RAIRO-Oper. Res. 60 (2026) 553–575
https://doi.org/10.1051/ro/2026005

RAIRO Operations Research
www.rairo-ro.org

THE PERMUTATION FLOW SHOP BATCH SCHEDULING PROBLEM: AN IMPROVED POPULATION-BASED ITERATIVE GREEDY ALGORITHM WITH SELF-ADAPTION AND SELF-VIEWING

Xiyang Liu[1,*], Fangjun Luan[1], Ming Zhao[1], Haomin Zhao[2] and Ye Zhang[1]

Abstract. To address production scheduling difficulties, efforts in academia and industry focus on achieving a balance of economic, environmental, and societal growth through green manufacturing scheduling. In this paper, the multi-objective permutation flow shop batch scheduling problem (MOPFSBSP) is optimized by taking makespan and machine emission noise into account. As a result of evaluating other NEH-based algorithms, the enhanced NEH_PR$_{SQ}$ algorithm yields a favorable initial solution after evaluating other NEH-based algorithms. The improved population-based iterative greedy algorithm (IPBIG) is then given a self-adaptation and self-viewing strategies to make it better at exploring. Then, a local search algorithm is suggested to apply mutation and replacement to sub-batches and sub-lots of each product to achieve the best solution. The algorithms presented in this research are tested on the Car, Rec, and Hel standard database instances and compared with traditional and innovative algorithms. The experimental data shows that the IPBIG algorithm outperforms other algorithms in optimizing over 74.19% of instances, particularly medium- and large-scale instances. Undoubtedly, the IPBIG algorithm offers a superior solution to the MOPFSBSP problem. It also significantly diminishes production noise, enhances operational efficiency for enterprises, and provides a novel trajectory for the sustained advancement of manufacturing firms.

Mathematics Subject Classification. 90B35, 90B36.

Received May 26, 2024. Accepted January 13, 2026.

1. Introduction

The green manufacturing [1–3] industry is gaining prominence due to heightened public consciousness and intensified competitiveness. The primary goal is to enhance efficiency and reduce environmental pollution [4, 5]. To improve productivity and minimize environmental harm, industrial companies should adopt efficient environmental preservation techniques [6] and utilize advanced technologies, such as green scheduling approaches [7].

The shop scheduling problem was categorized into three types: job shop problem (JSP) [8–10], flow shop problem (FSP) [11, 12], and open shop problem (OSP) [13, 14]. FSP allocates resources to jobs to improve target efficiency. While the permutation flow shop problem (PFSP), which aims for productivity, has been

Keywords. Green manufacturing scheduling, machine emission noise, MOPFSBSP, IPBIG.

[1] School of Computer Science and Engineering, Shenyang Jianzhu University, Shenyang 110168, P.R. China.
[2] 33 LiXian Street, Qixianling, Dalian High-tech Industrial Park, Daian, P.R. China.
*Corresponding author: liuxiyang0305@163.com

extensively addressed, few prioritize machine emission noise reduction. The multi-objective permutation flow shop batch scheduling problem is introduced in this paper. It adds batch [15] and lot streaming [16] to the general PFSP problem. The problem involves multiple sub-batches within each product, where each sub-batch consists of multiple jobs forming a sub-lot. The objectives of it include minimizing machine emission noise [17] and makespan [18]. It is different from the usual PFSP for multiple jobs, so it is more challenging to deal with sub-lots, and the processing time needs to be calculated according to the sub-lots of the product. It is a development and advancement in the field of PFSP. Although more sophisticated, MOPFSBSP corresponds to the current production environment and reduces stockpiling and completion dates.

Meta-heuristic algorithms [19] have shown excellent performance in scheduling problems, such as particle swarm optimization [20], differential evolution [21], ant colony optimization [22], firefly algorithm [23], cuckoo search algorithm [24], and bat algorithm [25]. Furthermore, the iterative greedy algorithm (IG) exhibits excellent performance [26]. The IG algorithm is a straightforward yet highly efficient method that achieves state-of-the-art results in many flow shop problems and their variations [27]. In contrast to approaches based on complex metaphors, IG is a search technique that does not possess memory and has very little structure. Simple encoding, comprehension, and adaptation to additional situations make the IG method appealing [28,29]. Many researchers have achieved successful results by utilizing the IG algorithm [27]. Considering the factors mentioned above, it can be anticipated that employing IG-based algorithms to solve MOPFSBP is going to produce superior outcomes.

This paper establishes two objective functions. The first objective, known as makespan, signifies the completion of the final job. Minimizing this objective can significantly enhance production efficiency and economic benefits for the enterprise while concurrently decreasing carbon emissions and minimizing noise generation duration. The second objective is the average minimum noise, which impacts the living environment of nearby residents and the health of enterprise personnel. Mitigating noise produced during the enterprise's operations is crucial for the long-term development of both the nation and its populace, so these two objectives have critical importance for the enterprise's long-term success. And then the MOPFSBSP is solved using the NEH algorithm [30] and the proposed IPBIG algorithm. We simplified the improved NEH-based heuristic method [31] to create the NEH_PR$_{SQ}$ algorithm for first-solution searching. Next, we design the IPBIG algorithm with three improvements over the classical IG algorithm [32] to improve exploration, solution quality, and algorithm stability. First, the IG algorithm uses a population, starting with the initial solution from the NEH_PR$_{SQ}$ algorithm. The population makes more quality solutions possible and prevents deviance from the track of high quality solutions. Second, the IPBIG algorithm incorporates self-adaption and self-viewing. The self-adaption strategy uses a formula to set the bottom border of the length of the destroyed coding segments; as a result, it adapts to changing sizes of problems and simulates real manufacturing. The self-viewing strategy can specify the upper limit of the destroyed coding segments to check if the objectives converge throughout iteration. It can then adjust the remaining iterations based on the outcome to discover the optimum solution in a more detailed optimization. Finally, three different local search algorithms are established to further investigate superior solutions. Then, the algorithm is evaluated through two experiments, comparing it with other classical and novel algorithms and demonstrating its superior exploration and stability capabilities.

2. Literature review

Experts have recently begun using heuristic algorithms, meta-heuristic algorithms, and mixed heuristic algorithms more frequently to address the PFSP [33]. Afterwards, this study will analyze them from multiple viewpoints.

2.1. Intelligent optimization algorithms

Intelligent optimization algorithms, such as heuristics and metaheuristics, are used to solve the FSP [34]. For example, the NEH algorithm [30], initially designed to solve the FSP, has become less accurate due to its simplicity but is still widely used to solve the initial solution. Hao-Xiang Qin *et al.* [35] developed an improved IG

algorithm to tackle the block hybrid flow shop problem. They used a standard simulated annealing criterion, a global perturbation approach based on a semi-exchange operator, to search for likely optimal solutions. However, the history demonstrates that experts have employed numerous intelligent algorithms to solve FSP. McCormick *et al.* [36] suggested using contour fitting to minimize blocking time and idle time. Ronconi [37] created the MME constructive heuristics approach for the FSP with blocking constraints. The heuristics mentioned previously demonstrated outstanding performance in initializing solutions. Therefore, these methods are also integrated into the initialization of the FSP. Suash Deb *et al.* [38] created RSA, a meta-heuristic search algorithm that draws inspiration from rhinoceros behavior. It enhances production scheduling by achieving a 3% improvement across various scales of the PFSP problem.

2.2. Permutation flow shop with single-objective

Since its initial discovery and investigation, the permutation flow shop problem, a well-established and thoroughly studied variant of FSP, has received significant contributions from several scholars. Jan Gmys *et al.* [39] developed a branch-and-bound algorithm (B & B) for effectively solving the permutation flow shop problem (PFSP). Their approach emphasizes parallel-tree exploration on multi-core central processing units (CPUs), with the goal of reducing the makespan. Mohamed Abdel-Basset [40, 41] proposed a new technique that combines the whale optimization algorithm with a local search approach to address the PFSP problem. This method specifically targets discrete search spaces and aims to optimize the makespan by using the largest rank value (LRV). Noor Aldeen Alawad [42] proposed DJRL3M, an improved version of discrete Jaya that incorporates refraction learning to produce numerous initial solutions and three mutation techniques to limit makespan. To minimize the makespan for the permutation flow shop scheduling problem (PFSP), Zixiao Pan [43] developed an effective deep reinforcement learning optimization technique. He created PFSPNet, a deep neural network that can produce end-to-end outputs regardless of problem size, trained it *via* behavioral critique reinforcement learning, and established an enhancement strategy to improve its solutions. It uses Deep Reinforcement Learning (DRL), Graph Isomorphic Network (GIN), and the IG algorithm to minimize the Permutation Flow Shop Problem (PFSP). Experimental results show that the RL+IG hybrid approach yields superior solutions quickly. Zhuoran Dong [44] suggested a novel learning-based method to minimize the Permutation Flow Shop Problem (PFSP) with Deep Reinforcement Learning (DRL), Graph Isomorphic Network (GIN), and IG algorithm. The experimental findings indicate that the suggested RL+IG combinatory method can achieve superior solutions rapidly. According to the research, it is evident that scholars studying PFSP typically fail to consider practical variables like energy consumption and environmental issues. As a result, there is a growing preference for multi-objective optimization in flow shop problems, especially in complicated industrial settings.

2.3. Green permutation flow shop

Recently, FSP scholars have become interested in green scheduling. Research on it dates back to the 1990s, including Boukas [45] and Janiak [46]. Chao Lu [47] researched an energy-efficient PFSP with a sequence-dependent setup and adjustable transport time using a hybrid multi-objective backtracking search algorithm (HMOBSA). By combining differential evolution, enhanced search, and job-switching mutation techniques, Xu Xin [48] improved the discrete whale swarm optimization algorithm (IDWSO). Its goals include performance and PFSP with sequential setup time. It also employs a new conveyor speed management method to reduce makespan and energy consumption. Yaping Fu [49] created a stochastic simulation-based multi-objective brainstorming optimization approach to solve a distributed permutation flow shop problem with real delay constraints. It reduces waste and energy use. As previously stated, research on green permutation flow shops focuses on time and cost savings. Environmental protection, energy consumption, workforce, productivity, and economic efficiency must be studied thoroughly and methodically.

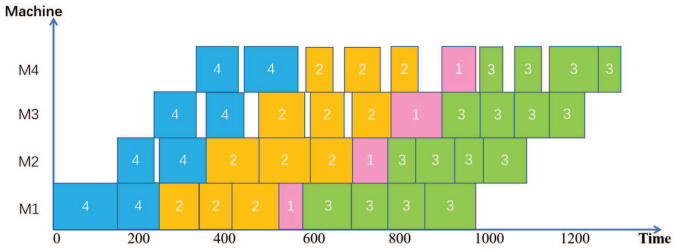


FIGURE 1. Gantt chart of MOPFSBSP.

2.4. Permutation flow shop batch scheduling problem

Feldmann [50] studied the multi-product batch flow problem in a PFSP and proposed a mixed integer programming formulation to find the optimal sub-batch size and sequence. Yuxia Pan [51] uses five meta-heuristic algorithms to solve this problem: Particle Swarm Optimization (PSO) [52], Genetic Algorithm (GA) [53], Harmony Search (HS) [54], Artificial Bee Colony (ABC) [55], and Jaya Algorithm [56]. These algorithms find the best way to divide up the jobs among the factories and arrange them so that the makespan is as short as possible. Wenyan Wang [57] proposed a two-stage discrete water wave optimisation (TSDWWO) algorithm to optimise sub-batching and sequencing for solving the permutation flow shop with lot-streaming.

From the above review and presentation, we can see that the literature in Section 2.1 primarily addresses general flow shop scenarios. Section 2.2 concentrates on the single-objective permutation flow shop scheduling problem. Section 2.3 introduces green scheduling within the permutation flow shop context, focusing on the dual objectives of makespan and total energy consumption. Section 2.4 discusses permutation flow shop batch scheduling, primarily targeting the optimization of sub-batch sizes or a single objective. Consequently, it is evident that previous studies have not simultaneously addressed both production efficiency and noise pollution as dual optimization objectives. The incorporation of sub-batches into the objective increases the complexity of the problem. This paper aims to construct a multi-objective production scheduling problem with sub-batches (MOPFSBP), optimizing it to enhance the economic efficiency of manufacturing enterprises while also contributing to environmental protection efforts.

3. Problem description and modelling

MOPFSBSP is an extension of PFSP that incorporates lot streaming. To clarify the problem presented in this paper, we can refer to the Gantt chart in Figure 1. The figure shows four machines, each assigned to produce a specific process. Products are manufactured in the same sequence on each machine. A sub-lot [58] of products is produced without waiting for the completion of all other remaining sub-batches of the product on that machine, and it continues directly to the next production stage. This kind of problem enhances the process, reduces delivery time [59], work in process inventory [60], transportation constraints [61], and loads on the material transport systems [62], etc. As a result, learning how to optimize these problems benefits the current manufacturing industry by improving its economic efficiency and development.

To describe the MOPFSBSP, the following assumptions are proposed:

(1) Each machine can process only one sub-batch of a product at a time.
(2) Each sub-batch can be processed by only one machine at a time.
(3) The products are processed in the same order on each machine.
(4) Once a sub-batch has started processing on one machine, it cannot be interrupted.
(5) The preparation time for all sub-batches is not taken into account.
(6) The start time is zero.

The symbols used in the mathematical model are defined in Table 1.

TABLE 1. Symbols definition.

Indexes	Meaning
i	Machine index, $i = 1, 2, \ldots, M$
k	Index of the position of the product's sub-batch in the production sequence, $k = 1, 2, \ldots, K$
j_k	Index of the jobs at position k of the production sequence, $k = 1, 2, \ldots, K$
Parameters	**Meaning**
M	Number of machines
K	Total number of all sub-batches in the production sequence
J_k	Total number of jobs contained in the sub-batch at position k of the production sequence
$p_{j_k,i,k}$	Processing time of job j_k on machine i at position k of the production sequence
$P_{j_k,i,k}$	Total processing time on machine i at position k of the production sequence for the sub-batch in which job j_k is located
$C_{i,k}$	Completion time of the sub-batch at position k of the production sequence on machine i
v	Speed gear, $v = 1, 2, 3$
No_v	Noise per unit time at different speed gears, $\mathrm{No}_1 = 200$, $\mathrm{No}_2 = 250$, $\mathrm{No}_3 = 300$

The loudness of sound can generally be described using physical quantities such as sound pressure and sound pressure level, sound intensity and sound intensity level, sound power and sound power level. Typically, the ratio of two physical quantities – the sound pressure level, measured in decibels (dB) is used to quantify sound, describing the magnitude of the physical quantity in the numerator relative to the reference in the denominator. In flow shop problem, noise primarily originates from machine operation and the contact between jobs and machinery during processing. For these structural noises, when the machinery operates at a certain power level, the noise radiates as a steady broadband signal. According to the "Environmental Noise Emission Standard for Boundaries of Industrial Enterprises", this paper evaluates the radiation of machining noise using Class A sound pressure, where the equivalent sound level can be expressed as:

$$\mathrm{Leq} = 10 \lg \left(\frac{1}{T} \int_0^T 10^{0.1 l_i} \, \mathrm{d}t \right) \tag{1}$$

where l_i is the instantaneous A-weighted sound level at time t; T is the measurement time interval. When the machine operates at a constant power, the radiated sound levels during each phase can be considered fixed and unchanging. Therefore, the sound power level can also be expressed as:

$$\mathrm{Leq} = 10 \lg \frac{\sum_i 10^{0.1 l_i} T_i}{\sum_i T_i} \tag{2}$$

where l_i denotes the instantaneous A-weighted sound level for time interval i; T_i represents time interval i. Given the research focus of this paper and considering that different noise sources from each machine result in variations in instantaneous sound pressure levels, the noise generated throughout the entire machining process is determined by the distinct sound pressure levels produced during the operation of each machine, it is also the first objective studied in this paper:

$$\min f_1 = 10 \log \left(\frac{\sum_{i=1}^{M} \sum_{k=1}^{K} 10^{0.1 \mathrm{No}_v} P_{j_k,i,k}}{\sum_{i=1}^{M} \sum_{k=1}^{K} P_{j_k,i,k}} \right). \tag{3}$$

In this paper, we solve two objectives: the average minimum emission noise of the machine [63] (As in Eq. (3)) and the makespan (As in Eq. (4)). The model is proposed below:

$$\min f_2 = C_{\max} \tag{4}$$

$$C_{i,k} = \begin{cases} P_{j_1,1,1}, & \text{if } i = 1, k = 1 \\ C_{1,k-1} + P_{j_k,1,k}, & \text{if } i = 1, k = 2, 3, \ldots, K \\ C_{i-1,1} + P_{j_1,i,1}, & \text{if } k = 1, i = 2, 3, \ldots, M \\ \max\{C_{i-1,k}, C_{i,k-1}\} + P_{j_k,i,k}, & \text{if } i > 1, k > 1 \end{cases} \tag{5}$$

$$C_{\max} \geq C_{i,k} \tag{6}$$

$$P_{j_k,i,k} = \sum_{j_k=1}^{J_k} p_{j_k,i,k}, \qquad k = 1, 2, \ldots, K. \tag{7}$$

Equation (3) figures out the average minimum noise made during production using the "Emission standard for industrial enterprises noise at boundary" [64] (another goal). The standards set environmental noise emission limit values and measurement methods for domestic industrial enterprises and can be used for managing, evaluating, and controlling industrial emissions. Equation (4) minimizes makespan, which is the main goal of this paper. Equation (5) represents the constraints for flow shop batch scheduling problem. The first equation in the expression denotes the completion time of the first sub-batch produced on the first machine, which is equal to the total processing time of that sub-batch. The second equation denotes the completion time of the sub-batch at position k of the production sequence on the first machine. The third equation represents the completion time of the sub-batch at the first production position on machine i, and the fourth equation represents the constraint on the completion time of the sub-batches. Equation (6) constrains the completion time, ensuring that the completion time of the last sub-batch is no more than the makespan. Equation (7) defines the processing time of a sub-batch as the total of the processing times of its jobs, calculated by multiplying the processing time of a single job by its number of jobs. The paper mathematical model adds parameters and indices to represent sub-batches and sub-lots to the regular PFSP model, making it better for MOPFSBSP.

4. Description of the proposed algorithm

This section details the operational procedures of the algorithms presented in the paper, covering the experimentation and selection of the NEH-based algorithms, the implementation of self-adaption and self-viewing strategies in the IPBIG algorithm, and the optimization of the solution of the local search algorithm.

4.1. NEH-based algorithm for the initial high-quality solutions

The NEH algorithm is a classical heuristic method. Based on the NEH, we first propose a swap method named NEH_SM, which significantly reduces the running time of the classic NEH, and then propose a new priority rule, denoted as NEH_PR_{SQ}, to obtain a valid initial sequence.

4.1.1. Classical NEH algorithm

The NEH algorithm [30] provides an efficient heuristic method for solving the FSP and is commonly used for creating initial solutions. Like most of the heuristic algorithms used for solving FSP, it is a heuristic method designed for solving a scheduling scheme with m machines and n jobs. It is straightforward and practical for implementation.

The operation of the NEH can be described in the following steps:

(1) Creates an input sequence for all products in a non-increasing order of processing time.
(2) Adds the first product in the input sequence to an empty planned partial sequence.
(3) Adds each remaining product in the input sequence to the partial sequence of planned products until all products in the input sequence have been designed.

As described above, the classical NEH algorithm prioritizes objects with longer processing times, and the limitations are clear: outliers or extreme values in processing time will directly impact the optimization capability of algorithms.

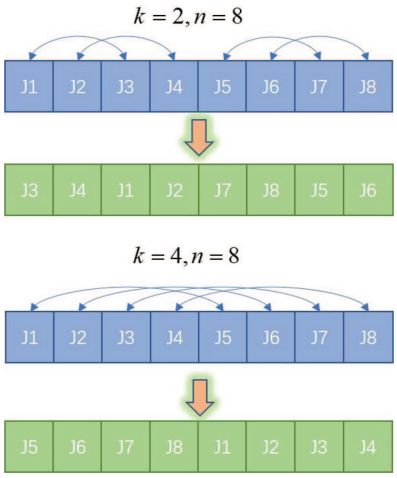


FIGURE 2. Process of NEH_SM algorithm.

4.1.2. NEH_SM algorithm

This section modifies the way the NEH algorithm creates input sequences by using the Swap Method (SM) to improve the N-NEH+ algorithm [65], namely NEH_SM, which does not need to re-insert the products one by one like the classical NEH algorithm. By directly exchanging product positions, the running time is significantly reduced. On the other hand, this will negatively impact the capability of schedule optimization. Thus, the decision to utilize the NEH_SM algorithm should be based on the current production circumstances. The pseudo-code is as follows:

Algorithm NEH_SM

Input: Sequence of Product $J_1, \ldots, J_n$
Output: Sequence of Product recorded by using Swap Method
for $i = 1$ to $n/2k$ do
    $nr = (i-1)2k$
    for $j = 1$ to k do
        SWAP $(J[nr+j], J[nr+j+k])$
    end for
end for

As shown in Figure 2, the initial products are produced in the order of $\{J_1, J_2, J_3, J_4, J_5, J_6, J_7, J_8\}$.

4.1.3. NEH_PR$_{SQ}$ algorithm algorithm

The classical NEH prioritizes products with longer processing times. The prioritization rule was updated by Dong *et al.* [66] to include the standard deviation of processing time for products. Their experiments successfully demonstrated the use of the standard deviation as a metric for quantifying the processing time of a product variability. According to Liu *et al.* [67], mean absolute deviation, kurtosis, and skewness have been introduced to the distribution of processing time to improve accuracy. This paper selects the NEH_PR$_{SQ}$ heuristic algorithm, which considers both the standard deviation and the inclusion of the quartile deviation. Unlike the mean and standard deviation, the quartile bias [68] is not affected by extreme observations, which makes it a better measure of central tendency and dispersion for highly skewed data. As the quartile deviation increases, the data in the center becomes more dispersed. The priority rule of the NEH_PR$_{SQ}$ algorithm is to prioritize products

with a longer average processing time, taking into account both the standard and quartile deviations. So, the higher the standard deviation and quartile deviation of the processing time, the higher the priority of product.

To eliminate dimensional effects between indicators, normalization is performed to balance the concentration trend and dispersion of the processing time of the products on the machine. As shown below, the products are arranged in ascending order:

$$A(i) = \eta \times \frac{\mathrm{AVG}_i - \min(\mathrm{AVG})}{\max(\mathrm{AVG}) - \min(\mathrm{AVG})} + (1-\eta) \times \left(\frac{\mathrm{STD}_i - \min(\mathrm{STD})}{\max(\mathrm{STD}) - \min(\mathrm{STD})} + \frac{\mathrm{QD}_i - \min(\mathrm{QD})}{\max(\mathrm{QD}) - \min(\mathrm{QD})} \right) \tag{8}$$

where $\eta \in (0,1)$, AVG_i denotes the average processing time of product i, $T_{[i],k}$ is the processing time of the ith product on machine k for the production sequence π in the PFSP, STD_i and QD_i represent the standard deviation and quartile bias of the processing time of product i. As a mathematical expression, we have the following:

$$\mathrm{AVG}_i = \frac{1}{m} \sum_{k=1}^{m} T_{[i],k} \tag{9}$$

$$\mathrm{STD}_i = \left[\frac{1}{m-1} \sum_{k=1}^{m} \left(T_{[i],k} - \mathrm{AVG}_i \right)^2 \right]^{\frac{1}{2}}. \tag{10}$$

We consider each product as an individual and sort these products in ascending order of processing time. x_1 and x_3 are the positions where the first and third quartiles of the product sequence are located, Q_1 and Q_3 refer to the products represented by the positions x_1 and x_3, respectively, and n is the number of products, defined as follows:

$$\mathrm{QD}_i = \frac{Q_3 - Q_1}{2} \tag{11}$$

$$x_1 = \frac{n+1}{4} \tag{12}$$

$$x_3 = \frac{3(n+1)}{4}. \tag{13}$$

The pseudo-code for NEH_$\mathrm{PR_{SQ}}$ is shown below:

4.1.4. Comparison of NEH-based algorithms

To optimize the objectives, we introduced three NEH-based algorithms and conducted a comparison experiment between them. The instances are chosen from the Rec standard database, and the results are generated by running each instance randomly ten times. Table 2 demonstrates that the NEH_$\mathrm{PR_{SQ}}$ algorithm outperforms the other two algorithms in terms of the maximum, minimum, and average values of the two objectives. It has been reported that the favorable rate for the average minimum noise has reached 55.56%, and the dominance rate for makespan has increased by 66.67%. As a result of the above analysis, the NEH_$\mathrm{PR_{SQ}}$ algorithm will be utilized in this paper to produce the first solution.

4.2. IPBIG algorithm with self-adaption and self-viewing strategies

The IG algorithm improves solutions *via* heuristics. It improves local search by using destruction and reconstruction to uncover better solution schemes. They have effectively been utilized to address PFSP optimization [69]. However, when local exploration algorithms are used, the normal IG algorithms, which use a constant parameter value in the destruction, are not good enough to deal with MOPFSBP problems of different sizes, such as small, medium, and large problems. If the parameter setting is excessively large, it will significantly enhance the time cost advantage of small problem sizes, thereby increasing time complexity. Conversely, if the parameter setting is too small, the optimization of the objective function for medium and large-scale problems

Algorithm NEH_PR$_{SQ}$

//Initial ordering

for $i = 1$ to n do, Compute

$$A(i) = \eta \times \frac{\mathrm{AVG}_i - \min(\mathrm{AVG})}{\max(\mathrm{AVG}) - \min(\mathrm{AVG})} + (1-\eta) \times \left(\frac{\mathrm{STD}_i - \min(\mathrm{STD})}{\max(\mathrm{STD}) - \min(\mathrm{STD})} + \frac{\mathrm{QD}_i - \min(\mathrm{QD})}{\max(\mathrm{QD}) - \min(\mathrm{QD})} \right)$$

end for

//Insert procedure

Input: $A(i)$

Output: Sequence with minimum makespan.

Initial phase: Sort n products in non-increasing order of $A(i)$ into the initial list of products $L = \{1, \ldots, n\}$

// Step 1

Schedule the first product and remove it from L

// Step 2

for $j = 2$ to n do

Insert the product j in the place that minimizes the partial makespan among the j possible ones

// Step 3

Remove product j from the list.

end for

TABLE 2. Comparative experiments of NEH algorithms.

Instance	Size	NEH						NEH_SM						NEH_PR$_{SQ}$					
		No_{min}			Makespan			No_{min}			Makespan			No_{min}			Makespan		
		Min	Mean	Max	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
Rec1	20 × 5	687.7296	688.1581	689.0541	333 081	351 114	385 514	687.7296	688.1856	689.0541	339 616	379 086	418 649	687.1787	688.1380	689.2264	310 121	327 423	342 733
Rec3	20 × 5	687.4299	688.1466	688.5565	278 830	301 260	351 119	687.4299	688.1064	689.2105	268 798	334 843	400 644	687.6368	688.1050	688.5002	280 247	293 826	306 309
Rec5	20 × 5	686.0601	687.8363	689.0859	321 930	330 753	340 986	686.0601	687.8276	689.0223	335 982	391 677	439 889	685.9663	687.8270	689.0546	319 649	337 976	353 972
Rec7	20 × 10	687.1990	687.5357	687.9728	369 298	390 444	420 155	687.1990	687.5570	687.9728	390 883	441 872	527 050	687.1920	687.5547	687.9143	365 016	391 110	437 047
Rec9	20 × 10	687.3819	687.8592	688.1574	377 939	403 455	460 045	687.3819	687.8383	688.1574	379 128	431 319	473 871	687.4812	687.8669	688.2681	379 128	389 568	395 628
rec11	20 × 10	687.0620	687.7616	687.9946	340 106	359 860	423 932	687.0620	687.7625	687.9946	390 745	400 928	423 932	687.0301	687.7736	687.9877	324 060	353 330	381 865

will be considerably less effective. To improve solution results, local search algorithms must avoid search range limitations and prioritize diversity [70]. Thus, new algorithms are needed to handle the different sizes of problems while providing high-quality solutions at the start and end of performance, as well as further research. The IPBIG algorithm enhances the adaptability and exploration of IG algorithm without increasing temporal complexity. This spurred algorithmic development.

The initial solution produced by the NEH_PR$_{SQ}$ algorithm is the first in the population; subsequent initial solutions are generated at random. The IG algorithm comprises two components: destruction and reconstruction. During the destruction process, the IPBIG algorithm introduces self-adaption and self-viewing, allowing for a variable number of destroyed products. The self-adaption strategy establishes lower bounds on the length of destroyed coding segments based on the size of instances to achieve self-adaption. As the number of iterations increases, the self-viewing strategy lengthens the destroyed coding segments, depending on the degree of convergence of the objectives. An upper limit on the number of destroyed coding segments will be determined in order to prevent convergence stagnation, enhance the exploration process, and increase the likelihood of producing a high-quality solution.

4.2.1. Encoding method

Encoding is an important strategy for the FSP. We utilize the sliding buoy approach to partition sub-batches, allowing for a straightforward determination of sub-batch size, making it more appropriate for the best solution. Its particular execution is illustrated in Figure 3, assuming a total of ten lots of the product J. A partition is used to locate the sliding buoy. If two sliding buoys are in the same position, one sub-batch will be divided less. The product J_1 in Figure 3 is segmented into three sub-batches; these sub-batches contain the number of jobs in the first half of the encoding, and the second half contains the order in which the products are destroyed.

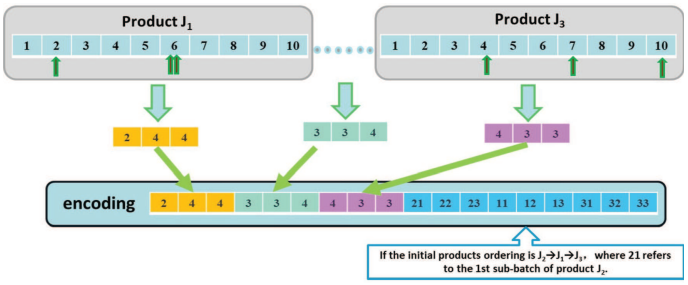


FIGURE 3. Encoding method.

4.2.2. Destruction of IPBIG

The destruction of the IPBIG algorithm refers to removing a partial coding segment from a code. Since we consider the batch scheduling problem, the destruction is to delete the whole coding segment representing a product rather than the partial coding segment representing a sub-batch in the product. The length of destroyed coding segments affects the result of objectives, so we set the initial length of destroyed coding segments according to the size and processing time of the instance, which can better reflect the information contained in the instance than the IG algorithm with fixed destroyed coding segments, preventing the algorithm from running for a long time redundantly or with poor exploratory ability due to the destroyed coding segments being too long or too short.

The equations of the destruction in the proposed IPBIG algorithm are as follows:

$$d = \frac{N(f_{\text{NEH}} - f_{\text{mean}})}{f_{\text{NEH}}} + m \tag{14}$$

$$d_{\min} = \begin{cases} d, d < \frac{N}{2} \\ \left[\frac{n}{3}\right], d \geq \frac{N}{2} \end{cases} \tag{15}$$

$$d_{\max} = \left[\frac{N}{2}\right] \tag{16}$$

where, f_{mean} is the average value of objectives of each coding in the initial population generated, f_{NEH} is the value of the objectives of the initial solution obtained by the NEH_PR$_{\text{SQ}}$ algorithm, N is the number of products, m is a constant, generally set $m = 2$, and $d_{\min}$ is the length of the initially destroyed coding segments, and the value of $d_{\min}$ needs to be set according to the size of d.

Many experiments in the previous period have proved that if the number of times to view the convergence of the objectives is set to two times during the iteration process, it will have less impact on improving the ability to optimize the objectives. Selecting more than three viewing opportunities will waste viewing time in small and medium-sized problems, increase the difficulty of optimizing the solution in large-scale problems, and significantly increase the running time. Therefore, taking into account the time cost factor, we select three viewing opportunities. The operation is as follows: the viewing position is evenly distributed in three places according to the number of iterations. Suppose the objectives of the last five generations no longer converge at the time we see them. In this case, the length of the coding segment is destroyed by adding one before processing, with the remaining iterative until the end of the three-time view operation. However, the maximum length of the destruction coding segment is set to $d_{\max}$. Suppose $d_{\max}$ is reached, but the times of view does not get three times. In that case, the next self-viewing will no longer change the length of the destroyed coding segment to prevent it from being too long and causing unnecessary running time and reconstruction difficulty to the algorithm.

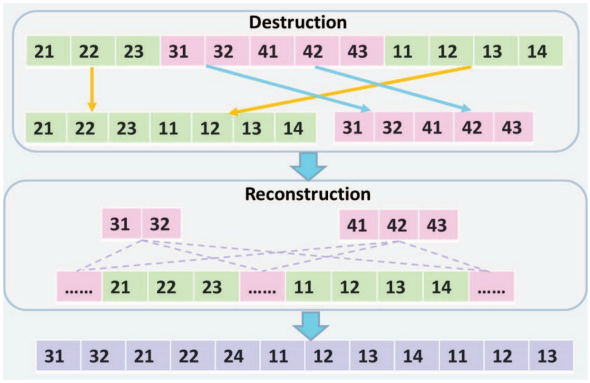


FIGURE 4. IPBIG algorithm.

4.2.3. Reconstruction of IPBIG

The reconstruction of the IPBIG involves sequentially reinserting the destroyed code segments into all possible positions of the undestroyed code. The optimal position is selected based on objectives, and the process continues until all code segments are fully reinserted.

The precise process of the destruction and reconstitution of the IPBIG algorithm is illustrated in the example diagram of Figure 4. It illustrates four products: Product 1 comprises four sub-batches, Product 2 contains three sub-batches, Product 3 has two sub-batches, and Product 4 includes three sub-batches. In the destruction phase, we eliminate Products 3 and 4 from the original sort. Next, we consolidate all remained sub-batches from each product and re-insert the sub-batches of Products 3 and 4 into all available positions in the sort. After calculating the objective, we must arrange the two sub-batches of Product 3 before all sub-batches of Product 2, and the sub-batches of Product 4 after all sub-batches of Product 1 to achieve the optimal production sorting using the latest sorting methodology.

Figure 5 illustrates the flowchart of the IPBIG algorithm, detailing the progression from acquiring an initial high-quality solution *via* the NEH_PR$_{SQ}$ algorithm to the local search strategy outlined in Section 4.3, thereby depicting the entire optimization process of the IPBIG algorithm.

The pseudo-code of the IPBIG algorithm is shown below in Algorithm IPBIG:

4.3. Local search algorithm for deep optimization

Following the NEH_PR$_{SQ}$ and IPBIG algorithms, each code in the population will undergo a local search algorithm. The proposed local search algorithm can be executed independently for sub-batches, making it a good fit for the MOPFSBSP. It is categorized into three sorts of operations: (a) batch mutation, (b) batch replacement, and (c) lot replacement. Figure 6 displays the specific operations.

Batch mutation: as shown in Figure 6, product J_1 has three sub-batches with sub-lots of 2, 4, and 4. The sub-lot size of the first sub-batch of product J_1 mutates from 2 to 3; the sub-lot size of the second sub-batch of J_1 mutates from 4 to 3, and the sub-lot size of the third sub-batch of J_1 is determined by the size of the first two sub-lots of J_1, so it should be 4. Where, for this size problem we set the sub-lot of the sub-batch with a maximum of four and a minimum of two.

Batch Replacement: Batch replacement is the replacement of the production order of sub-batches within a particular product, as shown in Figure 6, where the first sub-batch and the second sub-batch of product J_2 are replaced, the first sub-batch and the third sub-batch of product J_1 are replaced, and the first sub-batch and the second sub-batch of product J_3 are replaced.

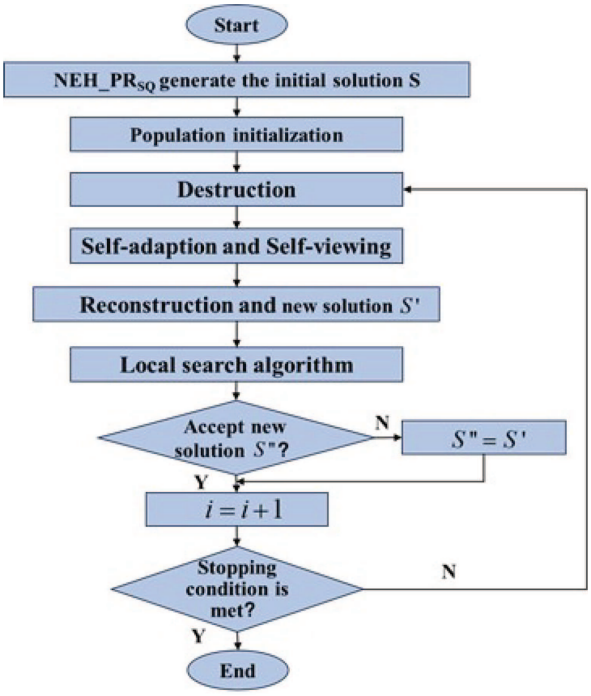


FIGURE 5. Flowchart of IPBIG algorithm.

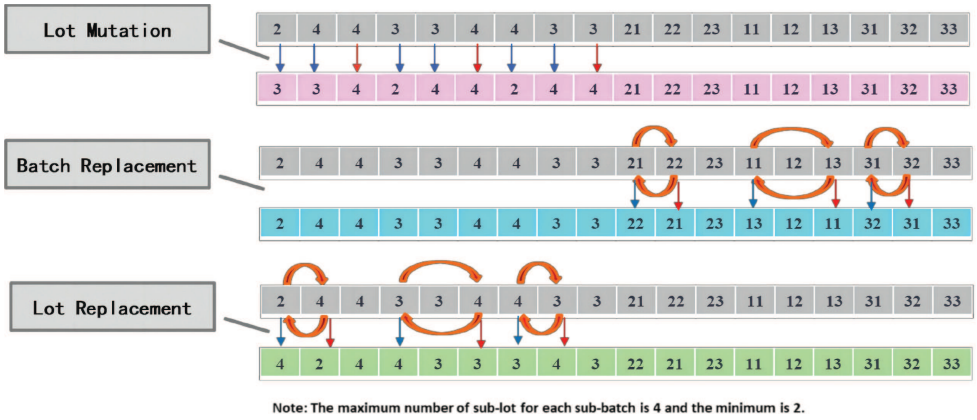


FIGURE 6. Local search algorithm.

Lot Replacement: Lot replacement is the replacement of sub-lots of each sub-batch within a product, *e.g.*, the number of sub-lots of the first sub-batch and the second sub-batch of product J_1 are replaced, and the number of sub-lots of the first sub-batch and the third sub-batch of product J_2 are replaced.

5. COMPARISON AND RESULTS

This section includes two crucial experiments. In the first experiment, the objectives and evaluation indexes of multiple IG-based algorithms are examined. In contrast, the second experiment compares and analyzes the

Algorithm IPBIG

```
def main(file):
    Read test cases from specified file and store them in a 2D array
    Generate initial individual using cursor method based on parameters
    //Perform NEH_PRSQ
    Perform special NEH optimization which requires reordering and creating new work objects
    // Sliding buoy approach
    Generate pop-1 random codings, and then segment sub-batches for the codings with the sliding
buoy approach
    // Destruction and reconstruction
    for generation in range(numGenerations):
        if the destruction factor reaches the increase condition
            Destruction factor will be increased one
        end if
        Select the destroyed coding and record its position in the whole coding
        Sort the destroyed coding at the recorded position in each coding
            if reconstructionCodingFitness originalCodingFitness
                Replace the original coding
            end if
        //Local search
        for each product in works:
            Determine whether to change based on random probability
            Execute a randomly chosen local search method
                if changedCodingFitness originalCodingFitness
                    Replace the original coding
                end if
        The best coding is the best coding from the last generation.
        Class Local search:
            Lot mutation: Reconstruct each sub-lot of batches in one product
            Batch replacement:Randomly swap the order of batches of one product and the
              number of its sub-lots is adjusted accordingly
            Lot replacement: Randomly swap the number of sub-lots in one product
        Auxiliary functions:
            splitRemake(split, subNum):
                Randomly adjust the batch allocation to maintain the total quantity
            randomChangeWithVariation(split):
                Adjust the value between two random positions
            arrayChange(split):
                Randomly swap values in the array at two positions
                arrayChange(split, subNum):
                Randomly swap values in the array at two positions and their corresponding
batch quantities
end
```

evaluation indexes of multiple intelligent optimization algorithms with the IPBIG algorithm. The results prove that the suggested IPBIG algorithm is a considerable and practical improvement over other techniques.

To ensure the fairness of the experiments, the experiments are all conducted on a computer with Windows 11 (64 bit), AMD Ryzen 7 5800U with Radeon Graphics 1.90 GHz, and 16 GB RAM. Each experiment data is executed ten times before the conclusive experimental outcomes are achieved.

5.1. Comparison and results of various IG algorithms

This section of the experiment focuses on the comparison of the objectives for different IG-based algorithms with the proposed IPBIG algorithm in all the instances in the Car database. It demonstrates the advantages of

TABLE 3. Parameter settings for IG-based algorithms.

Algorithm	Pop	Length of fragment destroyed	NEH	Self-adaption and self-viewing	LS algorithm
IG	1	3	/	/	/
IG_1	1	3	√	/	/
IG_2	1	*d*	√	√	/
IG_3	1	*d*	√	√	√
PBIG	100	3	/	/	/
$PBIG_1$	100	3	√	/	/
$PBIG_2$	100	*d*	√	√	/
AIG	100	*d*	√	√	√
IPBIG	100	*d*	√	√	√

TABLE 4. Experimental results of multiple IG-based algorithms.

Instance	Size	IG		IG_1		IG_2		IG_3		PBIG		$PBIG_1$		$PBIG_2$		AIG		IPBIG	
		No_{min}	makespan	No_{min}	makespan	No_{min}	makespan	No_{min}	makespan	No_{min}	makespan	No_{min}	makespan	No_{min}	makespan	No_{min}	makespan	No_{min}	makespan
Car1	11 × 5	687.7549	1 332 411	687.9010	1 332 350	688.0294	1 299 367	688.1799	1 287 324	687.9576	1 293 921	687.9207	1 296 565	688.9226	1 234 557	689.4596	1 213 988	689.3874	1 209 935
Car2	13 × 4	688.0824	1 918 423	688.1029	1 916 254	687.8783	1 937 579	687.8514	1 911 092	688.0535	1 924 732	688.2830	1 915 531	689.1183	1 874 206	689.6988	1 854 579	689.6703	1 854 543
Car3	12 × 5	688.0973	1 758 183	687.9586	1 756 752	687.9970	1 735 717	688.1219	1 854 998	687.9077	1 899 190	687.6438	1 883 445	688.9095	1 806 647	689.3508	1 665 833	689.2446	1 775 729
Car4	14 × 4	687.7087	2 059 882	687.7477	2 049 167	687.8977	2 030 911	688.1495	2 022 750	688.0911	2 029 479	687.9716	2 045 910	688.8525	1 982 200	689.4551	1 962 001	689.4079	1 961 979
Car5	10 × 6	687.7084	1 967 647	687.8288	1 969 627	687.7481	1 937 509	687.7462	1 936 712	687.7972	1 943 853	688.0183	1 991 995	688.6225	1 835 141	689.2033	1 801 395	689.1650	1 793 891
Car6	8 × 9	688.0270	2 183 851	687.9825	2 176 932	688.0658	2 086 485	688.0444	2 100 559	687.8616	2 167 695	688.3922	2 068 174	688.4281	1 917 790	689.0665	1 861 752	689.1977	1 859 360
Car7	7 × 7	687.8544	1 353 129	687.9271	1 345 219	688.1622	1 304 758	688.3270	1 292 140	687.4303	1 357 163	687.6936	1 338 832	688.7142	1 240 445	689.4406	1 223 691	689.3507	1 201 974
Car8	8 × 8	688.0556	1 767 782	688.5189	1 771 927	688.0110	1 836 247	687.4229	1 912 033	688.3260	1 865 251	687.4180	1 888 836	688.9983	1 617 698	689.2782	1 579 748	689.2309	1 577 062

the IPBIG algorithm in terms of solution quality and algorithm stability by comparing two evaluation indexes: RPI and ARPI.

5.1.1. Setting of parameters

Table 3 displays the settings of IG algorithms to conduct the comparative studies more transparently. IG is the basic algorithm, IG_1 is an enhanced version that uses the NEH algorithm for the initial solution; IG_2 builds upon IG_1 with self-adaption and self-viewing strategies; and IG_3 further improves upon IG_2 with a local search algorithm. PBIG is a primary population-based iterative greedy algorithm; $PBIG_1$ is an enhanced version that uses the NEH_PR_{SQ} algorithm for the initial solution; $PBIG_2$ expands on $PBIG_1$ with additional self-adaption and self-viewing strategies; In order to make a fairer and more advanced comparison, this paper also uses the AIG algorithm [71] to participate in the comparison, and IPBIG is the algorithm presented in this paper.

5.1.2. Experiments based on the IG-based algorithm

This section presents the comparison experiments conducted on the different types of IG algorithms mentioned earlier. The results of these experiments are displayed in Table 4. The data in the table reveals that the IPBIG algorithm, which incorporates the concept of population, only shows a marginal improvement in terms of average minimum noise compared to the IG algorithm, which does not incorporate this concept. For example, in the Car1, the value of No_{min}, IG is 0.16% lower than IPBIG. However, when compared to the makespan, which is another objective, it is evident that the IPBIG algorithm outperforms the IG algorithm by being 9.19% lower. Compared to the advanced AIG algorithm, only in the Car3 algorithm there is no advantage of the IPBIG algorithm, in all other instances the IPBIG algorithm has some advantage over the AIG algorithm. After analysing the results, it is evident that although it is not guaranteed that both objectives can be significantly reduced, it can acquire a significant reduction trend in makepsan, which is the most important and commonly studied objective in the field of production scheduling, and it hardly brings about a change in the average minimum noise, which greatly shortens the time cost and thus shortens the time of noise generation, which is also a very good result we want to get in the development of the scheduling.

(a) RPI of the objective -No_{min} (b) RPI of the objective -makespan

FIGURE 7. Boxplots of RPI values for the objectives of IG algorithms.

(a) ARPI of the objective -No_{min} (b) ARPI of the objective -makespan

FIGURE 8. ARPI values of objectives of IG algorithms.

This paper calculates the relative percentage increase (RPI) based on experimental results to estimate the difference between the current and best value for the optimization problem [72,73]. RPI is calculated as follows:

$$\mathrm{RPI} = \frac{f_{\mathrm{mean}} - f_{\mathrm{best}}}{f_{\mathrm{best}}} \times 100\%. \tag{17}$$

RPI is the percentage of relative growth, f_{mean} is the average value of the objective obtained by the algorithm executed independently ten times, and f_{best} is the minimum of the objective by all the comparison algorithms implemented separately ten times. To get Figure 7, we first calculate the RPI for each instance in Car. It can be seen that the IPBIG algorithm has excellent RPI values on both objectives through (a) and (b) in Figure 7. The boundary value gap of the proposed IPBIG algorithm is small, and the objectives are all more concentrated with a low degree of discretization. At the same time, in terms of the median RPI comparison, it is somewhat higher than the AIG algorithm but lower than the other algorithms in the RPI of average minimum noise, indicating that the average condition of the optimization of the objectives of the IPBIG algorithm is better than that of other IG algorithms. The quartile range of RPI is smaller, indicating that the proposed algorithm is more stable. So, the IPBIG algorithm has a more substantial advantage over other IG algorithms regarding solution quality and algorithmic stability.

Afterwards, we compute the average RPI, also known as ARPI, for all the algorithms. According to Figure 8, the IPBIG algorithm exhibits a 61.16% lower ARPI than the worst IG_2 algorithm in terms of average minimum noise. Additionally, the IPBIG algorithm demonstrates an 88.86% lower ARPI than the IG_2 algorithm in terms of makespan. Although the IPBIG algorithm is similar to the AIG algorithm in terms of average minmum noise, the makespan of the IPBIG algorithm is 36.93% lower than the AIG algorithm.

Therefore, based on the above experimental results, the proposed algorithm differs from other IG-based algorithms in different degrees in terms of objectives and evaluation indexes in all the standard instances of the Car database. Overall, the IPBIG algorithm proposed in this paper exhibits the best performance.

TABLE 5. Setting of algorithm parameters.

Algorithm	Pop	Length of destroyed coding	Crossover rate	Mutation rate	pr	$T_{\max}$	h	R	Mutation length	Crossover length
IPBIG	100	d	/	/	/	/	/	/	/	/
GA [53]	100	/	0.9	0.05	/	/	/	/	/	/
SA [74]	100	/	/	/	0.1	/	/	/	/	/
DWPA [75]	100	/	/	/	/	12	40	60	/	/
DBMEA [76]	100	/	0.6	0.8	200	/	/	/	4	4

5.2. Comparison between intelligent optimization algorithms

We conducted ten tests on each instance in the Car, Rec, and Hel standard databases to evaluate the effectiveness of the IPBIG algorithm. With the same batch and instance, we compared the IPBIG with GA [53], SA [74], the newer Discrete Wolf Pack Algorithm (DWPA) [75] and the discrete bacterial memetic evolutionary algorithm (DBMEA) [76]. Because the SA algorithm only works on a single solution, the one used in this paper is based on a population that was generated at random and has simulated annealing operations for each individual in the population. Evaluation indexes are used to compare all the algorithms.

5.2.1. Setting of parameters

This section will set the parameters for all the algorithms. In this paper, the IPBIG algorithm proposes a variable parameter for the length of the destroyed coded segment, named as d. The GA algorithm [53] has two parameters: the crossover rate and the mutation rate. The SA algorithm [74] has a parameter called pr, and the initial temperature is determined using equation (17). The DWPA algorithm [75] has three key parameters: $T_{\max}$ for the number of wanderings, h for the probing direction, and R for the number of updated wolf packs. The DBMEA algorithm [76] has several key parameters: the crossover rate and the mutation rate, mutation length and crossover length and the pr.

$$T_0 = -\frac{F_w - F_b}{\ln pr} \tag{18}$$

where F_w is the worst objective and F_b is the best objective. Table 5 explains how parameters of each algorithm are configured.

5.2.2. Evaluation indexes

Several performance indicators are available for evaluating multi-objective optimization techniques. In the experiments in this section, we select three evaluation indexes to examine the performances of multiple algorithms. The PF indicated in the evaluation indexes refers to the Pareto solutions obtained by NTCM method [77], while PF* refers to the Pareto solutions acquired from all algorithms used.

This paper utilizes performance indexes such as Spread, GD [78], and IGD [79]. The specifics are explained below.

Spread:

$$\text{Spread} = \frac{\sum_{j=1}^{no} d_j^e + \sum_{i=1}^{n} \left| d_i - \bar{d} \right|}{\sum_{j=1}^{no} d_j^e + n \cdot \bar{d}} \tag{19}$$

where d_i denotes the Euclidean distance between the ith member of the obtained PF and its nearest member, $\bar{d}$ denotes the average Euclidean distance over all d_i, d_j^e denotes the Euclidean distance between the extreme solution in the direction of the jth objective and the corresponding extreme solution in the PF*, n is the number of PF, and no is the number of objectives. A smaller spread means a more uniform distribution along the PF.

Generational Distance (GD):

GD indicates convergence performance. It is the average distance from PF to PF*. It is expressed as:

$$\mathrm{GD} = \frac{\sqrt{\sum_{i=1}^{n} D_i^2}}{n} \tag{20}$$

where D_i denotes the Euclidean distance between the ith point of the PF and the nearest point of the PF*, and n denotes the number of PFs.

Inverse Generational Distance (IGD):

It is a variant of GD, represents a composite index. It calculates as the average distance from PF* to PF obtained by the algorithm. IGD can be defined as follows:

$$\mathrm{IGD} = \frac{\sqrt{\sum_{i=1}^{n^*} D_i^{*2}}}{n^*}. \tag{21}$$

Similar to GD, D_i^* is the distance between a point in PF* and the nearest point in PF. But n^* in equation (20) is the number of points in the PF*. Therefore, the lower the value of IGD, the more desirable is the convergence of diversity, homogeneity and solution set.

5.2.3. Analyses and discussions

Table 6 presents the spread, GD, and IGD values for the five compared algorithms across all 31 instances in the Car, Rec, and Hel datasets. In Car1, Car2, Car3, Car5, Rec3, Rec9, the Spread value of the IPBIG algorithm does not perform better. However, in the remaining 25 instances, the IPBIG algorithm outperforms the others when it comes to the value of Spread. Similarly, when analyzing the IGD, the IPBIG algorithm does not achieve the lowest values for Car1-4, Rec1, Rec9, Rec11 and Rec15 compared to the other four algorithms. Nevertheless, IPBIG consistently exhibits lower IGD than the other algorithms in the remaining 23 cases. And in addition to Rec15, the advantage of the GD value of the IPBIG algorithm is obvious.

After analysing, it is found that these few instances can be considered as small-scale instances, and the IPBIG algorithm performs better in terms of Spread and IGD on them, and the GD value performs extraordinarily well, so the following conclusions can be drawn from the experimental data in Table 6: The IPBIG algorithm achieves the best results for the Spread value in more than 80.65% of the experimental instances, the IGD value in more than 74.19% of the experimental instances and the best performance for the GD values in 96.77% of the experimental instances. Based on the excellent performance of the small-scale cases, the proposed IPBIG algorithm on the scheduling scheme for the medium- and large-scale cases has more prominent dominance in solving the better quality solution.

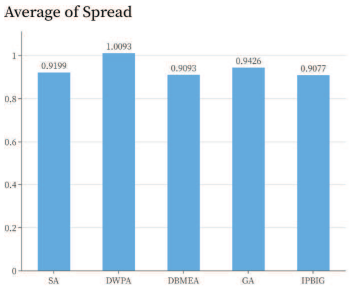
Furthermore, we compute the average of the three evaluation indexes for all algorithms, as illustrated in Figure 9. The average value of three evaluation indexes of the proposed IPBIG algorithm is lower than that of the other three algorithms, demonstrating its advantage in stability and efficiency in solving the MOPFSBSP.

We look at the Pareto solution sets that the IPBIG algorithms make for the Car8, Rec1, Rec25, and He1 datasets to show that they work better than other algorithms. The experiment was mainly to record the Pareto solution set every 5 iterations for all algorithms. It is then compared to the Pareto solution set in PF* that was acquired in the current iteration. The comparison results are displayed in Figure 10. It shows that the Pareto solution sets in PF* and those generated by the IPBIG method largely overlap, suggesting that most of the Pareto solutions in PF* are derived from the IPBIG algorithm. As a result, the proposed IPBIG algorithm can provide superior Pareto answers.

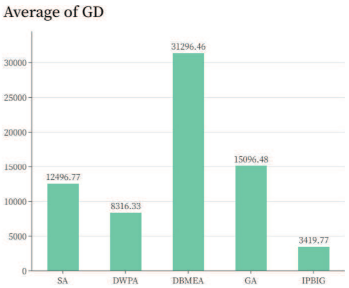
In Figure 10, many of the points that show the Pareto solution set of the IPBIG algorithm overlap with the points that show the PF*. It means that the points that show the Pareto solution set of the IPBIG algorithm are erased, which proves the above conclusion once more. We are providing the analysis results from Table 7. It indicates that the percentage of Pareto solution sets obtained by the IPBIG algorithm that are selected for

TABLE 6. Evaluation indicators of the algorithm.

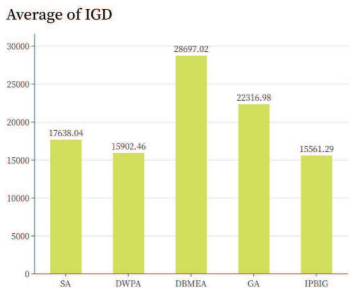
Instance	Size	SA			DWPA			GA			DBMEA			IPBIG		
	(J × S)	Spread	GD	IGD	Spread	GD	IGD	Spread	GD	IGD	Spread	GD	IGD	Spread	GD	IGD
Car1	11 × 5	0.9619	11 601	18 553	1.0363	16 493	21 569	0.9945	17 697	21 812	0.9131	59 416	39 092	0.9604	1358	21 207
Car2	13 × 4	0.9569	23 192	17 817	1.0098	5183	27 370	0.8440	28 783	18 009	0.8746	59 594	45 361	0.9651	4417	26 372
Car3	12 × 5	0.9133	19 369	43 402	1.1489	9675	36 118	0.9804	19 201	43 438	0.9254	59 647	54 831	0.9143	2983	39 265
Car4	14 × 4	0.9330	13 391	31 017	1.1975	19 669	18 638	0.9659	19 041	37 658	0.8826	84 045	49 969	0.8764	9093	25 676
Car5	10 × 6	0.8955	36 260	38 292	0.9430	17 335	40 707	0.9532	29 818	54 089	0.8992	119 455	51 046	0.9343	3495	37 158
Car6	8 × 9	0.8836	24 428	37 656	0.9737	9680	43 484	0.9497	44 545	49 298	0.8841	84 359	65 093	0.8735	6674	34 350
Car7	7 × 7	0.9313	13 207	30 705	1.0613	12 962	28 347	0.9176	10 136	33 248	0.9082	47 118	34 038	0.9069	3786	28 148
Car8	8 × 8	0.9283	9850	21 200	0.9027	9344	25 919	0.9873	27 493	35 338	0.8553	82 530	49 286	0.8480	7899	21 085
Rec1	20 × 5	0.8563	4483	5231	1.1059	1505	5817	0.9184	4865	6517	0.8610	6214	8881	0.8526	305	5843
Rec3	20 × 5	0.8895	3547	4033	0.9537	1018	3897	0.9687	4602	5057	0.8282	9746	7240	0.9673	307	3808
Rec5	20 × 5	0.9130	3550	4503	0.9956	1092	4873	0.9345	4062	5479	0.8861	9255	8983	0.8793	343	4410
Rec7	20 × 10	0.9222	7350	10 003	0.9909	6826	8030	0.9710	9429	14 725	0.8896	12 425	16 485	0.8809	859	7601
Rec9	20 × 10	0.9028	10 377	18 605	0.9225	17 878	26 137	0.9403	20 923	22 854	0.9124	23 537	32 201	0.9361	26 353	27 795
Rec11	20 × 10	0.8986	6223	7322	0.9509	4914	7111	0.9260	6122	10 278	0.8865	8818	10 376	0.8815	3307	7196
Rec13	20 × 10	0.9345	4471	5347	0.9211	1587	4237	0.9002	5459	7508	0.9008	10 288	12 206	0.8896	622	4176
Rec15	20 × 15	0.9074	7925	16 808	0.9240	12 919	14 626	0.9039	6431	8960	0.9389	7986	11 735	0.9027	16 278	17 037
Rec17	20 × 15	0.8820	8738	8966	0.9181	3998	7626	0.9558	12 892	15 292	0.9456	22 480	22 149	0.8816	1837	7587
Rec19	20 × 15	0.9375	5885	6380	1.0768	3465	7037	0.9528	8825	10 327	0.9457	18 198	17 659	0.9353	1385	5931
Rec21	30 × 10	0.9538	13 457	15 224	0.9718	7058	10 898	0.9334	16 709	19 276	0.9325	18 229	20 143	0.9233	831	10 640
Rec23	30 × 10	0.8792	9954	10 908	1.0444	4701	8599	0.8854	11 052	16 021	0.8663	19 403	19 364	0.8642	1259	8616
Rec25	30 × 15	0.9289	12 491	12 801	1.0482	5709	13 568	0.9400	10 124	18 185	0.9275	16 789	26 167	0.9127	1676	11 973
Rec27	30 × 15	0.9307	11 234	11 420	1.0230	5615	10 730	0.9541	14 705	18 842	0.9225	15 555	23 989	0.9146	1553	10 554
Rec29	30 × 15	0.9050	8790	11 664	0.9124	5977	8749	0.9200	10 905	18 064	0.9450	16 714	25 620	0.9013	1267	8496
Rec31	50 × 10	0.9842	13 342	19 933	0.9572	6974	11 580	0.9592	17 573	24 738	0.9518	21 304	27 086	0.9269	810	11 487
Rec33	50 × 10	0.9068	13 435	15 424	0.9352	4025	11 276	0.9625	14 731	20 792	0.9275	22 726	28 674	0.8916	798	11 808
Rec35	50 × 10	0.9401	11 466	13 872	0.9911	5383	9716	0.9657	11 653	18 470	0.9183	18 092	22 360	0.9152	574	9127
Rec37	75 × 20	0.9526	19 531	34 599	0.9843	21 366	25 179	0.9457	17 127	41 669	0.9344	15 054	48 850	0.9182	1747	24 550
Rec39	75 × 20	0.9701	32 906	36 061	0.9813	14 959	27 277	0.9475	41 030	44 928	0.8920	45 233	52 133	0.8896	2142	26 951
Rec41	75 × 20	0.9184	24 494	36 373	0.9619	19 746	22 137	0.9520	19 830	47 460	0.9262	32 736	54 404	0.9153	1904	21 813
Hel1	100 × 10	0.9461	2115	2082	1.0757	398	1280	0.9309	1950	2570	0.9442	2336	3102	0.9414	64	1247
Hel2	20 × 10	0.8533	338	579	1.3705	354	442	0.9588	280	924	0.9634	910	1084	0.9379	85	491



(a) Average of Spread

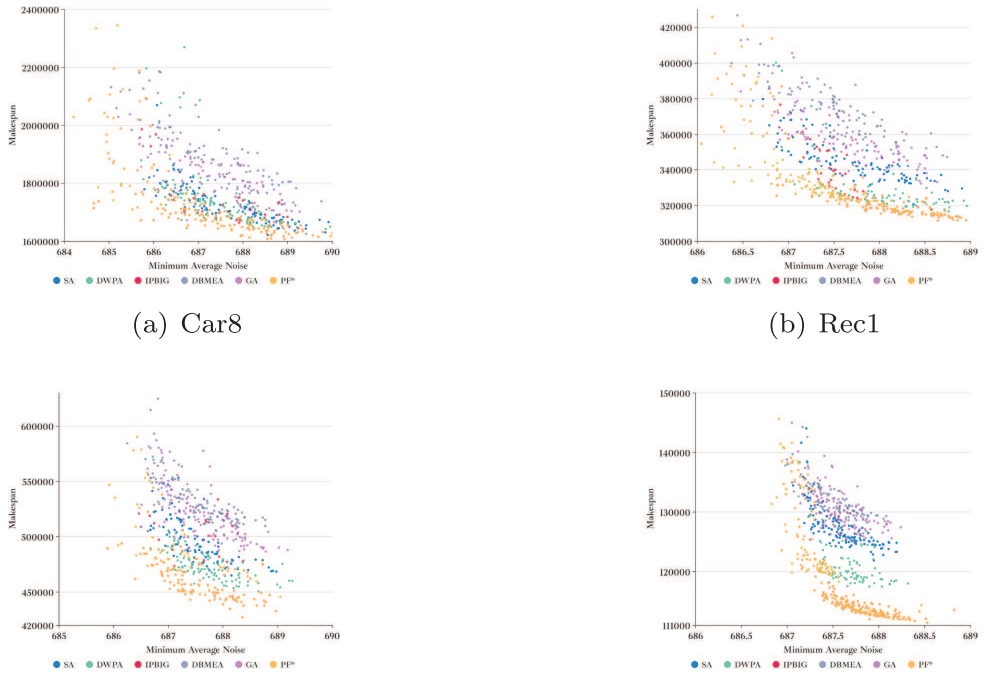


(b) Average of GD



(c) Average of IGD

FIGURE 9. Average values of evaluation indexes.



(a) Car8 (b) Rec1

(c) Rec25 (d) He1

FIGURE 10. Pareto solution sets of the algorithms.

TABLE 7. Comparison of the proportion of Pareto solution sets.

	Proportion of Pareto solution sets obtained by IPBIG that are selected for inclusion in PF*	Proportion of Pareto solution sets obtained by IPBIG in PF*
Car8	79.84%	56.25%
Hel1	100%	64.23%
Rec1	83.73%	53.26%
Rec25	88.19%	67.47%

inclusion in the PF* only does not reach more than 80% in the Car8 instance, but it is more than 75%, while in other algorithms this percentage is even higher than 83%. The IPBIG algorithm in PF* achieves a Pareto solution set proportion of at least 53%, demonstrating its superior performance compared to the other four algorithms in addressing multi-objective problems.

Based on these results, the IPBIG algorithm shows exceptional performance. It delves deeply into exploring solutions compared to traditional intelligent optimization algorithms like GA and SA, as well as the novel swarm intelligent optimization algorithm DWPA. Finding a sequence of products that significantly reduces the makespan, maintains an average minimum noise level that does not increase, and has a high probability of decreasing is easier. The IPBIG algorithm has shown superior performance to other algorithms, although it has yet to achieve ideal results for all issue sizes or evaluation measures. Its excellence is attributed to the self-adaption and self-viewing strategies outlined in Section 4.2, as well as the local search algorithm introduced in Section 4.3. These approaches all enhance population variety and the capability to find high-quality solutions.

6. Conclusion

This paper explores MOPFSBSP with green scheduling and proposes an efficient IPBIG algorithm to solve it. The proposed algorithm demonstrated higher performance and efficiency compared to current classical algorithms, novel algorithms, and different IG algorithms when tested on standard instances in the Standard databases. As a result of the analysis, the outstanding efficiency of the algorithm is mostly due to the following factors:

(1) Different strategies for the NEH algorithm can efficiently predetermine the beginning product sequence and reduce the time needed to acquire optimal solution schemes by preventing random generation.
(2) Self-adaption and self-viewing strategies of IPBIG algorithm. The adaptive strategy prevents the inapplicability of the fixed values of the destroyed coded segments, and thus the unnecessary trouble caused to the overall objectives and the running time of the algorithm. The self-viewing strategy facilitates a more comprehensive exploration of the solution's local neighborhood, diminishes the likelihood of quick convergence, and guarantees ongoing convergence even after other algorithms have stabilized at a fixed value.
(3) The local search algorithm aims to go beyond the local optimal by expanding the search space to potentially discover superior solutions, thereby enhancing local exploitation capabilities.

According to the experimental data and evaluation index presented in this paper, it is clear that the strategies mentioned are effective. As a result, the algorithms presented in this paper can practically solve MOPFSBSP.This paper did not conduct optimization research regarding the algorithm's time complexity, an area the authors have contemplated but failed to achieve satisfactory results in. We aspire that future studies will address this gap and apply the findings to manufacturing companies seeking collaboration.

Data availability statement

No new data/codes were created or analyzed in this study.

References

[1] C. Rusinko, Green manufacturing: an evaluation of environmentally sustainable manufacturing practices and their impact on competitive outcomes. *IEEE. Trans. Eng. Manage.* **54** (2007) 445–454.
[2] A.M. Deif, A system model for green manufacturing. *J. Clean. Prod.* **19** (2011) 1553–1559.
[3] S.-H. Ahn, An evaluation of green manufacturing technologies based on research databases. *Int. J. Precis. Eng. Manuf.-Green Technol.* **1** (2014) 5–9.
[4] D. Marsetiya Utama and M. Dines Primayesti, A novel hybrid aquila optimizer for energy-efficient hybrid flow shop scheduling. *Results Control. Optim.* **9** (2022) 100177.
[5] L. Xue and X. Wang, A multi-objective discrete differential evolution algorithm for energy-efficient two-stage flow shop scheduling under time-of-use electricity tariffs. *Appl. Soft. Comput.* **133** (2023) 109946.
[6] R.-H. Huang, S.-C. Yu and P.-H. Chen, Energy-saving scheduling in a flexible flow shop using a hybrid genetic algorithm. *J. Environ. Prot.* **8** (2017) 1037–1056.
[7] X. Xin, Q. Jiang, C. Li, S. Li and K. Chen, Permutation flow shop energy-efficient scheduling with a position-based learning effect. *Int. J. Prod. Res.* **61** (2023) 382–409.
[8] Y. Zhou, S. Du, M. Liu and X. Shen, Machine-fixture-pallet resources constrained flexible job shop scheduling considering loading and unloading times under pallet automation system. *J. Manuf. Syst.* **73** (2024) 143–158.
[9] D. Applegate and W. Cook, A computational study of the job-shop scheduling problem. *ORSA J. Comput.* **3** (1991) 149–156.
[10] B. Çaliş and S. Bulkan, A research survey: review of ai solution strategies of job shop scheduling problem. *J. Intell. Manuf.* **26** (2015) 961–973.
[11] J. Mark Wilson, Alternative formulations of a flow-shop scheduling problem. *J. Oper. Res. Soc.* **40** (1989) 395–399.
[12] M. Ben-Daya and M.A-Fawzan, A tabu search approach for the flow shop scheduling problem. *Eur. J. Oper. Res.* **109** (1998) 88–95.
[13] C.-F. Liaw, A hybrid genetic algorithm for the open shop scheduling problem. *Eur. J. Oper. Res.* **124** (2000) 28–42.

[14] M. Mahdi Ahmadian, M. Khatami, A. Salehipour and T. Cheng, Four decades of research on the open-shop scheduling problem to minimize the makespan. *Eur. J. Oper. Res.* **295** (2021) 399–426.

[15] O. Shahvari and R. Logendran, Hybrid flow shop batching and scheduling with a bi-criteria objective. *Int. J. Prod. Econ.* **179** (2016) 239–258.

[16] C. Potts and K. Baker, Flow shop scheduling with lot streaming. *Oper. res. lett.* **8** (1989) 297–303.

[17] G. Gong, R. Chiong, Q. Deng, W. Han, L. Zhang, W. Lin and K. Li, Energy-efficient flexible flow shop scheduling with worker flexibility. *Expert. Syst. Appl.* **141** (2020) 112902.

[18] B. Naderi and Rubén Ruiz, A scatter search algorithm for the distributed permutation flowshop scheduling problem. *Eur. J. Oper. Res.* **239** (2014) 323–334.

[19] S. Luo, W. Liang and G. Zhao, Hybrid PSO-WDBA method for the site selection of tailings pond. *Comput. Ind. Eng.* **143** (2020) 106429.

[20] D. Chang, H. Shi, C. Liu and F. Meng, Scheduling optimization of flexible flow shop with buffer capacity limitation based on an improved discrete particle swarm optimization algorithm. *Eng. Optim.* **57** (2025) 571–597.

[21] M. de Fátima Morais, M.H.D.M. Ribeiro, R.G. da Silva, V.C. Mariani and L. dos Santos Coelho, Discrete differential evolution metaheuristics for permutation flow shop scheduling problems. *Comput. Ind. Eng.* **166** (2022) 107956.

[22] X. Zheng, S. Zhou, R. Xu and H. Chen, Energy-efficient scheduling for multi-objective two-stage flow shop using a hybrid ant colony optimisation algorithm. *Int. J. Prod. Res.* **58** (2020) 4103–4120.

[23] A. Kareem Jabari and A. Hasan, Energy-aware scheduling in hybrid flow shop using firefly algorithm. *J. Teknik Ind.* **22** (2021) 18–30.

[24] X. Li, X. Guo, H. Tang, R. Wu and J. Liu, An improved cuckoo search algorithm for the hybrid flow-shop scheduling problem in sand casting enterprises considering batch processing. *Comput. Ind. Eng.* **176** (2023) 108921.

[25] J. Robert Bellabai, B.N. Murugadhas Leela and S.M. Ramesh Kennedy, Testing the performance of bat-algorithm for permutation flow shop scheduling problems with makespan minimization. *Braz. Arch. Biol. Tech.* **65** (2022) 20210840.

[26] F. Zhao, Z. Xu, L. Wang, N. Zhu, T. Xu and J. Jonrinaldi, A population-based iterated greedy algorithm for distributed assembly no-wait flow-shop scheduling problem. *IEEE. Trans. Ind. Inf.* **19** (2022) 6692–6705.

[27] R. Ruiz, Q.-K. Pan and B. Naderi, Iterated greedy methods for the distributed permutation flowshop scheduling problem. *Omega* **83** (2019) 213–222.

[28] S.-W. Lin, K.-C. Ying and C.-Y. Huang, Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm. *Int. J. Prod. Res.* **51** (2013) 5029–5038.

[29] G.-G. Wang and Y. Tan, Improving metaheuristic algorithms with information feedback models. *IEEE. Trans. Cybern.* **49** (2017) 542–555.

[30] M. Nawaz, E. Emory Enscore Jr. and I. Ham, A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega* **11** (1983) 91–95.

[31] Q. Wu, Q. Gao, W. Liu and N. Shuai Cheng, Improved NEH-based heuristic for the blocking flow-shop problem with bicriteria of the makespan and machine utilization. *Eng. Optim.* **55** (2023) 399–415.

[32] R. Ruiz, Q.-K. Pan and B. Naderi, Iterated greedy methods for the distributed permutation flowshop scheduling problem. *Omega* **83** (2019) 213–222.

[33] X.-R. Tao, J.-Q. Li, T.-H. Huang and P. Duan, Discrete imperialist competitive algorithm for the resource-constrained hybrid flowshop problem with energy consumption. *Complex Intell. Syst.* **7** (2021) 311–326.

[34] G. Moslehi and D. Khorasanian, Optimizing blocking flow shop scheduling problem with total completion time criterion. *Comput. Oper. Res.* **40** (2013) 1874–1883.

[35] H.-X. Qin, Y.-Y. Han, B. Zhang, L.-L. Meng, Y.-P. Liu, Q.-K. Pan and D.-W. Gong, An improved iterated greedy algorithm for the energy-efficient blocking hybrid flow shop scheduling problem. *Swarm Evol. Comput.* **69** (2022) 100992.

[36] S. Thomas McCormick, M.L. Pinedo, S. Shenker and B. Wolf, Sequencing in an assembly line with blocking to minimize cycle time. *Oper. Res.* **37** (1989) 925–935.

[37] D.P. Ronconi, A note on constructive heuristics for the flowshop problem with blocking. *Int. J. Prod. Econ.* **87** (2004) 39–48.

[38] S. Deb, Z. Tian, S. Fong, R. Tang, R. Wong and N. Dey, Solving permutation flow-shop scheduling problem by rhinoceros search algorithm. *Soft Comput.* **22** (2018) 6025–6034.

[39] J. Gmys, M. Mezmaz, N. Melab and D. Tuyttens, A computationally efficient branch-and-bound algorithm for the permutation flow-shop scheduling problem. *Eur. J. Oper. Res.* **284** (2020) 814–833.

[40] M. Abdel-Basset, G. Manogaran, D. El-Shahat and S. Mirjalili, A hybrid whale optimization algorithm based on local search strategy for the permutation flow shop scheduling problem. *Future Gener. Comput. Syst.* **85** (2018) 129–145.

[41] M. Abdel-Basset, R. Mohamed, M. Abouhawwash, R.K. Chakrabortty and M.J. Ryan, A simple and effective approach for tackling the permutation flow shop scheduling problem. *Mathematics* **9** (2021) 270.

[42] N.A. Alawad and B.H. Abed-alguni, Discrete jaya with refraction learning and three mutation methods for the permutation flow shop scheduling problem. *J. Supercomput.* **78** (2022) 3517–3538.

[43] Z. Pan, L. Wang, J. Wang and J. Lu, Deep reinforcement learning based optimization algorithm for permutation flow-shop scheduling. *IEEE. Trans. Emerg. Top Comput. Intell.* **7** (2021) 983–994.

[44] Z. Dong, T. Ren, J. Weng, F. Qi and X. Wang, Minimizing the late work of the flow shop scheduling problem with a deep reinforcement learning based approach. *Appl. Sci.* **12** (2022) 2366.

[45] E.-K. Boukas, A. Haurie and F. Soumis, Hierarchical approach to steel production scheduling under a global energy constraint. *Ann. Oper. Res.* **26** (1990) 289–311.

[46] A. Janiak, Single machine scheduling problem with a common deadline and resource dependent release dates. *Eur. J. Oper. Res.* **53** (1991) 317–325.

[47] C. Lu, L. Gao, X. Li, Q. Pan and Q. Wang, Energy-efficient permutation flow shop scheduling problem using a hybrid multi-objective backtracking search algorithm. *J. Clean. Prod.* **144** (2017) 228–238.

[48] X. Xin, Q. Jiang, S. Li, S. Gong and K. Chen, Energy-efficient scheduling for a permutation flow shop with variable transportation time using an improved discrete whale swarm optimization. *J. Clean. Prod.* **293** (2021) 126121.

[49] Y. Fu, G. Tian, A.M. Fathollahi-Fard, A. Ahmadi and C. Zhang, Stochastic multi-objective modelling and optimization of an energy-conscious distributed permutation flow shop scheduling problem with the total tardiness constraint. *J. Clean. Prod.* **226** (2019) 515–525.

[50] M. Feldmann and D. Biskup, Lot streaming in a multiple product permutation flow shop with intermingling. *Int. J. Prod. Res.* **46** (2008) 197–216.

[51] Y. Pan, K. Gao, Z. Li and N. Wu, Improved meta-heuristics for solving distributed lot-streaming permutation flow shop scheduling problems. *IEEE. Tran. Autom. Sci. Eng.* **20** (2022) 361–371.

[52] Q. Feng, Q. Li, W. Quan and X.-M. Pei, Overview of multiobjective particle swarm optimization algorithm. *Chin. J. Eng.* **43** (2021) 745–753.

[53] J. Robert RB and R. Rajkumar, An effective genetic algorithm for flow shop scheduling problems to minimize makespan. *Mechanics* **23** (2017) 594–603.

[54] Z. Woo Geem, J. Hoon Kim and G. Vasudevan Loganathan, A new heuristic optimization algorithm: harmony search. *Simulation* **76** (2001) 60–68.

[55] E. Kaya, B. Gorkemli, B. Akay and D. Karaboga, A review on the studies employing artificial bee colony algorithm to solve combinatorial optimization problems. *Eng. Appl. Artif. Intell.* **115** (2022) 105311.

[56] E.H. Houssein, A.G. Gad and Y.M. Wazery, Jaya algorithm and applications: a comprehensive review. *Metaheuristics Optim. Comput. Electr. Eng.* (2021) 3–24.

[57] W. Wang, Z. Xu and X. Gu, A two-stage discrete water wave optimization algorithm for the flowshop lot-streaming scheduling problem with intermingling and variable lot sizes. *Knowl.-Based Syst.* **238** (2022) 107874.

[58] Q.-K. Pan and R. Ruiz, An estimation of distribution algorithm for lot-streaming flow shop problems with setup times. *Omega* **40** (2012) 166–180.

[59] W. Qin, J. Zhang and D. Song, An improved ant colony algorithm for dynamic hybrid flow shop scheduling with uncertain processing time. *J. Intell. Manuf.* **29** (2018) 891–904.

[60] S.K. Paul and A. Azeem, Minimization of work in process inventory in hybrid flow shop scheduling using fuzzy logic. *Int. J. Ind. Eng. Theory Appl. Pract.* **17** (2010) 115–127.

[61] W. Shao, D. Pi and Z. Shao, A pareto-based estimation of distribution algorithm for solving multiobjective distributed no-wait flow-shop scheduling problem with sequence-dependent setup time. *IEEE. Trans. Autom. Sci. Eng.* **16** (2019) 1344–1360.

[62] B. Farooq, J. Bao and Q. Ma, Flow-shop predictive modeling for multi-automated guided vehicles scheduling in smart spinning cyber–physical production systems. *Electronics* **9** (2020) 799.

[63] C. Lu, L. Gao, Q. Pan, X. Li and J. Zheng, A multi-objective cellular grey wolf optimizer for hybrid flowshop scheduling problem considering noise pollution. *Appl. Soft. Comput.* **75** (2019) 728–749.

[64] L. Yin, X. Li, L. Gao, C. Lu and Z. Zhang, A novel mathematical model and multi-objective method for the low-carbon flexible job shop scheduling problem. *Sustain. Comput. Inf. Syst.* **13** (2017) 15–30.

[65] R. Puka, J. Duda, A. Stawowy and I. Skalna, N-NEH+ algorithm for solving permutation flow shop problems. *Comput. Oper. Res.* **132** (2021) 105296.

[66] X. Dong, H. Huang and P. Chen, An improved NEH-based heuristic for the permutation flowshop problem. *Comput. Oper. Res.* **35** (2008) 3962–3968.

[67] W. Liu, Y. Jin and M. Price, A new Nawaz–Enscore–Ham-based heuristic for permutation flow-shop problems with bicriteria of makespan and machine idle time. *Eng. Optim.* **48** (2016) 1808–1822.

[68] Q. Wu, Q. Gao, W. Liu and S. Cheng, Improved NEH-based heuristic for the blocking flow-shop problem with bicriteria of the makespan and machine utilization. *Eng. Optim.* **55** (2023) 399–415.

[69] R. Ruiz and T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *Eur. J. Oper. Res.* **177** (2007) 2033–2049.

[70] J.-P. Huang, Q.-K. Pan and L. Gao, An effective iterated greedy method for the distributed permutation flowshop scheduling problem with sequence-dependent setup times. *Swarm Evol. Comput.* **59** (2020) 100742.

[71] Y.-Z. Li, Q.-K. Pan, J.-Q. Li, L. Gao and M.F. Tasgetiren, An adaptive iterated greedy algorithm for distributed mixed no-idle permutation flowshop scheduling problems. *Swarm Evol. Comput.* **63** (2021) 100874.

[72] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE. Trans. Evol. Comput.* **6** (2002) 182–197.

[73] E. Zitzler, Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications. Vol. 63. Shaker, Ithaca (1999).

[74] Y. Fu, Y. Hou, Z. Chen, X. Pu, K. Gao and A. Sadollah, Modelling and scheduling integration of distributed production and distribution problems via black widow optimization. *Swarm Evol. Comput.* **68** (2022) 101015.

[75] R. Xie and H. Zhang, Discrete wolf pack algorithm for permutation flow shop scheduling problem. *Control Eng. Chin.* **27** (2020) 288.

[76] A. Agárdi, K. Nehéz, O. Hornyák and L.T. Kóczy, A hybrid discrete bacterial memetic algorithm with simulated annealing for optimization of the flow shop scheduling problem. *Symmetry* **13** (2021) 1131.

[77] F. Wang, Y. Rao, Q. Tang, X. He and L. Zhang, Fast construction method of pareto non-dominated solution for multi-objective decision-making problem. *J. Syst. Eng. Theor. Pract.* **36** (2016) 454–463.

[78] J.S Neufeld, S. Schulz and U. Buscher, A systematic review of multi-objective hybrid flow shop scheduling. *Eur. J. Oper. Res.* **309** (2023) 1–23.

[79] Y. Hu, J. Peng, J. Ou, Y. Li, J. Zheng, J. Zou, S. Jiang, S. Yang and J. Li, The IGD-based prediction strategy for dynamic multi-objective optimization. *Swarm Evol. Comput.* **91** (2024) 101713.