Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

Heuristics and metaheuristics to minimize makespan for flowshop with peak power consumption constraints

Yuan-Zhen Li[a], Kaizhou Gao[b*], Lei-Lei Meng[a], Xue-Lei Jing[a] and Biao Zhang[a]

[a]*School of Computer Science, Liaocheng University, Shandong Liaocheng, 252000, P.R. China*
[b]*Institute of Systems Engineering, Macau University of Science and Technology, Taipa, 999078, Macao*

C H R O N I C L E

Article history:
Received May 10 2022
Accepted February 14 2023
Available online
February, 14 2023

Keywords:
Permutation flowshop scheduling
Peak power consumption
Makespan
Heuristics
Artificial bee colony algorithm
Iterated local search algorithm

A B S T R A C T

This paper addresses the permutation flowshop scheduling problem with peak power consumption constraints (PFSPP). The real-time power consumption of the PFSPP cannot exceed a given peak power at any time. First, a mathematical model is established to describe the concerned problem. The sequence of operations is taken as a solution and the characteristics of solutions are analyzed. Based on the problem characteristics, eight heuristics are proposed, including balanced machine-job decoding method, balanced machine-job insert method, balanced job-machine insert method, balanced machine-job group insert method, balanced job-machine group insert method, greedy algorithm, beam search algorithm, and improved beam search algorithm. Similarly, the canonical artificial bee colony algorithm and iterated local search algorithm are modified based on the problem characteristics to solve the PFSPP. A large number of experiments are carried out to evaluate the performance of new proposed heuristics and metaheuristics. The results and discussion show that the proposed heuristics and metaheuristics perform well in solving the PFSPP.

1. Introduction

Faced with serious global environmental problems, green manufacturing has become a mode increasingly concerned by modern production (Li & Wang, 2022). With the rapid consumption of non-renewable energy and the continuous increase of energy prices, the efficient use of energy has attracted extensive attention (Renna & Materi, 2021). Green scheduling (Li et al., 2021) is the key to green manufacturing. Efficient green scheduling method can synergistically optimize the economic indicators and green indicators of enterprises to achieve efficiency increase, energy conservation, emission reduction, consumption reduction, and cost reduction (Ramezanian et al., 2019; Ribas & Companys, 2021). Early research on green scheduling focused on the design of energy management strategies, that is, the processing speed of equipment can be adjusted. It is generally believed that the faster the processing time, the greater the energy consumption (Wang and Wang, 2020). In view of the time of use prevailing in the industry, some researchers have proposed scheduling methods to optimize electricity charges (Luo et al., 2013; Zhao et al., 2021). The main energy source of most manufacturing enterprises is electric. The maximum total power consumption of all machines in the production process is called peak power consumption. If the peak power consumption exceeds an established threshold, the electricity price will be much higher than the rated price. Therefore, controlling the peak power consumption within a reasonable range can effectively save costs. At present, there is little research on scheduling with peak power consumption constraints. The permutation flowshop scheduling problem (PFSP) (González-Neira et al., 2017; Mishra & Shrivastava, 2020) has a wide application background and plays an important role in manufacturing systems (Fernandez-Viagas et al., 2022; Yu et al., 2022). With the serious attention of manufacturing enterprises to energy, peak power consumption is considered to be one of the important issues. For PFSP with peak power constraints (PFSPP), real-time power consumption cannot exceed a given peak power at any time. Fang et al. (2011) took the lead in studying the flowshop scheduling problem considering peak power consumption. They proposed a mathematical programming model and a simple case to demonstrate the novel mathematical model. In their following paper (Fang et al.,

* Corresponding author
E-mail: kzgao@must.edu.mo (K. Gao)
ISSN 1923-2934 (Online) - ISSN 1923-2926 (Print)
2023 Growing Science Ltd.
doi: 10.5267/j.ijiec.2023.2.004

2013), Feng *et al.* named the PFSP with peak power constraints as PFSPP. The properties of concurrently running jobs were studied and a special case of two machines and zero intermediate storage was solved. The time window technique (Cui and Lu, 2020), which reduced complexity of the problem and simplified calculation, was used to solve the PFSPP to minimize makespan and power consumption. Chou *et al.* (2020) studied a scheduling problem with peak power consumption, where the energy consumption of machines was random and depended on scheduling. The research on the PFSPP is few but very meaningful. Unlike most classical scheduling problems, it is necessary to track which jobs are running at the same time at any time to consider the peak power consumption in the PFSPP. If the power consumption of concurrent operations exceeds a given threshold, some operations will be delayed. Wang *et al.* (2019) proposed a solution for the PFSPP which contains two parts, i.e. the job processing sequence and processing speed of each job on each machine. In fact, this coding scheme does not provide enough information to indicate the sequence of operations. Wang *et al.* (2019) proposed an earliest processing rule (EPR) to determine the start time of each operation to meet the peak power consumption constraint. Then, five decoding methods were presented to decode a solution to obtain the sequence of operations. In this paper, we take the sequence of operations as the representation of solutions. Specifically, this paper studies a reasonable scheduling method to meet the demand of peak power consumption in the PFSPP when the processing sequence of jobs is determined and the processing speeds are fixed. In this way, the five decoding methods in the paper (Wang and Wang, 2019) are actually equivalent to five heuristics. The mathematical model of the concerned problem is established and the problem characteristics are analyzed. Based on the problem characteristics, eight constructive heuristics and two metaheuristics derived from artificial bee colonies and iterated local search are proposed to solve this concerned problem. The experimental results show that both the presented constructive heuristics and metaheuristics are very effective for solving the PFSPP.

The remainder of this paper is organized as follows. Section 2 presents the problem description, establishes a mathematical model, introduces an illustrative example, and analyzes the characteristics of the considered problem. Section 3 proposes 8 heuristics, including balanced machine-job decoding method, balanced machine-job insert method, balanced job-machine insert method, balanced machine-job group insert method, balanced job-machine group insert method, greedy algorithm, beam search algorithm, and improved beam search algorithm. Section 4 presents ABC algorithm and ILS algorithm customized according to problem characteristics. We report the computational results and comparisons in Section 5. Finally, Section 6 provides the concluding remarks and suggests some future work.

2. Problem Description

2.1 Problem Definition

There are n jobs to be processed in a flowshop, where there are m machines in the fixed permutation. The set of jobs is $\mathcal{J} = \{1,2,\dots,n\}$, the set of machines is $\mathcal{M} = \{1,2,\dots,m\}$, and n and m are known constants. The operation of job j on machine i is denoted as O_{ij}. At the beginning, all jobs and machines are ready. A machine can only process one job at a time, and a job can only be processed on one machine at a time. No interruption is allowed during jobs processing. There are no setup times, facility malfunctions or maintenance issues. Suppose the processing sequence and processing speed of all jobs have been determined. The processing sequence of jobs are $\pi = \{1,2,\dots,n\}$. We need to determine the sequence of all operations. The processing time and power consumption of O_{ij} are p_{ij} and q_{ij}, respectively. In addition, the power consumption at any time cannot exceed a threshold, denoted as Q_{max}. Another assumption is that $\max_{i\in\mathcal{M},j\in\mathcal{J}}\{q_{ij}\} \le Q_{max}$ to ensure the existence of feasible schedule. A feasible schedule is defined as a schedule in which the total power consumption at any time is no more than the given threshold Q_{max}. The goal is to minimize the completion time under the condition of meeting the peak power consumption constraint. The indices, parameters and decision variables used are shown in Table 1.

Table 1
Indices, parameters and decision variables

Parameters:	
n	The number of jobs.
m	The number of machines.
O_{ij}	The operation of job $i \in N$ on machine $j \in M$.
p_{ij}	The processing time of O_{ij}.
q_{ij}	The power consumption of O_{ij}.
C_{max}	The completion time of a schedule.
C_{ij}	The completion time of job *j* on machine *i*.
S_{ij}	The start time of job *j* on machine *i*.
D	A large positive number.
Variables	
u_{hkij}	is equal to 1 if the start time of job k on machine h is less than or equal to the start time of job j on machine *i* (in other words, $S_{hk} \le S_{ij}$), and 0 otherwise.
v_{hkij}	is equal to 1 if the completion time of job k on machine h is greater than the start time of job j on machine *i* (in other words, $C_{hk} > S_{ij}$), and 0 otherwise.
y_{hkij}	is equal to 1 if the start time of job j on machine i occurs during the processing of job k on machine h (in other words, $S_{hk} \le S_{ij} < C_{hk}$), and 0 otherwise.

2.2 Problem Formulation

The following mathematical model of PFSPP is expanded from the modes presented by Fang *et al.* (2013) and Wang *et al.* (2019).

$$min \quad C_{max}; \tag{1}$$

Subject to

$$C_{max} \geq C_{mnf}; \tag{2}$$

$$C_{11} \geq p_{11}; \tag{3}$$

$$C_{ik} \geq C_{i-1,k} + p_{ik}; \text{ for } i \in \mathcal{M}\backslash\{1\}; k \in \{1,2,\dots,n\}; \tag{4}$$

$$C_{ik} \geq C_{i,k-1} + p_{ik}; \text{ for } i \in \mathcal{M}; k \in \{2,\dots,n\}; \tag{5}$$

$$C_{ik} = S_{ik} + p_{ik}; \text{ for } i \in \mathcal{M}; k \in \{1,\dots,n\}; \tag{6}$$

$$S_{ij} - S_{hk} \leq Du_{hkij} - 1; \text{ for } i \in \mathcal{M}; k \in \{1,\dots,n\}; \tag{7}$$

$$S_{hk} - S_{ij} \leq D(1 - u_{hkij}); \text{ for } i,h \in \mathcal{M}; j,k \in \{1,\dots,n\}; \tag{8}$$

$$C_{hk} - S_{ij} \leq Dv_{hkij}; \text{ for } i,h \in \mathcal{M}; j,k \in \{1,2,\dots n\}; \tag{9}$$

$$S_{ij} - C_{hk} \leq D(1 - v_{hkij}) - 1; \text{ for } i,h \in \mathcal{M}; j,k \in \{1,2,\dots,n\}; \tag{10}$$

$$u_{hkij} + v_{hkij} = 1 + y_{hkij}; \text{ for } i,h \in \mathcal{M}; j,k \in \{1,2,\dots n\}; \tag{11}$$

$$y_{hkij} \leq u_{hkij}; \text{ for } i,h \in \mathcal{M}, j,k \in \{1,2,\dots n\}; \tag{12}$$

$$y_{hkij} \leq v_{hkij}; \text{ for } i,h \in \mathcal{M}, j,k \in \{1,2,\dots,n\}; \tag{13}$$

$$q_{ij} + \sum_{h\in\mathcal{M},h\neq i;} \sum_{l\in\mathcal{J}} q_{hl} y_{hkij} \leq Q_{max}; \text{ for } i \in \mathcal{M}, j,k \in \{1,2,\dots,n\}; \tag{14}$$

$$u_{hkij}, v_{hkij}, y_{hkij} \in \{0,1\}; \text{ for } i,h \in \mathcal{M}, j,l,k \in \{1,2,\dots,n\}; \tag{15}$$

The objective (1) and the constraint (2) ensures that the makespan of the schedule is equal to the completion time of the last job on the last machine. Constraints (3)(4)(5) ensure that the completion times are consistent with a flowshop. Constraints (6) ensure that jobs are processed nonpreemptively. Constraints (7)(8)(9)(10)(11)(12)(13) ensure that the concurrent job variables *u*, *v*, and *y* take their intended values. Finally, constraints (14) ensure that at any time, the total power consumption of all machines is at most Q_{max}. Constraints (15) define the value range of the variables.

2.3 Illustrative Example

There is an example with *m*=3 and *n*=6. The processing times and power consumption are given in Table 2.

Table 2
The processing times p_{ij} and power consumption q_{ij} of jobs on machines.

	p_{ij}			q_{ij}		
	i=1	*i* =2	*i* =3	*i*=1	*i* =2	*i* =3
***j*=1**	11	30	16	16	9	16
***j*=2**	6	37	12	16	4	16
***j*=3**	24	24	8	16	16	9

Jobs 1, 2, and 3 are processed on machines in sequence. The scheduling Gantt without peak power consumption constraint is shown in Fig. 1. As can be seen from the scheduling Gantt, the power consumption at time t ($t \in (78,90]$) is $Q(\mathrm{t}) = 32$. It exceeds Q_{max}, which is equal to 30. Operation $O_{3,2}$ or $O_{2,3}$ should be postponed meeting the peak power consumption constraint.

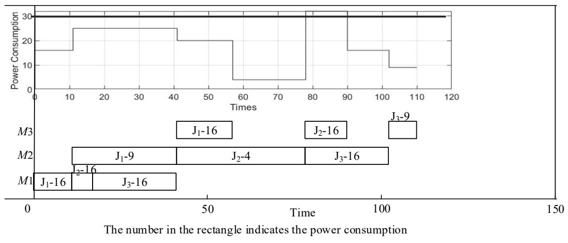


Fig. 1. Gantt Chart for a solution to an example without peak power consumption constraint.

We postpone operation $O_{2,3}$. The consequence of the postponing is that other subsequent operations will also be postponed. The scheduling Gantt after postponing $O_{2,3}$ is shown in Fig. 2. All postponed jobs are shaded. The completion time is 122. Finally, the completion time of this schedule considering peak power consumption constraint is 122.

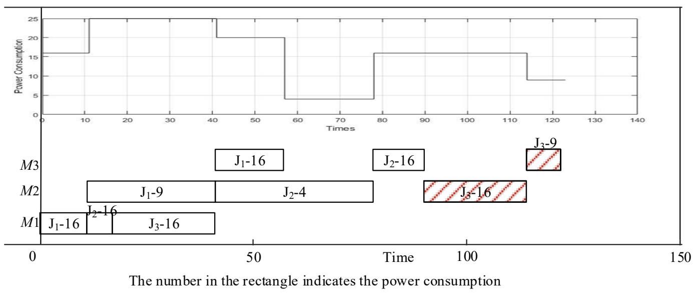


Fig. 2. Gantt Chart for a solution to an example with peak power consumption constraint

2.4 Representation, Characteristic Analysis and Decoding of Solutions

Wang et al. (2019) proposed a solution representation method, which consists of two parts. One part represents the order of all jobs, and the other part represents the processing speeds of jobs. When the speeds of all jobs are determined, the order of jobs represents a solution. This scheme can only indicate the sequence of jobs, not the sequence of operations. To further determine the sequence of operations, five decoding schemes are proposed, including job-precedence (JP) decoding method, machine-precedence (MP) decoding method, largest total remaining load (LTRL) decoding method, earliest completion time (ECT) decoding method and balanced job-machine (BJM) decoding method. These five decoding methods generate five operation sequences according to the sequence of jobs. The actual processing scheme is obtained by using the early processing rule. The five decoding methods can be regarded as heuristics. The decoded results are not the optimal solution and can be further optimized. We improved the representation of solutions. A solution is a sequence of operations rather than a sequence of jobs, representing the implementation order of operations. A solution can be expressed as: $\Pi = (O_1, O_2, \dots, O_k \dots, O_{n*m})$, where O_k indicates an operation O_{ij} which means the kth element in the solution is the operation of job j on machine i. Obviously, the number of elements of a solution is $n \times m$.

The characteristics of flowshop and fixed processing sequence of jobs determine that a solution has the following characteristics. (1) The first operation of a feasible solution is $O_{1,1}$, and the last one is $O_{n,m}$. (2) In a feasible solution, all operations $O_{pq} (p \le i, q \le j)$ must precede the operation O_{ij}. (3) In a feasible solution, the reasonable interval for operating O_{ij} is $[i \times j, (m-i+1) \times (n-j+1)]$.

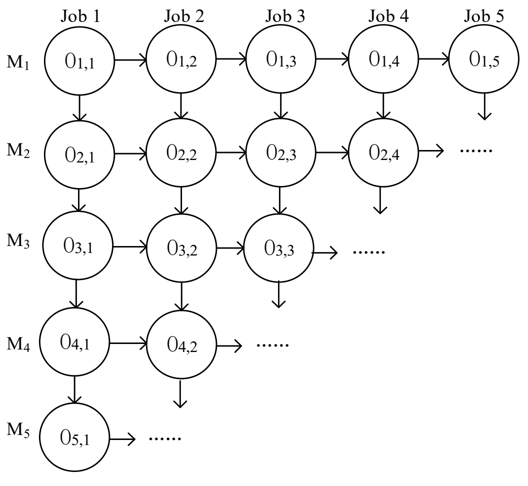


Fig. 3. Properties of a feasible solution.

A solution represents the sequence of all operations. Indeed, peak power consumption should also be considered when scheduling. It is necessary to process operations as early as possible while meeting peak power constraint at any time.

3. Proposed Heuristics

This section presents eight heuristics newly designed, including balanced machine-job decoding method, balanced machine-job insert method, balanced job-machine insert method, balanced machine-job group insert method, balanced job-machine group insert method, greedy algorithm, beam search algorithm, and improved beam search algorithm.

3.1 Balanced Machine-Job Method

Wang et al. (2019) proposed balanced job-machine decoding method (BJM) which considers the sequence of jobs and machine comprehensively when determining all operations sequence. The operation sequence obtained by BJM is $O_{1,1}, O_{2,1}, O_{1,2}, O_{3,1}, O_{2,2}, O_{1,3}, \dots, O_{n,m}$. A simple modification has been made to BJM and a balanced machine-job decoding method (BMJ) is proposed. The BMJ generates a sequence of all operations according to the gray arrows in Fig. 4. The operation sequence generated by BMJ is $O_{1,1}, O_{1,2}, O_{2,1}, O_{1,3}, O_{2,2}, O_{3,1}, \dots, O_{n,m}$. The pseudo-code of BMJ algorithm is shown in Algorithm 1.

Algorithm 1. BMJ(π)

```
1:  $j = 1, i = 1, k = i + j, \Pi = \emptyset$;
2:  **While** $k \le n + m$ **do**
3:      **While** $i > 0$ **do**
4:          Append the $O_{i,j}$ to $\Pi$;
5:          $j++$;
6:          $i--$;
7:      **endwhile**
8:      $k++$;
9:      $j = 1$;
10:     $i = k - j$;
11: **endwhile**
12: **return** $\Pi$;
```

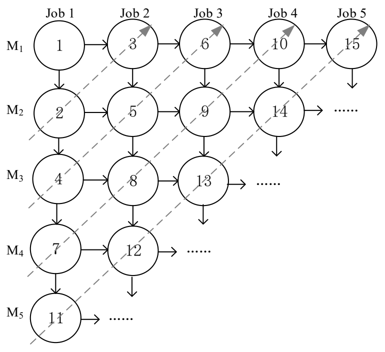


Fig. 4. Illustration of BMJ decoding

3.2 Balanced Machine-Job Insert Method

The pseudo-code of the balanced machine-job insert (BMJI) algorithm is shown in Algorithm 2. First, a temporary intermediate sequence is obtained using BMJ. Then, each job in the temporary intermediate sequence is taken out in turn and inserted into the optimal position of the final solution.

Algorithm 2. BMJI(π)

```
1:  $\Pi = \emptyset$;
2:  $\lambda = \mathrm{BMJ}(\pi)$;
3:  **While** sizeof($\lambda$) $>$ **0 do**
4:      Extract the first operation $O$ from $\lambda$;
5:      Test operation $O$ at all possible positions of $\Pi$;
6:      Insert operation $O$ at the position which result minimum increase of makespan;
7:  **endwhile**
8:  **return** $\Pi$;
```

3.3 Balanced Job-Machine Insert Method

The pseudo-code of the balanced job-machine insert (BMJI) algorithm is shown in Algorithm 3. The difference between the BJMI and the BMJI is that the temporary intermediate sequence of the BJMI is generated by the BJM.

Algorithm 3. BJMI(π)

```
1:  $\Pi = \emptyset$;
2:  $\lambda = $ BJM($\pi$);
3:  **While** sizeof($\lambda$) **do**
4:      Extract the first operation $O$ from $\lambda$;
5:      Test operation $O$ at all possible positions of $\Pi$;
6:      Insert operation $O$ at the position which result minimum increase of makespan;
7:  **endwhile**
8:  **return** $\Pi$;
```

3.4 Balanced Machine-Job Group Insert Method

The balanced machine-job group insert algorithm is based on BMJ algorithm, which divides all operations into *m*+*n* groups. As shown in Fig. 4, the operations connected by an arrow form a group. Compared with BMJ, the positions of operations in a group can be adjusted. The operations in each group can be inserted into the optimal position. The pseudo-code of BMJGI algorithm is shown in Algorithm 4.

Algorithm 4. BMJGI(π)

```
1:  $j = 1, i = 1, k = i + j, \Pi = \emptyset$;
2:  **While** $k \leq n + m$ **do**
3:      $l = $ sizeof($\Pi$);
4:      $\lambda = \emptyset$;
5:      **While** $i > 0$ **do**
6:          Append the $O_{i,j}$ to $\lambda$;
7:          $j++$;
8:          $i--$;
9:      **endwhile**
10:     $k++$;
11:     $j = 1$;
12:     $i = k - j$;
13:     **While** sizeof($\lambda$) $> 0$ **do**
14:         Extract the first operation $O$ from $\lambda$;
15:         Test operation $O$ from the $l^{th}$ position to the last position of $\Pi$;
16:         Insert operation $O$ at the position which result minimum increase of makespan;
17:     **endwhile**
18: **endwhile**
19: **return** $\Pi$;
```

3.5 Balanced Job-Machine Group Insert Method

This balanced job-machine group insert algorithm is based on BJM algorithm, which divides all operations into *m*+*n* groups, and the operations in each group can be inserted into the optimal position. The pseudo-code of BJMGI algorithm is shown in Algorithm 5.

Algorithm 5. BJMGI(π)

```
1:  $j = 1, i = 1, k = i + j, \Pi = \emptyset$;
2:  **While** $k < n + m$ **do**
3:      $l = $ sizeof($\Pi$);
4:      $\lambda = \emptyset$;
5:      **While** $j > 0$ **do**
6:          Append the $O_{i,j}$ to $\lambda$;
7:          $j--$;
8:          $i++$;
```

```
9:      **endwhile**
10:     $k++$;
11:     $i = 1$;
12:     $j = k - i$;
13:     **While** sizeof($\lambda$) $> 0$ **do**
14:         Extract the first operation $O$ from $\lambda$;
15:         Test operation $O$ from the $l^{th}$ position to the last position of $\Pi$;
16:         Insert operation $O$ at the position which result minimum increase of makesan ;
17:     **endwhile**
18: **endwhile**
19: **return** $\Pi$;
```

3.6 Greedy Algorithm

The pseudo-code of the greedy algorithm is shown in Algorithm 6. The greedy algorithm divides all operations into three categories. The first category is operations that have been added to the final solution Π. The second category is candidate operations (in Set S), which means these operations can be performed. The third category is the remaining operations (in Set V) that cannot be performed at present. The following steps are performed in each cycle. All operations in the candidate operation set S are attempted to be attached to the final solution Π, and the operation with the lowest completion time is selected as the current operation. The current operation is marked as completed, removed form S and appended to Π. Then, the operations in V that can be performed are moved to S.

Algorithm 6. Greedy(π)

```
1:  $V=\{O_{i,j}\}, i \in \mathcal{M}, j \in \mathcal{J}$;// the set of all operations
2:  $\Pi=\{O_{1,1}\}$; // the first operation;
3:  $S=\{O_{2,1},O_{1,2}\}$;// the set of candidate operations which can be performed;
4:  $V=V - S - \Pi$; // the set of operations which cannot be performed;
5:  **While** sizeof($\Pi$) $\leq m * n$ **do**
6:      Try to append each operation in $S$ to $\Pi$;
7:      $O'$ is the operation with the minimum completion time after being append to $\Pi$;
8:      $O'$ is removed from $S$ and append to $\Pi$;
9:      **for** each operation $O_{i,j}$ in $V$ **do**;
10:         **if** $O_{i-1,j} \in \Pi$ and $O_{i,j-1} \in \Pi$ **do**;
11:             $O_{i,j}$ is removed from $V$ and append to $S$;
12:         **endif**
13:     **endfor**
14: **endwhile**
15: **return** $\Pi$;
```

3.7 Beam Search Algorithm

The greedy algorithm can get results quickly, but its result may not be optimal. The enumeration algorithm can achieve the best result, but its computation is often too large. There is a compromise algorithm, named beam search (BS) algorithm, which can be used for the considered problem. The BS algorithm is an improved version of greedy algorithm, which differs from greedy algorithm in that greedy algorithm selects the best one each time, while the BS selects the best b candidates each time. The parameter b is an algorithm parameter, generally called beam size. The pseudo-code of beam search is shown in Algorithm 7.

Algorithm 7. BS(b)

```
1:  $\Pi_1=\{O_{1,1}\}$; // the first operation;
2:  $X=\{\Pi_1\}$;// the set of solutions
3:  **for** $k = 1$ to $m * n$ **do**
4:      $X_1=\emptyset$ ;
5:      **for** each solution $\Pi_k$ in $X$ **do**
6:          Try to append each operation that can be performed to $\Pi_k$, and put all new
              temporary solutions obtained into $X_1$;
7:      **endfor**
```

```
8:      $X$=$\emptyset$ and select the best $b$ solutions in $X_1$ and put them to $X$;
9:   endfor
10:  $\Pi$=best solution in $X$;
11:  return $\Pi$;
```

3.8 Improved Beam Search Algorithm (iBS)

The improved beam search algorithm (iBS) divides all operations into *m*+*n* groups, and the operation in each group resulting in minimum increase of makespan is append the final solution. The pseudo-code of the iBS algorithm is shown in Algorithm 8.

Algorithm 8. iBS()

```
1:   $j = 1, i = 1, k = i + j, \Pi = \emptyset$;
2:   While $k \leq n + m$ do
3:     $l$ = sizeof(Π);
4:     $\lambda = \emptyset$;
5:     While $i > 0$ do
6:       Append operation $O_{i,j}$ to $\lambda$;
7:       $j++$;
8:       $i--$;
9:     endwhile
10:    $k++$;
11:    $j = 1$;
12:    $i = k - j$;
13:    While sizeof($\lambda$) $> 0$ do
14:      for each operation in $\lambda$ do
15:        Try to append each operation to Π;
16:      endfor
17:      $O$ is the operation resulting in minimum increase of makespan;
18:      Remove operation $O$ from $\lambda$ and append it to $\Pi$;
19:    endwhile
20:  endwhile
21:  return $\Pi$;
```

4. Proposed Metaheuristics

First, three operators based on the problem characteristics are designed. Subsequently, based on three operators, ILS and ABC algorithms are redesigned for the considered problem.

4.1 Operators

Three operators are shift, swap, and hybrid.

The pseudo-code of shift is shown in Algorithm 9. An operation is randomly selected except the first and last one, because the positions of these two operations, namely, $O_{1,1}$ and $O_{m,n}$, are fixed (Line 3 in Algorithm 9). Suppose the selected operation is $O_{i,j}$. Then a random position (Line 5 in Algorithm 9) in its feasible position interval is selected. Then, it is checked whether the new solution generated after $O_{i,j}$ moves to the new location is feasible (Lines 6-8 in Algorithm 9). If there is no conflict, operation $O_{i,j}$ shifts to the new position (Line 9 in Algorithm 9). If there is conflict, try again.

Algorithm 9. Shift(Π)

```
1:   Bool Conflict = true;
2:   While Conflict == true do
3:     $k$ = rand()%$(m \times n - 2) + 2$;// Randomly select an operation expect $O_{1,1}$ and $O_{mn}$;
4:     $O_{i,j}$=$\Pi_k$;// Assume that the operation at position $k$ is $O_{i,j}$;
5:     $k'$=rand()%$((m - i + 1) \times (n - j + 1) - i \times j)$+$i \times j$;
         // Randomly select a position within its reasonable range.
6:     for each operation $O_{i',j'}$ between $k$ and $k'$ do
7:       if $O_{i',j'}$ and $O_{i,j}$ are conflicting do Conflict = true, break, endif;
8:     endfor
```

```
9:       if Conflict == false do Operation $O_{i,j}$ moves from position $k$ to position $k'$, endif
10:  endwhile
11:  return Π;
```

The pseudo-code of swap is shown in Algorithm 10. The swap operator is very similar to the shift operator. The swap operator exchanges the operations of two positions randomly selected. Of course, conflict checking is also required.

Algorithm 10. Swap(Π)

```
1:   Bool Conflict = true;
2:   While Conflict == true do
3:       $k_1$ = rand()%($m \times n - 2$) + 2;// Randomly select an operation expect $O_{1,1}$ and $O_{mn}$;
4:       $O_{i1,j1}$=$\Pi_{k1}$;// Assume that the operation at position $k_1$ is $O_{i1,j1}$;
5:       $k_2$=rand()%($(m - i1 + 1) \times (n - j1 + 1) - i1 \times j1$)+$i1 \times j1$;
             // Randomly select a position within its reasonable range.
6:       $O_{i2,j2}$=$\Pi_{k2}$;// Assume that the operation at position $k_2$ is $O_{i2,j2}$;
7:       for each operation $O_{i',j'}$ between $k_1$ and $k_2$ do
8:           if $O_{i',j'}$ and $O_{i1,j1}$ are conflicting do Conflict = true, break, endif;
9:           if $O_{i',j'}$ and $O_{i2,j2}$ are conflicting do Conflict = true, break, endif;
10:      endfor
11:      if Conflict == false do Swap $O_{i1,j1}$ and $O_{i2,j2}$, endif
12:  endwhile
13:  return Π;
```

The pseudo-code of hybrid operator is shown in Algorithm 11. The hybrid operator is a combined operator that performs shift with 50% probability and swap with 50% probability.

Algorithm 11. Hybrid(Π)

```
1:   $op = rand()\%2$;
2:   if $op == 0$ do
3:       Shift(Π);
4:   else
5:       Swap(Π);
6:   endif
7:   return Π;
```

4.2 Random Initialization

The random initialization algorithm generates a random initial solution, whose pseudo-code is shown in Algorithm 12. The random initialization method is very similar to Greedy Algorithm, except that the operations in S are randomly selected (Line 6 in Algorithm 12).

Algorithm 12. InitIndividualRandom()

```
1:   $V$=$\{O_{i,j}\}, i \in \mathcal{M}, j \in \mathcal{J}$; // the set of all operations
2:   $\Pi$=$\{O_{1,1}\}$; // the first operation;
3:   $S$=$\{O_{2,1}$,$O_{1,2}\}$;// the set of candidate operations which can be performed;
4:   $V$=$V - S - \Pi$; // the set of operations which cannot be performed;
5:   While sizeof($\Pi$) $\leq m * n$ do
6:       Randomly selected an operation $O'$ in $S$;
8:       $O'$ is removed from $S$ and append to $\Pi$;
9:       for each operation $O_{i,j}$ in $V$ do;
10:          if $O_{i-1,j}$∈$\Pi$ and $O_{i,j-1}$∈$\Pi$ do;
11:              $O_{i,j}$ is removed from $V$ and append to $S$;
12:          endif
13:      endfor
14:  endwhile
15:  return $\Pi$;
```

4.3 Iterated Local Search Algorithm

The iterated local search (ILS) algorithm is an improvement of local search method. It adds perturbation to the local optimal solution obtained by local search, and then carries out local search again (Qin et al., 2022). It is widely used in the field of combinatorial optimization because of its good performance. The ILS algorithm we designed is shown in Algorithm 13. An initial solution is generated by using the random initialization method in section 4.2. The initial solution is marked as the optimal solution. After initialization, ILS iteratively performs the perturbation procedure and acceptance criterion, until a stop condition is reached. A perturbation solution is obtained by performing a number $PLen$ of neighbourhood operators to the current local optimum. A total of $nPers$ perturbation solutions are obtained and then the best one is retained. For the acceptance criterion, the simulated annealing type criterion with temperature *t* calculated as in Eq.(16) is used.

$$t = TF \times \frac{\sum_{j=1}^{n}\sum_{k=1}^{m} p_{kj}}{10nm}; \tag{16}$$

Algorithm 13. ILS($nPers, OperType, PLen, TF$)

```
1:  $\Pi$ = InitIndividualRandom();
2:  $\Pi^* = \Pi$;// best solution so far
3:  $t$=temperature obtained from Eq.(16) with parameter $TF$;
4:  While (stop condition not met) do
5:     for $i$ = 1 to $nPers$ do
6:        $\Pi'(i) = \Pi$;
7:        for $j$ = 1 to $PLen$ do
8:           switch $OperType$ do
9:              case 0: $\Pi'(i)$ = Shift($\Pi'(i)$)
10:             case 1: $\Pi'(i)$ = Swap($\Pi'(i)$)
11:             case 2: $\Pi'(i)$ = Hybrid($\Pi'(i)$)
12:          endswitch
13:       endfor
14:    endfor
15:    $\Pi''$ = best solution among {$\Pi'(1)$,$\Pi'(2)$,…,$\Pi'(nPers)$};
16:    for C($\Pi''$)<C($\Pi$) then
17:       $\Pi = \Pi''$;
18:    elseif rand()<exp((C($\Pi$)-C($\Pi''$))/$t$) then
19:       $\Pi = \Pi''$;
20:    endif
21:    if C($\Pi''$)<C($\Pi^*$) then
22:       $\Pi^* = \Pi''$;
23:    endif
24: endwhile
25: return $\pi^*$;
```

4.4 Artificial Bee Colony Algorithm

The artificial bee colony algorithm is a new intelligent optimization algorithm to simulate the honey gathering process of bees (Yu et al., 2022). It is composed of three parts: food source, employed bees and non-employed bees (Tao et al., 2022). Its advantages are less control parameters, strong robustness and fast convergence.

4.4.1 Population Initialization

The pseudo-code of population initialization is shown in Algorithm 14. The population contains *PSize* individuals, each of which is generated by InitIndividualRandom.

Algorithm 14. InitPopRandom($PSize$)

```
1:  for i = 1 to PSize do
2:     $\Pi(i)$ = InitIndividualRandom();
3:     Put $\Pi(i)$ into $X$
4:  endfor
5:  return $X$;
```

4.4.2 Employed Bee Stage

The pseudo-code of the employed bee stage is shown in Algorithm 15. According to the parameter $OperType$, operator shift, swap, or hybrid is performed for each solution in the current population. All new solutions are stored in a temporary set X'.

Algorithm 15. Employed_Bee_Stage($X, OperType$)

```
1:  $X' = \emptyset$;
2:  for each solution $\Pi(i)$ in $X$ do
3:      $\Pi'(i) = \Pi(i)$;
4:      switch $OperType$ do
5:          case 0: $\Pi'(i) = \text{Shift}(\Pi'(i))$
6:          case 1: $\Pi'(i) = \text{Swap}(\Pi'(i))$
7:          case 2: $\Pi'(i) = \text{Hybrid}(\Pi'(i))$
8:      endswitch
9:      Put $\Pi'(i)$ into $X'$
10: endfor
11: return $X'$;
```

4.4.3 Onlooker Bee Stage

The pseudo-code of the onlooker bee stage is shown in Algorithm 16. For each onlooker bee, randomly select two individuals, and then select the better one from the two individuals for subsequent operation. According to the parameter $OperType$, operator shift, swap, or hybrid is performed for the better individual. All new solutions are stored in a temporary set X''.

Algorithm 16. Onlooker_Bee_Stage($X, OperType$)

```
1:  $X'' = \emptyset$;
2:  for i = 1 to $PSize$ do
3:      $\Pi'(i)$= solution selected from $X$ using binary tourmonent selection;
4:      switch $OperType$ do
5:          case 0: $\Pi'(i) = \text{Shift}(\Pi'(i))$
6:          case 1: $\Pi'(i) = \text{Swap}(\Pi'(i))$
7:          case 2: $\Pi'(i) = \text{HybridOperator}(\Pi'(i))$
8:      endswitch
5:      Put $\Pi'(i)$ into $X''$;
6:  endfor
7:  return $X''$;
```

4.4.4 Computational Procedure of the ABC algorithm

Finally, the overall flow of the ABC algorithm is shown in Algorithm 17. The initialization method is used to generate $PSize$ initial solutions, and the one with the highest fitness is marked as the optimal one. Then, it enters a loop. Then the operation in the employed bee stage and onlooker bee stage are executed. The solutions obtained in the employed bee stage are saved in X', while the solutions obtained in the onlooker bee stage is stored in X''. The best $PSize$ solutions in $X \cup X' \cup X''$ are selected as the population for next generation. The best solution is updated. Finally, when the termination condition is satisfied, ABC algorithm ends.

Algorithm 17. ABC($PSize, OperType$)

```
1: $X = \text{InitPopRandom}(PSize)$;//Randomly generate $PSize$ solutions;
2: $\Pi^* =$ best solution found so far;
3: while (terminiation criterion not satisfied) do
4:     $X' = \text{Employed\_Bee\_Stge}(X, OperType)$;
5:     $X'' = \text{Onlooker\_Bee\_Stge}(X', OperType)$;
6:     $X =$ best $PSize$ solutions from $X \cup X' \cup X''$;  //population updating
7:     Update $\Pi^*$;
8: endwhile
9: return $\Pi^*$;
```

4.5 Calibration

The parameters of the proposed ABC and ILS algorithm are adjusted to obtain their best performance. We determine the general range of parameters based on preliminary experiments. The parameters of the ABC algorithm are set as follows: $PSize \in \{100,200,300,\}$, $OperType\in\{0,1,2\}$. The parameters of the ILS algorithm are set as follows: $nPerS \in \{5,10,15,\}$, $OperType\in\{0,1,2\}$, $PLen\in\{1,3,5\}$, $TF\in\{0.3,0.5,0.7\}$. There are 9 and 27 parameter combinations for the ABC and ILS algorithm, respectively.

The instances for calibration experiments are derived from the literature (Wang and Wang, 2019) on peak power consumption. The number of machines m and jobs n in instances are set as follows: $n\in\{20,40,60,80,100\}$, $m\in\{4,8,16\}$. The standard processing time of jobs on machines is evenly selected from [5,50]. The set of speeds of machines is $S = \{1,1.3,1.55,1.75,2.1\}$. The actual processing time of jobs on machines is equal to the standard processing time divided by its speed. The instantaneous power consumption PP_s when a machine works at speed s equals $4s^2$(kw). We define the minimum power value $\underline{Q} = 4 \times 1^2 = 4$, and the maximum power value $\overline{Q} = 4 \times 2.1^2 \times m$. If $Q_{max} < \underline{Q}$, there is no feasible solution; if $Q_{max} \geq \overline{Q}$, all solutions are feasible. Ten power peaks are defined as $Q_k = \underline{Q} + \frac{\overline{Q}-\underline{Q}}{9} \times k$, (k = 0,1, … ,9), and are regarded as candidate values of Q_{max}. For m, n, and Q_{max}, there are 5×3×10=150 combination. An instance is generated for each combination where the processing sequence and processing speed of jobs are randomly generated with the constraint that the instantaneous power consumption of each job shall not be greater than Q_{max}. Finally, a total of 5×3×10×1=150 instances are generated.

For each instance, the ABC algorithm of each parameter combination independently repeated 10 times, and finally 150×10×9=13500 results were obtained. Similarly, the ILS algorithm of each parameter combination solved each instance once, and finally 150×1×81=12150 results were obtained. All algorithms in this paper were coded C++ in Visual Studio 2017. All experiments in this paper were conducted on an Inter(R) Xeon(R) CPU E5-2630 2.4 GHz with 16 GB of RAM running Windows 7 Standard 64 bits. The termination time of ABC and ILS was 6nm ms, where n and m are the numbers of jobs and machines in instances, respectively. The relative percentage increase (RPI) was used to evaluate the experimental results, shown in Eq. (17).

$$RPI = \frac{100 * (C - C^*)}{C^*} \tag{17}$$

where, C^* is the best solution obtained by all competing algorithms. Analysis of variance (ANOVA) (Li et al., 2022) are used to analyze the results. Fig. 5 and Fig. 6 show the results from ANOVA. Obviously, for ABC algorithm, $PSize$ and $OperType$ should be fixed at 200 and 0, respectively. For ILS algorithm, the optimal parameter configuration should be: $nPerS = 5$, $OperType = 2$, $PLen = 3$, $TF = 0.5$.

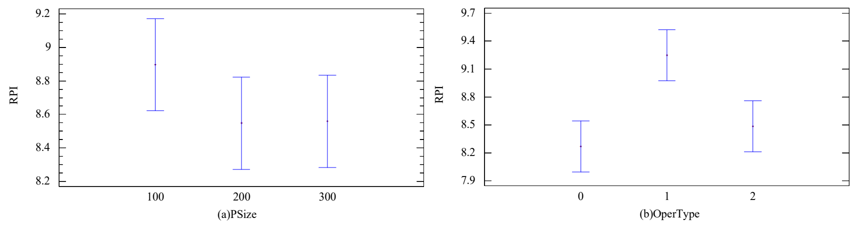


Fig. 5. Means plots for all factories for the ABC calibration.

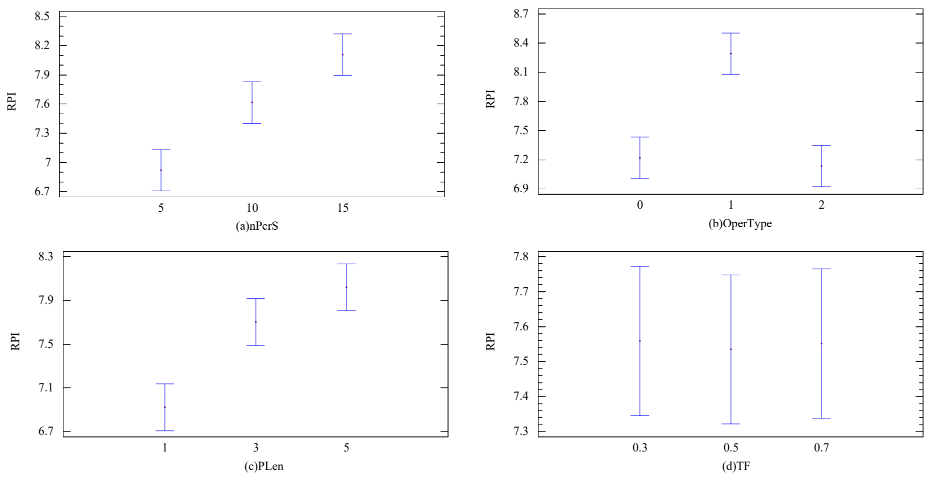


Fig. 6. Means plots for all factories for the ILS calibration.

5. Computational evaluation

Two groups of experiments were designed. The first group compared the heuristics. The second group compared ABC, ILS, and the best heuristic.

5.1 Comparison of Constructive Heuristics

Eight heuristics proposed in this paper are compared with 5 heuristics in the literature (Wang and Wang, 2019), including JP, MP, TLRL, ECT, and BJM. Thirteen heuristics are tested on small- and large-scale instances. For small-scale instances, $n \in \{8,12,16,20\}$, $m \in \{2,4\}$. Similarly, Q_{max} has ten values, as described in Section 4.5. For m, n, and Q_{max}, there are 4×2×10=80 combinations. One hundred instances are generated for each combination. A total of 4×2×10×100=8000 small-scale instances are generated. Other details of the small-scale instances are the same as those in Section 4.5. For large-scale instances, $n \in \{20,40,60,80,100\}$, $m \in \{4,8,16\}$. A total of 5×3×10×100=15000 large-scale instances are generated. The indicators compared include average RPI (ARPI) and NSR. NSR means the number of successes runs (Wang and Wang, 2019). Obviously, an algorithm with large NSR or small ARPI is better. Table 3 lists the ARPI for small-scale instances. BS($1m$) means b equal to $1*m$. The new heuristics have shown good performance. BJMI achieved the best results, and BJMGI was the second best. The fourth was iBS. BS($1m$) was the fifth. Greedy algorithm was the sixth.

Table 3
ARPI for heuristics(small instance)

(n, m)	BJM	BJMGI	BJMI	BMJ	BMJGI	BMJI	BS($1m$)	iBS	ECT	Greedy	JP	LTRL	MP
(8,2)	1.217	1.217	**1.124**	4.564	3.748	3.748	1.906	1.962	5.425	2.364	4.564	3.328	8.644
(12,2)	1.481	1.481	**1.308**	5.472	4.57	4.57	2.675	2.681	6.228	2.972	5.472	3.626	9.932
(16,2)	1.757	1.757	**1.394**	5.783	4.833	4.833	2.491	2.927	7.833	2.75	5.783	4.78	11.919
(20,2)	1.852	1.852	**1.569**	6.252	5.28	5.28	2.3	3.273	8.021	2.439	6.252	5.269	13.185
(8,4)	1.861	1.383	**1.193**	3.356	2.141	2.258	3.339	2.308	6.752	3.732	7.202	2.703	8.619
(12,4)	1.646	1.236	**0.974**	3.571	2.076	2.153	3.403	2.347	7.233	3.755	7.935	3.203	10.782
(16,4)	1.608	1.155	**0.811**	3.709	2.298	2.424	3.423	2.394	7.941	3.614	8.07	3.977	13
(20,4)	1.655	1.276	**0.844**	3.999	2.456	2.633	2.653	2.284	8.164	2.964	8.382	4.075	13.378
Mean	1.635	1.42	**1.152**	4.588	3.425	3.487	2.774	2.522	7.199	3.074	6.708	3.87	11.182
Rank	3	2	1	10	7	8	5	4	12	6	11	9	13
Q_{max}													
Q_1	2.978	2.255	**1.818**	6.112	4.831	4.801	3.694	3.474	11.549	4.339	11.405	5.751	19.88
Q_2	3.136	2.46	**1.986**	7.356	5.617	5.758	4.102	4.002	13.491	4.899	14.04	6.846	23.663
Q_3	2.956	2.374	**2.074**	7.739	5.934	6.108	5.087	4.351	14.055	5.658	13.075	6.782	22.006
Q_4	2.234	2.112	**1.706**	7.75	5.373	5.633	5.527	4.669	11.601	5.879	10.638	5.935	15.323
Q_5	2.105	1.999	**1.609**	7.365	5.563	5.615	4.025	3.828	10.177	4.301	8.164	6.402	16.705
Q_6	1.626	1.663	**1.343**	5.129	3.873	3.899	2.81	2.707	5.963	2.978	5.292	3.728	8.306
Q_7	0.975	0.989	**0.739**	3.242	2.27	2.27	1.888	1.603	3.872	2.05	3.27	2.438	4.585
Q_8	0.19	0.194	**0.135**	0.675	0.456	0.456	0.347	0.326	0.735	0.356	0.675	0.427	0.754
Q_9	0.146	0.149	**0.113**	0.516	0.334	0.334	0.259	0.261	0.551	0.277	0.516	0.39	0.601
Q_{10}	0	0	**0**	0	0	0	0	0	0	0	0	0	0
AVE	1.635	1.42	**1.152**	4.588	3.425	3.487	2.774	2.522	7.199	3.074	6.708	3.87	11.182

Fig. 7 (a) shows mean plots with 95.0% Tukey honest significant difference (HSD) confidence intervals of ARPI for small instances of all heuristics. For clarity, Fig. 7 (b) only shows mean plots and 95.0% Tukey HSD Intervals of ARPI for small instances of the best six heuristics. Fig. 7 (c) and (d) show the interactions of the algorithms and number of jobs, interactions of the algorithms and number of machines, respectively. As can be seen, the new proposed BJMI algorithm has achieved good results in each case.

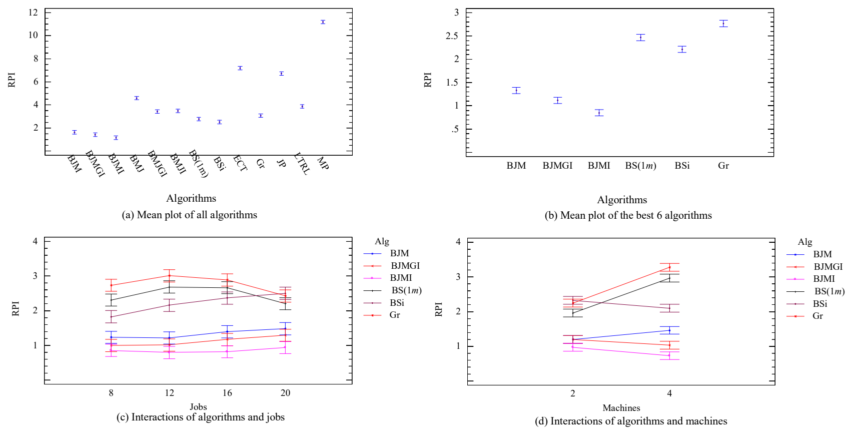


Fig. 7. Means plots, interactions and 95.0% Tukey HSD intervals of ARPI (small instances).

Table 4 lists the NSR of small-scale instances, which shows the advantages of the new heuristics. The best six heuristics are BJMI, BJMGI, BJM, BS($1m$), LTRL, and iBS, four (BJMI, BJMGI, BS($1m$), and iBS) of which are newly proposed.

Table 4
NSR for heuristics(small instance)

(n, m)	BJM	BJMGI	BJMI	BMJ	BMJGI	BMJI	BS($1m$)	iBS	ECT	Greedy	JP	LTRL	MP
(8,2)	809	809	**819**	491	567	567	679	688	478	624	491	594	427
(12,2)	688	688	**715**	356	406	406	508	499	352	484	356	499	374
(16,2)	610	610	**663**	314	364	364	487	441	303	465	314	405	304
(20,2)	560	560	**611**	251	280	280	483	345	272	453	251	368	280
(8,4)	565	617	**661**	497	558	560	488	511	463	470	472	578	514
(12,4)	537	567	**640**	439	480	486	445	457	412	417	426	540	467
(16,4)	507	538	**630**	396	438	438	402	431	364	384	387	459	397
(20,4)	464	504	**619**	358	388	384	425	404	352	402	355	466	411
Mean	592.5	611.63	**669.75**	387.75	435.13	435.63	489.6	472	374.5	462.38	381.5	488.6	396.8
Rank	3	2	1	11	9	8	4	6	13	7	12	5	10
Q_{max}													
Q_1	244	272	**358**	55	100	113	224	160	41	180	45	130	17
Q_2	248	296	**378**	56	88	88	220	139	26	163	42	110	14
Q_3	244	299	**352**	46	86	79	191	135	31	159	36	109	11
Q_4	317	308	**389**	63	107	110	161	149	68	143	57	179	75
Q_5	343	362	**421**	90	132	128	206	166	77	186	85	204	103
Q_6	448	457	**509**	235	296	295	292	301	223	270	231	389	281
Q_7	591	594	**632**	395	456	456	415	490	359	399	394	521	448
Q_8	749	749	**757**	664	695	695	684	707	668	680	664	727	700
Q_9	756	756	**762**	698	721	721	724	729	703	719	698	740	725
Q_{10}	800	800	**800**	800	800	800	800	800	800	800	800	800	800
AVE	474	489.3	**535.8**	310.2	348.1	348.5	391.7	377.6	299.6	369.9	305.2	390.9	317.4

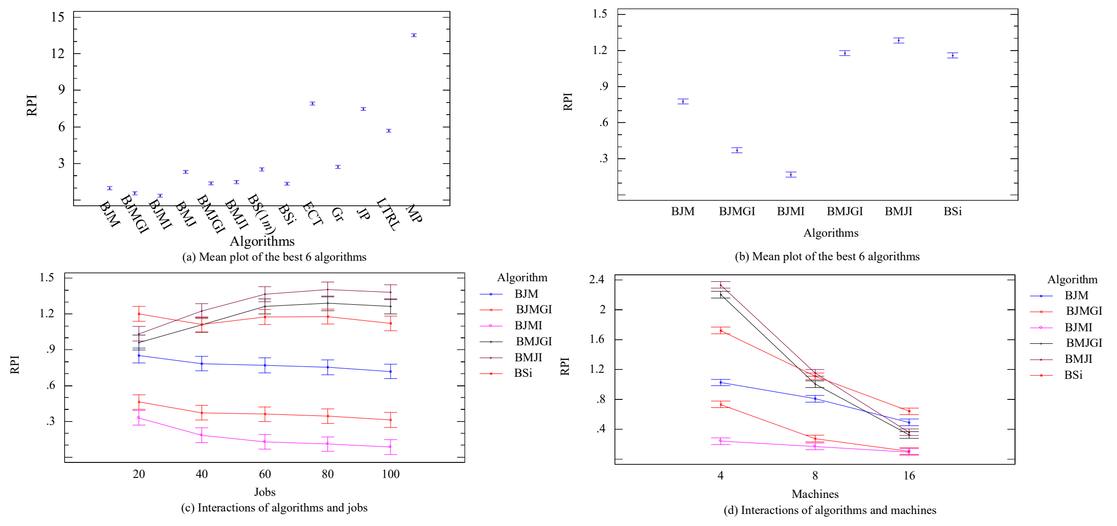
Tables 5 and 6 list the ARPI and NSR of large-scale instances, respectively. Fig. 8 shows means plots, interactions and 95.0% Tukey HSD intervals for large instances. The new heuristics also shows better performance for large-scale instances. BJMI is still the best. The best six heuristics for large-scale instances are BJMI, BJMGI, BJM, iBS, BMJGI, and BMJI, five (BJMI, BJMGI, iBS, BMJGI, and BMJI) of which are our new heuristics.

Table 5
ARPI for heuristics(large instance)

(n, m)	BJM	BJMGI	BJMI	BMJ	BMJGI	BMJI	BS($1m$)	iBS	ECT	Greedy	JP	LTRL	MP
(20,4)	1.497	1.165	**0.787**	3.631	2.257	2.337	3.456	2.151	8.556	3.599	8.002	4.513	14.09
(40,4)	1.375	1.061	**0.609**	4.01	2.474	2.637	2.32	1.969	9.919	2.451	8.628	5.862	17.32
(60,4)	1.548	1.277	**0.713**	4.512	2.885	3.012	1.734	2.295	9.79	1.806	9.315	5.861	18.31
(80,4)	1.59	1.316	**0.758**	4.644	2.972	3.104	1.515	2.38	10.55	1.643	9.391	6.018	18.62
(100,4)	1.482	1.198	**0.68**	4.397	2.839	2.987	1.145	2.198	10.11	1.295	9.147	6.353	19.25
(20,8)	0.898	0.405	**0.363**	1.501	0.766	0.877	3.125	1.166	6.905	3.319	7.004	3.07	9.848
(40,8)	0.869	0.33	**0.234**	1.844	0.997	1.16	2.939	1.131	8.049	3.148	7.949	5.871	13.85
(60,8)	0.889	0.335	**0.216**	2.075	1.212	1.344	2.815	1.22	8.283	3.089	8.417	6.669	15.5
(80,8)	0.897	0.341	**0.217**	2.123	1.227	1.403	2.499	1.22	8.911	2.777	8.647	7.732	17.07
(100,8)	0.922	0.382	**0.241**	2.166	1.248	1.421	2.338	1.243	8.795	2.6	8.648	7.723	17.15
(20,16)	0.524	0.177	**0.197**	0.572	0.222	0.254	2.293	0.648	4.476	2.508	4.449	2.002	5.182
(40,16)	0.517	0.136	**0.116**	0.682	0.272	0.294	2.772	0.647	5.75	3.05	5.243	3.89	7.277
(60,16)	0.497	0.094	**0.08**	0.767	0.329	0.379	2.902	0.643	6.07	3.206	5.496	6.038	9.178
(80,16)	0.511	0.113	**0.091**	0.892	0.421	0.462	2.961	0.683	6.187	3.157	5.945	6.532	9.679
(100,16)	0.5	0.102	**0.08**	0.928	0.468	0.501	2.872	0.676	6.333	3.138	5.901	7.373	10.44
Mean	0.968	0.562	**0.359**	2.316	1.373	1.478	2.512	1.351	7.912	2.719	7.479	5.701	13.52
Rank	3	2	1	7	5	6	8	4	12	9	11	10	13
Q_{max}													
Q_1	3.151	0.919	**0.381**	4.168	1.536	1.51	4.555	2.129	20.56	5.606	23	5.988	39.22
Q_2	1.958	0.818	**0.662**	3.402	1.643	2.054	5.737	2.54	19.13	6.285	18.57	10.47	32.03
Q_3	1.53	0.737	**0.47**	4.317	2.29	2.641	6.451	3.193	17.51	6.81	16.1	15.45	26.96
Q_4	0.959	0.88	**0.422**	4.929	3.237	3.483	4.422	2.616	12	4.462	9.575	14.55	21.21
Q_5	0.927	1.029	**0.633**	3.621	2.739	2.806	2.273	1.603	6.405	2.343	4.658	7.492	11.51
Q_6	0.684	0.747	**0.583**	1.772	1.448	1.452	1.067	0.904	2.5	1.06	1.913	2.22	3.139
Q_7	0.367	0.385	**0.342**	0.773	0.668	0.669	0.519	0.421	0.856	0.526	0.792	0.719	0.963
Q_8	0.084	0.09	**0.079**	0.155	0.143	0.143	0.084	0.089	0.139	0.084	0.155	0.102	0.127
Q_9	0.016	0.017	**0.015**	0.026	0.023	0.023	0.015	0.018	0.031	0.014	0.026	0.013	0.015
Q_{10}	0	0	**0**	0	0	0	0	0	0	0	0	0	0
AVE	0.968	0.562	**0.359**	2.316	1.373	1.478	2.512	1.351	7.912	2.719	7.479	5.701	13.52

Table 6
NSR for heuristics (large instances)

(n, m)	BJM	BJMGI	BJMI	BMJ	BMJGI	BMJI	BS($1m$)	iBS	ECT	Greedy	JP	LTRL	MP
(20,4)	484	527	**622**	394	424	424	389	425	346	371	386	427	385
(40,4)	427	430	**620**	323	344	345	377	377	283	355	323	354	310
(60,4)	362	361	**593**	261	276	276	398	304	287	370	261	341	316
(80,4)	336	330	**554**	245	250	250	387	279	275	352	246	327	302
(100,4)	331	329	**543**	259	260	260	405	290	247	364	258	309	283
(20,8)	614	701	**726**	577	625	618	481	580	486	481	559	583	545
(40,8)	557	652	**752**	497	541	524	423	540	424	421	491	479	461
(60,8)	519	630	**748**	435	454	448	385	493	395	380	433	464	452
(80,8)	488	586	**722**	414	448	427	373	463	384	372	412	423	417
(100,8)	459	581	**707**	401	429	414	379	449	359	367	399	426	411
(20,16)	721	791	**791**	717	780	767	610	708	628	609	697	717	678
(40,16)	663	743	**786**	621	687	673	523	643	539	521	601	620	599
(60,16)	636	753	**786**	577	641	600	474	620	512	472	572	565	550
(80,16)	612	708	**765**	558	626	584	467	598	492	469	547	549	537
(100,16)	603	706	**766**	525	577	541	443	577	484	443	523	528	511
Mean	520.8	588.5	**698.7**	453.6	490.8	476.7	434.27	489.7	409.4	423.1	447.2	474.1	450.5
Rank	3	2	1	8	4	6	11	5	13	12	10	7	9
Q_{max}													
Q_1	2	250	**782**	2	181	159	84	33	0	40	0	1	0
Q_2	27	466	**491**	4	276	94	113	22	0	60	0	6	0
Q_3	99	433	**805**	25	74	70	78	31	0	46	5	24	2
Q_4	444	491	**923**	145	153	151	104	229	39	98	113	144	93
Q_5	746	721	**904**	460	465	465	324	640	216	289	431	458	364
Q_6	985	960	**1040**	810	827	827	629	922	626	636	803	916	818
Q_7	1168	1171	**1194**	1080	1100	1099	949	1141	990	946	1078	1189	1124
Q_8	1384	1379	**1383**	1333	1339	1339	1287	1377	1321	1285	1333	1399	1384
Q_9	1457	1457	**1459**	1445	1447	1447	1446	1451	1449	1447	1445	1475	1472
Q_{10}	1500	1500	**1500**	1500	1500	1500	1500	1500	1500	1500	1500	1500	1500
AVE	781.2	882.8	**1048**	680.4	736.2	715.1	651.4	734.6	614.1	634.7	670.8	711.2	675.7



(a) Mean plot of the best 6 algorithms
(b) Mean plot of the best 6 algorithms
(c) Interactions of algorithms and jobs
(d) Interactions of algorithms and machines

Fig. 8. Means plots, interactions and 95.0% Tukey HSD Intervals (large instances)

5.2 Comparison of Metaheuristics

Large-scale instances are used to verify the performance of metaheuristics. The algorithms compared include two metaheuristic algorithms, ABC and ILS, and BJMI, which is the best among all heuristics. ABC and ILS have three termination times, namely 10, 20 and 30 *nm* ms, where *n* and *m* are the numbers of jobs and machines in instances.

Table 7 and 8 show the ARPI and NSR of three compared algorithms respectively. It can be seen that ILS achieves the best performance when it terminates at 30*nm* ms. In general, metaheuristic algorithms have better performance than heuristic.

Table 7
ARPI for metaheuristics (large instances, minimum ARPI values are in bold).

Termination time	BJMI	10*nm*		20*nm*		30*nm*	
Algorithm		ABC	ILS	ABC	ILS	ABC	ILS
(20,4)	4.309	0.546	0.494	0.462	0.163	0.109	**0.072**
(40,4)	5.052	1.004	0.903	0.84	0.317	0.132	**0.054**
(60,4)	5.515	1.135	1	0.938	0.369	0.153	**0.05**
(80,4)	5.671	1.359	1.168	1.078	0.509	0.178	**0.032**
(100,4)	5.641	1.549	1.294	1.201	0.594	0.214	**0.032**
(20,8)	2.677	0.84	0.767	0.733	0.205	0.068	**0.003**
(40,8)	2.834	1.245	1.111	1.051	0.347	0.113	**0**
(60,8)	2.795	1.356	1.17	1.091	0.566	0.182	**0**
(80,8)	2.747	1.419	1.156	1.051	0.715	0.248	**0.002**
(100,8)	2.349	1.162	0.865	0.727	1.026	0.508	**0.167**
(20,16)	1.137	0.657	0.599	0.572	0.222	0.068	**0**
(40,16)	0.689	0.638	0.527	0.483	0.501	0.182	**0.021**
(60,16)	0.494	0.708	0.586	0.52	0.707	0.402	**0.224**
(80,16)	0.442	0.725	0.597	0.527	1.027	0.71	**0.514**
(100,16)	0.418	0.792	0.661	0.586	0.861	0.598	**0.433**
Mean	2.851	1.009	0.86	0.791	0.542	0.258	**0.107**
	7	6	5	4	3	2	1
Q_{max}							
Q_1	6.291	3.028	0.696	2.4	0.239	2.118	**0.012**
Q_2	6.045	2.605	0.959	2.181	0.367	1.989	**0.068**
Q_3	5.397	2.336	1.819	2.073	1.031	1.954	**0.576**
Q_4	4.975	1.32	1.287	1.186	0.663	1.119	**0.33**
Q_5	3.777	0.702	0.502	0.664	0.198	0.636	**0.04**
Q_6	1.418	0.088	0.141	0.082	0.071	0.08	**0.04**
Q_7	0.509	0.011	0.007	0.01	0.003	0.01	**0.002**
Q_8	0.086	0	0.007	0	0.004	0	**0.003**
Q_9	0.016	0	0	0	0	0	0
Q_{10}	0	0	0	0	0	0	0
AVE	2.851	1.009	0.542	0.86	0.258	0.791	**0.107**

Table 8
NSR for metaheuristics (large instances, minimum ARPI values are in bold).

Termination time	BJMI	10*nm*		20*nm*		30*nm*	
Algorithm		ABC	ILS	ABC	ILS	ABC	ILS
(20,4)	421	612	791	630	868	642	**924**
(40,4)	348	498	585	527	751	544	**937**
(60,4)	291	458	486	476	650	493	**939**
(80,4)	264	443	435	453	551	474	**947**
(100,4)	273	434	412	444	490	454	**948**
(20,8)	589	658	710	662	813	668	**995**
(40,8)	513	564	598	565	667	565	**1000**
(60,8)	452	551	575	552	598	552	**997**
(80,8)	432	512	509	512	545	515	**993**
(100,8)	425	532	475	561	505	632	**863**
(20,16)	716	732	758	734	816	735	**1000**
(40,16)	663	673	668	679	703	685	**964**
(60,16)	702	645	585	650	606	660	**836**
(80,16)	711	667	550	684	571	696	**745**
(100,16)	743	642	549	664	559	681	**685**
Mean	502.9	574.7	579.1	586.2	646.2	599.7	**918.2**
Rank	7	6	5	4	2	3	1
Q_{max}							
Q_1	0	17	27	41	77	280	**1466**
Q_2	150	11	25	46	90	270	**1316**
Q_3	329	70	109	180	137	284	**1054**
Q_4	192	433	512	583	349	526	**1150**
Q_5	524	823	841	856	838	1019	**1396**
Q_6	882	1307	1318	1329	1245	1340	**1409**
Q_7	1135	1460	1461	1461	1468	1485	**1490**
Q_8	1373	1500	1500	1500	1483	1489	**1492**
Q_9	1458	1500	1500	1500	1499	1500	**1500**
Q_{10}	1500	1500	1500	1500	1500	1500	**1500**

AVE	754.3	862.1	879.3	899.6	868.6	969.3	**1377.3**

6. Conclusions

Energy conservation is a hot research topic at present, and this paper investigates the flowshop scheduling problems with peak energy consumption (PFSPP). To meet the limit of peak power consumption, the concurrency of jobs on different machines should be considered. The sequence of operations is taken as a solution and the problem characteristics are analyzed. Based on the problem characteristics, eight heuristics and two metaheuristics are proposed to solve the PFSPP. Many numerical experiments and comparisons show the superiority of the eight heuristics and two metaheuristics for solving the concerned problems.

The scheduling with peak power constrained is a very interesting topic, and there are many problems to be studied in the future. Future research work includes considering peak power consumption in hybrid flowshop scheduling problems (Çolak and Keskin, 2022; Meng et al., 2022) and multi-objective optimization of flow shop (F. et al., 2022; Zhang et al., 2022).

Acknowledgment

This research is partially supported by the National Natural Science Foundation of China 62173356 and 52205529, the Science and Technology Development Fund (FDCT), Macao SAR, under Grant 0019/2021/A, the Natural Science Foundation of Shandong Province (ZR2021QE195), and the Zhuhai Industry-University-Research Project with Hongkong and Macao under Grant ZH22017002210014PWC.

References

Chou, Y., Yang, J. & Wu, C. (2020). An energy-aware scheduling algorithm under maximum power consumption constraints. *Journal of Manufacturing Systems, 57*,182-197.

Çolak, M. & Keskin, G.A. (2022). An extensive and systematic literature review for hybrid flowshop scheduling problems. *International Journal of Industrial Engineering Computations, 13*(2),185-222.

Cui, W. & Lu, B. (2020). A Bi-Objective Approach to Minimize Makespan and Energy Consumption in Flow Shops with Peak Demand Constraint. *Sustainability, 12*(10),4110.

Zhao, F., Ma, R., & Wang, L. (2021). A self-learning discrete jaya algorithm for multiobjective energy-efficient distributed no-idle flow-shop scheduling problem in heterogeneous factory system. *IEEE Transactions on Cybernetics*, *52*(12), 12675-12686.

Fang, K., Uhan, N., Zhao, F. & Sutherland, J.W. (2011). A new approach to scheduling in manufacturing for power consumption and carbon footprint reduction. *Journal of Manufacturing Systems, 30*(4),234-240.

Fang, K., Uhan, N.A., Zhao, F. & Sutherland, J.W. (2013). Flow shop scheduling with peak power consumption constraints. *Annals of Operations Research, 206*(1),115-145.

Fernandez-Viagas, V., Sanchez-Mediano, L., Angulo-Cortes, A., Gomez-Medina, D. & Molina-Pariente, J.M. (2022). The Permutation Flow Shop Scheduling Problem with Human Resources: MILP Models, Decoding Procedures, NEH-Based Heuristics, and an Iterated Greedy Algorithm. *Mathematics, 10*(19),3446.

González-Neira, E., Montoya-Torres, J., & Barrera, D. (2017). Flow-shop scheduling problem under uncertainties: Review and trends. *International Journal of Industrial Engineering Computations*, *8*(4), 399-426.

Li, M., & Wang, G. G. (2022). A review of green shop scheduling problem. *Information Sciences*, *589*, 478-496.

Li, Y., Pan, Q., Gao, K., Tasgetiren, M.F., Zhang, B. & Li, J. (2021). A green scheduling algorithm for the distributed flowshop problem. *Applied Soft Computing, 109*(9),107526.

Li, Y., Pan, Q., Ruiz, R. & Sang, H. (2022). A referenced iterated greedy algorithm for the distributed assembly mixed no-idle permutation flowshop scheduling problem with the total tardiness criterion. *Knowledge-Based Systems, 239*(3),108036.

Luo, H., Du, B., Huang, G.Q., Chen, H. & Li, X. (2013). Hybrid flow shop scheduling considering machine electricity consumption cost. *International Journal of Production Economics, 146*(2),423-439.

Meng, L., Gao, K., Ren, Y., Zhang, B., Sang, H. & Chaoyong, Z. (2022). Novel MILP and CP models for distributed hybrid flowshop scheduling problem with sequence-dependent setup times. *Swarm and Evolutionary Computation, 71*, 101058.

Mishra, A., & Shrivastava, D. (2020). A discrete Jaya algorithm for permutation flow-shop scheduling problem. *International Journal of Industrial Engineering Computations*, *11*(3), 415-428.

Qin, S., Pi, D., & Shao, Z. (2022). AILS: A budget-constrained adaptive iterated local search for workflow scheduling in cloud environment. *Expert Systems with Applications*, *198*, 116824.

Ramezanian, R., Vali-Siar, M.M. & Jalalian, M. (2019). Green permutation flowshop scheduling problem with sequence-dependent setup times: a case study. *International Journal of Production Research, 57*(10),3311-3333.

Renna, P. & Materi, S. (2021). A Literature Review of Energy Efficiency and Sustainability in Manufacturing Systems. *Applied Sciences, 11*(16),7366.

Ribas, I. & Companys, R. (2021). A computational evaluation of constructive heuristics for the parallel blocking flow shop problem with sequence-dependent setup times. *International Journal of Industrial Engineering Computations, 12*(3),321-328.

Tao, X., Pan, Q. & Gao, L. (2022). An efficient self-adaptive artificial bee colony algorithm for the distributed resource-constrained hybrid flowshop problem. *Computers & Industrial Engineering, 169*,108200.

Wang, J. & Wang, L. (2019). Decoding methods for the flow shop scheduling with peak power consumption constraints. *International Journal of Production Research, 57*(10),3200-3218.

Wang, J. & Wang, L. (2020). A Knowledge-Based Cooperative Algorithm for Energy-Efficient Scheduling of Distributed Flow-Shop. *IEEE Transactions on Systems, Man, and Cybernetics: Systems, 50*(5),1-15.

Yu, Y., Zhang, F., Yang, G., Wang, Y., Huang, J. & Han, Y. (2022). A discrete artificial bee colony method based on variable neighborhood structures for the distributed permutation flowshop problem with sequence-dependent setup times. *Swarm and Evolutionary Computation, 75*,101179.

Zhang, B., Pan, Q., Meng, L., Lu, C., Mou, J. & Li, J. (2022). An automatic multi-objective evolutionary algorithm for the hybrid flowshop scheduling problem with consistent sublots. *Knowledge-Based Systems, 238*,107819.