Bi-objective optimization of identical parallel machine scheduling with flexible maintenance and job release times

Yarong Chen⁵, Zailin Guan⁵, Chen Wang⁶, Fuh-Der Chou⁶ and Lei Yue⁵*  

⁵School of Mechanical Science and Engineering, Huazhong University of Science and Technology, China  
⁶School of Mechanical and Electrical Engineering, Wenzhou University, Wenzhou 325035, Zhejiang, China  

ABSTRACT

This paper investigates an identical parallel machine scheduling problem with flexible maintenance and job release times and attempts to optimize two objectives: the minimization of the makespan and total tardiness simultaneously. A mixed-integer programming (MIP) model for solving small-scale instances is presented first, and then a modified NSGA-II (M-NSGA-II) algorithm is constructed for solving medium- and large-scale instances by incorporating several strategies. These strategies include: (i) the proposal of a decoding method based on dynamic programming, (ii) the design of dynamic probability crossover and mutation operators, and (iii) the presentation of neighborhood search method. The parameters of the proposed algorithm are optimized by the Taguchi method. Three scales of problems, including 52 instances, are generated to compare the performance of different optimization methods. The computational results demonstrate that the M-NSGA-II algorithm obviously outperforms the original NSGA-II algorithm when solving medium- and large-scale instances, although the time taken to solve the instances is slightly longer.

Keywords:  
Identical parallel machine scheduling  
Flexible maintenance  
Bi-objective optimization  
MIP  
M-NSGA-II

1. Introduction

Parallel machine scheduling (PMS) is an important branch of production scheduling (Cheng & Sin, 1990). In the literature on PMS, most studies assume that machines are available at all times (Liu et al., 2020; Kim et al., 2019). However, in real-world manufacturing, machines need to be maintained and hence may become unavailable during a certain period, which makes it highly important to jointly schedule production and maintenance tasks to simultaneously improve system availability and system throughput. The problem of scheduling jobs with maintenance reduces to the problem often referred to in the literature as scheduling with machine availability constraints. Lee and Chen (2000) first researched a parallel machine scheduling problem (PMSP) with maintenance in which each machine must be maintained once in the planning horizon, and they proposed branch and bound (B&B) algorithms based on the column generation approach. To date, many studies on various PMS with different preventive maintenance (PM) strategies have been carried out. Unfortunately, most research focuses on single-objective optimization, especially the makespan, and few studies focus on multiple objective optimization. Berrichi et al. (2009) researched the PMSP with machine reliability and simultaneously minimized the makespan and machine system unavailability. Based on this research, Wang and Liu (2015) proposed a multi-objective PMSP with two kinds of resources (machines and moulds) and flexible PM activities on resources, to simultaneously minimize the makespan, the machine system unavailability, and the mould system unavailability. With the development of intelligent manufacturing, customer satisfaction plays an increasingly vital role in both the manufacturing and service industries. On-time delivery is becoming increasingly important. Therefore, the bi-objective to simultaneously minimize the makespan and total tardiness is studied in this paper. In addition, most studies on the joint optimization of production scheduling and PM usually assume that jobs reach zero time.
However, in real-world manufacturing, it is very common for jobs to arrive at different times. Many PMSP studies consider job release times without considering integration optimization with PM (Kramer et al., 2020; Abedi et al., 2015; Nessah et al., 2008; Yalaoui & Chu, 2006). Cui and Lu (2017) studied the single machine scheduling problem with machine flexible maintenance and job release times. Chen et al. (2021) studied two identical parallel machines with flexible maintenance to minimize the makespan without considering the job's release time. To the best of our knowledge, the PMSP with PM and job release times has not been documented in the literature. In this research, we extended the number of machines from references (Cui & Lu, 2017; Chen et al., 2021) to multiple machines and considered job release times to simultaneously minimize the makespan and total tardiness. Our objective is to determine the assignment of jobs to the parallel machines, the decision of the maintenance activities, and the optimal sequence of jobs and PM on each machine. The MIP model and an M-NSGA-II are successfully customized to solve the considered problem.

The rest of the paper is organized as follows: the next section surveys the related literature. A problem description is presented in Section 3. The MIP model is proposed in Section 4. Section 5 provides details about the modified NSGA-II. The experimental results are demonstrated in Section 6, and finally, Section 7 includes conclusions and discusses future research.

2. Literature review

2.1. PMSP with preventive maintenance

PMSPs incorporated with PM can be divided into different models according to different criteria. The basic criterion is the number of maintenance tasks, including single maintenance (Xu et al., 2009; Wang & Wei, 2011) or multiple maintenance (Hashemian et al., 2014; Li et al., 2017; Xu et al., 2008). In addition, maintenance may be performed on one of the parallel machines or on all parallel machines. Some earlier studies assumed that maintenance is only performed on some parallel machines (Tan et al., 2011) or only once within the planning horizon (Lee & Chen, 2008; Yoo & Lee, 2016). Recent research has generally assumed that all parallel machines have multiple maintenance, and the related research is divided into three kinds: fixed interval maintenance, flexible maintenance, and variable maintenance in the form of maintenance activities. Fixed interval maintenance involves periodically performing fixed time maintenance according to a fixed time interval. Yoo and Lee (2016) studied the PMSP in which each machine requires maintenance activity once within a given time window. Li et al. (2017) studied the PMSP in which each machine is subject to periodic maintenance. Xu et al. (2009) considered two PMSPs in which one machine is periodically unavailable. Flexible maintenance is fixed time maintenance within a flexible time window or upper threshold. Xu et al. (2008) studied a PMSP with ε-almost periodic maintenance activities to minimize the makespan. Sun and Li (2010) studied a similar problem and assumed that the largest consecutive processing time for each machine cannot exceed an upper limit T. Chen et al. (2021) studied two identical PMSPs with flexible maintenance to minimize the makespan. Variable maintenance is a kind of maintenance in which the maintenance interval and time depend on the condition of the machines and/or jobs. Wang and Wei (2011) studied PMSPs with deteriorating maintenance activity, that is, when delaying maintenance increases the time required to perform it. Wu et al. (2020) considered the dispatching-dependent deterioration of machines and machine-health-dependent production rates and proposed a dynamic dispatching and PM model to minimize the weighted long-run average waiting costs of MTO systems. Moradi and Zandieh (2010) studied the joint production and maintenance problem of parallel machines, and they concluded that the availability of machines is related to the failure rate and repair rate.

2.2. Multi-objective optimization of PMSP

Instead of a single objective, multiple but conflicting objectives are often considered when a manager executes production scheduling. Thus, a Pareto front consisting of a set of non-dominated solutions is required so that the manager can choose one of the alternatives from the set while preparing the production plan. However, compared with a single objective, such as minimizing the makespan, multi-objective optimization of PMSPs is rarely studied. According to the number of objectives, this problem can be divided into two types: bi-objective and multi-objective. For bi-objective optimization, in the context of green manufacturing, minimizing the energy or power consumption and cost is an important objective; therefore, the objective of this problem can be divided into two types: bi-objective and multi-objective. For bi-objective optimization, in the context of green manufacturing, minimizing the energy or power consumption and cost is an important objective; therefore, the objective of this problem can be divided into two types: bi-objective and multi-objective. In addition, the bi-objective to minimize the makespan and the total weighted earliness and tardiness of jobs (Abedi et al., 2015) as well as the makespan and the system unavailability (Berrichi et al., 2009), Moradi and Zandieh (2010) can also be found in the literature. For multiple objective optimizations, based on the literature (Berrichi et al., 2009), the bi-objective is extended to three; Wang and Liu (2015) proposed a multi-objective PMSP with two kinds of resources (machines and moulds) and with flexible preventive maintenance activities on resources, and presented an integrated optimization method with NSGA-II adaptation. Bandyopadhyay and Bhattacharya (2013) proposed a modified version of NSGA-II with a new mutation algorithm for a PMSP with three objectives. Cochran et al. (2003) proposed a two-stage multi-population genetic algorithm to solve PMSPs with multiple objectives.

2.3. Solution methodology of the PMSP

The solutions to PMSP generally include two decisions: one assigns jobs to machines, and the second sorts of the jobs assigned to the machine. These two decisions can be made in parallel, i.e., allocation first and then sequencing, or sequencing can be performed during assignment. The solution method for PMSP in the literature mainly includes the exact method, rule-based heuristics, the meta-heuristic algorithm and the intelligent evolutionary algorithm.
The exact method can obtain the optimal solution to the problem, but it is only suitable for small-scale instances. The exact methods proposed for solving the PMSP are mainly the B&B algorithm (Nessah et al., 2008; Yalaloui & Chu, 2006) and mathematical models (Wu & Wang, 2018; Naderi & Roshanaei, 2020). To improve the efficiency of the exact models, the lower bounds and the dominance properties have been incorporated into the B&B algorithm. In addition, Alhadi et al. (2020) presented a polynomial-time approximation scheme to generate an approximate Pareto Frontier that minimizes the maximum lateness and makespan. Rule-based heuristics are widely used in the real-world manufacturing industry (Lee, 2018). The rules proposed for solving the problem \(Pm//C_{\text{max}}\) include the LPT, MULTIFIT, COMBINE, and LISTFIT (Gupta & Ruiz-Torres, 2001); those proposed for the problem \(Pm//\sum T_j\) include the EDD, SPT, TPI, MDD, KPM, Minimum Slack and BHG (Biskup et al., 2008); and those proposed for the problem \(Pm//\sum w_j T_j^2\) are the QB and QBP procedures (Schaller & Valente, 2018).

Considering the job’s release time, Akturk and Ozdemir (2001) proposed some rules, namely, the ATC (apparent tardiness cost), X-RM (X-dispatch ATC), KZRM (Kanet and Zhou approach to ATC), and COVERT (weighted cost over time), for the problem \(P_m/1/r\). Let \(d_{ij}\) be the due date of \(J_{ij}\); then, the tardiness of \(J_{ij}\) is calculated by the equation \(T_i = \max \{C_{i[k]} - d_{ij}, 0\}\). Let \(B_{ib}\) be the set of jobs of the \(b^{th}\) batch in machine \(M_i\), i.e., \(B_{ib} = \{J_{ij|1}, \ldots, J_{ij|k}\}\), and \(|B_{ib}| = k\). Let \(T_P\) be the sum processing time of the jobs in batch \(B_{ib}\), i.e., \(T_P = \sum_{J_{ij|k} \in B_{ib} \cup r_i} P_i\). \(P_{ib}\) is the \(ib^{th}\) PM of machine \(M_i\), and \(C_{ij|k}\) is the makespan of machine \(M_i\).

![Fig. 1. A Gantt chart of a solution for the identical PMSP with m machines and n jobs](image)

**Theorem 1.** Problem P: \(Pm//r_j, nr, FPM/(C_{\text{max}} \sum T_j)\) is NP-Hard.

**Proof.** The sequencing problem of each machine in \(Pm//r_j, nr, FPM/(C_{\text{max}} \sum T_j)\) is equivalent to the bin-packing problem. Because both the bin-packing problem and the \(P2//C_{\text{max}}\) problem have been proven to be NP-Hard, the problem \(P2//nr, FPM/C_{\text{max}}\) is NP-Hard. In addition, the problem \(1/r_j/\sum T_j\) is NP-Hard (Nessah et al., 2008); thus, \(Pm/r_j/\sum T_j\) is NP-Hard. The problem \(Pm//r_j, nr, FPM/(C_{\text{max}} \sum T_j)\) extends the number of identical parallel machines from 2 to \(m\) and considers the constraint of the job’s release time and the objective of minimizing total tardiness; thus, problem \(Pm//r_j, nr, FPM/(C_{\text{max}} \sum T_j)\) is NP-Hard.

**Definition:** Full loading (FL) means that the cumulative processing time of the jobs during two consecutive PMs, or the sum processing time of the jobs in a batch, is equal to the maintenance threshold UT. When \(TP_{ib} = UT\), batch \(B_{ib}\) is fully loaded; otherwise, batch \(B_{ib}\) is not fully loaded, and the waste time is equal to \(UT - TP_{ib}\).

4. **MIP model**

Based on the following assumptions, an MIP model for the problem studied in this paper is established: (1) each job has a release time; (2) the processing time, release time and due date of the job are known in advance; (3) the maintenance threshold and maintenance time are known in advance; (4) the time allocated to any machine for processing the same job is the same; (5) each machine can only process one job at any given time and shall maintain continuous processing, which shall not be
pre-empted and interrupted by other jobs; and (6) there is no job importance, that is, the weight is not considered. The input parameters and decision variables are introduced before developing the proposed model.

**Input Parameters**

- $J$: set of jobs
- $M$: set of machines
- $j, k$: job index from set $J$
- $i$: machine index from set $M$
- $p_j$: the processing time of job $J_j$
- $r_j$: the release time of job $J_j$
- $d_j$: the due date of job $J_j$
- $C_j$: the completion time of job $J_j$
- $ST_{ik}$: the start time at the $k^{th}$ position of machine $M_i$
- $Q_{ik}$: the cumulative processing time at the $k^{th}$ position of machine $M_i$
- $CT_{ik}$: the completion time of machine $M_i$ in position $k$
- $UT$: threshold of the cumulative processing time that the machine can process continuously
- $m_t$: time required for maintenance
- $L$: a sufficiently large positive integer, it is safe to set $L = \sum p_j + n \times m_t$

**Decision variables**

- $X_{ijk}$: equal to 1 if job $J_j$ is processed at the $k^{th}$ position of machine $M_i$, 0 otherwise
- $Y_{ik}$: equal to 1 if maintenance is conducted after the $k^{th}$ position of machine $M_i$, 0 otherwise

**Mathematical model**

\[
\begin{align*}
\text{min. } & C_{\text{max}} \\
\text{min. } & \sum_{i=1}^{n} T_i \\
\sum_{j=1}^{n} X_{ijk} & \leq 1, \forall i \in \{1, 2, 3, \ldots, m\} \quad (3) \\
\sum_{i=1}^{m} \sum_{k=1}^{n} X_{ijk} & = 1, \forall j \in \{1, 2, 3, \ldots, n\} \quad (4) \\
Y_{im} &= 0, \forall i \in \{1, 2, 3, \ldots, m\} \quad (5) \\
ST_{ik} & \geq \sum_{j=1}^{n} r_j \times X_{ijk}, \forall \{i = 1, 2, 3, \ldots, m; k = 2, 3, \ldots, n\} \quad (6) \\
CT_{ik} & \geq ST_{ik} + \sum_{j=1}^{n} p_j \times X_{ijk}, \forall \{i = 1, 2, 3, \ldots, m; k = 2, 3, \ldots, n\} \quad (8) \\
Q_{i1} &= \sum_{j=1}^{n} p_j \times X_{ij1}, \forall i = 1, 2, 3, \ldots, m \quad (9) \\
\left\{Q_{i(k-1)} + \sum_{j=1}^{n} p_j \times X_{ijk} \leq Q_{ik} + L \times Y_{i(k-1)} \right\}, \forall \{i = 1, 2, 3, \ldots, m; k = 2, 3, \ldots, n\} \quad (10) \\
Q_{ik} & \leq UT, \forall \{i = 1, 2, 3, \ldots, m; k = 1, 2, 3, \ldots, n\} \quad (11) \\
\sum_{j=1}^{n} X_{ijk} & \leq \sum_{j=1}^{n} X_{ij(k-1)}, \forall \{i = 1, 2, 3, \ldots, m; k = 2, 3, \ldots, n\} \quad (12) \\
C_{\text{max}} & \geq CT_{in}, \forall i = 1, 2, 3, \ldots, n \quad (13) \\
C_j & \geq CT_{ik} + L \times (X_{ijk} - 1), \forall \{i = 1, 2, 3, \ldots, m; j = 1, 2, 3, \ldots, n; k = 1, 2, 3, \ldots, n\} \quad (14) \\
\{T_j \geq C_j - d_j, \forall j = 1, 2, 3, \ldots, n\} \quad (15)
\end{align*}
\]

In this model, the objectives are to minimize $C_{\text{max}}$ and $\sum T_j$ as shown in Eq. (1) and Eq. (2). Constraints (3) and (4) ensure that each job can only be processed at one position of one machine and that each position of one machine can only be occupied by one job. Constraint (5) specifies that no PM is required after the last position of each machine. Constraints (6) and (7) limit the start time of each machine at each position. Constraint (8) defines the completion time of each machine at each position. Constraint (9) describes the cumulative processing time of each machine at the first position. Constraint (10) defines the cumulative processing time of each machine at each position. Constraint (11) ensures that the cumulative processing time of each machine at each position is not larger than UT. Constraint (12) ensures that the job is arranged in a more forward position.
Constraint (13) defines the maximum completion time. Constraint (14) defines the completion time of each job. Constraint (15) defines the tardiness of each job.

5. M-NSGA-II algorithm

For the proposed problem, an M-NSGA-II algorithm is presented by incorporating three strategies. (i) A decoding method based on dynamic programming (DP) is constructed. (ii) Dynamic probability crossover and mutation are designed. (iii) Neighbourhood search method is proposed. The general operation of the M-NSGA-II algorithm is shown in Algorithm 1.

Algorithm 1: M-NSGA-II algorithm
1 Create an initial population \( P_0 \) of size \( \text{Popsize} \) using a random rule
2 Decode the individual using the method based on DP, sort the population based on the fast-non-dominated sorting method
3 Create offspring population \( Q_0 \) by applying the dynamic probability crossover and mutation operators as well as elitist operators
4 While stopping criteria are not verified, do
   Create population \( R_{t+1} = P_t \cup Q_t \) of size \( 2 \text{Popsize} \) and construct the different \( F_i \) of \( R_t \) using the fast-non-dominated sorting procedure
   Put \( P_{t+1} = \emptyset \) and \( i = 0 \)
   While \( |P_{t+1}| + |F_i| < \text{Popsize} \) do
      \( P_{t+1} = P_{t+1} \cup F_i \)
      \( i = i + 1 \)
   End While
   Include in \( P_{t+1} \) the \( (\text{Popsize} - |P_{t+1}|) \) of \( F_i \) according to the crowding procedure
   Renew population \( P_{t+1} \) with neighbourhood search method
   Create offspring population \( Q_{t+1} \) by selection, crossover, and mutation with dynamic probability
End While

5.1. Chromosome representation

According to the decision of the proposed problem, the presentation of the chromosome is a vector with length \( n + m - 1 \), where \( n \) is the number of jobs, and \( m \) is the number of machines. An example is \( J_1[1], \ldots, J_1[n], 0, \ldots, 0, J_m[1], \ldots, J_m[m] \), where the jobs allocated to different machines are distinguished by 0, and the string of numbers divided by 0 represents the sequence of the jobs processed on one machine.

5.2. Decoding method

For the solution \( \pi = \{J_1[1], \ldots, J_1[n], 0, \ldots, 0, J_m[1], \ldots, J_m[m]\} \), although the job allocation and sequence on each machine have been determined according to the position of 0 in the chromosome, the maintenance or the job-grouping batch decision still needs to be made. For the identical PMSP without considering the job’s release time to minimize \( C_{\text{max}} \), a near-optimal solution can be obtained by a maintenance decision made based on the FL principle as shown in Appendix 1. However, it is not true for the same problem considering the job release times. An example with five jobs on a single machine is used to explain, and the jobs data is shown in Table 1. Assume the solution \( \pi = \{J_1, J_2, J_3, J_4, J_5\} \), \( UT = 10 \), and \( mt = 2 \). The Gantt chart of the solution obtained based on the FL and non-FL principle is shown in Fig. 2. The solution obtained without considering the FL is better than that obtained with the FL. The reason is that machine have to be idle for a period to be fully loaded. For the identical PMSP considering the job release times, the maintenance decision is a dynamic decision, and we propose a decoding method based on dynamic programming (DP process (Zhou et al., 2018)). In each stage, the different job-grouping batch schemes of a newly added job and the currently sorted job are compared, and the best optimal solution is selected as the final decoding result.

Table 1
Basic information for 5 jobs example

<table>
<thead>
<tr>
<th>Job</th>
<th>( J_1 )</th>
<th>( J_2 )</th>
<th>( J_3 )</th>
<th>( J_4 )</th>
<th>( J_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( j_1 )</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>( j_2 )</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>( j_3 )</td>
<td>3</td>
<td>4</td>
<td>28</td>
<td>17</td>
<td>16</td>
</tr>
</tbody>
</table>

Fig. 2. Solution obtained by different decoding methods for the example
To decode the solution based on the DP method, the individual \( \pi \) is first divided into \( m \) segments, representing \( m \) machines, and then the solution is decoded for each machine by the Algorithm 2. When decoding the job sequence on machine \( M_i \), the set of the first \( k \) jobs is \( \Delta^k = (J_{[1]'}, ..., J_{[k]'}) \), and the last \( g \) jobs in the set \( \Delta^k \) are \( \Psi^k = (J_{[k+1]'}, ..., J_{[k+g+1]'}) \). \( Z_0(\Delta^k) \) and \( Z_1(\Delta^k) \) represent the makespan and total tardiness of \( k \) jobs in a given sequence, respectively, so \( \Delta^0 = \emptyset \), \( Z_0(\Delta^0) = 0 \), \( Z_1(\Delta^0) = 0 \). The makespan and the total tardiness of \( k \) sequenced jobs are represented by the state function of the \( k \)th stage as shown in Eq. (16); the calculation method for the makespan and total tardiness is shown in Eq. (17) and Eq. (18).

\[
Z(\Delta^k) = (Z_0(\Delta^k), Z_1(\Delta^k)) = \min_{1 \leq g \leq k} \{ f_{g}^0(\Psi^k, Z_0(\Delta^{k-g})), \ f_{g}^1(\Psi^k, Z_1(\Delta^{k-g})) \} = \begin{cases} 
\max\{Z_0(\Delta^0), r_{[k]}\} + p_{[k]} & g = 1 \text{ and } k = g \\
\max\{Z_0(\Delta^{k-g}) + mt, r_{[k]}\} + p_{[k]} & g = 1 \text{ and } k > g \\
\max\{f_{g-1}^0(\Psi^k_{g-1}, Z_0(\Delta^{k-g-1})), r_{[k]}\} + p_{[k]} & g > 1 \text{ and } \sum_{\rho \in \Psi^k} p_{\rho} \leq UT \\
\infty & g > 1 \text{ and } \sum_{\rho \in \Psi^k} p_{\rho} > UT \end{cases} 
\]

\[
f_{g}^0(\Psi^k, Z_0(\Delta^{k-g})) = \begin{cases} 
Z_1(\Delta^{k-g}) + \max\{f_{g}^0(\Psi^k, Z_0(\Delta^{k-g})), d_{[k]}.0\} & g = 1 \\
f_{g-1}^0(\Psi^k_{g-1}, Z_1(\Delta^{k-g})), \max\{f_{g}^0(\Psi^k, Z_0(\Delta^{k-g})), d_{[k]}.0\} & g > 1 \text{ and } \sum_{\rho \in \Psi^k} p_{\rho} \leq UT \\
\infty & g > 1 \text{ and } \sum_{\rho \in \Psi^k} p_{\rho} > UT \end{cases} 
\]

\[
f_{g}^1(\Psi^k, Z_1(\Delta^{k-g})) = \begin{cases} 
Z_1(\Delta^{k-g}) + \max\{f_{g}^1(\Psi^k, Z_1(\Delta^{k-g})), d_{[k]}.0\} & g = 1 \\
f_{g-1}^1(\Psi^k_{g-1}, Z_1(\Delta^{k-g})), \max\{f_{g}^1(\Psi^k, Z_1(\Delta^{k-g})), d_{[k]}.0\} & g > 1 \text{ and } \sum_{\rho \in \Psi^k} p_{\rho} \leq UT \\
\infty & g > 1 \text{ and } \sum_{\rho \in \Psi^k} p_{\rho} > UT \end{cases} 
\]

Algorithm 2: DP decoding algorithm

0 Initialize the completion time set of machines \( MC = \emptyset \), \( \Delta^0 = \emptyset \), \( Z_0(\Delta^0) = 0 \), \( Z_1(\Delta^0) = 0 \), UT, and \( mc \)
1 For chromosome \( h = 1 \) to \( Popsize \) do
2 For machine \( i = 1 \) to \( m \) do
3 For the machine’s job position \( k = 1 \) to \( n_i \), \( \Delta^k = (J_{[1]'}, ..., J_{[k]'}) \)
4 For \( g = 1 \) to \( k \), \( \Psi^k = (J_{[k+1]'}, ..., J_{[k+g+1]'}) \)
5 // Compute the makespan and total tardiness
6 If \( g = 1 \)
7 If \( k = g \), then \( Z_0(\Delta^k) = f_{g}^0(\Psi^k, Z_0(\Delta^{k-g})), Z_1(\Delta^k) = \max\{Z_0(\Delta^0), r_{[k]}\} + p_{[k]} \)
8 Else if \( k > g \), then
9 \( Z_0(\Delta^k) = f_{g}^0(\Psi^k, Z_0(\Delta^{k-g})), Z_1(\Delta^k) = \max\{Z_0(\Delta^{k-g}) + mt, r_{[k]}\} + p_{[k]} \)
10 If \( g > 1 \)
11 If \( \sum_{\rho \in \Psi^k} p_{\rho} \leq UT \), then
12 \( Z_0(\Delta^k) = f_{g}^0(\Psi^k, Z_0(\Delta^{k-g})), Z_1(\Delta^k) = \max\{f_{g-1}^0(\Psi^k_{g-1}, Z_0(\Delta^{k-g-1})), r_{[k]}\} + p_{[k]} \)
13 \( Z_1(\Delta^k) = f_{g}^1(\Psi^k, Z_1(\Delta^{k-g})), Z_1(\Delta^k) = \max\{f_{g}^1(\Psi^k, Z_0(\Delta^{k-g})), d_{[k]}.0\} \)
14 Else if \( \sum_{\rho \in \Psi^k} p_{\rho} > UT \), then
15 \( Z_0(\Delta^k) = f_{g}^0(\Psi^k, Z_0(\Delta^{k-g})), Z_1(\Delta^k) = \max\{f_{g-1}^0(\Psi^k_{g-1}, Z_1(\Delta^{k-g-1})), r_{[k]}\} + p_{[k]} \)
16 \( Z_1(\Delta^k) = f_{g}^1(\Psi^k, Z_1(\Delta^{k-g})), Z_1(\Delta^k) = \infty \)
7 End For
8 End For
9 \( C_i = Z_0(\Delta^k), MC = MC \cup \{C_i\}, T_i = Z_1(\Delta^k) \)
10 End For
11 \( C_{max} = \max\{C_i, C_i \in MC\}, TT = \sum_{i=1}^{m} T_i \)
12 End For

5.3. Neighbourhood Search operators

For the identical PMSP, it is easy to generate redundant solutions in the population because of the symmetry of the machines. For example, the solution \( \pi_1 = (I_4, I_3, I_6, 0, I_1, I_2, I_3) \) is equal to the solution \( \pi_2 = (I_1, I_2, I_5, 0, I_6, I_3, I_4) \). Too many redundant
solutions will lead to premature convergence of the algorithm. To improve the diversity of the population, we firstly need to identify the redundant solution, then to renew it with a neighbourhood search method. If chromosome \( q_1 \) and \( q_2 \) in the population is different, then the difference between the two chromosomes is \( d(q_1, q_2) > 0 \), otherwise \( d(q_1, q_2) = 0 \). If the difference between chromosome \( q_1 \) and other chromosomes in the population is equal to 0, the solution corresponding to chromosome \( q_1 \) is redundant solution. Because each chromosome is composed of \( m \) segments, firstly the lengths of the \( m \) segments are compared. If they are different, the chromosomes are different; otherwise, the difference is determined by comparing the genes. The Eq. (19) and Eq. (20) respectively give the calculation method of the difference of each segment and the difference of the whole chromosome between two chromosomes \( q_1 \) and \( q_2 \) with the same length.

\[
d(q_{1i}, q_{2i'}) = \frac{\sum_{i=1}^{n_i}|g_{1it} - g_{2it'}|}{n_i} \quad (19)
\]

\[
d(q_1, q_2) = \frac{\sum_{i=1}^{m} d(q_{1i}, q_{2i'})}{n} \quad (20)
\]

where \( q_{1i} \) and \( q_{2i'} \) is gene segments on chromosomes \( q_1 \) and \( q_2 \), respectively; \( n_i \) is the length of the gene segment, and \( g_{1it} \) and \( g_{2it'} \) is the locus in the gene segment.

Let \( N_1 \) represent the number of redundant solutions in the population, and \( dw \) represents the density of the redundant solutions; then, \( dw = N_1/P_{\text{opsize}} \). If the \( dw \) is larger than the threshold, a neighbourhood search method is used to renew the individuals in the redundant solutions population. If the new solution is better than the original solution, it replaces the original solution; otherwise, the original solution is kept.

Because the two objectives are optimized, two kinds of four neighbourhood search operators are designed. The average makespan \( \bar{C} \) of the maximum makespan \( \text{Max}\bar{C}_{\max} \) and the minimum makespan \( \text{Min}\bar{C}_{\max} \), the average \( \sum \bar{T} \) of the maximum total tardiness \( \text{Max} \sum \bar{T}_j \), and the minimum total tardiness \( \text{Min} \sum \bar{T}_j \) in the redundant solution’s population are calculated firstly, then the neighbourhood search operators are used.

### 5.3.1 Neighbourhood search guided by makespan improvement

If the objective of the \( k \)th individual satisfies \( C_k \geq \bar{C} \), one of the two M-neighbourhood search methods is randomly selected. The purpose of the M-neighbourhood search method is to improve the \( C_{\text{max}} \) of the solution, including M-Insert and M-Swap.

**M-Insert:** Select a batch with waste time \( P_{\text{wet}} \) from machine \( M_{\text{minct}} \) with the minimum completion time, search for a job with a processing time equal to or less than \( P_{\text{wet}} \) in machine \( M_{\text{maxct}} \) with the maximum completion time, and insert it into the selected batch on machine \( M_{\text{minct}} \).

**M-Swap:** The job with a longer processing time selected from machine \( M_{\text{maxct}} \) is swapped with the job with a shorter processing time selected from machine \( M_{\text{minct}} \). The difference in processing time between the swapped jobs should be less than the difference in the completion time of the two machines.

### 5.3.2 Neighbourhood search guided by total tardiness improvement

If the objective of the \( k \)th individual satisfies \( \sum T_k > \sum \bar{T} \), one of the two T-neighbourhood search methods is randomly selected. The purpose of the T-neighbourhood search method is to improve the \( \sum \bar{T}_j \) of the solution, including T-Insert and T-Swap.

**T-Insert:** Select the job with the shortest processing time from among the delayed jobs and insert them one by one into the position before the job with a larger due date time.

**T-Swap:** Select the job with the longest processing time from among the delayed jobs and swap its position with the job whose subsequent due date is smaller than the job one by one.

### 5.4. Genetic operators

#### 5.4.1 Crossover operator

A fixed crossover probability \( p_c = 0.5 \) is set for the NSGA-II algorithm, and a dynamic crossover probability \( p'_c = 1.5 - \frac{2e^{-g}}{1 + e^{-g}} \times p_c \) is designed for the M-NSGA-II algorithm. \( g \) represents the number of iterations at present, and \( G \) represents the total number of iterations to be performed.

A two-point crossover method is adopted. A random chromosome \( p_{x} \) in the first Pareto front \( F_1 \) of the population after fast non-dominated sorting and a random chromosome \( p_{y} \) in the other front are selected as parents. A random gene in \([1, n + m - 1]\) is selected from each parent for crossover, and the other genes are obtained by mapping. If the offspring is better than the parent \( p_{y} \), it will be retained. Otherwise, the original parent is retained. An example of crossover is shown in Fig. 3.
5.4.2 Mutation operator

A fixed mutation probability \( p_m = 0.05 \) is set for the NSGA-II algorithm, and a dynamic mutation probability \( p_m' = \frac{2e^{-\frac{t}{1+e^{-\frac{t}{5}}}}} \) is designed for the M-NSGA-II algorithm. Two methods, MO₁ and MO₂, are used for the mutation operation of the first 50% generation population and the last 50% generation population, respectively. MO₁ randomly selects an individual, selects a gene fragment of length \( \lfloor \frac{n}{m} \rfloor \), and reverses the sequence of jobs in this gene fragment. MO₂ selects an individual and randomly selects two genes to exchange. If the randomly selected gene is 0, reselection is required. An example of the mutation operation is shown in Fig. 4.

6. Parameter tuning and computational experiments

The NSGA-II and the M-NSGA-II algorithms were written using the Dev C++ language, and the MIP model was constructed on the IBM ILOG CPLEX Optimization Studio Ver. 12.7.1 platform. The three algorithms’ simulation experiments were all run on an Intel(R) Xeon(R) E5-2650 v4 @2.20 GHz CPU and 64 GB RAM workstation.

6.1. Performance measures

This paper compares different algorithms in terms of operation efficiency and effectiveness. The operation efficiency is directly expressed by the operating time. Four measures are used to compare the operation effectiveness or different aspects of the non-dominated fronts obtained by MIP, the NSGA-II algorithm and the M-NSGA-II algorithm, including the number of Pareto solutions \( N_Ω \), the \( C \) metric, the inverted generational distance (IGD) and the distance \( \Delta_Ω \) between the different solutions. The \( C \) metric is used to show differences between two Pareto fronts (Wang & Liu, 2015). The \( C(A, B) \) states the percentage of a solution in \( B \) dominated by at least one solution of \( A \). It is calculated by Eq. (21):

\[
C(A, B) = \frac{|\{ b \in B \mid \exists a \in A : a < b \}|}{|B|} \tag{21}
\]

where \( a < b \) means that \( a \) dominates \( b \) or \( b \) is dominated by \( a \). It can be seen from Equation (21) that if all individuals in solution set \( A \) dominate all individuals in solution set \( B \), then \( C(A, B) = 1 \); otherwise, \( C(A, B) = 0 \). In this metric, \( C(A, B) \neq C(B, A) \); thus, both \( C(A, B) \) and \( C(B, A) \) must be calculated, and \( A \) is better than \( B \) if \( C(A, B) > C(B, A) \).

The IGD is the inverse mapping of the generational distance (Zhang et al., 2021), which is represented by the average distance from the individual in the Pareto solution set to the non-dominated solution set solved by the algorithm. The computing formula is shown in Eq. (22):

\[
IGD(A, \Omega) = \frac{\sum_{x \in \Omega} \min_{a \in A} d_{xa}}{|\Omega|} \tag{22}
\]

where \( d_{xa} \) is the Euclidean distance between solutions \( x \) and \( a \), and \( \Omega \) is the approximate Pareto front, and is acquired by merging the Pareto solutions obtained by all algorithms. The smaller the \( IGD(A, \Omega) \) is, the better of the algorithm.

\( \Delta_Ω \) is used in the literature (Wang et al., 2018), and it is defined by Eq. (23):
where \( d_x \) is the Euclidean distance between two consecutive solutions in the obtained non-dominated set \( A \), and \( \bar{d} \) is the average of all distances \( d_x \). The smaller the \( \Delta_{D_1} \) is, the better of the algorithm.

6.2. Parameter tuning

Parameters largely affect the performance of the M-NSGA-II algorithm. Therefore, it is necessary to tune the parameters using appropriate methods. The Taguchi method is a strong stochastic technique from among the design of experiments (DOE) methods. It aims to improve quality and reduce cost when designing a process or production, and its main idea is to combine different factors and their degrees using vertical arrays and factorial designs. Then, the analysis is conducted with fewer numbers of experiments (Wang et al., 2018). The parameters that affect the performance of the M-NSGA-II algorithm include the population number \( \text{Popsize} \), the number of iterations \( \text{Gensize} \), the crossover rate \( p_c \), and the mutation rate \( p_m \). The levels of these four parameters were determined according to the preliminary experiments as shown in Table 2.

A set of 13 typical problem instances with three scales (small, medium, and large) were generated to perform the parameter tuning experiments, and a decoding method based on FL was adopted. The signal-to-noise (S/N) ratio of the parameters was analysed to determine the parameters of the M-NSGA-II algorithm. Using \( \text{Popsize} \), \( \text{MaxGen} \), \( p_c \), and \( p_m \) as control factors where each factor has 3 levels, the number of orthogonal experiments is \( L_{3^4} \), the objective function is used as the response variable, and the number of experiments is \( 13 \times 9 \times 10 = 1170 \). The results for the small-scale problem instances are shown in Table 3. Because there are two objectives, the same data normalization method used in reference (Yue et al., 2019) is applied. Through normalization of the S/N at different levels of different response variables, the comprehensive S/N ratio at the final parameter level was obtained, and Fig. 5 shows the mean S/N for the small-scale problems.

<table>
<thead>
<tr>
<th>No.</th>
<th>Popsize</th>
<th>Gensize</th>
<th>( p_c )</th>
<th>( p_m )</th>
<th>( C_{\max} )</th>
<th>( TT )</th>
<th>( C_{\max} )</th>
<th>( TT )</th>
<th>( C_{\max} )</th>
<th>( TT )</th>
<th>( C_{\max} )</th>
<th>( TT )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
<td>50</td>
<td>0.5</td>
<td>0.01</td>
<td>24.017</td>
<td>7.983</td>
<td>30.880</td>
<td>9.640</td>
<td>38.708</td>
<td>25.483</td>
<td>31.202</td>
<td>14.369</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>75</td>
<td>0.7</td>
<td>0.05</td>
<td>24.033</td>
<td>7.567</td>
<td>30.733</td>
<td>9.200</td>
<td>38.992</td>
<td>28.300</td>
<td>31.253</td>
<td>15.022</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td>100</td>
<td>0.9</td>
<td>0.1</td>
<td>24.067</td>
<td>8.200</td>
<td>30.800</td>
<td>10.825</td>
<td>38.700</td>
<td>26.592</td>
<td>31.189</td>
<td>15.206</td>
</tr>
<tr>
<td>4</td>
<td>75</td>
<td>50</td>
<td>0.7</td>
<td>0.1</td>
<td>24.167</td>
<td>7.633</td>
<td>30.850</td>
<td>11.650</td>
<td>38.668</td>
<td>28.573</td>
<td>31.228</td>
<td>15.952</td>
</tr>
<tr>
<td>5</td>
<td>75</td>
<td>75</td>
<td>0.9</td>
<td>0.01</td>
<td>24.233</td>
<td>7.767</td>
<td>30.700</td>
<td>9.258</td>
<td>39.058</td>
<td>26.333</td>
<td>31.331</td>
<td>14.453</td>
</tr>
<tr>
<td>6</td>
<td>75</td>
<td>100</td>
<td>0.5</td>
<td>0.05</td>
<td>23.967</td>
<td>8.117</td>
<td>31.067</td>
<td>11.867</td>
<td>39.150</td>
<td>27.737</td>
<td>31.394</td>
<td>15.907</td>
</tr>
<tr>
<td>7</td>
<td>100</td>
<td>50</td>
<td>0.9</td>
<td>0.05</td>
<td>24.033</td>
<td>7.533</td>
<td>30.700</td>
<td>10.067</td>
<td>38.705</td>
<td>25.762</td>
<td>31.146</td>
<td>14.454</td>
</tr>
<tr>
<td>8</td>
<td>100</td>
<td>75</td>
<td>0.5</td>
<td>0.1</td>
<td>24.333</td>
<td>7.533</td>
<td>30.767</td>
<td>9.875</td>
<td>39.145</td>
<td>27.940</td>
<td>31.415</td>
<td>15.116</td>
</tr>
<tr>
<td>9</td>
<td>100</td>
<td>100</td>
<td>0.7</td>
<td>0.01</td>
<td>24.242</td>
<td>7.775</td>
<td>30.783</td>
<td>10.917</td>
<td>38.945</td>
<td>28.135</td>
<td>31.323</td>
<td>15.609</td>
</tr>
</tbody>
</table>

Due to the two objectives of the minimization functions, the level of the parameter at which the minimum value of the evaluation criteria is obtained is preferred. The parameters of NSGA-II for different sets of problem instances were also selected using the Taguchi method. The optimum level of parameters for each algorithm for each instance set is illustrated in Table 4.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Popsize</th>
<th>Gensize</th>
<th>( p_c )</th>
<th>( p_m )</th>
<th>Popsize</th>
<th>Gensize</th>
<th>( p_c )</th>
<th>( p_m )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>50</td>
<td>50</td>
<td>0.9</td>
<td>0.05</td>
<td>50</td>
<td>50</td>
<td>0.9</td>
<td>0.05</td>
</tr>
<tr>
<td>Medium</td>
<td>200</td>
<td>200</td>
<td>0.9</td>
<td>0.15</td>
<td>200</td>
<td>200</td>
<td>0.9</td>
<td>0.15</td>
</tr>
<tr>
<td>Large</td>
<td>200</td>
<td>250</td>
<td>0.9</td>
<td>0.1</td>
<td>200</td>
<td>250</td>
<td>0.9</td>
<td>0.1</td>
</tr>
</tbody>
</table>
6.3. Computational experiments and discussion

6.3.1 Data generation

The experimental data in this paper are based on the data in the literature (Kramer et al. (2020), Nessah et al. (2008), Chen et al. (2021) and Li et al. (2019)). The experiments were conducted using the machine parameters, job parameters and PM activity parameters. The machine parameters include the number of machines \( m \); the job parameters include the number of jobs \( n \), processing time \( p_j \), release time \( r_j \) and due date \( d_j \); and the PM activity parameters include the threshold \( UT \) and maintenance time \( mt \). The instances and the range of experimental parameter are shown in Table 5.

### Table 5

<table>
<thead>
<tr>
<th>Instances</th>
<th>( m )</th>
<th>( n )</th>
<th>( p_j )</th>
<th>( UT )</th>
<th>( mt )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>2, 3</td>
<td>6, 8, 10</td>
<td>U[1,9]</td>
<td>10</td>
<td>2, 5</td>
</tr>
<tr>
<td>Medium</td>
<td>3, 5</td>
<td>15, 20, 25</td>
<td>U[1,9]</td>
<td>10, 15</td>
<td>5, 8</td>
</tr>
<tr>
<td>Large</td>
<td>5, 10</td>
<td>50, 100</td>
<td>U[1,9]</td>
<td>15, 20</td>
<td>5, 8</td>
</tr>
</tbody>
</table>

The due date \( d_j \) satisfies the uniform distribution \( U[r_j, r_j + \frac{(1+Q-C)\sum_{j=1}^{m}p_j}{m}] \), where \( Q = [0.4,0.5] \) and \( C = [0.4,0.3] \). The release time \( r_j \) of the job is set to obey a uniform distribution \( U[0, k \times n \times \alpha/m] \), where \( k = [2.02,3.03] \) and \( \alpha = [0.4,0.5,0.6] \).

6.3.2 Comparison results for different decoding methods

To validate the performance of the decoding method based on DP, we solved the problem instances using the DP-based and FL-based decoding methods. For each instance, the method was run 10 times by considering randomness, and the average values of the performance metrics were recorded. The fronts obtained by the two methods mentioned above were combined, and all the non-dominated solutions were used to form set \( \Omega \) to compute the IGD. The results are shown in Table 6.

### Table 6

<table>
<thead>
<tr>
<th>Size</th>
<th>Instances</th>
<th>FL: A</th>
<th>DP: B</th>
<th>IGD</th>
<th>( \Delta_{IGD} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( m, n ) (UT, mt)</td>
<td>( t(s) )</td>
<td>( N_d )</td>
<td>( C(B,A) )</td>
<td>( \Delta_{IGD} )</td>
</tr>
<tr>
<td>Small</td>
<td>m2n6(10,2)</td>
<td>2.30</td>
<td>2</td>
<td>0.65</td>
<td>1.30</td>
</tr>
<tr>
<td></td>
<td>m2n8(10,2)</td>
<td>3.30</td>
<td>2.4</td>
<td>0.57</td>
<td>0.95</td>
</tr>
<tr>
<td></td>
<td>m2n10(10,2)</td>
<td>3.99</td>
<td>2.4</td>
<td>0.62</td>
<td>1.71</td>
</tr>
<tr>
<td></td>
<td>m3n6(10,2)</td>
<td>2.52</td>
<td>2.1</td>
<td>0.92</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td>m3n8(10,2)</td>
<td>3.16</td>
<td>2.1</td>
<td>0.65</td>
<td>0.67</td>
</tr>
<tr>
<td></td>
<td>m3n10(10,2)</td>
<td>4.07</td>
<td>2.8</td>
<td>0.49</td>
<td>1.23</td>
</tr>
<tr>
<td></td>
<td>m2n6(10,5)</td>
<td>2.52</td>
<td>3.0</td>
<td>0.93</td>
<td>0.18</td>
</tr>
<tr>
<td></td>
<td>m2n8(10,5)</td>
<td>3.14</td>
<td>3.0</td>
<td>0.87</td>
<td>0.28</td>
</tr>
<tr>
<td></td>
<td>m2n10(10,5)</td>
<td>3.44</td>
<td>3.5</td>
<td>0.30</td>
<td>1.96</td>
</tr>
<tr>
<td></td>
<td>m3n6(10,5)</td>
<td>1.94</td>
<td>3.0</td>
<td>0.83</td>
<td>0.36</td>
</tr>
<tr>
<td></td>
<td>m3n8(10,5)</td>
<td>2.83</td>
<td>3.5</td>
<td>0.63</td>
<td>0.57</td>
</tr>
<tr>
<td></td>
<td>m3n10(10,5)</td>
<td>4.37</td>
<td>3.0</td>
<td>0.48</td>
<td>1.40</td>
</tr>
</tbody>
</table>
The results show that the DP-based decoding method performs better than the FL-based decoding method in terms of the C metric and IGD, especially for large-scale problem instances composed of many jobs. There is no significant difference between the two decoding methods in terms of the $\Delta_{12}$ metric. The $N_d$ metric obtained by the FL-based decoding method is slightly larger than that obtained by the DP-based decoding method. This result shows that the solution set obtained by the DP-based decoding method is more diversity and better approximates the Pareto front. In this paper, the $N_d$ metric obtained by the FL-based decoding method is slightly larger than that obtained by the DP-based decoding method. This result shows that the solution set obtained by the NSGA-II and M-NSGA-II algorithms is limited since minimizing the makespan indirectly leads to minimizing the total tardiness, which is why the $N_d$ metric of the FL-based decoding method slightly outperforms the DP-based decoding method.

The DP-based decoding method searches for the job-grouping solution from the first position to the current position when searching for the solution. Thus, the DP-based decoding method requires more time than the FL-based decoding method.

6.3.3 Comparison results for different neighbourhood search methods

To validate the performance of the proposed neighbourhood search method, the M-NSGA-II algorithm with and without the neighbourhood search method was used to solve the instances. The results are shown in Fig. 6. The results show that the M-NSGA-II method with neighbourhood search can obtain better non-dominated solutions compared to the method without neighbourhood search in terms of the C metric and IGD. This advantage becomes more evident as the size of the problem increases. However, there is no significant difference between the two methods in terms of the $\Delta_{12}$ metric and $N_d$. Since neighbourhood search adds further exploration and searching of the solution space, the time required is longer than that without neighbourhood search. These results fully show that the use of neighbourhood search can improve the search quality of the solution, improve the distribution and quality of the solution, and requires a longer time to obtain a better solution.
6.3.4 Comparison results for different algorithms

To compare the performance of MIP, M-NSGA-II, and NSGA-II, all three methods were used to solve the small-scale instances, and the M-NSGA-II and NSGA-II algorithms were used to solve the medium- and large-scale instances. The results are shown in Table 7 and Table 8.

Table 7
Performance results for different algorithms (small-scale problem instances)

| Size | Instance | t(s) | Na | C(A,B) | C(A,C) | IGD | Δ<sub>IGD</sub> | t(s) | N<sub>a</sub> | C(B,A) | C(B,C) | IGD | Δ<sub>IGD</sub> | t(s) | N<sub>a</sub> | C(C,A) | C(C,B) | IGD | Δ<sub>IGD</sub> |
|------|----------|------|----|--------|--------|-----|-------------|------|----|--------|--------|-----|-----|-------------|------|----|--------|--------|-----|-----|-------------|------|----|--------|--------|-----|
| Small | m2n6 (10,2) | 5.3  | 2.2 | 1  | 1  | 1  | 0  | 0.39 | 0.88 | 2.1  | 0.72 | 0.80 | 0.84 | 1.35 | 3.62 | 2.2  | 0.62 | 0.65 | 0.72 | 0.75 |
|      | m2n8 (10,2) | 32.4 | 2.4 | 1  | 1  | 1  | 0  | 2.20 | 1.64 | 2.6  | 0.45 | 0.71 | 1.14 | 1.49 | 4.68 | 2.5  | 0.37 | 0.58 | 1.32 | 2.08 |
|      | m2n10 (10,2) | 496.0 | 2.9 | 1  | 1  | 1  | 0  | 3.02 | 2.76 | 2.6  | 0.28 | 0.73 | 2.56 | 4.64 | 6.28 | 2.4  | 0.15 | 0.48 | 3.13 | 6.19 |
|      | m3n6 (10,2) | 7.2  | 2.4 | 1  | 1  | 1  | 0  | 0.55 | 0.82 | 2.4  | 0.95 | 1.00 | 0.05 | 0.55 | 2.01 | 2.1  | 0.79 | 0.79 | 0.41 | 0.46 |
|      | m3n8 (10,2) | 49.7  | 2.7 | 1  | 1  | 1  | 0  | 1.55 | 1.91 | 2.4  | 0.40 | 0.52 | 1.15 | 1.37 | 3.53 | 2.2  | 0.59 | 0.68 | 0.96 | 0.62 |
|      | m3n10 (10,2) | 3449.5 | 3.4 | 1  | 1  | 1  | 0  | 1.62 | 2.90 | 2.8  | 0.35 | 0.58 | 1.38 | 1.34 | 6.31 | 2.5  | 0.22 | 0.46 | 1.97 | 1.62 |
|      | m2n6 (10,5) | 10.6 | 3   | 1  | 1  | 1  | 0  | 2.56 | 0.91 | 2.7  | 0.77 | 0.75 | 1.06 | 1.27 | 4.46 | 2.6  | 0.73 | 0.70 | 0.61 | 3.91 |
|      | m2n8 (10,5) | 55.0 | 2.8 | 1  | 1  | 1  | 0  | 1.18 | 1.65 | 2.4  | 0.42 | 0.67 | 1.26 | 1.84 | 5.08 | 2.4  | 0.45 | 0.63 | 1.12 | 1.82 |
|      | m2n10 (10,5) | 13264.7 | 3.4 | 1  | 1  | 1  | 0  | 2.29 | 2.76 | 3.1  | 0.32 | 0.73 | 2.00 | 2.33 | 7.10 | 2.1  | 0.22 | 0.40 | 4.03 | 2.71 |
|      | m3n6 (10,5) | 25.5 | 3   | 1  | 1  | 1  | 0  | 1.08 | 0.82 | 3.1  | 0.97 | 1.00 | 0.03 | 1.13 | 2.76 | 2.9  | 0.93 | 0.92 | 0.15 | 1.09 |
|      | m3n8 (10,5) | 210.0 | 3.4 | 1  | 1  | 1  | 0  | 1.70 | 1.80 | 3.5  | 0.58 | 0.67 | 1.06 | 2.17 | 5.10 | 2.5  | 0.36 | 0.47 | 1.69 | 1.51 |
|      | m3n10 (10,5) | 15269.2 | 3.2 | 1  | 1  | 1  | 0  | 2.45 | 3.22 | 2.5  | 0.35 | 0.57 | 2.77 | 2.19 | 6.19 | 2.2  | 0.22 | 0.45 | 3.30 | 1.76 |
Table 8
Performance results for different algorithms (medium- and large-scale problem instances)

<table>
<thead>
<tr>
<th>Size</th>
<th>Instance</th>
<th>t(s)</th>
<th>Nₐ</th>
<th>C(A, B)</th>
<th>IGD</th>
<th>Δ₁,₀</th>
<th>t(s)</th>
<th>C(B, A)</th>
<th>IGD</th>
<th>Δ₁,₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>Medium</td>
<td>m₃n₁₅(10,5)</td>
<td>47.64</td>
<td>2.3</td>
<td>0.25</td>
<td>3.47</td>
<td>2.00</td>
<td>206.14</td>
<td>2.2</td>
<td>0.65</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>m₃n₂₀(10,5)</td>
<td>71.72</td>
<td>2.4</td>
<td>0.13</td>
<td>6.78</td>
<td>5.12</td>
<td>338.95</td>
<td>2.3</td>
<td>0.50</td>
<td>3.27</td>
</tr>
<tr>
<td></td>
<td>m₃n₂₅(10,5)</td>
<td>89.94</td>
<td>2.6</td>
<td>0.30</td>
<td>6.88</td>
<td>12.19</td>
<td>365.60</td>
<td>2.9</td>
<td>0.57</td>
<td>4.27</td>
</tr>
<tr>
<td></td>
<td>m₅n₁₅(10,5)</td>
<td>47.49</td>
<td>2.2</td>
<td>0.47</td>
<td>1.08</td>
<td>4.69</td>
<td>94.94</td>
<td>2.2</td>
<td>0.57</td>
<td>1.89</td>
</tr>
<tr>
<td></td>
<td>m₅n₂₀(10,5)</td>
<td>68.28</td>
<td>2.3</td>
<td>0.58</td>
<td>1.54</td>
<td>1.52</td>
<td>151.99</td>
<td>2.3</td>
<td>0.26</td>
<td>2.52</td>
</tr>
<tr>
<td></td>
<td>m₅n₂₅(10,5)</td>
<td>96.35</td>
<td>3.1</td>
<td>0.22</td>
<td>4.73</td>
<td>2.68</td>
<td>269.25</td>
<td>2.6</td>
<td>0.49</td>
<td>2.52</td>
</tr>
<tr>
<td></td>
<td>m₅n₁₅(10,8)</td>
<td>45.01</td>
<td>2.5</td>
<td>0.28</td>
<td>4.67</td>
<td>5.82</td>
<td>162.42</td>
<td>2.4</td>
<td>0.63</td>
<td>4.12</td>
</tr>
<tr>
<td></td>
<td>m₅n₂₀(10,8)</td>
<td>78.37</td>
<td>2.4</td>
<td>0.03</td>
<td>8.95</td>
<td>3.64</td>
<td>320.89</td>
<td>2.5</td>
<td>0.89</td>
<td>0.61</td>
</tr>
<tr>
<td></td>
<td>m₅n₂₅(10,8)</td>
<td>84.82</td>
<td>2.6</td>
<td>0.25</td>
<td>12.08</td>
<td>7.87</td>
<td>437.04</td>
<td>2.5</td>
<td>0.55</td>
<td>2.91</td>
</tr>
<tr>
<td></td>
<td>m₅n₁₅(10,8)</td>
<td>52.80</td>
<td>2.6</td>
<td>0.45</td>
<td>0.76</td>
<td>1.77</td>
<td>144.50</td>
<td>1.8</td>
<td>0.64</td>
<td>0.82</td>
</tr>
<tr>
<td></td>
<td>m₅n₂₀(10,8)</td>
<td>75.58</td>
<td>2.8</td>
<td>0.32</td>
<td>2.93</td>
<td>4.87</td>
<td>189.48</td>
<td>2.8</td>
<td>0.40</td>
<td>4.14</td>
</tr>
<tr>
<td></td>
<td>m₅n₂₅(10,8)</td>
<td>96.21</td>
<td>2.2</td>
<td>0.18</td>
<td>9.34</td>
<td>2.33</td>
<td>278.11</td>
<td>2.3</td>
<td>0.61</td>
<td>3.06</td>
</tr>
<tr>
<td></td>
<td>m₅n₁₅(15,5)</td>
<td>38.69</td>
<td>2.1</td>
<td>0.30</td>
<td>2.40</td>
<td>8.20</td>
<td>145.51</td>
<td>1.8</td>
<td>0.68</td>
<td>1.12</td>
</tr>
<tr>
<td></td>
<td>m₅n₂₀(15,5)</td>
<td>66.65</td>
<td>2.3</td>
<td>0.26</td>
<td>2.89</td>
<td>1.02</td>
<td>204.21</td>
<td>2.2</td>
<td>0.53</td>
<td>1.53</td>
</tr>
<tr>
<td></td>
<td>m₅n₂₅(15,5)</td>
<td>88.78</td>
<td>4</td>
<td>0.21</td>
<td>7.24</td>
<td>3.57</td>
<td>248.33</td>
<td>2.7</td>
<td>0.55</td>
<td>5.79</td>
</tr>
<tr>
<td></td>
<td>m₅n₁₅(15,5)</td>
<td>47.68</td>
<td>1.9</td>
<td>0.63</td>
<td>0.68</td>
<td>2.61</td>
<td>107.42</td>
<td>1.7</td>
<td>0.66</td>
<td>1.38</td>
</tr>
<tr>
<td></td>
<td>m₅n₂₀(15,5)</td>
<td>67.78</td>
<td>2.6</td>
<td>0.44</td>
<td>1.29</td>
<td>3.26</td>
<td>189.53</td>
<td>2.2</td>
<td>0.53</td>
<td>2.15</td>
</tr>
<tr>
<td></td>
<td>m₅n₂₅(15,5)</td>
<td>100.24</td>
<td>2.2</td>
<td>0.28</td>
<td>2.69</td>
<td>0.80</td>
<td>253.04</td>
<td>2.5</td>
<td>0.68</td>
<td>1.72</td>
</tr>
<tr>
<td></td>
<td>m₃n₁₅(15,8)</td>
<td>42.04</td>
<td>2.3</td>
<td>0.00</td>
<td>4.61</td>
<td>4.04</td>
<td>176.97</td>
<td>2.3</td>
<td>0.92</td>
<td>0.12</td>
</tr>
<tr>
<td></td>
<td>m₃n₂₀(15,8)</td>
<td>64.78</td>
<td>3.1</td>
<td>0.27</td>
<td>5.12</td>
<td>4.21</td>
<td>245.88</td>
<td>3.4</td>
<td>0.63</td>
<td>2.41</td>
</tr>
<tr>
<td></td>
<td>m₃n₂₅(15,8)</td>
<td>89.45</td>
<td>2.9</td>
<td>0.05</td>
<td>9.56</td>
<td>5.11</td>
<td>415.90</td>
<td>2.8</td>
<td>0.81</td>
<td>4.14</td>
</tr>
<tr>
<td></td>
<td>m₅n₁₅(15,8)</td>
<td>51.59</td>
<td>2.2</td>
<td>0.63</td>
<td>1.54</td>
<td>2.66</td>
<td>151.53</td>
<td>2.2</td>
<td>0.67</td>
<td>0.70</td>
</tr>
<tr>
<td></td>
<td>m₅n₂₀(15,8)</td>
<td>74.89</td>
<td>2.2</td>
<td>0.43</td>
<td>2.20</td>
<td>4.52</td>
<td>223.01</td>
<td>2.7</td>
<td>0.43</td>
<td>2.48</td>
</tr>
<tr>
<td></td>
<td>m₅n₂₅(15,8)</td>
<td>98.04</td>
<td>2.1</td>
<td>0.30</td>
<td>3.22</td>
<td>4.24</td>
<td>308.37</td>
<td>2.3</td>
<td>0.52</td>
<td>3.52</td>
</tr>
</tbody>
</table>

As seen in Table 7, the MIP method can obtain the optimal Pareto solution for all 12 instances, although the time needed for solving the problem greatly increased with the increase in the size of the problem. The NSGA-II algorithm slightly outperforms the M-NSGA-II algorithm in terms of the IGD, the C metric and Nₐ. There is no significant difference between the two methods in terms of Δ₁,₀. The NSGA-II algorithm is more efficient than the MIP and M-NSGA-II algorithms, and this advantage becomes more evident as the size of the problem increases. It can be seen from Table 8 that the M-NSGA-II algorithm outperforms the NSGA-II algorithm in terms of the IGD and C metrics for medium- and large-scale problem instances, and this advantage also becomes more evident as the size of the problem increases. However, there is no significant difference between the two methods in terms of Nₐ and Δ₁,₀. The efficiency of the NSGA-II algorithm is better than that of M-NSGA-II because M-NSGA-II uses the DP-based decoding method and the neighbourhood search method.

6.3.5 Influence of PM parameters

To judge the influence of the PM parameters on the NSGA-II and M-NSGA-II algorithms, their four metrics were analysed using variance analysis (the confidence interval was 95%), and the results are shown in Table 9.

As shown in Table 9, for the different UT and mt, all the indicators for the M-NSGA-II algorithm are not significantly different. In contrast, for the NSGA-II algorithm, the Nₐ indicator for small instances and the IGD indicator for medium instances are significantly different for parameter mt. This result shows that the M-NSGA-II algorithm is stable and effective for all instances.
Table 9
Significance level of the different algorithms

<table>
<thead>
<tr>
<th>NSGA-II</th>
<th>Size</th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>$N_d$</td>
<td>IGD</td>
<td>C(A,B)</td>
<td>$\Delta_{0.01}$</td>
</tr>
<tr>
<td>UT</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>0.955</td>
</tr>
<tr>
<td>$m_t$</td>
<td>0.018</td>
<td>0.548</td>
<td>0.902</td>
<td>0.509</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>M-NSGA-II</th>
<th>Size</th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>$N_d$</td>
<td>IGD</td>
<td>C(A,B)</td>
<td>$\Delta_{0.01}$</td>
</tr>
<tr>
<td>UT</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>1</td>
</tr>
<tr>
<td>$m_t$</td>
<td>0.330</td>
<td>0.256</td>
<td>0.899</td>
<td>0.857</td>
</tr>
</tbody>
</table>

7. Conclusions

In this paper, an identical PMSP with machine flexible maintenance and job release times has been investigated to minimize the makespan and total tardiness simultaneously. A MIP model was first established to obtain the exact Pareto fronts for small-scale instances. To tackle the medium- and large-scale instances, an M-NSGA-II algorithm was constructed with three strategies. After tuning the M-NSGA-II parameters using the DOE and the Taguchi method, computational experiments were conducted. The results show that the M-NSGA-II algorithm obviously outperforms the original NSGA-II algorithm when solving medium- and large-scale instances, although the time required to solve these instances is slightly longer.

In future research, more realistic parallel machine manufacturing environments such as unrelated parallel machines should be investigated. The maintenance period depends on the machine’s reliability, and the objective is to minimize the energy and power costs. Furthermore, considering the development of Big Data and Intelligent Manufacturing, parallel machine dynamic scheduling and intelligent scheduling methods based on reinforcement learning should be developed.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No.51705370).

Disclosure statement

No potential conflict of interest was reported by the author(s).

Data availability statement

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

References


Yoo, J., & Lee, I.S. (2016). Parallel machine scheduling with maintenance activities. *Computers & Industrial Engineering,


**Appendix 1. Decoding algorithm based on FL**

**Algorithm 3: Full loading decoding algorithm**

0. Initialize the cumulative processing time of each machine $Q_1 = \cdots = Q_m = 0$, the completion time set of jobs $JC = \{\emptyset\}$, the tardiness set of jobs $JT = \{\emptyset\}$, and the completion time set of machines $MC = \{\emptyset\}$, UT, mt, etc.

1. For chromosome $h = 1$ to $P_{\text{size}}$
2. For machine $i = 1$ to $m$
3. For the machine’s job position $k = 1$ to $n_i$, select job $J_{ik}$
4. If $(Q_i + p_{ik} < UT)$ then $C_{ik} = \max(C_{i(k-1)}, r_{ik}) + p_{ik}$, $Q_i = Q_i + p_{ik}$
5. If $(Q_i + p_{ik} = UT)$ then $C_{ik} = \max(C_{i(k-1)}, r_{ik}) + p_{ik} + mt, Q_i = 0$
6. If $(Q_i + p_{ik} > UT)$ then $C_{ik} = \max(C_{i(k-1)}, r_{ik}) + p_{ik} + mt, Q_i = p_{ik}$
7. $T_{ik} = \max\{C_{ik} - d_{ik}, 0\}$, $JC = JC \cup \{C_{ik}\}$, $JT = JT \cup \{T_{ik}\}$
8. End For
9. $C_i = \max\{C_{ik}\}$, $MC = MC \cup \{C_i\}$, $T_i = \sum T_{ik}$
10. End For
11. $C_{\text{max}} = \max\{C_i, C_i \in MC\}$, $TT = \sum_{i=1}^m T_i$
12. End For

© 2022 by the authors; licensee Growing Science, Canada. This is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC-BY) license (http://creativecommons.org/licenses/by/4.0/).