, 2 \times 10 - (17 + 0)] = 0$ |
| 3   | 2     | 2     | $\max[0, 2 \times 10 - (17 + 0)] = 0$ |
| 4   | 2     | 3     | $\max[0, 2 \times 10 - (17 + 0)] = 0$ |
| 5   | 3     | 4     | $\max[0, 3 \times 10 - (22 + 13)] = 0$ |

#### 5.1. Full Batch (FB)

Select the job in order from the given JS and place it in the first batch in which there is enough slack time to accommodate it, i.e., $P_j \leq s_k$ (slack time). Otherwise, create a new batch, and place the job in it.

#### 5.2. Best Fit (BF)

- **Step 1:** Let $UJS = \{J_{[1]}, J_{[2]}, \ldots, J_{[n]}\}$, $i = 1$, $B_i = B_i = \{\emptyset\}$, $sk = T$.
- **Step 2:** Select all unscheduled jobs if $P_j \leq sk$, $j \in UJS$, and put the jobs in set CJS.
- **Step 3:** If CJS = $\{\emptyset\}$, $i = i + 1$, create a new batch, i.e., $B_i = B_i = \{\emptyset\}$, $sk = T$, and go to Step 2. Otherwise, select the job $J_{[a]}$ in order from set CJS if $P_j \leq sk$.
- **Step 4:** $B_i = B_i \cup \{J_{[a]}\}$, $sk = sk - P_{[a]}$, delete the job from set UJS, i.e., $UJS = UJS \backslash \{J_{[a]}\}$.
- **Step 5:** If $UJS = \{\emptyset\}$, calculate makespan using $C_{\max} = (T + t) \cdot (i - 1) + \sum_{j \in B_i} P_j$, and stop. Otherwise, let CJS = $\{\emptyset\}$ and go to Step 2.

#### 5.3. Minimum Waste (MW)

- **Step 1:** Let $i = 1$, $B_i = B_i = \{\emptyset\}$, $sk = T$.
- **Step 2:** If $UJS \neq \{\emptyset\}$, select all unscheduled jobs if $P_j \leq sk$, and put the jobs in order in set CJS. Otherwise, go to Step 11.
- **Step 3:** If CJS = $\{\emptyset\}$, $i = i + 1$, create a new batch, i.e., $B_i = B_i = \{\emptyset\}$, $sk = T$, and go to Step 2.
- **Step 4:** If the sum of the processing time of all jobs in set CJS is less than or equal to sk, i.e., $\sum_{j \in CJS} P_j \leq sk$, place all the jobs of set CJS into the batch, i.e., $B_i = B_i \cup CJS$, $sk = sk - \sum_{j \in CJS} P_{[j]}$, and $UJS = UJS \backslash CJS$; go to Step 2.
- **Step 5:** If $\sum_{j \in CJS} P_{[j]} > sk$, select job $J_{[a]}$ in the top of set CJS. Let $SS_1 = \{J_{[a]}\}$, and $Waste_1 = sk - P_{[a]}$.
- **Step 6:** If $Waste_1 = 0$, place $J_{[a]}$ in $B_i$, $B_i = B_i \cup \{J_{[a]}\}$, $i = i + 1$, $B_i = B_i = \{\emptyset\}$, $sk = T$, and delete the $J_{[a]}$ from set UJS; go to Step 2.
- **Step 7:** If $Waste_1 > 0$, select any pair of jobs, $J_{[b]}; J_{[c]}; J_{[b]} \in CJS$ such that $\min(sk - P_{[b]} - P_{[c]}), SS_2 = \{J_{[b]}; J_{[c]}\}$, and go to Step 8. If none exists, place $J_{[a]}$ in $B_i$, $B_i = B_i \cup \{J_{[a]}\}$, $i = i + 1$, $B_i = B_i = \{\emptyset\}$, $sk = T$, $UJS = UJS \backslash \{J_{[a]}\}$, and go to Step 2.
- **Step 8:** If $Waste_2 = 0$, place the jobs of the SS_2 in the $B_i$, $B_i = B_i \cup SS_2 = B_i \cup \{J_{[b]}; J_{[c]}\}$, $i = i + 1$, $B_i = B_i = \{\emptyset\}$, $sk = T$, $UJS = UJS \backslash SS_2$, and go to Step 2.
Step 9: If Waste_3 > 0, select any three jobs, \( J[x], J[y], J[z] \); \( J[x], J[y], J[z] \) ∈ CJS such that \( \min (sk - p[x] - p[y] - p[z]) \geq 0 \). Let Waste_3 = \( sk - p[x] - p[y] - p[z] \) and \( SS_3 = \{ J[x], J[y], J[z] \} \), and go to Step 10. If none exists, Waste_3 = \( T \) and \( SS_3 = \{ \} \).

Step 10: Select the minimum waste from Waste_1, Waste_2, and Waste_3.

Case 1: If the minimum waste is equal to Waste_1, \( B_i = B_l \cup \{ J[x] \}, sk = sk - p[x], UJS = UJS \setminus \{ J[x] \} \), and go to Step 2.

Case 2: If the minimum waste is equal to Waste_2, \( B_i = B_l \cup SS_2, sk = sk - p[y] - p[z], UJS = UJS \setminus \{ J[y], J[z] \} \), and go to Step 2.

Case 3: If the minimum waste is equal to Waste_3, \( B_i = B_l \cup SS_3, sk = sk - p[x] - p[y] - p[z], UJS = UJS \setminus \{ J[x], J[y], J[z] \} \), and go to Step 2.

Step 11: Calculate makespan using \( C_{max} = (T + t) \cdot (i - 1) + \sum_{j \in B_l} p_j \) and stop.

5.4 Minimum Lower Bound (LB’). Select the job with minimum lower bound (LB’) from the given JS and place it in the first batch in which there is enough slack time to accommodate it, i.e., \( p_j \leq \text{slack time} \). Otherwise, create a new batch, and place the job in it.

To illustrate these packing policies, an 8-job instance is given in Table 3 where \( T \) and \( t \) are equal to 10 and 3, respectively. First, the jobs are sorted in nondecreasing order of their processing times, i.e., \( [J_1, J_2, J_3, J_4, J_5, J_6, J_7, J_8] \) and then applying four different packing policies to assign jobs into batches, the results are obtained as shown in Table 4. In Table 4, \( H (**) \) indicates that the heuristic algorithm adopts different bin-packing policies; for example, \( H (FB) \) means that the full-batch packing policy is used. Additionally, the MLPT heuristic, borrowed from the MFFD algorithm for the bin-packing problem [40], is proposed by Yu et al. [41] to solve the problem with periodic preventive maintenance activities.

Many recent studies [42, 43] have mentioned the benefits of interchange or insertion moves for the scheduling problems with makespan criterion, especially applying for generating neighborhood solutions [44]. We also propose two insertions and two interchanges for the considered problems in this paper. That is, once the initial schedule is obtained by one of the abovementioned bin-packing policies, the solution is further improved by the proposed local search technique including interchange and insertion moves. Our insertion method includes two insertion-type moves: job-type insertion and batch-type insertion.

5.5 Job-Type Insertion. With respect to Property 1, it implies that a schedule in which it is possible to find a batch \( B_i \) and a job \( j \) if job \( j \) satisfies the condition of \( p_j \leq T - \sum_{\ell \in B_l} p_j, j \in B_l \), \( B_i \) is scheduled before \( B_l \), and the operation of inserting job \( j \) into \( B_i \) would not lead to an increase in the makespan, as shown in Figure 2.

| Table 3: An instance with eight jobs. |
|--------------------------------------|
| \( J_1 \) | \( J_2 \) | \( J_3 \) | \( J_4 \) | \( J_5 \) | \( J_6 \) | \( J_7 \) | \( J_8 \) |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 4 | 8 | 2 | 1 | 8 | 8 |

| Table 4: Comparisons of different heuristic algorithms. |
|--------------------------------------|
| Schedule | \( C_{max} \) |
|-----------|---------------|
| \( H (FB) \) | \( (J_1, J_2, J_3, J_4, J_5, J_6, J_7, J_8) \) | 68 |
| \( H (BF) \) | \( (J_1, J_3, J_2, J_4, J_5, J_6, J_7, J_8) \) | 62 |
| \( H (MV) \) | \( (J_1, J_3, J_2, J_4, J_5, J_6, J_3, J_8) \) | 60 |
| \( H (LB) \) | \( (J_1, J_3, J_2, J_4, J_6, J_5, J_7, J_8) \) | 60 |
| MLPT | \( (J_1, J_3, J_2, J_4, J_6, J_5, J_7, J_8) \) | 60 |

Job-type insertion attempts to insert a job \( j \) into the front batches based on Property 1 are mentioned. The complete steps are described as follows:

Step 1: Let \( \pi \) be the initial schedule. Let \( k = 1 \), and \( K \) is equal to the number of batches.

Step 2: If \( k < K \), \( l = k + 1 \); otherwise, go to Step 5.

Step 3: If \( l \leq K \); otherwise, go to Step 4.

Step 3.1: If none of the jobs in the batch \( B_l \) satisfies the condition of \( p_j \leq sk_k \), \( j \in B_l \), \( sk_k = T - \sum_{\ell \in B_l} p_j \), \( l = l + 1 \), and go to Step 3.

Step 3.2: Remove job \( j \) from batch \( B_l \) and insert the job into batch \( B_k \) if the condition of \( p_j \leq sk_k \), \( j \in B_l \) is satisfied; \( \pi \) is a new schedule after insertion, and recalculate the value of \( K \) based on \( \pi \). \( \pi = \pi \) and go to Step 1.

Step 4: \( k = k + 1 \), and go to Step 2.

Step 5: Calculate makespan using \( C_{max} = (T + t) \cdot (K - 1) + \sum_{j \in B_l} p_j \).

5.6 Batch-Type Insertion. Formula \( C_{max} = (T + t) \cdot (K - 1) + \sum_{j \in B_l} p_j \), implies that the nonfull batch with the lowest load should be the last batch in the schedule; thus, the solution can be improved by moving the nonfull batch with the lowest load into the last position in the schedule, as shown in Figure 3. Batch-type insertion attempts to move the batch with minimum load into the last position in the sequence based on Property 2. The complete steps are described as follows:

Step 1: Let \( \pi \) be the initial sequence after implementing Job-type insertion and Job interchange, and the solution is equal to \( C_{max} \). Let \( k = 1 \), and \( K \) is equal to the number of batches.

Step 2: For \( k = 1 \) to \( K \), find the batch with minimum load and move the batch into the last position. Form a new schedule \( \pi \) without any unnecessary machine idle periods.

Step 3: Calculate makespan using \( C_{max} = (T + t) \cdot (K - 1) + \sum_{j \in B_l} p_j \), for the schedule \( \pi \).

Because each batch is as full as possible except for the last batch, the job-type insertion is aimed at doing that.
Additionally, we could also apply the interchange of jobs without increasing the makespan for the same purpose. Here are two cases.

Property 4. If \( p_j \leq p_i + s_{k_j} \) and \( p_j > p_i; i \in B_k, j \in B_l, k < l \), and then the interchange of \( i \) and \( j \) will not lead to an increase in \( C_{max} \) as shown in Figure 4.

Property 5. If \( p_j + p_{j'} \leq p_i + s_{k_j} \) and \( p_j + p_{j'} > p_i; i \in B_k, j, j' \in B_l, k < l \), then the interchange of \( (j, j') \) and \( i \) will not lead to an increase in \( C_{max} \), as shown in Figure 5.

The abovementioned two interchanges, namely, interone and intertwo, aim to make the former batches as full as possible, and the complete steps for the two interchanges are described as follows:

5.7. Interone Type

Step 1: Let \( \pi \) be the initial sequence. Let \( k = 1 \), and \( K \) is equal to the number of batches.

Step 2: If \( k < K, l = k + 1 \); otherwise, go to Step 5.

Step 3: If \( l \leq K \); otherwise, go to Step 4.

Step 3.1: If none of the jobs in batch \( B_l \) satisfies the conditions \( p_j \leq p_i + s_{k_j} \) and \( p_j > p_i, l = l + 1 \), go to Step 3.

Step 3.2: Interchange the jobs \( i, j \) if the conditions of \( p_j \leq p_i + s_{k_j} \) and \( p_j > p_i \) are satisfied, \( \pi' \) is a new schedule after interchange, \( \pi = \pi' \), and \( l = l + 1 \), go to Step 1.

Step 4: \( k = k + 1 \); go to Step 2.

Step 5: Calculate makespan using \( C_{max} = (T + t) \cdot (K - 1) + \sum_{j \in B_k} p_j \).

5.8. Intertwo Type

Step 1: Let \( \pi \) be the initial schedule. Let \( k = 1 \) and \( K \) is equal to the number of batches.

Step 2: If \( k < K, l = k + 1 \); otherwise, go to Step 5.

Step 3: If \( l \leq K \); otherwise, go to Step 4.

Step 3.1: If none of the jobs in batch \( B_l \) satisfies the conditions \( p_j + p_{j'} \leq p_i + s_{k_j} \) and \( p_j + p_{j'} > p_i \), \( l - l + 1 \) and go to Step 3.

Step 3.2: Interchange the two jobs \( j, j' \) with job \( i \) if the conditions of \( p_j + p_{j'} \leq p_i + s_{k_j} \) and \( p_j + p_{j'} > p_i \) are satisfied; \( \pi' \) is a new schedule after interchange, \( \pi = \pi' \), \( l = l + 1 \), and go to Step 1.

Step 4: \( k = k + 1 \); go to Step 2.
Step 5: Calculate makespan using $C_{\text{max}} = (T + t) \cdot (K - 1) + \sum_{j \in B_k} p_j$

Our proposed heuristic algorithms with the improvement procedure are outlined as follows:

Step 1: Obtain an initial schedule by one of the bin-packing policies.

Step 2: Improvement procedure:

Step 2.1: Implement job-type insertion to obtain an improved schedule based on the sequence of Phase 1.

Step 2.2: Implement interone interchange to obtain an improved schedule based on the sequence of Step 2.1.

Step 2.3: Implement intertwo interchange to obtain an improved schedule based on the sequence of Step 2.2.

Step 2.4: Implement batch-type insertion to obtain a final schedule based on the sequence of Step 2.3.

Step 3: Calculate $C_{\text{max}}$ and stop.

6. Computational Experiments

In this section, two experiments are conducted on a PC Xeon E5-1620 CPU operating at 3.6 GHz with 12 GB of RAM. Experiment 1 is aimed at comparing the computational efficiencies of the two MIP models. Experiment 2 focuses on comparing our proposed algorithms with other heuristic algorithms in the literature; we generated a set of problems randomly in which the setting is the same as that employed by Perez-Gonzalez and Framinan [16]. That is, the processing times are generated from a discrete uniform distribution between 1 and 50, the availability period of $T$ is from a discrete uniform distribution between 50 and 100, and the three different values of maintenance time $t$ are 10, 25, and 50. For each experimental condition, thirty instances are generated randomly. Therefore, our test-bed consists of 1260 problems considering $n \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 250, 300\}$.

6.1. Experiment 1: Comparison of the Two MIP Models. In this section, we compare our MIP model with the PF model of Perez-Gonzalez and Framinan [16]; the two models are implemented in IBM ILOG CPLEX Optimization Studio Version 12.7.1. We set a limit of 7200 seconds as the stopping criterion for the two models to solve each instance and then reported the number of optimal solutions obtained by the two models within the time limit and the average computation time for each problem size. Table 5 shows the numbers of optimal solutions obtained and average computation times spent by the two models for each problem; overall, the total numbers of optimal
solutions obtained by PF and our models are 857 and 1165, respectively. Furthermore, our model is much more efficient than the PF model; this supports our anticipation, that is, more constraints in our model are advantageous for reducing solution space, resulting in a greater efficiency of our model.

In Table 5, our model can give optimal solutions in all instances up to the 50-job problem; however, as the number of jobs increases, the performance of our model gradually worsens, although, fortunately, not like the rapid deterioration of the PF model.

6.2. Experiment 2: Comparison of the Heuristic Algorithms.
In this section, our proposed heuristic algorithms, namely, H\((MW)\) and H\((LB^*)\), are compared with the well-known heuristic algorithms in the literature, including H\((BF)\), H\((FB)\), and MLPT. All the heuristic algorithms are coded in C++. To compare the algorithms, relative percentage deviation (RPD) is used as a performance measure for each instance. RPD is calculated as follows:

\[
RPD = \frac{\text{Alg}_{\text{sol}} - \text{MIN}}{\text{MIN}} \times 100\%,
\]

where \(\text{Alg}_{\text{sol}}\) is the makespan obtained by a given algorithm for an instance, and \(\text{MIN}\) is the optimal or lower bound value obtained for a given instance.

Table 6 provides results of different heuristic algorithms for small problems. All optimal solutions could be obtained by our model. The columns of RPD (%) give average RPD values from the optimal solutions over 90 instances. We can find that the heuristic H\((MW)\) is the best heuristic among the five heuristics if the improvement procedure is not involved, as shown in Table 6. The mean RPD value of H\((MW)\) is 0.154%, and the heuristic H\((MW)\) on average finds 71.5% of the optimal solutions in the experiments. The results for the heuristic algorithms with improvement procedure are shown in Table 7. It is obvious that using the improvement procedure leads to all the heuristics achieving similar better performances, and the average percentages of optimal solutions found by H\((BF)\), H\((FB)\), MLPT, H\((MW)\), and H\((LB^*)\) are 96.9%, 96.5%, 96.95%, 99.1%, and 98.5%, respectively. According to Tables 7 and 8, the solutions of all the heuristics with the improvement procedure are refined evidently without large increase in expenses. Hence, the improvement procedure presented in this paper is a significant addition to the literature.

Regarding large problems, the solutions obtained by all the heuristics are compared with lower bounds; when the solution obtained by the heuristics is equal to the lower bound, the number of optimal solutions increases by 1, and we could then obtain the percentage of optimal solutions found by each heuristic. As revealed in Table 9, the performances of average RPD value and average percentage of

| $n$ | PF model | Our model | PF model | Our model |
|-----|-----------|------------|-----------|------------|
| 10  | 90        | 90         | 0.180     | 0.090      |
| 20  | 90        | 90         | 0.349     | 0.182      |
| 30  | 90        | 90         | 21.911    | 1.328      |
| 40  | 88        | 90         | 181.034   | 7.533      |
| 50  | 81        | 90         | 1001.149  | 11.741     |
| 60  | 72        | 88         | 1599.318  | 242.726    |
| 70  | 66        | 87         | 2054.072  | 313.883    |
| 80  | 54        | 83         | 3085.476  | 627.217    |
| 90  | 46        | 84         | 3870.221  | 607.719    |
| 100 | 36        | 81         | 4445.117  | 897.274    |
| 150 | 24        | 55         | 5404.562  | 3161.652   |
| 200 | 13        | 58         | 6290.459  | 2936.797   |
| 250 | 9         | 43         | 6621.720  | 4175.544   |
| 300 | 8         | 46         | 6665.626  | 4535.073   |
| 857 |           | 1165       |           |            |

| $n$ | H\((BF)\) | H\((FB)\) | MLPT | H\((MW)\) | H\((LB^*)\) | H\((BF)\) | H\((FB)\) | MLPT | H\((MW)\) | H\((LB^*)\) |
|-----|------------|------------|------|-----------|------------|-----------|------------|------|-----------|------------|
| 8   | 0.418      | 18.558     | 0.418| 0.017     | 0.065      | 85.6      | 0.0        | 85.6| 98.9      | 95.6      |
| 10  | 2.149      | 17.907     | 2.149| 0.069     | 1.208      | 36.7      | 1.1        | 36.7| 93.3      | 48.9      |
| 20  | 1.060      | 24.969     | 1.060| 0.218     | 1.074      | 23.3      | 0.0        | 23.3| 85.6      | 24.4      |
| 30  | 0.831      | 26.504     | 0.831| 0.460     | 0.734      | 21.1      | 0.0        | 21.1| 28.9      | 25.6      |
| 40  | 0.764      | 27.524     | 0.764| 0.064     | 0.473      | 22.2      | 0.0        | 22.2| 75.6      | 22.2      |
| 50  | 0.522      | 27.898     | 0.522| 0.095     | 0.335      | 13.3      | 0.0        | 13.3| 46.7      | 40.0      |
| Ave.| 0.957      | 23.893     | 0.957| 0.154     | 0.648      | 33.7      | 0.2        | 33.7| 71.5      | 42.8      |
optimalsolutionfoundforeachheuristicworsen;however,
heuristic H (MW) is still the best among them, and fur-
thermore, the results of Table 10 verified again that the
improvement procedure indeed significantly induces the
quality of the final solutions; it could lead heuristics to obtain
better solutions, and adding the improvement procedure
into each heuristic does not increase the computational
burden a lot, as shown in Table 11. From the general analysis
results, the heuristics with improvement procedure can
obtain very good solutions very fast, which indicates that the
heuristics with improvement procedure can be applied to
real-world problem.

| n  | H (BF) | H (FB) | MLPT | H (MW) | H (LB*) | H (BF) | H (FB) | MLPT | H (MW) | H (LB*) |
|----|--------|--------|------|--------|---------|--------|--------|------|--------|---------|
| 8  | 0.000  | 0.000  | 0.000| 0.000  | 0.000   | 0.000  | 0.000  | 0.000| 0.000  | 0.000   |
| 10 | 0.000  | 0.000  | 0.000| 0.000  | 0.000   | 0.000  | 0.000  | 0.000| 0.000  | 0.000   |
| 20 | 0.002  | 0.004  | 0.002| 0.000  | 0.002   | 0.989  | 0.978  | 0.989| 0.989  | 0.989   |
| 30 | 0.004  | 0.010  | 0.002| 0.000  | 0.000   | 0.967  | 0.922  | 0.978| 0.989  | 1.000   |
| 40 | 0.005  | 0.005  | 0.006| 0.002  | 0.002   | 0.933  | 0.944  | 0.922| 0.978  | 0.978   |
| 50 | 0.007  | 0.005  | 0.010| 0.004  | 0.005   | 0.922  | 0.944  | 0.922| 0.967  | 0.944   |
| Ave.| 0.003  | 0.004  | 0.003| 0.001  | 0.002   | 0.969  | 0.965  | 0.969| 0.991  | 0.985   |

| n  | H (BF) | H (FB) | MLPT | H (MW) | H (LB*) | H (BF) | H (FB) | MLPT | H (MW) | H (LB*) |
|----|--------|--------|------|--------|---------|--------|--------|------|--------|---------|
| 60 | 0.502  | 8.901  | 0.502| 0.043  | 0.257   | 10.0   | 0.0    | 10.0 | 65.6   | 23.3    |
| 70 | 0.750  | 30.030 | 0.750| 0.279  | 0.438   | 5.6    | 0.0    | 5.6  | 46.7   | 26.7    |
| 80 | 0.611  | 31.142 | 0.611| 0.337  | 0.462   | 3.3    | 0.0    | 3.3  | 34.4   | 17.8    |
| 90 | 0.512  | 30.545 | 0.512| 0.324  | 0.358   | 2.2    | 0.0    | 2.2  | 21.1   | 17.8    |
| 100| 0.450  | 30.994 | 0.450| 0.300  | 0.324   | 7.8    | 0.0    | 7.8  | 22.2   | 23.3    |
| 150| 0.374  | 31.294 | 0.374| 0.290  | 0.300   | 8.9    | 0.0    | 8.9  | 34.4   | 32.2    |
| 200| 0.303  | 32.360 | 0.303| 0.205  | 0.228   | 13.3   | 0.0    | 13.3 | 65.6   | 32.2    |
| 250| 0.319  | 31.324 | 0.319| 0.265  | 0.278   | 12.2   | 0.0    | 12.2 | 61.1   | 45.6    |
| 300| 0.197  | 31.147 | 0.197| 0.161  | 0.184   | 17.8   | 0.0    | 17.8 | 52.2   | 55.6    |
| Ave.| 0.446  | 28.637 | 0.446| 0.247  | 0.314   | 9.0    | 0.0    | 9.0  | 44.8   | 29.8    |
7. Conclusion

In this paper, we address the single-machine scheduling problem with periodic preventive maintenance activities. This problem is commonly encountered in the real world, especially for capital-intensive companies, where periodically implemented preventive maintenance is very important to avoid machine malfunctions. For the problem, there is a vast number of studies in the literature, and different approaches, including mixed integer programming formulations, dynamic programming methods, and heuristic algorithms, are developed for obtaining optimal or near-optimal solutions. Inspired by the study of Perez-Gonzalez and Framinan [16] and the bin-packing problem, the constraint of the minimum bins (or batches) property added into our MIP model makes it more efficient; this is a novel contribution to the single-machine problem with periodic maintenance activities.

We also proposed two heuristic algorithms, H (MW) and H (LB*), in which two different bin-packing policies are used. As the results revealed, H (MW) and H (LB*) outperformed other heuristic algorithms. We further developed improvement procedures and embedded them into all the heuristic algorithms of the paper. Based on the results of a comprehensive experiment, the performances of the heuristic algorithms with improvement procedure could be improved evidently with a slight increase of computational effort, and after proceeding with the improvement procedure, the solution qualities are not significantly different among all the heuristic algorithms. The results help us conclude that the improvement procedure is very effective and robust regardless of the original heuristic algorithms that this paper used. The research can continue the three directions: (1) modify our proposed algorithms to be suitable for solving general problems such as parallel machine, flowshop, jobshop, and openshop problems; (2) consider different objective functions for the single problem; and (3) multiple objectives can also be investigated for this single-machine scheduling problem with periodic maintenance activities.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant nos. 51705370 and 71501143) and the Zhejiang Province Natural Science Foundation of China (Grant nos. LY18G010012 and LY19G010007).

Supplementary Materials

Supplementary Materials include two folders: In Data folder, the file name as HNXXX.TXT is given, where XXX identifies the number of jobs which is an input data for the proposed heuristic algorithm, and another name MXXX.TXT is for the mathematical programming model, and in Result folder, the results obtained by the proposed algorithm are identified as AAA-JXXX.TXT, where AAA is the algorithm, and JXXX is the number of jobs. (Supplementary Materials)

References

[1] X. Zhao, C. Qian, and T. Nakagawa, “Comparisons of replacement policies with periodic times and repair numbers,” Reliability Engineering & System Safety, vol. 168, pp. 161–170, 2017.
[2] X. Zhao and T. Nakagawa, “Over-time and over-level replacement policies with random working cycles,” Annals of Operations Research, vol. 244, no. 1, pp. 103–116, 2016.
[3] X. Zhao and T. Nakagawa, “Optimization problems of replacement first or last in reliability theory,” European Journal of Operational Research, vol. 223, no. 1, pp. 141–149, 2012.
[4] C.-Y. Lee, “Machine scheduling with an availability constraint,” Journal of Global Optimization, vol. 9, no. 3-4, pp. 395–416, 1996.
[5] T. C. E. Cheng and G. Wang, “An improved heuristic for two-machine flowshop scheduling with an availability constraint,” Operations Research Letters, vol. 26, no. 5, pp. 223–229, 2000.
[6] Y. He, W. Zhong, and H. Gu, ”Improved algorithms for two single machine scheduling problems,” Theoretical Computer Science, vol. 363, no. 3, pp. 257–265, 2006.
[7] J. J. Liao and W. J. Chen, “Single-machine scheduling with periodic maintenance and nonresumable jobs,” Computers & Operations Research, vol. 30, no. 9, pp. 1335–1347, 2003.
[8] C. Sadfi, B. Penz, C. Rapine, J. Blæzewicz, and P. Formanowicz, “An improved approximation algorithm for the single machine total completion time scheduling problem with availability constraints,” European Journal of Operational Research, vol. 161, no. 1, pp. 3–10, 2005.
[9] M. Ji, Y. He, and T. C. E. Cheng, “Single-machine scheduling with periodic maintenance to minimize makespan,”

| $n$  | $H(BF)$ | $H(FB)$ | MLPT | $H(MW)$ | $H(LB^*)$ | $H^2(BF)$ | $H^2(FB)$ | MLPT$^4$ | $H^2(MW)$ | $H^2(LB^*)$ |
|------|---------|---------|-------|----------|-----------|-----------|-----------|---------|-----------|-----------|
| 60   | 0.000   | 0.000   | 0.000 | 0.001    | 0.000     | 0.005     | 0.001     | 0.000   | 0.001     | 0.000     |
| 70   | 0.000   | 0.000   | 0.000 | 0.001    | 0.000     | 0.008     | 0.002     | 0.000   | 0.002     | 0.001     |
| 80   | 0.000   | 0.000   | 0.000 | 0.002    | 0.000     | 0.011     | 0.003     | 0.000   | 0.003     | 0.001     |
| 90   | 0.000   | 0.000   | 0.000 | 0.003    | 0.000     | 0.019     | 0.004     | 0.000   | 0.003     | 0.001     |
| 100  | 0.000   | 0.000   | 0.000 | 0.003    | 0.000     | 0.024     | 0.005     | 0.000   | 0.005     | 0.001     |
| 150  | 0.000   | 0.000   | 0.000 | 0.010    | 0.000     | 0.116     | 0.014     | 0.000   | 0.014     | 0.004     |
| 200  | 0.000   | 0.000   | 0.000 | 0.023    | 0.000     | 0.381     | 0.069     | 0.000   | 0.030     | 0.008     |
| 250  | 0.000   | 0.000   | 0.000 | 0.042    | 0.000     | 0.718     | 0.121     | 0.000   | 0.055     | 0.016     |
| 300  | 0.000   | 0.000   | 0.000 | 0.071    | 0.000     | 1.779     | 0.188     | 0.000   | 0.094     | 0.027     |
12 Mathematical Problems in Engineering

Computers & Operations Research, vol. 34, no. 6, pp. 1764–1770, 2007.

[10] W. J. Chen, "Scheduling of jobs and maintenance in a textile company," The International Journal of Advanced Manufacturing Technology, vol. 31, pp. 737–742, 2007.

[11] I. Kacem and C. Chu, "Efficient branch-and-bound algorithm for minimizing the weighted sum of completion times on a single machine with one availability constraint," International Journal of Production Economics, vol. 112, no. 1, pp. 138–150, 2008.

[12] M. Sbihi and C. Varnier, "Single-machine scheduling with periodic and flexible periodic maintenance to minimize maximum tardiness," Computers & Industrial Engineering, vol. 55, no. 4, pp. 830–840, 2008.

[13] F. Angel-Bello, A. Alvarez, J. Pacheco, and I. Martinez, "A single machine scheduling problem with availability constraints and sequence-dependent setup costs," Applied Mathematical Modelling, vol. 35, no. 4, pp. 2041–2050, 2011.

[14] J. Moncel, J. Thierry, and A. Waserhole, "Computational performances of a simple interchange heuristic for a scheduling problem with an availability constraint," Computers & Industrial Engineering, vol. 67, pp. 216–222, 2014.

[15] W. Jia, Z. Jiang, and Y. Li, "Scheduling to minimize the makespan in large-piece one-of-a-kind production with machine availability constraints," Expert Systems With Applications, vol. 42, no. 23, pp. 9174–9182, 2015.

[16] P. Perez-Gonzalez and J. M. Framanan, "Single machine scheduling with periodic machine availability," Computers & Industrial Engineering, vol. 123, pp. 180–188, 2018.

[17] X. Qi, T. Chen, and F. Tu, "Scheduling the maintenance on a single machine," The Journal of the Operational Research Society, vol. 50, no. 10, pp. 1071–1078, 1999.

[18] J. S. Chen, "Single-machine scheduling with flexible and periodic maintenance," Journal of the Operational Research Society, vol. 57, no. 6, pp. 703–710, 2006.

[19] G. Mosheiov and A. Sarig, "Scheduling a maintenance activity to minimize total weighted completion-time," Computers & Mathematics with Applications, vol. 57, no. 4, pp. 619–623, 2009.

[20] C. Low, C.-J. Hsu, and C.-T. Su, "A modified particle swarm optimization algorithm for a single-machine scheduling problem with periodic maintenance," Expert Systems with Applications, vol. 37, no. 9, pp. 6429–6434, 2010.

[21] C. Low, M. Ji, C.-J. Hsu, and C.-T. Su, "Minimizing the makespan in a single machine scheduling problems with flexible and periodic maintenance," Applied Mathematical Modelling, vol. 34, no. 2, pp. 334–342, 2010.

[22] S.-L. Yang, Y. Ma, D.-L. Xu, and J.-B. Yang, "Minimizing total completion time on a single machine with a flexible maintenance activity," Computers & Operations Research, vol. 38, no. 4, pp. 755–770, 2011.

[23] K.-C. Ying, C.-C. Lu, and J.-C. Chen, "Exact algorithms for single-machine scheduling problems with a variable maintenance," Computers & Industrial Engineering, vol. 98, pp. 427–433, 2016.

[24] A. Costa, F. A. Cappadonna, and S. Fichera, "Total tardiness minimization in a parallel machine system with flexible periodic maintenance," Journal of Industrial and Production Engineering, vol. 33, no. 7, pp. 485–494, 2016.

[25] J. Yoo and I. S. Lee, "Parallel machine scheduling with maintenance activities," Computers & Industrial Engineering, vol. 101, pp. 361–371, 2016.

[26] L.-H. Su and H.-M. Wang, "Minimizing total absolute deviation of job completion times on a single machine with cleaning activities," Computers & Industrial Engineering, vol. 103, pp. 242–249, 2017.

[27] M. A. Kubzin and V. A. Strusevich, "Planning machine maintenance in two-machine shop scheduling," Operations Research, vol. 54, no. 4, pp. 789–800, 2006.

[28] D. Xu, Y. Yin, and H. Li, "Scheduling jobs under increasing linear machine maintenance time," Journal of Scheduling, vol. 13, no. 4, pp. 443–449, 2010.

[29] W. Luo, T. C. E. Cheng, and M. Ji, "Single-machine scheduling with a variable maintenance activity," Computers & Industrial Engineering, vol. 79, pp. 168–174, 2015.

[30] S. Gawiejnowicz, Time-Dependent Scheduling, Springer-Verlag, New York, NY, USA, 2008.

[31] A. Janiak and R. Rudek, "Scheduling problems with position dependent job processing time," in Scheduling in Computer and Manufacturing Systems, A. Janiak, Ed., pp. 26–32, Springer, Warszawa, Poland, 2006.

[32] M. Ji, Y. He, and T. C. E. Cheng, "Scheduling linear deteriorating jobs with an availability constraint on a single machine," Theoretical Computer Science, vol. 362, no. 1–3, pp. 115–126, 2006.

[33] W.-C. Lee and C.-C. Wu, "Multi-machine scheduling with deteriorating jobs and scheduled maintenance," Applied Mathematical Modelling, vol. 32, no. 3, pp. 362–373, 2008.

[34] D.-L. Yang, T. C. E. Cheng, S.-J. Yang, and C.-J. Hsu, "Unrelated parallel-machine scheduling with aging effects and multi-maintenance activities," Computers & Operations Research, vol. 39, no. 7, pp. 1458–1464, 2012.

[35] S.-J. Yang, "Unrelated parallel-machine scheduling with deteriorating effects and deteriorating multi-maintenance activities for minimizing the total completion time," Applied Mathematical Modelling, vol. 37, no. 5, pp. 2995–3005, 2013.

[36] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling; a survey," Annals of Discrete Mathematics, vol. 5, pp. 287–326, 1979.

[37] J.-S. Chen, "Scheduling of nonresumable jobs and flexible maintenance activities on a single machine to minimize makespan," European Journal of Operational Research, vol. 190, no. 1, pp. 90–102, 2008.

[38] W.-W. Cui, Z. Lu, B. Zhou, C. Li, and X. Han, "A hybrid genetic algorithm for non-permutation flow shop scheduling problems with unavailability constraints," International Journal of Computer Integrated Manufacturing, vol. 29, no. 9, pp. 944–961, 2016.

[39] S. Martello and P. Toth, Bin-Packing Problen. Algorithms and Computer Implementation, p. 221, John Wiley & Sons, New York, NY, USA, 1990.

[40] M. Yue and L. Zhang, "A simple proof of the inequality MFFD (L) ≤ 71/60 OPT (L)+1, L for the MFFD bin-packing algorithm," Acta Mathematicae Applicatae Sinica, vol. 11, no. 3, pp. 318–330, 1995.

[41] X. Yu, Y. Zhang, and G. Steiner, "Single-machine scheduling with periodic maintenance to minimize makespan revisited," Journal of Scheduling, vol. 17, no. 3, pp. 263–270, 2014.

[42] E. Talaard, "Some efficient heuristic methods for the flow shop sequencing problem," European Journal of Operational Research, vol. 47, pp. 67–74, 1990.

[43] D. Kizilay, M. F. Tasgetiren, Q.-K. Pan, and L. Gao, "A variable block Insertion Heuristic for solving permutation flow shop
scheduling problem with makespan criterion,” *Algorithms*, vol. 12, p. 100, 2019.

[44] I. Alharkan, K. Bamataraf, M. A. Noman et al., “An order effect of neighborhood structures in variable neighborhood search algorithm for minimizing the makespan in an identical parallel machine scheduling,” *Mathematical Problems in Engineering*, vol. 2018, Article ID 3586731, 8 pages, 2018.