SwarmandEvolutionaryComputation69(2022)100992

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

An improved iterated greedy algorithm for the energy-efficient blocking hybrid flow shop scheduling problem

Hao-Xiang Qin[a], Yu-Yan Han[a,*], Biao Zhang[a], Lei-Lei Meng[a], Yi-Ping Liu[b], Quan-Ke Pan[c], Dun-Wei Gong[d]

[a] School of Computer Science, Liaocheng University, Liaocheng 252059, China
[b] The College of Computer Science and Electronic Engineering, Hunan University, 410082, China
[c] School of Mechatronic Engineering and Automation, Shanghai University, Shanghai 200072, China
[d] School of Information and Control Engineering, China University of Mining and Technology, Xuzhou 221116, China

ARTICLE INFO

Keywords:
Blocking
Hybrid flow shop scheduling problem
Iterated greedy algorithm
Energy efficiency
Local perturbation

ABSTRACT

With the continuous development of national economies, problems of various energy consumption levels and pollution emissions in manufacturing have attracted attention from researchers. Most existing research has focused on reducing economic costs and energy consumption. However, the Hybrid Flow Shop Scheduling Problem with energy-efficient criteria has not yet been well studied, especially with blocking constraints. This paper is the first to present a mathematical model of the blocking hybrid flow shop problem with an energy-efficient criterion and a modified Iterative Greedy algorithm based on a swap strategy designed to optimize the constructed model. In the proposed algorithm, first, a heuristic is adopted to generate the initial solution. Second, a local perturbation strategy based on a swap operator is designed to ensure the convergence of the algorithm. Third, a simple global perturbation strategy based on a half-swap operator is proposed as a means to further search for the potentially best solution with the traditional simulated annealing criterion. The proposed algorithm is applied to 150 test instances at different scales and compared to state-of-the-art algorithms. The experimental results demonstrate that the proposed algorithm outperforms the compared algorithms and can obtain a better solution.

1. Introduction

Under pressures of global warming and continuous competition among enterprises, energy-efficient manufacturing that aims to increase production efficiency and decrease energy wastage is attracting increasing attention [1]. In the manufacturing industry, scheduling is an important problem that directly affects the production efficiency and energy consumption of enterprises. It is necessary to establish a reasonable calculation model and to design an efficient scheduling optimization method that can improve the production efficiency of enterprises and reduce environmental pollution. The Flow Shop Scheduling Problem (FSP) is a commonly experienced optimization problem in many enterprises, and it has been proven to be an NP-hard problem [2].

The Hybrid Flow Shop Scheduling Problem (HFSP) is a more complex optimization problem than the FSP; covers all characteristics of the FSP and is commonly used in steelmaking and refining [3], film transistor-liquid crystal displays [4], semiconductor wafer fabrication facilities [5] and other production processes. For the HFSP, a collection of jobs must pass through all stages of the workshop, and the process of each job is independent. Unlike the FSP, the HFSP overcomes the unique constraints of machines; that is, in any processing stage, each job can be processed on one parallel machine. This setting of parallel machines can increase the productivity and flexibility of the scheduling process [6]. In production settings, due to limitations of storage capacity or technical constraints [7], when a job is finished in a certain stage and machines for the next stage are not available, the job must remain at the current machine until one of the machines of the next stage is available [8,9]. This situation is often called job blocking. The existence of blocking in machines prolongs the wait times of jobs and causes unnecessary energy wastage; thus, the blocking constraint reduces the processing efficiency of the job sequence. It is evident that for job sequences of different scales, the difference in job sequencing will cause different degrees of blocking. This arrangement can improve the production efficiency of the manufacturing industry and reduce energy wastage by applying an optimal job sequencing sequence as effectively as possible, mitigating the blocking problem caused by the absence of a buffer.

At present, many metaheuristic algorithms have been developed for solving FSP with blocking constraints, such as the Hybrid Multiobjective Artificial Bee Colony [10], the Evolutionary Multiobjective Robust Scheduling algorithm [11], the Iterated Greedy (IG) algorithm [12,13],

* Corresponding author.
E-mail address: hanyuyan@lcu-cs.com (Y.-Y. Han).

https://doi.org/10.1016/j.swevo.2021.100992
Received 18 February 2021; Received in revised form 29 July 2021; Accepted 3 October 2021
Available online 7 October 2021

the Hybrid Enhanced Discrete Fruit Fly Optimization algorithm [14], Discrete Invasive Weed Optimization [15] and the Discrete Gravitational Search Algorithm [16]. All of the above algorithms are used to solve the FSP with blocking constraints. However, these algorithms do not design corresponding methods for the HFSP with blocking constraints. In view of this, this paper studies means to reduce the energy consumption of the Blocking Hybrid Flow Shop Scheduling Problem (BHFSP).

The IG algorithm has been proven to be an effective method for solving the FSP [17]. The IG algorithm with blocking constraints also shows good performance relative to many algorithms [18,19]. Some typical advantages of the IG algorithm are that (1) the algorithm has a simple structure with few parameters and can integrate constructive heuristic and metaheuristic algorithms into its framework. (2) Unlike existing swarm intelligence algorithms, the IG algorithm generates only one solution in each iteration so it can focus on a deeper exploration of a solution. (3) The IG algorithm can arrange a job in an appropriate manner as much as possible through the local perturbation of the job sequence, effectively reducing energy wasted due to the blocking of the job sequence. Based on the advantages of the IG algorithm and job blocking problems, we propose an improved Iterative Greedy algorithm based on swap (IGS) to solve the above BHFSP. In the IGS, two perturbation strategies are designed to improve the local and global search abilities of this solution to reduce the impact of blocking constraints on the job sequence.

Our algorithm is the preferred choice when solving the HFSP with blocking constraints, since our algorithm is particularly designed to solve such a problem. This would be also a limitation of our algorithm. That is, our algorithm may be not the best choice to solve the HFSP without blocking constraints. According to the above description, the contributions of this paper are as follows:

(1) In real-world production, the blocking of jobs and machine idleness always lead to increased energy consumption. However, to the best of our knowledge, the BHFSP with energy consumption has not yet been well studied. Therefore, this paper makes a strategy design for blocking constraints and uses it to reduce the invalid processing of the machine. Then, this paper constructs the mathematical model of the BHFSP with energy consumption criteria to satisfy scheduling demands.
(2) The strategies proposed in this paper are particularly designed to handle job blocking constraints. To exploit promising subregions and explore irregular unknown regions, in the earlier stages of iteration, the MinMax and Nawaz, Enscore and Ham (MME) heuristic algorithm is used to reduce the impact of blocking on the job sequence. In the later stages, two perturbation strategies are designed to balance the local and global search ability of the proposed algorithm. By disturbing the blocking jobs, the energy consumption is effectively reduced.
(3) To handle the blocking problems of job sequences and improve the iterative performance of the algorithm for large-scale operations, local perturbation strategies based on swap are proposed as a means to improve the local search ability of the solution. Because blocked jobs may vary with changes in job sequences, local perturbation strategies based on swap effectively change the positions of blocked jobs. The strategy proposed in this paper can be executed more times due to its lower complexity. It can quickly disturb the job sequence, simplifying the iterative process and reducing energy wastage.
(4) To further disturb the blocked jobs and change the jobs order, a global perturbation strategy based on a half-swap is proposed in consideration of the conditions of the local optimum and the diversity of the solution. The proposed strategy can effectively adjust the arrangement of the current blocked job sequence, and improve the global search ability of the algorithm for the individuals of each generation.

The remainder of this paper is organized as follows. After a brief introduction provided in Sections 1, 2 presents a literature review. Section 3 describes the scheduling problem of the energy-efficient BHFSP. In Section 4, an IGS is developed to obtain the above mathematical model. Section 5 analyzes the experimental results. Finally, a summary of this paper and avenues for future work are presented in Section 6.

2. Literature review

2.1. HFSPs

Several exact algorithms have been proposed as tools used to solve the HFSP, including the branch and bound algorithm [24,25] and Lagrangian relaxation algorithm [26,27]. However, for a large-scale HFSP, it is difficult to solve such problems with exact algorithms. Heuristic, metaheuristic and hybrid heuristic algorithms have become scholars' favored means to solve such problems [28]. The following section provides a review of these algorithms.

Intelligent optimization algorithms are generally divided into heuristic and metaheuristic algorithms to solve the FSP [21]. The Nawaz–Enscore–Ham (NEH) constructive heuristic was first designed to solve the FSP [22]. Later, McCormick et al. [23] proposed profile fitting to minimize blocking and idle times of the job sequence. Ronconi [20] developed the MME constructive heuristic for the FSP with blocking constraints. The above heuristic methods show superior performance in initializing a solution. Thus, the methods are embedded into the initialization stage.

More recent studies on metaheuristics have also been conducted. Nejati et al. used the genetic algorithm (GA) to minimize the weighted completion time of the HFSP with a work shift constraint; the advantage lies in the algorithm's quick response to demands based on substantial streaming and machine utilization to adjust the sequencing problem. [29]. Pan et al. proposed a Discrete Artificial Bee Colony (DABC) to minimize the makespan of the HFSP [30] where a new control parameter is introduced into the algorithm to balance the abilities of local exploitation and global exploration. Zhang et al. [31] presented an Effective Modified Migrating Bird Optimization (EMBO) algorithm to solve the HFSP. In EMBO, two competitive mechanisms are used to increase the probability of locating better solutions and to enhance the interactions between two lines. Li et al. [32] proposed the Hybrid Variable Neighborhood Search (VNS) method, which combines Chemical-Reaction Optimization and the Estimation of Distribution algorithm in solving the HFSP. Later, Liu et al. [33] presented a hybrid EDA-DE algorithm that combines the Estimation of Distribution algorithm with the Differential Evolution algorithm to solve the HFSP.

In addition, for the optimization of energy consumption, Tao et al. [28] were the first to consider energy consumption and resource constraints and proposed the Discrete Imperialist Competitive algorithm (ICA) for solving the HFSP with a makespan objective. Marichelvam et al. proposed a Discrete Particle Swarm Optimization (DPSO) algorithm that is hybridized with the VNS method to reduce energy consumption for a flexible FSP [34]. Similarly, the Nondominated Sorti3ng Genetic Algorithm-II is adopted to solve the flexible job shop problem with energy consumption objectives [35], which can effectively select a suitable machine for each operation and undertake rational operation sequencing simultaneously without the interference of subjective factors. Later, Lei et al. proposed a two-phase metaheuristic based on the ICA and VNS to solve the flexible job shop scheduling problem with an energy consumption threshold [36].

2.2. IG algorithm

The IG algorithm is a simple and effective optimization algorithm. Due to its simple structure and embeddability, the algorithm has attracted much attention from researchers in manufacturing scheduling

Table 1
Advantages and limitations of the algorithms.

	Advantages	Limitations
GA [29]	The structure is simple, and in the same amount of termination time, the algorithm can iterate many times more than other algorithms, and the diversity of the solutions is good.	Due to using too many iterations, the local neighborhood of a single solution is not sufficiently explored.
DABC [30]	The three-layer structure is simple and clear. In the iterative process, the new solution is used to replace the worst solution of the population to improve the overall quality of the population.	Similarly, the exploration ability of the single solution is not sufficient.
EMBO [31]	The local and global search abilities of the algorithm are widely considered, and the algorithm combines the advantages of the DABC and GA algorithms.	The algorithm has a more complex structure and more parameters, necessitating extensive parameter adjustment efforts.
DPSO [34]	In this paper, the initialization strategy and subsequent iteration operation are shown to better balance the global and local search abilities of the algorithm.	With more parameters, the implementation process is more complicated than that of other swarm algorithms.
IGA [17]	The algorithm's structure is simple, the operation of only one solution can achieve a greater degree of exploration, and local search ability is strong.	The operation of a single solution is not conducive to maintaining solution diversity, global search ability is weak, the algorithm easy to falls into the local optimal.
IGRS [46]	The algorithm has fewer parameters and a simple structure. Compared to the traditional IG algorithm, this algorithm achieves better local solution exploration.	Local searches are not highly effective at improving the global search ability of the solution, and the algorithm easily falls into the local optimal.
IGT [46]	Compared to other IG algorithms, global search ability is improved, and fewer parameters are applied.	RIS and RSS policies are complex, and their execution takes a considerable amount of time.
IGTALL [46]	The IGT algorithm is further optimized after the destruction strategy, and the local search ability of the algorithm is further improved.	Local search ability is further strengthened, which reduces the exploration of global scope and causes the algorithm to easily fall into the local optimum.
VBIH [46]	Performing operations on blocks can reduce damage to the current good sequence and improve local search ability.	The algorithm takes more time to execute an iterative process and the irregular unknown regions of the solution are not sufficiently explored.

fields. After Ruben and Thomas first used the algorithm to solve the FSP [17], a series of improved IG algorithms were applied to solve the FSP. For the makespan criterion of the no-wait FSP, Ding et al. [37] proposed a tabu-based reconstruction strategy to enhance the exploration abilities of the IG algorithm. Next, Huang et al. modified the IG algorithm by using six different operators to solve the distributed permutation FSP [38]. Fernandez-Viagas et al. adopted IG-based algorithms with beam search initialization to minimize the overall tardiness of the FSP [39]. Ochi and Driss proposed the Bounded-Search IG algorithm for solving the Distributed Assembly Permutation Flow Shop Scheduling Problem (DAPFSP) [40]. Similarly, Huang et al. proposed an improved IG algorithm based on group think as a means to solve the DAPFSP [41].

To the best of our knowledge, there have been few studies on the use of the IG algorithm for solving the HFSP. We note the following related studies. Ying [42] proposed an IG algorithm for solving the HFSP with multiprocessor tasks. Rodriguez et al. used the IG algorithm to solve the large-scale unrelated parallel machine scheduling problem [43]. Shao et al. [44] used the IG algorithm to solve the distributed HFSP with a makespan objective. Fbo and Ms [45] proposed an IG search metaheuristic that minimizes the makespan of the HFSP encountered in a manufacturing plant with sequence-dependent setup times. Ztop et al. [46] designed a series of strategies and four IG variants, i.e., IGRS, IGT, IGTALL, and VBIH algorithms, to solve the HFSP. These variants have shown good performance in the literature. From experimental results, the IG algorithm shows good performance in various studies in solving HFSP.

2.3. The motivation of the proposed IGS

According to the above algorithms, we select nine representative algorithms related to our considered problem and state their advantages and limitations in Table 1. Through a comparative analysis of algorithms, we find that swarm intelligence algorithms, i.e., GA, DABC, EMBO, and DPSO, include more parameters and complicated structures. At the same time, the algorithms can provide multiple solutions that are helpful in improving the diversity of solutions. However, in the exploration of a single solution neighborhood, these swarm intelligence algorithms are slightly less effective than the IG algorithm. In actual production, we often only need an optimal scheduling scheme, which requires this scheduling scheme to minimize the optimal objective value as much as possible. Compared to traditional swarm intelligence algorithms, the improved IG algorithm can more intensively explore a single solution and is very effective at improving the reinforcement of the solution. According to the description of the advantages of the IG algorithm provided in the introduction and the current status of research problems, this paper identifies further improvements on the basis of the IG algorithm.

The studies conducted adopt a series of IG algorithms to solve the HFSP, demonstrating good results in local neighborhood exploration. However, it is difficult to have the solution jump out of the search range and reduce the diversity of the solution using local neighborhood exploration alone. The iterative improvement strategy of the traditional IG algorithm shows excellent performance in solving small-scale problems. However, with the continuous expansion of the job sequence, the advantages of the iterative improvement strategy gradually decline. Because this strategy is based on the insertion operation, the operation is more complex than the swap strategy. Therefore, if the termination time is the same, the number of insert operations will be lower than the number of swap operations, which will reduce the number of algorithm iterations and make it difficult to change the relative position of the job sequence. As a result, the improvement in job blocking is not obvious, and the solution easily falls into a local optima.

To avoid the above result, this paper proposes a local perturbation strategy based on swap to replace the iterative improvement strategy of the traditional IG algorithm. The proposed strategy can better solve large-scale problems. In addition, the traditional IG algorithm only uses the criterion of simulated annealing to improve the diversity of the solution, and this criterion has not been found to greatly improve the diversity of this solution after simulation testing [38]. Therefore, this paper designs a new global perturbation strategy for the original simulated annealing criterion to explore the solution across a broader range. In addition, the strategy can more effectively perturb the job sequence, improve the global search ability of the solution, and further reduce the impact of blocking constraints on the energy consumption of jobs, motivating our design of new strategies based on the IG algorithm.

3. Energy-efficient BHFSP

The related notation of BHFSP is given as follows:

J: The total number of jobs, indexed by $j = 1, 2, \ldots, J$.
S: The total number of stages, indexed by $s = 1, 2, ..., S$.
M_s: The number of identical parallel machines in stage s; $M_s = \{1, ..., m, ..., M_s\}$.
$M_{s,m}^{Idle}$: The available time of machine m in stage s.
$p_{s,j}$: The processing time of job j in stage s.
$B_{s,j}$: The start time of job j in stage s.
$E_{s,j}$: The end time of job j in stage s.
$Block_{s,j}$: The blocking time of job j in stage s.
m^{before}: The number of machines with job j processing in the preceding stage $s-1, s = 2, ...S$.
$m^{current}$: The number of machines with job j processing in the current stage s.
$P_{s,m}^{Process}$: The power of machine m in stage s processing a job per unit of time.
$P_{s,m}^{Idel}$: The power of machine m in stage s remaining in the idle state per unit of time.
$P_{s,m}^{Block}$: The power of machine m in stage s remaining in the block state per unit of time.

$$X_{j,m}=\begin{cases} 1 & The\ job\ is\ arranged\ to\ be\ processed\ on\ machine\ m \\ 0 & Others \end{cases}$$

The BHFSP contains a set of S stages in which one or more identical parallel machines are arranged for each stage. At least one stage uses more than one machine. A sequence of n jobs must traverse through all stages consecutively. Once a job is completed in the current stage, it must be transferred to the next stage for processing. In contrast to the HFSP, no intermediate buffers exist for any adjacent machines for the BHFSP considered in this paper. Thus, it is necessary to consider the idle and blocking time of each machine over two key steps: machine assignment for each job and the allocation of collections of jobs to the selected machines. In addition to the above constraints, the BHFSP is subject to the following eight constraints:

(1) All machines and jobs are available at time zero.
(2) Both the idle time and blocking time of machines are considered.
(3) Each job passes through all stages, and at a specific stage, a job is performed by exactly one machine at a time.
(4) At any given time, each machine can process at most one job, and each job can be processed on at most one machine.
(5) All jobs should be continuously processed and not preempted or interrupted.
(6) No intermediate buffers exist for adjacent machines.
(7) A completed job must be immediately transported to the next stage. If there is no available machine at the next stage, the job will be blocked on the current machine until the next downstream machine is available for processing.
(8) Both the transportation time and setup time are included in the processing time.

To more clearly describe the difference between the HFSP and BHFSP, a simple example is given. Suppose that the processing times of each job in 3 stages are {3,10,1}, {5,7,6}, {5,3,3}, {5,4,7}, {8,3,7}, and {4,7,3}. Each stage involves two identical parallel machines.

As shown in Fig. 1., when there are sufficient buffers between adjacent machines, the completion time of the HFSP equals 28. However, for the no-buffer case, the completion time of the BHFSP is increased to 30. This is the case because some jobs are blocked on the current machine until the next downstream machine is available for processing; i.e., because machines 3 and 4 are not available 8 times, job 3 is blocked on machine 1 (denoted by the shaded rectangle).

The blocking constraint increases the manufacturing time. Thus, we are encouraged to research the BHFSP and reduce the blocking time. First, we construct a mathematical model of the BHFSP, and then we provide an example of the calculation process of the makespan to more clearly describe the process.

Let a job permutation $\pi = \{1, 2, 3, 4, 5, 6\}$ represent the sequence of jobs to be processed, and let the processing time of each job be as follows:

$$[p_{s,j}]_{3\times 6} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & p_{1,4} & p_{1,5} & p_{1,6} \\ p_{2,1} & p_{2,2} & p_{2,3} & p_{2,4} & p_{2,5} & p_{2,6} \\ p_{3,1} & p_{3,2} & p_{3,3} & p_{3,4} & p_{3,5} & p_{3,6} \end{bmatrix} = \begin{bmatrix} 3 & 5 & 5 & 5 & 8 & 4 \\ 10 & 7 & 3 & 4 & 3 & 7 \\ 1 & 6 & 3 & 7 & 7 & 3 \end{bmatrix}$$

According to the mathematical model given above, the calculation of machine availability time and the completion time of every job is as follows:

1) $M_{1,1}^{Idle} = \inf\{M_{1,1}^{Idle}, M_{1,2}^{Idle}\} = \inf\{0, 0\} = 0$; $B_{1,1} = M_{1,1}^{Idle} = 0$; $E_{1,1} = B_{1,1} + p_{1,1} = 0 + 3 = 3$;

$M_{1,1}^{Idle} = E_{1,1} = 3$; $m^{before} = 1$;

1) $M_{2,1}^{Idle} = \inf\{M_{2,1}^{Idle}, M_{2,2}^{Idle}\} = \inf\{0, 0\} = 0$; $B_{2,1} = \sup\{M_{2,1}^{Idle}, E_{1,1}\} = \sup\{0, 3\} = 3$; $E_{1,1}=B_{2,1} = 3$; $M_{1,1}^{Idle} = E_{1,1} = 3$; $E_{2,1}=B_{2,1} + p_{2,1} = 3 + 10 = 13$; $M_{2,1}^{Idle} = E_{2,1} = 13$; $m^{before} = 1$;
2) $M_{3,1}^{Idle} = \inf\{M_{3,1}^{Idle}, M_{3,2}^{Idle}\} = \inf\{0, 0\} = 0$; $B_{3,1} = \sup\{M_{3,1}^{Idle}, E_{2,1}\} = \sup\{0, 13\} = 13$;

$E_{2,1}=B_{3,1} = 13$; $M_{2,1}^{Idle} = E_{2,1} = 13$; $E_{3,1}=B_{3,1} + p_{3,1} = 13 + 1 = 14$; $M_{3,1}^{Idle} = E_{3,1} = 14$; $m^{before} = 1$;

1) $M_{1,2}^{Idle} = \inf\{M_{1,1}^{Idle}, M_{1,2}^{Idle}\} = \inf\{3, 0\} = 0$; $B_{1,2} = M_{1,2}^{Idle} = 0$; $E_{1,2} = B_{1,2} + p_{1,2} = 0 + 5 = 5$;

$M_{1,2}^{Idle} = E_{1,2} = 5$; $m^{before} = 2$;

1) $M_{2,2}^{Idle} = \inf\{M_{2,1}^{Idle}, M_{2,2}^{Idle}\} = \inf\{13, 0\} = 0$; $B_{2,2} = \sup\{M_{2,2}^{Idle}, E_{1,2}\} = \sup\{0, 5\} = 5$;

$E_{1,2} = B_{2,2} = 5$; $M_{1,2}^{Idle} = E_{1,2} = 5$; $E_{2,2} = B_{2,2} + p_{2,2} = 5 + 7 = 12$; $M_{2,2}^{Idle} = E_{2,2} = 12$; $m^{before} = 2$;

1) $M_{3,2}^{Idle} = \inf\{M_{3,1}^{Idle}, M_{3,2}^{Idle}\} = \inf\{14, 0\} = 0$; $B_{3,2} = \sup\{M_{3,2}^{Idle}, E_{2,2}\} = \sup\{0, 12\} = 12$;

$E_{2,2} = B_{3,2} = 12$; $M_{2,2}^{Idle} = E_{2,2} = 12$; $E_{3,2} = B_{3,2} + p_{3,2} = 12 + 6 = 18$; $M_{3,2}^{Idle} = E_{3,2} = 18$; $m^{before} = 2$;

1) $M_{1,1}^{Idle} = \inf\{M_{1,1}^{Idle}, M_{1,2}^{Idle}\} = \inf\{3, 5\} = 3$; $B_{1,3} = M_{1,1}^{Idle} = 3$; $E_{1,3} = B_{1,3} + p_{1,3} = 3 + 5 = 8$;

$M_{1,1}^{Idle} = E_{1,3} = 8$; $m^{before} = 1$;

1) $M_{2,2}^{Idle} = \inf\{M_{2,1}^{Idle}, M_{2,2}^{Idle}\} = \inf\{13, 12\} = 12$; $B_{2,3} = \sup\{M_{2,2}^{Idle}, E_{1,3}\} = \sup\{12, 8\} = 12$;

$E_{1,3} = B_{2,3} = 12$; $M_{1,1}^{Idle} = E_{1,3} = 12$; $E_{2,3} = B_{2,3} + p_{2,3} = 12 + 3 = 15$; $M_{2,2}^{Idle} = E_{2,3} = 15$; $m^{before} = 2$;

1) $M_{3,1}^{Idle} = \inf\{M_{3,1}^{Idle}, M_{3,2}^{Idle}\} = \inf\{14, 18\} = 14$; $B_{3,3} = \sup\{M_{3,1}^{Idle}, E_{2,3}\} = \sup\{14, 15\} = 15$;

$E_{2,3} = B_{3,3} = 15$; $M_{2,2}^{Idle} = E_{2,3} = 15$; $E_{3,3} = B_{3,3} + p_{3,3} = 15 + 3 = 18$; $M_{3,1}^{Idle} = E_{3,3} = 18$; $m^{before} = 1$;

1) $M_{1,2}^{Idle} = \inf\{M_{1,1}^{Idle}, M_{1,2}^{Idle}\} = \inf\{8, 5\} = 5$; $B_{1,4} = M_{1,2}^{Idle} = 5$; $E_{1,4} = B_{1,4} + p_{1,4} = 5 + 5 = 10$;

$M_{1,2}^{Idle} = E_{1,4} = 10$; $m^{before} = 2$;

1) $M_{2,2}^{Idle} = \inf\{M_{2,1}^{Idle}, M_{2,2}^{Idle}\} = \inf\{13, 15\} = 13$; $B_{2,4} = \sup\{M_{2,2}^{Idle}, E_{1,4}\} = \sup\{13, 10\} = 13$;

$E_{1,4} = B_{2,4} = 13; M_{1,2}^{Idle} = E_{1,4} = 13; E_{2,4} = B_{2,4} + p_{2,4} = 13 + 4 = 17; M_{2,1}^{Idle} = E_{2,4} = 17; m^{before} = 1;$

1) $M_{3,1}^{Idle} = \inf\{M_{3,1}^{Idle}, M_{3,2}^{Idle}\} = \inf\{18, 18\} = 18;$ $B_{3,4} = \sup\{M_{3,1}^{Idle}, E_{2,4}\} = \sup\{18, 17\} = 18;$

$E_{2,4} = B_{3,4} = 18; M_{2,1}^{Idle} = E_{2,4} = 18; E_{3,4} = B_{3,4} + p_{3,4} = 18 + 7 = 25; M_{3,1}^{Idle} = E_{3,4} = 25; m^{before} = 1;$

1) $M_{1,1}^{Idle} = \inf\{M_{1,1}^{Idle}, M_{1,2}^{Idle}\} = \inf\{12, 13\} = 12;$ $B_{1,5} = M_{1,1}^{Idle} = 12;$ $E_{1,5} = B_{1,5} + p_{1,5} = 12 + 8 = 20;$

$M_{1,1}^{Idle} = E_{1,5} = 20;\ m^{before} = 1;$

1) $M_{2,2}^{Idle} = \inf\{M_{2,1}^{Idle}, M_{2,2}^{Idle}\} = \inf\{18, 15\} = 15;$ $B_{2,5} = \sup\{M_{2,2}^{Idle}, E_{1,5}\} = \sup\{15, 20\} = 20;$

$E_{1,5} = B_{2,5} = 20; M_{1,1}^{Idle} = E_{1,5} = 20; E_{2,5} = B_{2,5} + p_{2,5} = 20 + 3 = 23; M_{2,2}^{Idle} = E_{2,5} = 23; m^{before} = 2;$

1) $M_{3,2}^{Idle} = \inf\{M_{3,1}^{Idle}, M_{3,2}^{Idle}\} = \inf\{25, 18\} = 18;$ $B_{3,5} = \sup\{M_{3,2}^{Idle}, E_{2,5}\} = \sup\{18, 23\} = 23;$

$E_{2,5} = B_{3,5} = 23; M_{2,2}^{Idle} = E_{2,5} = 23; E_{3,5} = B_{3,5} + p_{3,5} = 23 + 7 = 30; M_{3,2}^{Idle} = E_{3,5} = 30; m^{before} = 2;$

1) $M_{1,2}^{Idle} = \inf\{M_{1,1}^{Idle}, M_{1,2}^{Idle}\} = \inf\{20, 13\} = 13;$ $B_{1,6} = M_{1,2}^{Idle} = 13;$ $E_{1,6} = B_{1,6} + p_{1,6} = 13 + 4 = 17;$

$M_{1,2}^{Idle} = E_{1,6} = 17;\ m^{before} = 2;$

1) $M_{2,2}^{Idle} = \inf\{M_{2,1}^{Idle}, M_{2,2}^{Idle}\} = \inf\{18, 23\} = 18;$ $B_{2,6} = \sup\{M_{2,2}^{Idle}, E_{1,6}\} = \sup\{18, 17\} = 18;$

$E_{1,6} = B_{2,6} = 18; M_{1,2}^{Idle} = E_{1,6} = 18; E_{2,6} = B_{2,6} + p_{2,6} = 18 + 7 = 25; M_{2,1}^{Idle} = E_{2,6} = 25; m^{before} = 1;$

1) $M_{3,1}^{Idle} = \inf\{M_{3,1}^{Idle}, M_{3,2}^{Idle}\} = \inf\{25, 30\} = 25;$ $B_{3,6} = \sup\{M_{3,1}^{Idle}, E_{2,6}\} = \sup\{25, 25\} = 25;$

$E_{2,6} = B_{3,6} = 25; M_{2,1}^{Idle} = E_{2,6} = 25; E_{3,6} = B_{3,6} + p_{3,6} = 25 + 3 = 28; M_{3,1}^{Idle} = E_{3,6} = 28; m^{before} = 1;$

The makespan of this example is $C_{\max} = E_{3,5} = 30$.

In addition to the above makespan objective, energy efficiency is a key objective in view of practical production. We know that energy consumption exists at any stage, i.e., the processing stage, blocking stage, machine idle stage and so on. In addition, different scheduling sequences may result in different idle and blocking time lengths, leading to increased energy consumption. Thus, for the BHFSP considered in this paper, we not only consider the energy consumption of the processing time of these jobs but also take into account the energy consumption of the idle and blocking times of all machines. The energy consumption objective is given as follows:

***Objective*:**

$$\boldsymbol{Argmin}\, TEC = EC_P + EC_B + EC_I \tag{12}$$

$$EC_P = \sum_{s\in\{1,\ldots,S\}} \sum_{m\in M_s} \sum_{j\in\{1,2,\ldots,J\}} X_{j,m} \times P_{s,m}^{Process} \times p_{s,j} \tag{13}$$

$$EC_B = \sum_{s\in\{2,\ldots,S\}} \sum_{m^{before}\in M_{s-1}} \sum_{m\in M_s} \sum_{j\in\{1,2,\ldots,J\}} X_{j,m^{before}} \times (M_{s,m}^{Idle} - E_{s-1,j}) \times P_{s-1,m^{before}}^{Block} \tag{14}$$

$$EC_I = \sum_{s\in\{2,\ldots,S\}} \sum_{m\in M_s} \sum_{j\in\{1,2,\ldots,J\}} (E_{s-1,j} - M_{s,m}^{Idle}) \times P_{s,m}^{Idle} \tag{15}$$

where TEC is total energy consumption, EC_P represents the energy consumption when the machines process all jobs, EC_B represents the energy consumption of machines that remain in the blocking state, and EC_I represents energy consumption when the machines remain in the idle state.

4. Proposed IGS algorithm for the BHFSP

4.1. Basic IG

The traditional IG algorithm includes five main parts: initialization, destruction and construction, iterative improvement, the acceptance criterion, and the termination condition. First, an initial solution is generated by utilizing the NEH heuristic, and then iterative improvement is employed to improve the performance of the obtained solution through continuous iterations and insertions (see lines 4-5 of Algorithm 1). In the destruction phase, the job sequence, $\pi=(\pi_1, \pi_2, ..., \pi_J)$, is partially destroyed, and the number of removed jobs is controlled by parameter d. The setting of d is very important. If it is too small, this will result in slow convergence and a small search range. If the value is too large, this will result in too much time spent and reduce the performance of the IG. Therefore, we test the setting of d in Section 4.3. When the destruction phase is finished, we obtain partial job sequences π^{Remove} and π^{origin} where π^{Remove} is the set of d removed jobs and π^{origin} consists of the remaining jobs (see line 8 of Algorithm 1). In the construction phase, every job in π^{Remove} is inserted into each possible position of π^{origin}, and the best position is selected with the minimal objective value (see line 9 of Algorithm 1). Then, the same local search as that applied before is used again to improve the convergence of the obtained solution. In the acceptance criterion, a simulated annealing algorithm is applied to determine whether the improved job sequence replaces the current sequence (see line 11 of Algorithm 1). The procedure of the basic IG algorithm is given below.

4.2. Proposed IGS

The existing literature has shown that the IG algorithm can achieve good results in solving flow shop and job shop scheduling problems. However, some strategies still need to be improved or modified. First, an efficient local perturbation strategy can be designed to reduce the time complexity of the insertion strategy in the IG algorithm, where a simple swap method is adopted to directly improve the current solution, which can not only ensure the convergence of the algorithm but also identify the global optimal solution as often as possible. Second, from the analysis of the simulation experiment results, we know that in the simulated annealing stage, the generated solution easily falls into a local optimum. Thus, to improve the performance of the IG algorithm in solving the BHFSP with energy consumption, in the simulated annealing stage, a global perturbation strategy based on a half-swap can be adopted to disturb the solution further to find a better solution. Based on the above motivations, we propose an improved IG algorithm based on a swap algorithm denoted as the IGS algorithm.

First, according to the blocking characteristic of the BHFSP, the MME heuristic [20] is adopted to generate an initial solution by reducing the critical path length. Second, a local perturbation strategy is designed to improve the current sequence to further strengthen the convergence of the proposed algorithm. Third, to prevent the solutions from being trapped in local optima, a new slight perturbation based on a half-swap is performed in the simulated annealing stage. Algorithm 2 presents the framework of the proposed the IGS algorithm.

4.2.1. Solution encoding and decoding

For the BHFSP, we adopt a permutation-based representation as the solution encoding method; that is, we apply an n-dimensional integer sequence $\pi=(\pi_1, \pi_2, ...\pi_j, ..., \pi_J)$ to represent a solution, where π_j denotes a job and J represents the number of job sequences. Please see Section 2 for further details on the decoding process; a specific example of a BHFSP with six jobs and three stages is given.

4.2.2. Destruction and construction phase

A good initial solution can accelerate the convergence speed of an algorithm. Thus, some heuristic methods, i.e., the NEH and MME, are used in the initialization stage. According to the blocking characteristic of the BHFSP, an MME heuristic is adopted to shorten the blocking time of jobs by using a shortest critical path. After initialization, destruction and construction operators are employed. Algorithm 3 shows the process of job sequence destruction, where π^{Remove} is generated by randomly extracting d jobs from π and entering them into π^{Remove} one by one. π^{origin} consists of the remaining jobs in π. The construction phase begins at the end of the destruction phase, and its main task is to construct a completed and feasible scheduling sequence. The specific process is shown by Algorithm 3.

4.3. Local perturbation strategies based on swap operator

In existing IG algorithms, the insertion operator is used as a local search strategy. Because all n jobs need to be inserted into (n-1) positions and each insertion will cause n-p+1 moves (where p is the best insertion position), its time complexity is $O(n^3)$. Moreover, if the objective value is further improved after all of the above operations, it will use too much execution time. Therefore, in this paper, to reduce time complexity, we abandon the traditional IG local search strategy and utilize a local perturbation strategy based on a swap operator whose time complexity is only $O(n^2)$, where n is the number of iterations, and each iteration contains a swap operator. Similarly, we propose another local perturbation for a comparison to the same kind of method, namely, the swap greedy method, and the efficiency of the proposed swap strategy is verified in Section 4 by comparing the insertion and insertion greedy strategies. The following provides an example of 6 jobs that clearly illustrate the above local perturbation strategies.

Suppose that the current sequence is $\pi^{temp}=(2, 4, 1, 3, 5, 6)$. The swap local perturbation strategy, namely, **IGS**, is as follows: (1) Randomly select two jobs, i.e., 2 and 3, and (2) exchange the two jobs to obtain a new sequence, $\pi^{temp2}=(3, 4, 1, 2, 5, 6)$. If π^{temp2} is better than π^{temp}, π^{temp2} replaces π^{temp}. Otherwise, π^{temp} remains unchanged. (3) Continue steps (1) and (2) until the termination condition is met.

For the same example, the swap greedy strategy, denoted as $\mathbf{IG_{SwapGreedy}}$, is as follows: (1) Swap the first job in π^{temp}, i.e., 2, with the remaining jobs, i.e., 4, 1, 3, 5, and 6, to obtain 5 new sequences, i.e., (4,2,1,3,5,6), (1,4,2,3,5,6), (3,4,1,2,5,6), (5,4,1,3,2,6), and (6,4,1,3,5,2), respectively. The generated sequence is denoted as π^{temp2}. (2) If π^{temp2} is better than π^{temp}, save π^{temp2} and set $\pi^{temp2}=\pi^{temp}$. Otherwise, only set $\pi^{temp2}=\pi^{temp}$. (3) Among the 5 sequences, choose the best sequence to replace current sequence π^{temp}. (4) Swap jobs 4,1,3,5, and 6 with the remaining jobs and repeat steps (1)–(3) until all positions are selected and the jobs in these positions are exchanged with all other jobs. The pseudocode of the local perturbation strategy is given in Algorithm 4.

4.4. Global perturbation strategy based on a half-swap operator

In the traditional IG algorithm, a simulated annealing acceptance criterion is used to prevent the solution from falling into a local optimum. However, the solution obtained by the previous stage is not necessarily good. To strengthen and expand the neighborhood search of the current solution, we develop a global perturbation strategy based on a half-swap denoted as $\mathbf{IG_{half\text{-}swap}}$ to disturb it further to find a better solution, and we integrate $\mathbf{IG_{half\text{-}swap}}$ into the simulated annealing stage. In this way, we can find a better solution more quickly. In addition, it is easier to find a solution that is similar to the global optimal solution, which is obviously different from other IG algorithms and is also an innovation of this paper. After the disturbance, the acceptance criterion is applied. If the new solution is better than the current best solution, it will replace the best solution.

To clearly illustrate the above steps of $\mathbf{IG_{half\text{-}swap}}$, an example is given. Let $\pi^{temp}=(2, 4, 1, 3, 5, 6)$. (1) We divide $\pi^{temp}=(2, 4, 1, 3, 5, 6)$ into two subsequences, $\pi^{Front}=(2, 4, 1)$ and $\pi^{Back}=(3, 5, 6)$. (2) We select the subsequence, i.e., π^{Front}, with more energy consumption to be further improved. (3) We swap the first job in π^{Front}, i.e., 2, with jobs 4 and 1, and obtain new sequences, i.e., (4,2,1) and (1,4,2), respectively, denoted as π^{Front_temp}. (4) If the energy consumption value of π^{Front_temp} is smaller than that of π^{Front}, π^{Front_temp} replaces π^{Front}. (5) A completed sequence, π^{temp_new}, is obtained by merging π^{Front} and π^{Back}. If π^{temp_new} is better than π^{temp}, then π^{temp} is replaced by π^{temp_new}. (6) We apply the second and third jobs in π^{Front} in turn; swap jobs 4 and 1 with the remaining jobs; and repeat steps (3), (4), and (5) above until all positions are selected and the jobs of these positions are exchanged with all other jobs. Finally, the best solution is obtained. Algorithm 5 illustrates the global perturbation strategy based on a half-swap.

5. Experimental results and analysis

5.1. Parameter settings

To verify the performance of the IGS algorithm, we randomly generate 150 test instances. Different numbers of jobs and stages can be combined to form various scale instances. We set the number of jobs as$J$and the number of stages as$S$, where $J\in${20,40,60,80,100,200,300} and $S\in${5, 10}. In addition, for the very large scale, we test ten instances in which J and S are set as 800 and 10, respectively. For each $J \times S$ combination, ten instances are randomly produced, so the total number of test instances is $7 \times 2 \times 10 + 10 \times 1 = 150$. The processing time is generated in a uniform distribution within the range [1,30], and the number of machines in each stage is randomly produced within the range [1,5], where the energy consumption per unit of time for idling, processing and blocking is sampled from uniform distribution ranges [1,3, 3,5] and [5,7], respectively.

By referring to [47], all compared algorithms in our paper followed the fair practices. First, all the algorithms are written in Visual Studio 2019 C++, realized on a PC with 16 GB RAM of memory and a 2.60 GHZ Intel Core i7 Pentium processor. Second, the same library functions and the same background running environment are adopted to make a fair comparison. No other programs are executed in parallel during implementing an algorithm. Third, to maintain consistency, we apply the same CPU elapsed time as the termination condition for all algorithms. For all tested cases, this termination condition is used in [28,31,38,46], and the dynamic setting of CPU parameters allows for more time as the number of stages or jobs increases. In this paper, we denote the termination condition as $TimeLimit$,$TimeLimit = J \times S \times \mathrm{CPU}$, and CPU $\in${5,7,9}.

To validate the effect of the proposed algorithm, we compare the IGS algorithm to nine different algorithms. We first compare our proposed IGS algorithm to classical swarm intelligence optimization algorithms, i.e., GA [29], DABC [30], EMBO [31], and DPSO [34]. Then, we compare the IGS algorithm to a series of IG algorithms, i.e., the original IG [17] and the latest improved IGRS, IGT, IGTALL, and VBIH [46], to further test the performance of the IGS algorithm. The main simulation parameters are set to the values recommended in the original literature. Table 2 lists the parameter settings of the compared algorithms.

All of the tested algorithms are compiled and coded in Visual Studio 2019, C++ run on a Microsoft Windows 10 operating system with 16 GB RAM of memory and a 2.60 GHZ Intel Core i7 Pentium processor. Each instance is repeated five times, and then the best instance is selected.

Table 2
Main simulation parameters of the comparison algorithms.

Algorithms	Population size *Psize*	Number of destruction jobs *d*	Crossover rate *a*	Variation rate *T*	Constant operator *Pc*	Temperature coefficient *Pm*
GA	100	/	4	0.85	0.7	0.1
DABC	20	/	30	/	/	/
EMBO	25	/	10	/	/	/
DPSO	100	/	200	/	/	/
IGA	1	3	10	0.5	/	/
IGRS	1	3	10	0.5	/	/
GT	1	2	10	0.5	/	/
IGTALL	1	3	10	0.5	/	/
VBIH	1	3	10	0.5	/	/
IGS	1	7	10	0.5	/	/

Table 3
Experimental results for energy consumption with and without local perturbation strategies.

Instance	**IGS**		**$IG_{SwapGreedy}$**		**IG_{Insert}**		**$IG_{InsertGreedy}$**		**$IGS_{N\text{-}Swap}$**		**$IGS_{N\text{-}HalfSwap}$**	
J × S	**MAX**	**MIN**	**MAX**	**MIN**	**MAX**	**MIN**	**MAX**	**MIN**	**MAX**	**MIN**	**MAX**	**MIN**
20 × 5	**16495**	**7873**	**16495**	7885	**16495**	7898	**16495**	7895	**16495**	**7873**	**16495**	7884
20 × 10	31262	**22953**	31284	22961	**31226**	22953	31331†	22971	31295	22961	31321†	**22953**
40 × 5	29408	**13136**	29439	13356‡	29448	13410‡	29437†	13477‡	29408	13295‡	**29403**	13171
40 × 10	69651	**47178**	69823†	47383‡	69888†	47378‡	69644	47291‡	69757†	47417‡	**69616**	47190
60 × 5	**55115**	**33450**	55270†	33608‡	55431†	33662‡	55562†	33783‡	55284†	33653‡	55497†	33550‡
60 × 10	106516	60277	106917†	60791‡	106722†	60481‡	106909†	61594‡	106707†	60940‡	**106465**	**60221**
80 × 5	115730	**27466**	115681	27713‡	115735	27656‡	115973†	27894‡	115959†	27816‡	**115680**	27526‡
80 × 10	**161993**	81981	162043†	82072‡	162203†	82346‡	162807†	81980	162117†	**81964**	162118†	82086‡
100 × 5	**117752**	**31361**	117822†	31767‡	117947†	31876‡	117842†	32233‡	117811†	32096‡	117770	31408‡
100 × 10	206722	**111060**	206765	111366‡	206913†	111782‡	207257†	112260‡	206696	111559‡	**206632**	111359‡
200 × 5	**250584**	60963	250703†	61248‡	251710†	62384‡	251578†	62385‡	251425†	62417‡	250975†	**60608**
200 × 10	**454244**	**222839**	456319†	223467‡	458128†	224068‡	459570†	224402‡	458606†	225179‡	455105†	223115‡
300 × 5	**409509**	**151898**	409791†	151898	410953†	151900	410449†	**151898**	411525†	151918	**409509**	**151898**
300 × 10	**672018**	**332151**	675080†	333085‡	675095†	333367‡	674797†	334175‡	677719†	334638‡	672049	**332151**

5.2. Evaluation indicator

We use the relative percentage increment (RPI) to evaluate the performance of all of the algorithms. The formula is given as follows:

$$RPI = (c_i - c_{best})/c_{best} \times 100\%, \tag{16}$$

where c_{best} is the minimum energy consumption value obtained by all of the algorithms and $c_i$is the energy consumption value yielded by algorithm i. Clearly, the lower the RPI is, the better the performance of the algorithm is.

From the objective analysis of the BHFSP, we know that the energy consumption value is large, and therefore, the difference between the numerator and denominator is small. The RPI values obtained by all of the algorithms are also very small. Thus, to intuitively and comprehensively evaluate the performance of the proposed algorithm, we compare not only the RPI value but also the maximum (MAX) and minimum (MIN) energy consumption levels, and the best results are marked in bold.

Tables 3–7 present the Wilcoxon rank sum tests with a significance level of 0.05. The tests are designed to determine whether the objective values obtained by the IGS algorithm are significantly different from those obtained by the other compared algorithms for different scale examples. In Tables 3–7, symbol '†' indicates that the MAX value of the comparison algorithm is significantly different from that of the IGS algorithm, '‡' indicates that the MIN value of the algorithm is significantly different from that of the IGS algorithm, and if there is no symbol identification, the difference between the two algorithms is not significant.

5.3. Sensitivity analysis of parameters

In this section, the impact of d values on IGS algorithm performance is investigated. In the proposed algorithm, parameter$d$, the number of removed jobs, is very important to the local perturbation strategy. Thus, we first present a sensitivity analysis of parameter d before demonstrating the effectiveness of the proposed algorithm. Fig. 2 shows the impact of different$d$values. In Fig. 2, all tests are carried out under condition CPU = 7. Without a loss of generality, the value of d in the abscissa is tuned from 2 to 9 with a step size of 1, and the ordinate is the interval of the energy consumption value of energy consumption for small-, moderate- and large-scale instances.

Fig. 2 shows that the average value of the interval gradually decreases as the value of d increases, and the objective reaches the minimum value when d=7, suggesting that the local search is always beneficial for the proposed algorithm. However, when $d > 7$, the average value of energy consumption gradually increases. A d with a large value means that more time is spent performing the insertion operator, which decreases opportunities to generate promising solutions in a certain number of iterations. Thus, from these results, we set the value of d to 7, which works well in terms of the average energy consumption value.

5.4. Evaluation with different perturbation strategies

The basis of the IGS algorithm lies in the proposed perturbation strategies. Therefore, it is necessary and interesting to verify performance and effectiveness by evaluating the use and absence of different strategies. This section compares the performance of the four proposed perturbation strategies and selects the best strategy to apply to the proposed algorithm. All of the compared strategies have the same parameter values and run at CPU = 7. For each scale, both the average and minimum energy consumption levels for the 10 cases are reported in Table 3, where **IGS** includes local and global perturbation strategies based on swap and half-swap operations. $\mathbf{IG_{SwapGreedy}}$ only contains the local perturbation strategy based on the swap greedy operator, $\mathbf{IG_{Insert}}$ includes the local perturbation strategy based on an insert operator, $\mathbf{IG_{InsertGreedy}}$ applies the local perturbation strategy with an insert greedy operator, and $\mathbf{IG_{SwapGreedy}}$ uses the swap greedy operation. $\mathbf{IGS_{N\text{-}Swap}}$ employs

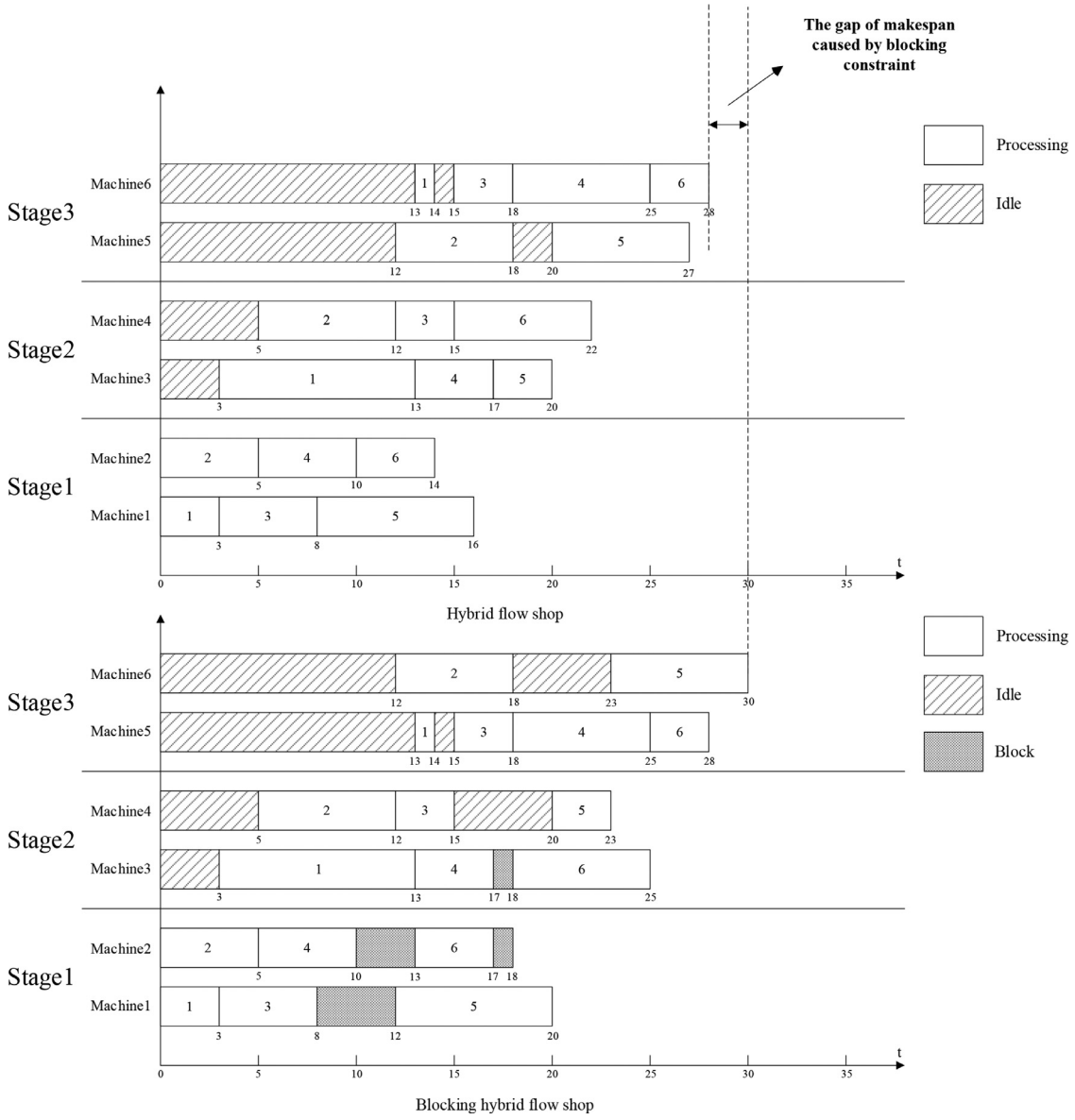

Fig. 1. Gantt comparison of the HFSP with and without blocking constraints.

the perturbation strategy without the swap strategy, and $\mathbf{IGS}_{\mathbf{N\text{-}HalfSwap}}$ adopts the perturbation strategy without the half-swap strategy.

For consistency, these strategies are tested individually under the same termination criterion. According to the experimental results given in Table 3, we observe that the IGS algorithm generates 8/14 best MAX values and 11/14 MIN objective values and thus significantly more than the other strategies, suggesting that the swap operator achieves good performance in terms of convergence and diversity. In addition, from the results of Wilcoxon rank sum tests in Table 3, the IGS algorithm is significantly different from the other compared strategies. The reasoning here may be that the insertion operator changes the relative position of only one job at a time, while the swap operator can alter the relative positions of two jobs at a time, which is conducive to exploiting promising subregions and exploring irregular unknown regions. Because the time complexity of the insertion operation is O (n^3) and that of the exchange operation is O (n^2), the number of swap operations adopted in the same time period is bound to be more than the number of insertion operations. Swap operation can improve the blocking status of the job sequence faster and reduce the corresponding energy consumption.

Based on the experimental results, we adopt the **IGS** algorithm, which contains both perturbation strategies, to solve the BHFSP.

5.5. Performance comparison of all the compared algorithms

For the sake of fairness and consistency, all of the compared algorithms have the same termination time and experimental environment. For $Timelimit = J \times S \times$CPU, we set three different CPU parameter values, i.e., 5,7, and 9, to more comprehensively apply the algorithms. In Tables 4–7, the best result of each comparative method is bolded.

To prove the presented results of statistical validity, the algorithms are tested by a one-factor analysis of variance (ANOVA) in which the type of algorithm is regarded as a single factor. Figs. 3 and 4 display the mean plots with Tukey HSD intervals at the 95% confidence level for all compared algorithms under different value of the CPU parameter. In addition, Figs. 5–7 presents the box plots of the results generated from three scales in all compared algorithms, where '20 × 10', '80 × 10' and '200 × 5' represent the small-, moderate- and large-scale instances, respectively.

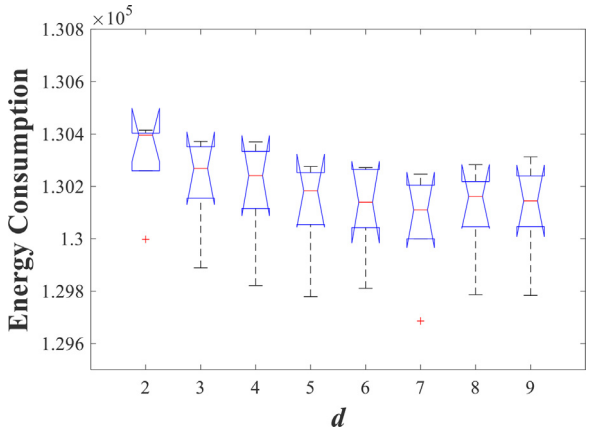

Fig. 2. Box plot of d values of the IGS algorithm.

5.6. Analysis and discussion

For the 15 different instance sets, Tables 4–7 list the MAX and MIN values of energy consumption obtained by the 10 compared algorithms when CPU = 5 and 7, respectively. In addition, the RPI values of all of the algorithms are shown in brackets in Tables 4–7. According to these experimental results for the two stopping criteria, considering all RPI values, the IGS algorithm is obviously better than the other compared algorithms in all test sets. The reason may be that the perturbation strategies designed for blocking constraints reduce the invalid operation of the machines. At all RPI scales, the IG series algorithms perform better than the compared swarm intelligence algorithms. It shows that the IG algorithm for a single solution can indeed explore the solution more deeply, and then find the solution with lower energy consumption. Among the IG algorithms, the IGS algorithm proposed in this paper achieves the best RPI value in all test sets. In Tables 4 and 5, regarding MIN values, the IGS algorithm can obtain 13 and 9 lowest values for 15 test sets, respectively. For MAX values, we observe that the IGS algorithm obtains 15 and 7 lowest values for 15 test sets. In Tables 6 and 7, the IGS algorithm obtains 13/15 and 11/15 best MIN values. In addition, the IGS algorithm gains 15/15 and 10/15 best MAX values. Clearly, from the number of best values obtained, the IGS algorithm performs the best. The large-scale experimental results show that the perturbation strategies designed for HFSP with blocking constraints can effectively alleviate the situation of machine blocking and reduce energy consumption more effectively than other current algorithms. Furthermore, Tables 4–7 show that for the Wilcoxon rank sum test, the compared algorithms show statistically significant differences from the IGS algorithm at most scales.

In addition, Table 8 (see the Appendix) shows the best values for energy consumption for all test instances to result from the proposed algorithm in updating the upper bound. As shown in Table 8, (1) for the 15 test sets with all scales, 14 and 8 of the best average and minimum values are obtained by the IGS algorithm, exceeding the corresponding numbers of best values yielded by IGA (0 and 0), IGRS (0 and 0), IGT (1 and 2), IGTALL (0 and 2), VBIH (0 and 5), GA (0 and 0), DABC (0 and 0), EMBO (0 and 2), and DPSO (0 and 0); (2) for the upper bounds of the 150 test instances, 17/150, 14/150, 33/150, 19/150, 42/150, 13/150, 5/150, 16/150, and 0/150 best objective values are produced by IGA, IGRS, IGT, IGTALL, VBIH, GA, DABC, EMBO, and DPSO, respectively, while 106 out of 150 (106/150=70.7%) of the best upper bounds are obtained by the proposed algorithm. These experimental results clearly demonstrate that the proposed algorithm can update more upper bounds than the compared algorithms.

From these results, the proposed IGS algorithm shows outstanding performance. Compared to the classical swarm intelligence algorithms, i.e., GA, DABC, EMBO and DPSO, IGS is very prominent in its in-depth exploration of a single solution, which makes it easier to find the job sequence with low energy consumption. Compared to the IG series algorithms, the IGA, IGRS, IGT, IGTALL and VBIH, although the proposed IGS algorithm does not achieve the best value on all scales, its performance has surpassed similar algorithms. The superiority of the algorithm is attributed to the local and global perturbation strategies proposed in Sections 4.3 and 4.4, the proposed strategies can reduce the time of ineffective machining, improving machine operation efficiency, it proves that the proposed strategies strengthen the exploitation and exploration abilities of the IGS algorithm. From the MIN and MAX energy consumption values obtained, the proposed strategy can directly reduce the occurrence of blocking conditions in the job sequence by changing job positions several times, reducing the energy wastage caused by blocking.

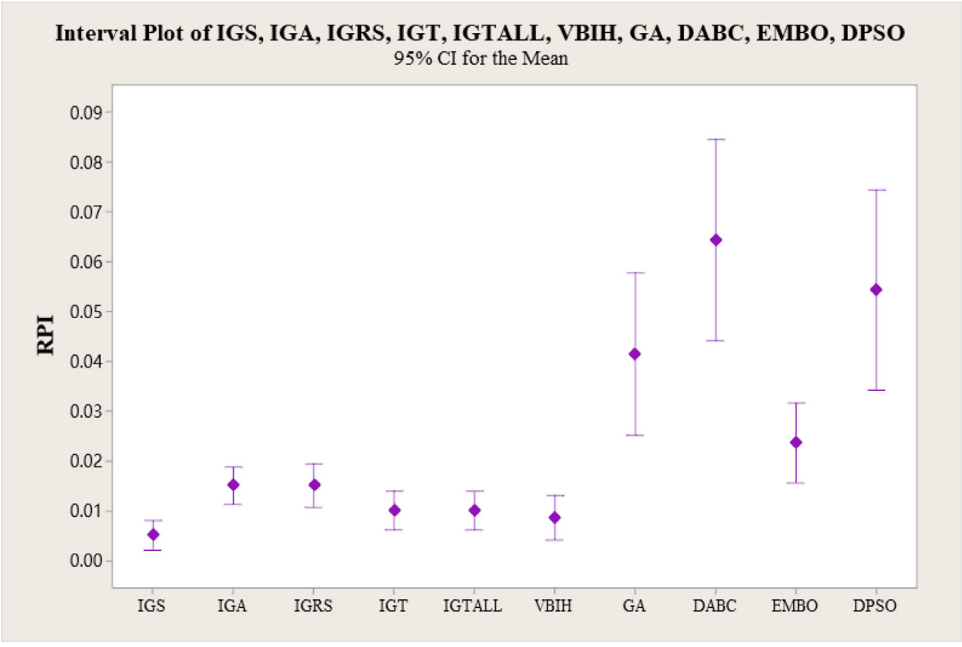

Fig. 3. Comparison of the 95% Tukey HSD confidence intervals of the algorithms when CPU = 5.

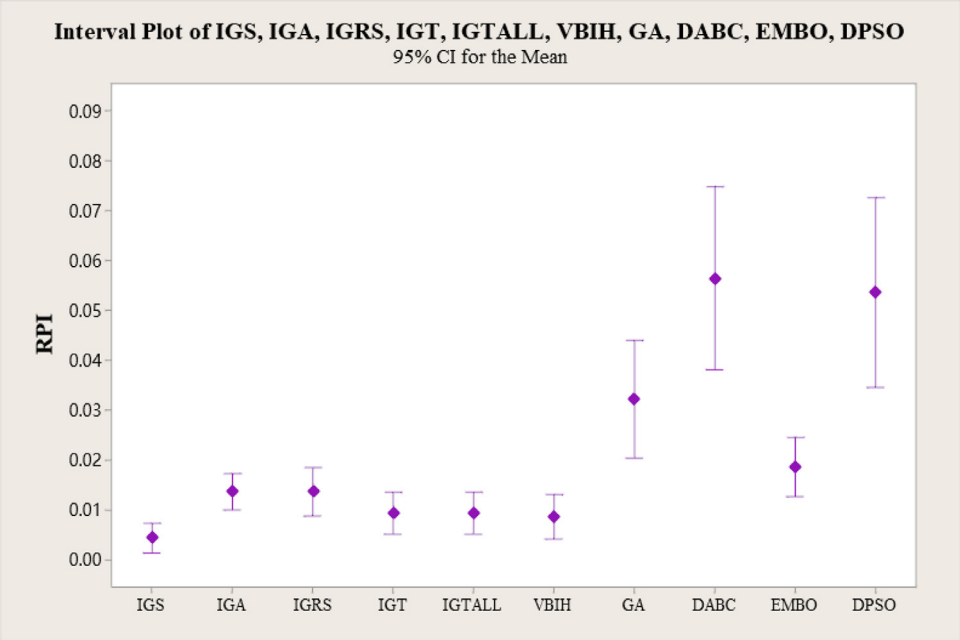

Fig. 4. Comparison of the 95% Tukey HSD confidence intervals of the algorithms when CPU = 7.

For statistical analysis, Tables 4–7 present the Wilcoxon rank sum tests with a significance level of 0.05. This test is used to determine whether the results obtained by one algorithm are statistically significantly different from those obtained by the other algorithms for all test sets. From Tables 4–7, for most BHFSP test instances, the IGS algorithm is significantly different from the other compared algorithms. Figs. 3 and 4 present the 95% Tukey HSD confidence intervals obtained when CPU = 5 and 7, respectively. From Figs. 3 and 4, the IGS algorithm performs statistically better than the other algorithms for different elapsed CPU times. The IGT, IGTALL and VBIH algorithms follow sequentially, and all show good performance. In summary, the above experimental results prove that the proposed IGS algorithm is much more effective than the compared algorithms in addressing the BHFSP to minimize energy consumption. Similarly, we present box plot tests on small, moderate and large scales. Figs. 5–7 show that our proposed IGS algorithm obtains the lowest interval values and shows a relatively stable condition. It may be that the proposed perturbation strategies can better explore unknown neighborhoods, prevent the solution from falling into a local optimum, and find the optimal solution more easily in less time.

Table 8 (*continued*)

Instance		IGS	IGA [17]	IGRS [46]	IGT [46]	IGTALL [46]	VBIH [46]	GA [29]	DABC [30]	EMBO [31]	DPSO [34]
	TI_127	345348	345387	345390	345551	345309	**345257**	345754	347513	345543	346099
	TI_128	368513	369733	369714	**368083**	368527	368490	374894	375806	370511	376012
	TI_129	174394	176641	176797	174515	175028	**174155**	188319	195450	177894	179260
	TI_130	**281037**	281225	281381	281297	281249	281229	281697	282158	281427	281622
AVG MIN		250561.4	251193.6	251212.8	**250502.1**	250563.7	250537.2	255113	256825.2	251959.1	252950.1
		151898	151900	151904	**151898**	151957	**151898**	152067	152387	151899	151957
300 × 10	TI_131	**666405**	676646	674984	**667142**	669107	668920	712606	725135	685899	687680
	TI_132	481295	481739	481666	**480496**	480678	480750	489009	490627	483717	485307
	TI_133	332143	333644	333098	332326	**332126**	**332126**	339170	340710	335127	334636
	TI_134	442818	443296	443366	**442224**	443149	442527	452904	453978	448413	445026
	TI_135	364282	368653	368775	364964	365513	**363962**	392956	405058	374241	372889
	TI_136	**447017**	448227	447996	447373	447332	447332	468228	472960	458943	450200
	TI_137	**670020**	676707	675292	670769	672350	673020	699223	702676	688807	693996
	TI_138	**341310**	341814	341881	341463	341633	341993	362716	366687	355436	344179
	TI_139	**591013**	594840	594900	592804	592821	591870	602333	608240	598170	599330
	TI_140	**555131**	572159	571014	563633	562554	560533	627487	657175	589207	581875
AVG MIN		**489143.4**	493772.5	493297.2	490319.4	490726.3	490303.3	514663.2	522324.6	501796	499511.8
		332143	333644	333098	332326	**332126**	**332126**	339170	340710	335127	334636
800 × 10	TI_141	**1315359**	1342235	1343653	1343869	1337845	1336124	1435260	1473989	1417609	1434307
	TI_142	**1496199**	1501016	1501164	1498880	1498094	1497954	1515740	1534072	1510872	1518832
	TI_143	**1686289**	1708099	1710540	1693788	1687032	1686791	1785319	1987264	1774098	1800338
	TI_144	986171	1002024	1002705	990196	984012	**983570**	1055757	1143463	1050037	1066300
	TI_145	**1702741**	1722703	1720849	1715497	1712496	1712830	1796089	1833326	1783880	1797921
	TI_146	1299959	1300590	1300887	1300402	**1299281**	1299475	1304817	1307121	1303061	1305643
	TI_147	1100771	1104621	1105342	1102492	1096290	**1096266**	1210776	1376197	1195650	1215806
	TI_148	1112917	1117658	1117735	1112883	**1109212**	1109440	1180009	1270559	1169799	1186142
	TI_149	**1502239**	1512538	1512414	1510936	1506354	1507324	1555247	1574675	1549271	1560674
	TI_150	**1155174**	1160750	1161022	1161508	1156686	1157048	1235090	1289732	1218561	1232795
AVG MIN		**1335781.9**	1347223.4	1347631.1	1343045.1	1338730.2	1338682.2	1407410.4	1479039.8	1397283.8	1411875.8
		986171	1002024	1002705	990196	984012	**983570**	1055757	1143463	1050037	1066300

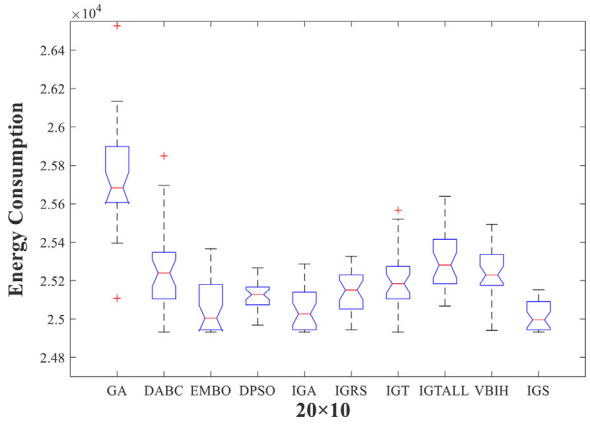

Fig. 5. Box plot of all algorithms for the small-scale case.

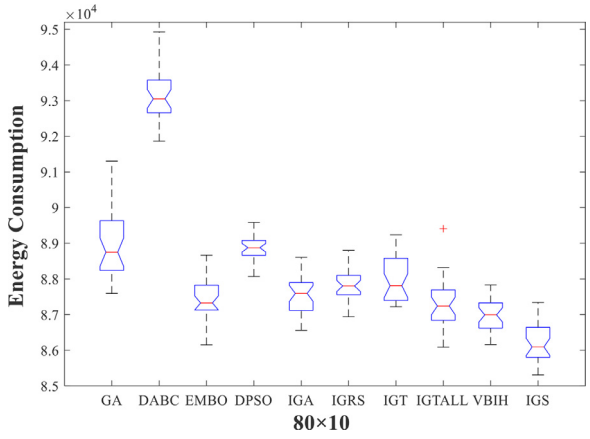

Fig. 6. Box plot of all algorithms for the moderate-scale case.

Remarks: as the above experimental results and analysis show, the proposed IGS algorithm is an effective algorithm for solving the BHFSP. The reasons can be concluded as follows.

(1) Local and global strategies based on swap operations are used to disturb the job sequence. The local perturbation strategy is used to replace the iterative improvement strategy. This strategy can carry out more iterations of the current solution within a limited amount of time and has a high probability of helping the solution jump out of the current neighborhood to avoid falling into a local optimum. Different from the traditional IG algorithm, the global perturbation strategy is integrated into the simulated annealing criteria, which improves the global search ability of the algorithm; once again, this helps the solution explore a broader neighborhood. At the same time, the characteristics of the BHFSP are considered in these two strategies.

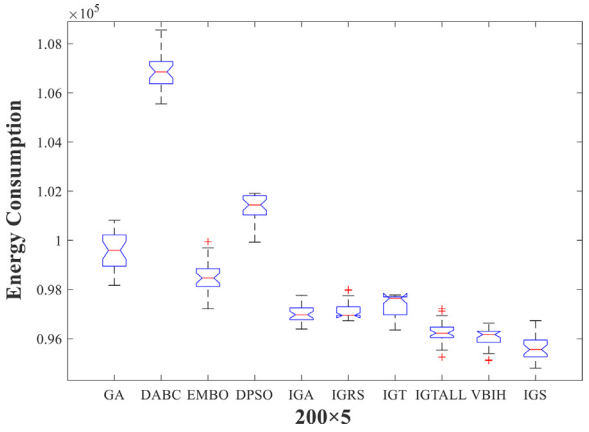

Fig. 7. Box plot of all algorithms for the large-scale case.

(2) The IGS algorithm has only one solution, since it always iterates over only one solution; therefore, the algorithm can explore the solution at a deeper level, allowing it to easily obtain scheduling sequences with less energy consumption and effectively change the positions of blocked jobs. Moreover, the IGS algorithm performs well not only on a small scale but also on a very large scale as shown by the 800 × 10 scale cases. This result indicates that the algorithm achieves stable performance, good robustness and strong applicability.

5.7. Gantt charts of specific cases

To observe the optimal scheduling sequence processing and blocking status more intuitively, we provide an optimal scheduling plan for managers of factories. Figs. 8 and 9 show the Gantt charts of TI_02 (20 × 5 scale) and TI_22 (40 × 5 scale) under condition CPU = 7, respectively. Each color represents the operational status of a specific job in all stages. The numbers of scheduling jobs and stages (Job-Stage) are marked with rectangles. The horizontal axis represents the makespan value. After the completion time of the last job, we give the energy consumption level of all machines. From Fig. 8, the optimal scheduling sequence is 3-15-1-8-18-14-9-7-11-10-16-13-2-12-5-4-20-19-17-6, the makespan value is equal to 223, and the energy consumption is equal to 7875. From Fig. 9, the optimal scheduling sequence is 37-34-2-17-19-1-36-40-33-3-30-27-23-12-15-13-26-16-4-28-24-22-25-21-7-5-6-20-39-31-11-10-32-18-14-9-8-29-38-35, the makespan value is 423, and the energy consumption is 18247.

6. Conclusion

This paper addressed the BHFSP with energy consumption criteria. In this paper, a reasonable mathematical model of the energy-efficient

Algorithm 4
IG Swap greedy

Input: π^{temp}
Output: π^{temp}
Begin
For $j = 1$ **to** J // the number of iterations
$\pi^{temp2} = \pi^{temp}$
Select the job in π^{temp2} that equals π_j^{temp} to swap with the rest jobs and save better ones
Find the best Sequence denoted as π^{temp2}
If (π^{temp2} is better than π^{temp})
$\pi^{temp} = \pi^{temp2}$
EndIf
EndFor
End

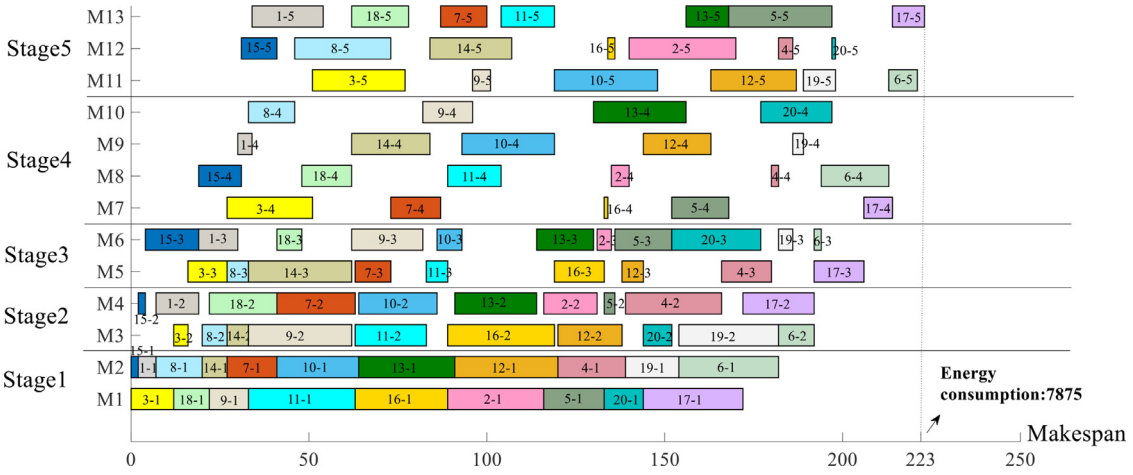

Fig. 8. Gantt chart of the TI_02 case when CPU = 7.

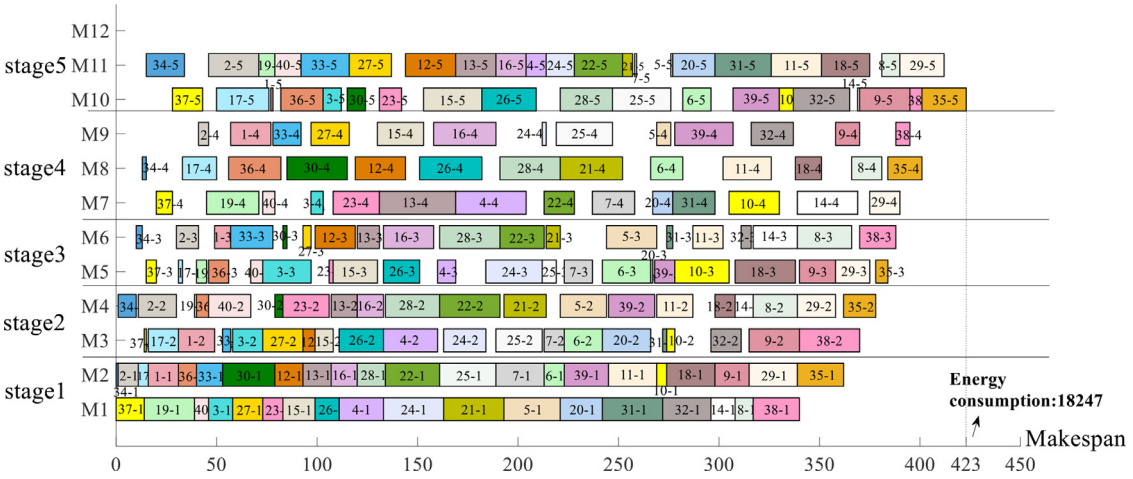

Fig. 9. Gantt chart of the TI_22 case when CPU = 7.

BHFSP that considers job blocking and idle machines is proposed, and an effective IGS algorithm is designed to solve the BHFSP. From the 150 example tests, the proposed algorithm performs better than the existing classical algorithms and improved IG algorithms. In this algorithm, a heuristic method called the MME is utilized to generate an initial solution, and then, two perturbation strategies based on swap and half-swap methods are developed to improve the exploitation and exploration of this algorithm. Simulation experiments show that our strategies are more effective than these existing algorithms. The superior performance of this algorithm is mainly attributed to the following.

(1) The strategies designed for blocking constraints can effectively adjust the scheduled job sequence and reduce energy consumption caused by blocking constraints.
(2) Two perturbation strategies are integrated into the IG framework to balance global and local search abilities.
(3) The local perturbation strategy can reduce the computational complexity of the original local search strategy, and with an increasing number of iterations, the local neighborhood of the solution is more fully explored.
(4) The global perturbation strategy can further improve the exploration of unknown subregions and enhance the diversity of solutions, further reducing the energy consumption of job processing.

Although the IGS algorithm outperforms the algorithms considered, the algorithm's performance could be further improved by using reinforcement learning methods or strategies of adaptive learning when devising it. Our future research will treat energy efficiency as the main optimization goal. Similarly, the algorithm can be extended from single-objective to multiobjective research. In addition, we may consider a distributed situation or consider other optimization goals, such as the makespan and total flow time, in exploring the issues addressed. We may also consider uncertain conditions such as machine breakdowns, changing due dates, and job processing times. Some practical and realistic conditions, for instance, various setup times and no-wait scenarios, would be interesting to consider.

Declaration of Competing Interest

We declare that we have no personal relationships with other people or organizations that can inappropriately influence our work. We declare that we do not have any commercial or associative interest that represents a conflict of interest in connection with the work submitted.

CRediT authorship contribution statement

Hao-Xiang Qin: Conceptualization, Visualization, Funding acquisition, Formal analysis, Data curation, Writing – original draft, Writing – review & editing. **Yu-Yan Han:** Conceptualization, Visualization, Funding acquisition, Formal analysis, Data curation, Writing – original draft, Writing – review & editing. **Biao Zhang:** Data curation, Writing – original draft, Writing – review & editing. **Lei-Lei Meng:** Data curation, Writing – original draft, Writing – review & editing. **Yi-Ping Liu:** Data

Table 4
Experimental results for energy consumption compared to those of classic algorithms when CPU = 5.

Instance	IGS			GA [29]			DABC [30]			EMBO [31]			DPSO [34]		
J × S	MAX	MIN	RPI	MAX	MIN	RPI	MAX	MIN	RPI	MAX	MIN	RPI	MAX	MIN	RPI
20 × 5	**16495**	7895	**0.00**	**16495**	7918‡	0.01	16526†	8007‡	0.02	**16495**	**7870**	0.01	18042†	8581‡	0.09
20 × 10	**31262**	**22953**	**0.00**	31429†	22990‡	0.01	31263†	22982‡	0.02	31297	22961	0.01	33583†	24114‡	0.05
40 × 5	**29435**	**13030**	**0.01**	29739†	13431‡	0.04	29914†	14118‡	0.09	29498	13168‡	0.03	30000†	15092‡	0.11
40 × 10	**69611**	**47190**	**0.01**	69851†	48044‡	0.02	70723†	49178‡	0.05	69744†	47935‡	0.02	71301†	51338‡	0.05
60 × 5	**55382**	**33478**	**0.00**	55657†	33946‡	0.02	56817†	34625‡	0.03	55961†	33646‡	0.01	57848†	34338‡	0.03
60 × 10	**106513**	**60377**	**0.01**	107263†	61576‡	0.03	111245†	64227‡	0.07	107746†	60997‡	0.02	113750†	62015‡	0.05
80 × 5	**115926**	**27474**	**0.00**	116528†	28002‡	0.02	118207†	28727‡	0.04	116080†	27782‡	0.01	119820†	28410‡	0.03
80 × 10	**161899**	**81925**	**0.01**	165359†	85828‡	0.05	166244†	88624‡	0.09	162836†	83899‡	0.02	165378†	83237‡	0.04
100 × 5	**117786**	**31413**	**0.00**	118251†	32512‡	0.03	120287†	34609‡	0.05	118196†	32193‡	0.02	119504†	33702‡	0.03
100 × 10	**206726**	111465	**0.01**	208039†	117072‡	0.07	220981†	119878‡	0.11	207924†	**113305**‡	0.04	208177†	113537‡	0.04
200 × 5	**250975**	**60608**	**0.01**	269297†	65860‡	0.07	274196†	68022‡	0.09	253339†	64081‡	0.04	253619†	63627‡	0.14
200 × 10	**455083**	**222839**	**0.01**	480557†	240067‡	0.10	489810†	240855‡	0.13	466725†	234505‡	0.05	472748†	224800‡	0.03
300 × 5	**409509**	**151898**	**0.00**	414430†	152914‡	0.03	417100†	152737‡	0.03	412287†	151904‡	0.01	413836†	151957‡	0.05
300 × 10	**671373**	**332151**	**0.00**	726368†	341201‡	0.08	728528†	341669‡	0.08	697574†	336586‡	0.04	693996†	334636‡	0.02
800 × 10	**1702741**	**986171**	**0.00**	1823892†	1077554‡	0.07	2010612†	1156467‡	0.13	1801636†	1065969‡	0.06	1830261†	1086087‡	0.07

Table 5
Experimental results for energy consumption compared to those of other existing IG algorithms when CPU = 5.

Instance	IGS			IGA [17]			IGRS [46]			IGT [46]			IGTALL [46]			VBIH [46]		
J × S	MAX	MIN	RPI	MAX	MIN	RPI	MAX	MIN	RPI	MAX	MIN	RPI	MAX	MIN	RPI	MAX	MIN	RPI
20 × 5	**16495**	**7895**	**0.00**	16535†	**7895**	0.01	**16495**	7950‡	0.01	**16495**	7941‡	0.01	**16495**	7906	0.01	**16495**	**7895**	0.01
20 × 10	**31262**	**22953**	**0.00**	31263	22953	0.01	31321	22982	0.01	31263	22953	0.01	31331†	22970	0.01	31451†	**22953**	**0.00**
40 × 5	29435	**13030**	**0.01**	29424	13563‡	0.03	29508†	13665‡	0.03	**29404**	13274‡	0.03	29472	13621‡	0.03	29444	13404‡	0.03
40 × 10	**69611**	**47190**	**0.01**	69976†	48093‡	**0.01**	69929†	48092‡	0.02	69845†	47975‡	**0.01**	69990†	47731‡	**0.01**	69889†	47725‡	**0.01**
60 × 5	55382	**33478**	**0.00**	55661†	33613‡	0.01	55697†	33837‡	0.01	**55310**	33639‡	0.01	55399	33770‡	0.01	55411†	33589‡	0.01
60 × 10	**106513**	**60377**	**0.01**	107121†	60964‡	0.02	107923†	60847‡	0.02	107044†	60953‡	**0.01**	107216†	60730‡	**0.01**	106749†	60730‡	**0.01**
80 × 5	115926	27474	**0.00**	116061†	27795‡	0.01	116151†	27825‡	0.01	115926	**27468**	**0.00**	116009†	27638‡	0.01	**115898**	27554‡	**0.00**
80 × 10	161899	**81925**	**0.01**	162009†	82199‡	0.02	162374†	82550‡	0.02	161937†	82705‡	**0.01**	162104†	82495‡	**0.01**	**161842**	82007‡	**0.01**
100 × 5	**117786**	**31413**	**0.00**	118071†	32036‡	0.01	118253†	31671‡	0.01	117933†	31732‡	0.01	118250†	31418	0.01	118041†	31515‡	0.01
100 × 10	206726	111465	**0.01**	206851†	112098‡	0.02	206922†	112477‡	0.02	206665	111257	**0.01**	206911†	111615‡	**0.01**	**206599**	**111171**	**0.01**
200 × 5	250975	60608	**0.01**	251715†	62218‡	0.02	252575†	62501‡	0.02	250801	**60591**	**0.01**	250752	61125‡	**0.01**	**250727**	61356‡	**0.01**
200 × 10	455083	222839	**0.01**	464337†	222944‡	0.02	464622†	222943‡	0.02	**454349**	222525	**0.01**	458805†	**221619**	**0.01**	456859†	221771	**0.01**
300 × 5	409509	**151898**	**0.00**	410962†	151912‡	0.01	410584†	151904	**0.00**	**408865**	**151898**	0.01	408809	151957‡	**0.00**	409397	**151898**	**0.00**
300 × 10	**671373**	332151	**0.00**	678698†	333824‡	0.01	675292†	333098‡	0.01	672102†	332326‡	**0.00**	672350†	**332126**	**0.00**	673020†	**332126**	**0.00**
800 × 10	**1702741**	986171	**0.00**	1722703†	1002024‡	0.01	1720849†	1002705‡	0.01	1715497†	990196‡	0.01	1712496†	984012	**0.00**	1712830†	**983570**	**0.00**

Table 6
Experimental results of energy consumption compared to those of classic algorithms when CPU = 7.

Instance	IGS			GA [29]			DABC [30]			EMBO [31]			DPSO [34]		
J × S	MAX	MIN	RPI	MAX	MIN	RPI	MAX	MIN	RPI	MAX	MIN	RPI	MAX	MIN	RPI
20 × 5	**16495**	7873	**0.00**	**16495**	7918‡	0.01	16526†	8007‡	0.01	**16495**	**7870**	0.01	18042†	8581‡	0.09
20 × 10	**31262**	**22953**	**0.00**	31429†	22990‡	0.01	31263	22982‡	0.01	31297	22961	0.01	33583†	24114‡	0.05
40 × 5	**29408**	13136	**0.01**	29627†	13313‡	0.03	29893†	14118‡	0.08	29497†	**13111**	0.02	30000†	15092‡	0.11
40 × 10	**69651**	**47178**	**0.01**	69851†	48044‡	0.02	70535†	49161‡	0.05	69744†	47734‡	0.01	71301†	51338‡	0.05
60 × 5	**55115**	**33450**	**0.00**	55594†	33875‡	0.01	56725†	34368‡	0.03	55953†	33604‡	0.01	57848†	34338‡	0.03
60 × 10	**106516**	**60277**	**0.00**	107028†	61466‡	0.02	109667†	63640‡	0.06	107564†	60830‡	0.02	113750†	62015‡	0.05
80 × 5	**115730**	**27466**	**0.00**	116114†	27893‡	0.02	118163†	28647‡	0.03	116020†	27701‡	0.01	119820†	28410‡	0.03
80 × 10	**161993**	**81981**	**0.01**	164080†	84901‡	0.04	165911†	88391‡	0.07	162669†	83455‡	0.02	165378†	83237‡	0.04
100 × 5	**117752**	**31361**	**0.00**	118131†	32222‡	0.02	119684†	34537‡	0.05	118127†	31935‡	0.01	119504†	33702‡	0.03
100 × 10	**206722**	**111060**	**0.01**	207722†	115279‡	0.05	217169†	118392‡	0.10	207778†	112876‡	0.03	208177†	113537‡	0.04
200 × 5	**250584**	**60963**	**0.01**	262797†	65085‡	0.06	272974†	67613‡	0.08	251895†	62918‡	0.03	253619†	63627‡	0.13
200 × 10	**454244**	**222839**	**0.01**	477360†	237289‡	0.07	486860†	239394‡	0.11	463635†	233131‡	0.04	472748†	224800‡	0.03
300 × 5	**409509**	**151898**	**0.00**	414041†	152301‡	0.03	416301†	152639‡	0.03	411959†	151902‡	0.01	413836†	151957‡	0.05
300 × 10	**672018**	**332151**	**0.00**	718748†	339903‡	0.06	726522†	341194‡	0.08	693195†	336002‡	0.03	693996†	334636‡	0.02
800 × 10	**1702741**	**986171**	**0.00**	1803626†	1063092‡	0.06	1995258†	1151241‡	0.13	1786842†	1054766‡	0.05	1818535†	1074096‡	0.07

Table 7
Experimental results of energy consumption compared to those of other existing IG algorithms when CPU =7.

Instance	IGS			IGA [17]			IGRS [46]			IGT [46]			IGTALL [46]			VBIH [46]		
J × S	MAX	MIN	RPI	MAX	MIN	RPI	MAX	MIN	RPI	MAX	MIN	RPI	MAX	MIN	RPI	MAX	MIN	RPI
20 × 5	**16495**	**7873**	**0.00**	16535†	7895	0.01	**16495**	7936‡	0.01	**16495**	7941‡	0.01	**16495**	7906‡	0.01	**16495**	7895	0.01
20 × 10	**31262**	**22953**	**0.00**	31263	**22953**	0.01	31284	22982	**0.00**	31263	**22953**	**0.00**	**31262**	22970	**0.00**	31451†	**22953**	**0.00**
40 × 5	29408	**13136**	**0.01**	29412	13559‡	0.03	29508†	13419‡	0.03	**29404**	13274‡	0.03	29472†	13616‡	0.03	29444	13404‡	0.03
40 × 10	**69651**	**47178**	**0.01**	69976†	47921‡	**0.01**	69784†	47888‡	**0.01**	69845†	47975‡	**0.01**	69990†	47731‡	**0.01**	69889†	47256‡	**0.01**
60 × 5	**55115**	**33450**	**0.00**	55661†	33592‡	0.01	55697†	33765‡	0.01	55310†	33639‡	0.01	55399†	33770‡	0.01	55411†	33589‡	0.01
60 × 10	**106516**	**60277**	**0.00**	107094†	60763‡	0.01	107755†	60647‡	0.02	106958†	60953‡	0.01	107216†	60730‡	0.01	106749†	60561‡	0.01
80 × 5	**115730**	**27466**	**0.00**	116058†	27756‡	0.01	116028†	27822‡	0.01	115926†	27468	**0.00**	115968†	27638‡	0.01	115898†	27554‡	**0.00**
80 × 10	161993	**81981**	**0.01**	161997	82199‡	**0.01**	162374†	82463‡	0.02	161937	82683‡	**0.01**	162104†	82184‡	**0.01**	**161842**	82007‡	**0.01**
100 × 5	**117752**	**31361**	**0.00**	118071†	31978‡	0.01	118205†	31671‡	0.01	117894†	31526‡	0.01	118250†	31418‡	0.01	117954†	31515‡	0.01
100 × 10	206722	**111060**	**0.01**	206754	112062‡	0.02	206922†	112477‡	0.02	206665	111257‡	**0.01**	206911†	111615‡	**0.01**	**206599**	111171‡	**0.01**
200 × 5	**250584**	60963	**0.01**	251555†	62203‡	0.02	252311†	62501‡	0.02	**250801**	60591	**0.01**	250752†	61125‡	**0.01**	250727†	**60931**	**0.01**
200 × 10	**454244**	222839	**0.01**	463468†	222741	0.02	464622†	222943‡	0.02	454349†	222525	**0.01**	458805†	**221619**	**0.01**	456859†	221771	**0.01**
300 × 5	409509	**151898**	**0.00**	410600†	151912‡	0.01	410584†	151904	**0.00**	**408713**	**151898**	0.01	408809	151957‡	**0.00**	409397	**151898**	**0.00**
300 × 10	672018	332151	**0.00**	676960†	333748‡	0.01	675292†	333098‡	0.01	**670769**	332326‡	**0.00**	672350†	**332126**	**0.00**	673020†	**332126**	**0.00**
800 × 10	**1702741**	986171	**0.00**	1722703†	1002024‡	0.01	1720849†	1002705‡	0.01	1715497†	990196‡	0.01	1712496†	984012	**0.00**	1712830†	**983570**	**0.00**

Table 8
Specific energy consumption values in all cases when CPU = 9.

Instance		IGS	IGA [17]	IGRS [46]	IGT [46]	IGTALL [46]	VBIH [46]	GA [29]	DABC [30]	EMBO [31]	DPSO [34]
20 × 5	TI_01	**10117**	**10117**	**10117**	**10117**	**10117**	10144	10220	**10117**	**10117**	11751
	TI_02	7885	7895	7936	7941	7906	7895	7918	8004	**7870**	8581
	TI_03	**10428**	10438	**10428**	**10428**	10437	**10428**	**10428**	10437	10438	11268
	TI_04	**8514**	8581	8532	8558	8539	8539	8538	8593	8552	9436
	TI_05	**8357**	8467	8361	8503	8397	8389	8423	8459	8419	8746
	TI_06	**16495**	16535	**16495**	**16495**	**16495**	**16495**	**16495**	16526	**16495**	18042
	TI_07	**12228**	**12228**	**12228**	**12228**	**12228**	**12228**	**12228**	**12228**	**12228**	12310
	TI_08	**12245**	**12245**	**12245**	**12245**	12256	**12245**	**12245**	**12245**	**12245**	12354
	TI_09	**9617**	9642	9623	9642	**9617**	9642	9623	9690	9710	10493
	TI_10	**9875**	9982	9903	**9875**	9962	9903	9883	9926	10069	10685
AVG MIN		**10576.1**	10613	10586.8	10603.2	10595.4	10590.8	10600.1	10622.5	10614.3	11366.6
	7885	7895	7936	7941	7906	7895	7918	8004	**7870**	8581	
20 × 10	TI_11	**22953**	**22953**	22982	**22953**	22970	**22953**	22990	22959	22961	24114
	TI_12	**29001**	29015	29037	29037	29081	**29001**	29015	29001	29015	30008
	TI_13	**25412**	**25412**	**25412**	**25412**	**25412**	**25412**	**25412**	**25412**	25447	28219
	TI_14	**28612**	28634	28634	28634	28704	**28612**	28642	28761	**28612**	28930
	TI_15	**30622**	**30622**	**30622**	**30622**	30647	**30622**	30829	**30662**	**30622**	33583
	TI_16	**27325**	27348	27348	27331	27414	27331	27336	27331	**27325**	28029
	TI_17	**28031**	**28031**	**28031**	28326	**28031**	28188	28623	28492	28577	29615
	TI_18	**29670**	29738	29710	**29670**	29707	29681	29708	29774	29548	29977
	TI_19	**26079**	26437	**26079**	**26079**	**26079**	**26079**	26178	26162	**26079**	27576
	TI_20	**31262**	31263	31284	31263	**31262**	31339	31429	31263	31297	33278
AVG MIN		**27896.7**	27945.3	27913.9	27932.7	27930.7	27921.8	28016.2	27981.7	27948.3	29332.9
	22953	**22953**	22982	**22953**	22970	**22953**	22990	22959	22961	24114	
40 × 5	TI_21	13154	13559	13419	13274	13616	13404	13235	14098	**13043**	15092
	TI_22	**17803**	18169	18226	18010	18271	18237	18382	18965	17983	19329
	TI_23	**17023**	17127	17206	17096	17206	17072	17198	17564	17188	17625
	TI_24	**29408**	29409	29508	29404	29472	29444	29552	29893	29497	30000
	TI_25	**15299**	15698	15718	15785	15701	15399	15752	16745	15638	17744
	TI_26	20514	20569	20839	20994	20655	20553	**20331**	21712	20993	22828
	TI_27	**16436**	16812	16732	16964	16983	16858	16540	17897	16452	18708
	TI_28	16908	17031	17133	**16863**	17086	17032	17179	17620	16971	17659
	TI_29	**19229**	19450	19656	19561	19439	19431	19539	19599	19372	20907
	TI_30	**19718**	20262	20340	20509	20024	20039	19965	21667	20154	23388
AVG MIN		**18549.2**	18808.6	18877.7	18846	18845.3	18746.9	18767.3	19576	18729.1	20328
		13154	13559	13419	13274	13616	13404	13235	14098	**13043**	15092
40 × 10	TI_31	**55613**	57457	56414	56398	56141	56444	56839	59407	56553	61190
	TI_32	**56597**	56626	56717	56622	56647	56647	56688	57027	56662	58091
	TI_33	**69681**	69960	69784	69845	69990	69864	69851	70377	69744	71301
	TI_34	51037	51040	51673	51462	50920	**50849**	51686	53448	51331	53128
	TI_35	**68881**	69489	69251	69383	69528	68940	69175	70310	69261	70806
	TI_36	**51557**	51704	51886	51811	51835	51692	52079	52505	51998	53325
	TI_37	**52700**	53086	53052	52950	53064	52884	53681	55548	53117	54117
	TI_38	**64394**	65004	65296	64831	64997	64802	64700	67238	64845	68406
	TI_39	**47190**	47550	47888	47975	47731	47256	47879	49161	47734	52763
	TI_40	**50889**	**50889**	**50889**	**50889**	**50889**	**50889**	**50889**	50891	**50889**	51338
AVG MIN		**56853.9**	57280.5	57285	57216.6	57174.2	57026.7	57346.7	58591.2	57213.4	59446.5
		47190	47550	47888	47975	47731	47256	47879	49161	47734	51338
60 × 5	TI_41	40042	40013	40172	40075	40075	**40000**	40756	41200	40126	41633
	TI_42	**33428**	33592	33741	33639	33770	33589	33810	34268	33521	34338
	TI_43	**52348**	**52348**	52379	**52348**	52361	**52348**	**52348**	52361	**52348**	52741
	TI_44	**55251**	55646	55697	55310	55399	55411	55497	56394	55941	57848
	TI_45	**42588**	42894	43081	43045	42924	42844	43402	44215	42890	44219
	TI_46	**45684**	**45684**	**45684**	**45684**	45785	**45684**	**45684**	45710	**45684**	45785
	TI_47	35919	36338	36616	**35915**	36274	36208	36237	37039	36187	37791
	TI_48	**41336**	41593	41738	41745	41444	41612	41847	42254	41763	42929
	TI_49	**36211**	36233	36233	36245	36268	36233	36384	36929	36223	36412
	TI_50	49603	**49548**	49648	49613	49737	49608	49648	49862	49673	49853
AVG MIN		**43241**	43388.9	43498.9	43361.9	43403.7	43353.7	43561.3	44023.2	43435.6	44354.9
		33428	33592	33741	33639	33770	33589	33810	34268	33521	34338
60 × 10	TI_51	71151	71102	**70969**	71239	71422	71104	71759	72959	71404	72700
	TI_52	**99695**	99725	99816	99737	99803	99743	99995	101169	99788	100623
	TI_53	**103316**	104946	104489	103861	104561	104040	103952	109553	104933	113750
	TI_54	**60012**	60645	60647	60953	60730	60537	60996	63267	60579	62015
	TI_55	**80878**	81954	82083	81473	81538	81670	82994	85943	82058	84313
	TI_56	**85484**	85794	85996	85840	85922	85806	86746	87626	85822	87408
	TI_57	**70509**	71414	71908	70874	71197	70767	71630	75291	71592	74100
	TI_58	**106568**	107006	107106	106877	107216	106749	106847	107934	107210	110015
	TI_59	**82540**	85113	85073	84638	83633	83617	83414	88150	83989	91605
	TI_60	**85164**	86166	86394	85797	86361	85543	85625	89809	85875	89219
AVG MIN		**84531.7**	85386.5	85448.1	85128.9	85238.3	84957.6	85395.8	88170.1	85325	88574.8
		60012	60645	60647	60953	60730	60537	60996	63267	60579	62015
80 × 5	TI_61	**70661**	70738	70830	70711	70839	70724	70980	71629	70813	71595
	TI_62	**45028**	45054	45191	45091	45078	44991	45568	46102	45128	45441
	TI_63	**57313**	**57313**	57316	**57313**	57335	57316	**57313**	57395	**57313**	57694

(*continued on next page*)

Table 8 (*continued*)

Instance		IGS	IGA [17]	IGRS [46]	IGT [46]	IGTALL [46]	VBIH [46]	GA [29]	DABC [30]	EMBO [31]	DPSO [34]
	TI_64	**40327**	40935	41279	40697	40779	40591	41348	42633	40613	42853
	TI_65	**38815**	38881	38925	38909	38988	38904	39132	39692	38830	39673
	TI_66	33555	33615	33781	33597	33605	**33511**	34185	35245	33725	34543
	TI_67	**115771**	116051	116028	115926	115968	115898	115928	117568	116002	119820
	TI_68	**27413**	27651	27822	27468	27638	27554	27775	28573	27615	28410
	TI_69	**55873**	55875	55881	55875	55883	55879	55879	55904	55879	56147
	TI_70	**67531**	68361	68447	67792	67808	67813	68835	69614	67862	69744
AVG MIN		**55228.7**	55447.4	55550	55337.9	55392.1	55318.1	55694.3	56435.5	55378	56592
		27413	27651	27822	27468	27638	27554	27775	28573	27615	28410
80 × 10	TI_71	**112163**	113225	113013	**112251**	112629	113065	116354	122218	113575	115910
	TI_72	**81810**	82038	82463	82572	82184	81999	84065	87610	83022	83237
	TI_73	**120237**	120505	120540	120467	120600	120290	120892	122396	120545	121525
	TI_74	101502	103466	103143	102778	101336	**101067**	105582	110037	102488	108122
	TI_75	161877	161997	162287	161937	162104	**161842**	163467	165911	162382	165378
	TI_76	108975	110154	111450	110254	109709	**108935**	113325	119788	109985	118857
	TI_77	**138131**	138912	139685	139035	139362	138823	139995	141476	138991	142040
	TI_78	**125764**	127082	126974	126691	126044	126273	129610	132348	127095	130229
	TI_79	**131407**	132469	133814	132929	132674	132121	134378	136994	132786	137668
	TI_80	**105934**	108457	109789	107457	108872	107696	110357	113015	107084	112857
AVG MIN		**118780**	119830.5	120315.8	119637.1	119551.4	119211.1	121802.5	125179.3	119795.3	123582.3
		81810	82038	82463	82572	82184	81999	84065	87610	83022	83237
100 × 5	TI_81	**54677**	56067	56171	55184	55471	55295	56456	57784	55236	57756
	TI_82	**63387**	**63387**	**63387**	**63387**	63392	**63387**	**63387**	63515	63391	63627
	TI_83	**100022**	100904	101067	100650	100519	100167	100511	104211	100223	102478
	TI_84	**117693**	118071	118205	117894	118250	117915	118011	119089	118034	119504
	TI_85	**42107**	42842	42552	42250	42216	42360	43794	45468	42828	43882
	TI_86	**67067**	**67067**	67080	**67067**	67135	**67067**	**67067**	67184	67071	67386
	TI_87	**55149**	**55149**	55316	**55149**	55221	**55149**	55208	55439	55150	55814
	TI_88	**64941**	65003	65140	64976	65028	64912	65985	67528	65322	65704
	TI_89	52041	52385	52365	52034	**52033**	51949	53004	54466	52320	53378
	TI_90	**31292**	31861	31671	31526	31418	31515	32026	34339	31583	33702
AVG MIN		**64837.6**	65273.6	65295.4	65011.7	65068.3	64971.6	65544.9	66902.3	65115.8	66323.1
		31292	31861	31671	31526	31418	31515	32026	34339	31583	33702
100 × 10	TI_91	**137236**	139228	139791	139154	138666	138486	144569	152704	139734	144033
	TI_92	**113604**	114593	115167	114516	114527	113940	119579	129511	116460	118345
	TI_93	**144155**	144886	145143	144814	144713	144225	146893	148797	145779	146870
	TI_94	189754	192520	193719	190252	189398	**188359**	197253	215329	192647	203076
	TI_95	**123281**	123834	125088	123929	123419	123370	128736	138449	125819	128332
	TI_96	206616	206746	206904	206665	206911	**206599**	207532	208557	207457	208177
	TI_97	**140814**	142406	143020	141704	142016	141921	144932	150141	142591	148722
	TI_98	111434	111926	112477	111257	111615	**111171**	114239	118392	112117	113537
	TI_99	**141974**	142674	142840	142545	142213	142107	143432	146429	143366	144991
	TI_100	**179439**	186433	186153	182769	180943	180591	188024	196485	183687	190974
AVG MIN		**148830.7**	150524.6	151030.2	149760.5	149442.1	149076.9	153518.9	160479.4	150965.7	154705.7
		111434	111926	112477	111257	111615	**111171**	114239	118392	112117	113537
200 × 5	TI_101	239966	240263	240298	239951	240116	**239879**	241768	242920	240498	243084
	TI_102	210021	210142	210776	210129	210070	**209992**	211287	212279	210509	212273
	TI_103	**75246**	76440	76910	75561	75694	75629	82158	84007	78007	79172
	TI_104	**83844**	89333	90444	84991	85321	84393	91777	94771	87884	93200
	TI_105	**134537**	134564	134539	**134537**	134554	**134537**	**134537**	134701	**134537**	134669
	TI_106	173586	174301	174214	173067	**172907**	172916	178020	183296	175325	176649
	TI_107	250796	251555	252023	250801	250752	**250727**	259591	270593	251290	253619
	TI_108	**60768**	62184	62290	**60591**	61125	60931	64495	67420	62533	63627
	TI_109	136066	136757	137748	136266	136128	**136032**	143038	149421	137741	139801
	TI_110	67612	69905	69770	67621	67708	**67433**	72327	76381	69906	72442
AVG MIN		**143244.2**	144544.4	144901.2	143351.5	143437.5	143246.9	147899.8	151578.9	144823	146853.6
		60768	62184	62290	**60591**	61125	60931	64495	67420	62533	63627
200 × 10	TI_111	333482	335635	335746	335325	**333286**	333475	345734	346737	341512	338600
	TI_112	**267224**	270757	270730	268472	267881	267884	282025	287715	277159	273841
	TI_113	**325940**	331710	332726	328092	327785	327761	339414	354789	334960	337238
	TI_114	247599	248109	248667	**247401**	247571	247530	257608	261465	252855	251389
	TI_115	235915	236629	236781	237120	235968	**235380**	249417	258515	241645	241079
	TI_116	**452701**	463305	464622	454349	458805	456859	474372	482249	461946	472748
	TI_117	344453	346278	347349	345302	**344121**	344338	359109	365208	351955	353608
	TI_118	222724	222681	222943	222525	**221619**	221771	235221	238120	230684	224800
	TI_119	333274	338963	338574	334614	334450	**332799**	371818	409231	352913	354514
	TI_120	254805	260794	265544	254037	255466	**253883**	276408	298306	263380	272447
AVG MIN		**301811.7**	305486.1	306368.2	302723.7	302695.2	302168	319112.6	330233.5	310900.9	312026.4
		222724	222681	222943	222525	**221619**	221771	235221	238120	230684	224800
300 × 5	TI_121	**153656**	**153656**	153705	**153656**	153669	153673	154270	154920	153658	153792
	TI_122	409445	410560	410584	**408673**	408809	409397	413372	414947	411100	413836
	TI_123	263192	264533	264248	263215	**262928**	263343	275249	278539	267367	267762
	TI_124	183077	183247	183351	**183079**	183107	182876	190298	191004	185138	183998
	TI_125	**175054**	**175054**	**175054**	**175054**	**175054**	**175054**	175210	175528	**175054**	175163
	TI_126	**151898**	151900	151904	**151898**	151957	**151898**	152067	152387	151899	151957

(*continued on next page*)

Algorithm 1
Basic iterated greedy algorithm.

01: **Begin**
02: Set parameters$d$, termination criterion, job sequence,$\pi=(\pi_1,\pi_2,...,\pi_J)$.
03: $\pi=NEH(\pi)$
04: $\pi=IterativeImprove(\pi)$
05: $\pi^{best}=\pi$
06: **While** (termination criterion is not satisfied) **do**
07: $\pi^{origin}=\pi$
08: $\pi^{Remove}=Destruction(\pi^{origin},d),\pi^{origin}=\pi^{origin}\backslash\pi^{Remove}$
09: $\pi^{origin}=Construction(\pi^{origin},\pi^{Remove})$
10: $\pi^{temp}=IterativeImprove(\pi^{origin})$
11: $\pi=AcceptanceCriterion(\pi^{temp},\pi^{best},\pi)$
12: **Endwhile**
13: **Return**π^{best}
14: **End**

Algorithm 2
Proposed improved iterated greedy-swap algorithm.

Input:d, termination criterion, Sequence$\pi=(\pi_1,\pi_2,...,\pi_J)$
Output:π^{best}
01: **Begin**
02: $\pi=MME(\pi)$ /*heuristic algorithm which we used*/
03: $\pi^{best}=\pi$
04: **While** (termination criterion is not satisfied) **do**
05: $\pi^{origin}=\pi$
06: $\pi^{origin},\pi^{Remove}=Destruction(\pi^{origin},d)$
07: $\pi^{origin}=Construction(\pi^{origin},\pi^{Remove})$
08: $\pi^{temp}=\pi^{origin}$
09: **If** (π^{temp} better thanπ^{best})
10: $\pi^{best}=\pi^{temp}$
11: **EndIf**
12: ***The following parts represent the main innovative contributions of this paper***
13: $\pi^{temp}=LocalPerturbationStrategy(\pi^{temp})$ /*based on swap strategy*/
14: **If** (π^{temp} better than π)
15: $\pi=\pi^{temp}$
16: **If** (π better than π^{best})
17: $\pi^{best}=\pi$
18: **EndIf**
19: **EndIf**
20: **Else**
21: $\pi^{temp}=GlobalPerturbationStrategy(\pi^{temp})$ /*based on Half Swap strategy*/
22: $\pi=\pi^{temp}$
23: **If** (π better thanπ^{best})
24: $\pi^{best}=\pi$
25: **EndIf**
25: **EndIf**
26: **Endwhile**
27: **End**

Algorithm 3
Destruction and construction phase.

Input:$\pi=(\pi_1,\pi_2,...,\pi_J)$, parameter$d$
Output:π^{origin}
01: **Begin**
02: $count=1$, $\pi^{Remove}=\Phi$, $\pi^{origin}=\pi$
03: **While** ($count<=d$) **do**
04: $num=rand()\%J+1$
05: **If** ($\pi^{origin}_{num}$is not selected)
06: Extract$\pi^{origin}_{num}$from$\pi^{origin}$and add toπ^{Remove}
07: $\pi^{origin}=\pi^{origin}\backslash\{\pi^{origin}_{num}\}$,$count++$
08: **EndIf**
09: **Endwhile**
10: **For** $j=1$ **to**d
11: $\pi_j=\pi_j^{Remove}$
12: **For** $p=1$ **to** $J-d$ // p is the insertion position
13: insert$\pi_j$in the p*th* position of π^{origin}
14: calculate the energy consumption value, Min_{EC}
15: **EndFor**
16: select position p with minimalMin_{EC}, and insertπ_j into the p*th* ofπ^{origin}
17: **EndFor**
18: **End**

Algorithm 5
IG half-swap

Input:π^{temp}
Output:π^{temp}
01: **Begin**
02: Sequence$\pi^{temp}$is divided equally into subsequencesπ^{Front}andπ^{Back}
03: **If** (The energy consumption of$\pi^{Back}$is lower thanπ^{Front})
04: $\pi^{Front\_temp}=\pi^{Front}$ /* improve π^{Front}*/
05: **For** $p=1$ **to** $Sizeof(\pi^{Front\_temp})$ /*select these jobs in order */
06: **For** $j=1$ **to** $Sizeof(\pi^{Front\_temp})$ /* swap with other jobs*/
07: **If** ($\pi_p^{Front_temp}$is not equal to $\pi_j^{Front\_temp}$)
08: Swap $\pi_p^{Front_temp}$with$\pi_j^{Front_temp}$
09: **EndIf**
10: **If** ($\pi^{Front_temp}$is better thanπ^{Front})
11: $\pi^{Front}=\pi^{Front\_temp}$
12: Mergeπ^{Front}andπ^{Back}, denoting the new sequence asπ^{temp_new}
13: **If** ($\pi^{temp_new}$is better thanπ^{temp})
14: $\pi^{temp}=\pi^{temp\_new}$
15: **EndIf**
16: **EndIf**
17: **EndFor**
18: **EndFor**
19: **ElseIf** (The energy consumption of π^{Front} is lower thanπ^{Back})
20: Similar to the steps performed above.
21: **EndIf**
22: **End**

curation, Writing – original draft, Writing – review & editing. **Quan-Ke Pan:** Data curation, Writing – original draft, Writing – review & editing. **Dun-Wei Gong:** Data curation, Writing – original draft, Writing – review & editing.

Acknowledgments

This work was jointly supported by the National Natural Science Foundation of China under Grant Nos. 61803192, 61973203, 61966012, 61773192, 61603169, 61773246, and 71533001. We are grateful for Youth Innovation Talent Introduction and Education Program support received from Shandong province colleges and universities.

Appendix. xxx

Table 8.

References

[1] J.J. Wang, L. Wang, A knowledge-based cooperative algorithm for energy-efficient scheduling of distributed flow-shop, IEEE Trans. Syst. Man Cybern. Syst. (2018) 1–15.
[2] V. Fernandez-Viagas, R. Ruiz, J.M. Framinan, A new vision of approximate methods for the permutation flow shop to minimize makespan: State-of-the-art and computational evaluation, Eur. J. Oper. Res. 257 (3) (2016).
[3] K. Peng, Q.K. Pan, L. Gao, et al., An improved artificial bee colony algorithm for real-world hybrid flow shop rescheduling in steelmaking-refining continuous casting process, Comput. Ind. Eng. 122 (8) (2018) 235–250.
[4] H.S. Choi, J.S. Kim, D.H. Lee, Real-time scheduling for reentrant hybrid flow shops: a decision tree based mechanism and its application to a TFT-LCD line, Expert Syst. Appl. 38 (4) (2011) 3514–3521.
[5] Y.D. Kim, S.O. Shim, B. Choi, et al., Simplification methods for accelerating simulation-based real-time scheduling in a semiconductor wafer fabrication facility, IEEE Trans. Semicond. Manuf. 16 (2) (2003) 290–298.
[6] S. Jun, J. Park, A hybrid genetic algorithm for the hybrid flow shop scheduling problem with nighttime work and simultaneous work constraints: a case study from the transformer industry, Expert Syst. Appl. 42 (15-16) (2015) 6196–6204.
[7] G. Zhang, K. Xing, F. Cao, Discrete differential evolution algorithm for distributed blocking flow shop scheduling with makespan criterion, Eng. Appl. Artif. Intell. 76 (11) (2018) 96–107.
[8] S. Schulz, J.S. Neufeld, U. Buscher, A multi-objective iterated local search algorithm for comprehensive energy-aware hybrid flow shop scheduling, J. Clean. Prod. 224 (7) (2019) 421–434.
[9] B. Rolf, T. Reggelin, A. Nahhas, et al., Assigning dispatching rules using a genetic algorithm to solve a hybrid flow shop scheduling problem, Procedia Manuf. 42 (2020) 442–449.
[10] D. Gong, Y. Han, J. Sun, A novel hybrid multi-objective artificial bee colony algorithm for blocking lot-streaming flow shop scheduling problems, Knowl. Based Syst. 148 (MAY15) (2018) 115–130.

[11] Y. Han, D. Gong, Y. Jin, et al., Evolutionary multiobjective blocking lot-streaming flow shop scheduling with machine breakdowns, IEEE Trans. Cybern. 49 (1) (2019) 184–197.
[12] M.F. Tasgetiren, D. Kizilay, Q.K. Pan, et al., Iterated greedy algorithms for the blocking flow shop scheduling problem with makespan criterion, Comput. Oper. Res. 77 (1) (2017) 111–126.
[13] I. Ribas, R. Companys, X. Tort-Martorell, An iterated greedy algorithm for solving the total tardiness parallel blocking flow shop scheduling problem, Expert Syst. Appl. 121 (2019) 347–361.
[14] Z. Shao, D. Pi, W. Shao, Hybrid enhanced discrete fruit fly optimization algorithm for scheduling blocking flow-shop in distributed environment, Expert Syst. Appl. 145 (2019) 113147.
[15] Z. Shao, D. Pi, W. Shao, et al., An efficient discrete invasive weed optimization for blocking flow shop scheduling problem, Eng. Appl. Artif. Intell. 78 (2) (2019) 124–141.
[16] F. Zhao, F. Xue, Y. Zhang, et al., A discrete gravitational search algorithm for the blocking flow shop problem with total flow time minimization, Appl. Intell. 49 (2) (2019).
[17] R. Ruiz, T. Stützle, A simple and effective iterated greedy algorithm for the permutation flow shop scheduling problem, Eur. J. Oper. Res. 177 (3) (2007) 2033–2049.
[18] I. Ribas, R. Companys, X. Tort-Martorell, An iterated greedy algorithm for the flow shop scheduling problem with blocking, Omega 39 (3) (2011) 293–301.
[19] N. Nouri, T. Ladhari, An efficient iterated greedy algorithm for the makespan blocking flow shop scheduling problem, Polibits 53 (53) (2016) 91–95.
[20] D.P. Ronconi, A note on constructive heuristics for the flowshop problem with blocking, Int. J. Prod. Econ. 87 (1) (2004) 39–48.
[21] S. Toumi, B. Jarboui, M. Eddaly, et al., Optimizing blocking flow shop scheduling problem with total completion time criterion, Comput. Oper. Res. 40 (7) (2013) 1874–1883.
[22] M.M. Nawaz, E. Enscore, J.I. Ham, A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, Omega 11 (1) (1983) 91–95.
[23] S.T. McCormick, M.L. Pinedo, S. Shenker, B. Wolf, Sequencing in an assembly line with blocking to minimize cycle time, Oper. Res. 37 (6) (1989) 925–935.
[24] S. Wang, M. Liu, C. Chu, A branch-and-bound algorithm for two-stage no-wait hybrid flow-shop scheduling, Int. J. Prod. Res. 53 (4) (2015) 1143–1167.
[25] H.M.H. FattahiP, T.M.R. JolaiF, A branch and bound algorithm for hybrid flow shop scheduling problem with setup time and assembly operations, Appl. Math. Model. 38 (1) (2014) 119–134.
[26] L.X. Tang, H. Xuan, Lagrangian relaxation algorithms for real-time hybrid flow shop scheduling with finite intermediate buffers, J. Oper. Res. Soc. 57 (3) (2006) 316–324.
[27] L. Tang, H. Xuan, J. Liu, A new Lagrangian relaxation algorithm for hybrid flow shop scheduling to minimize total weighted completion time, Comput. Oper. Res. 33 (11) (2006) 3344–3359.
[28] X.R. Tao, J.Q. Li, T.H. Huang, et al., Discrete imperialist competitive algorithm for the resource-constrained hybrid flow shop problem with energy consumption, Complex Intell. Syst. (2020) 1–16.
[29] M. Nejati, I. Mahdavi, R. Hassanzadeh, et al., Multi-job lot streaming to minimize the weighted completion time in a hybrid flow shop scheduling problem with work shift constraint, Int. J. Adv. Manuf. Technol. 70 (1-4) (2014) 501–514.
[30] Q.K. Pan, L. Wang, J.Q. Li, et al., A novel discrete artificial bee colony algorithm for the hybrid flow shop scheduling problem with makespan minimisation, Omega 45 (2014) 42–56 (jun.).
[31] B. Zhang, Q.K. Pan, L. Gao, et al., An effective modified migrating birds optimization for hybrid flow shop scheduling problem with lot streaming, Appl. Soft Comput. 52 (2017) 14–27.
[32] J.Q. Li, Q.K. Pan, F.T. Wang, A hybrid variable neighborhood search for solving the hybrid flow shop scheduling problem, Appl. Soft Comput. J. 24 (2014) 63–77.
[33] S. Liu, J. Pei, H. Cheng, et al., Two-stage hybrid flow shop scheduling on parallel batching machines considering a job-dependent deteriorating effect and non-identical job sizes, Appl. Soft Comput. 84 (2019) 105701.
[34] M.K. Marichelvam, M. Geetha, Tosun, An improved particle swarm optimization algorithm to solve hybrid flow shop scheduling problems with the effect of human factors - a case study, Comput. Oper. Res. 114 (2019) 104812.
[35] Z. Zhang, L. Wu, T. Peng, et al., An improved scheduling approach for minimizing total energy consumption and makespan in a flexible job shop environment, Sustainability 11 (1) (2018).
[36] D. Lei, M. Li, L. Wang, A two-phase meta-heuristic for multiobjective flexible job shop scheduling problem with total energy consumption threshold, IEEE Trans. Cybern. 49 (3) (2019) 1097–1109.
[37] J.Y. Ding, R. Chiong, et al., An improved iterated greedy algorithm with a Tabu-based reconstruction strategy for the no-wait flow shop scheduling problem, Appl. Soft Comput. 30 (2015) 604–613.
[38] J.P. Huang, Q.K. Pan, L. Gao, An effective iterated greedy method for the distributed permutation flow shop scheduling problem with sequence-dependent setup times, Swarm Evolut. Comput. (2020) 100742.
[39] V. Fernandez-Viagas, J.M.S. Valente, J.M. Framinan, Iterated-greedy-based algorithms with beam search initialization for the permutation flow shop to minimise total tardiness, Expert Syst. Appl. (2017) S0957417417307327.
[40] H. Ochi, O.B. Driss, Scheduling the distributed assembly flow shop problem to minimize the makespan, Procedia Comput. Sci. 164 (2019) 471–477.
[41] Y.Y. Huang, Q.K. Pan, J.P. Huang, et al., An improved iterated greedy algorithm for the distributed assembly permutation flow shop scheduling problem, Comput. Ind. Eng. 152 (2020) 107021.
[42] K.C. Ying, An iterated greedy heuristic for multistage hybrid flow shop scheduling problems with multiprocessor tasks, J. Oper. Res. Soc. 60 (6) (2009) 810–817.
[43] F.J. Rodriguez, M. Lozano, C. Blum, et al., An iterated greedy algorithm for the large-scale unrelated parallel machines scheduling problem, Comput. Oper. Res. 40 (7) (2013) 1829–1841.
[44] W. Shao, Z. Shao, D. Pi, Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem, Knowl. Based Syst. (2020) 105527.
[45] A. Fbo, B. Ms, Iterated greedy algorithms enhanced by hyper-heuristic based learning for hybrid flexible flow shop scheduling problem with sequence dependent setup times: A case study at a manufacturing plant, Comput. Oper. Res. (2020) 125.
[46] H. Ztop, M.F. Tasgetiren, D. Türsel Eliiyi, et al., Metaheuristic algorithms for the hybrid flow shop scheduling problem, Comput. Oper. Res. 111 (2019) 177–196.
[47] E. Osaba, E. Villar-Rodriguez, J. Del Ser, A.J. Nebro, D. Molina, A. LaTorre, P.N. Suganthan, C.A. Coello Coello, F. Molina, A tutorial on the design, experimentation and application of metaheuristic algorithms to real-world optimization problems, Swarm Evolut. Comput. 100888 (2021) 1–21.