Knowledge-Based Systems 200 (2020) 106032

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

An improved Jaya algorithm for solving the flexible job shop scheduling problem with transportation and setup times

Jun-qing Li [a,b,*], Jia-wen Deng [a], Cheng-you Li [a], Yu-yan Han [a], Jie Tian [b], Biao Zhang [a], Cun-gang Wang [a]

[a] College of Computer Science, Liaocheng University, Liaocheng 252059, PR China
[b] School of Information Science and Engineering, Shandong Normal University, Jinan, 25014, PR China

ARTICLE INFO

Article history:
Received 3 January 2020
Received in revised form 11 April 2020
Accepted 13 May 2020
Available online 18 May 2020

Keywords:
Flexible job shop
Improved Jaya algorithm
Energy consumption
Transportation time
Setup time

ABSTRACT

Flexible job shop scheduling has been widely researched due to its application in many types of fields. However, constraints including setup time and transportation time should be considered simultaneously among the realistic requirements. Moreover, the energy consumptions during the machine processing and staying at the idle time should also be taken into account for green production. To address this issue, first, we modeled the problem by utilizing an integer programming method, wherein the energy consumption and makespan objectives are optimized simultaneously. Afterward, an improved Jaya (IJaya) algorithm was proposed to solve the problem. In the proposed algorithm, each solution is represented by a two-dimensional vector. Consequently, several problem-specific local search operators are developed to perform exploitation tasks. To enhance the exploration ability, a SA-based heuristic is embedded in the algorithm. Meanwhile, to verify the performance of the proposed IJaya algorithm, 30 instances with different scales were generated and used for simulation tests. Six efficient algorithms were selected for detailed comparisons. The simulation results confirmed that the proposed algorithm can solve the considered problem with high efficiency.

1. Introduction

In realistic industrial production systems, scheduling problem has been considered as the key issue for production efficiency. The flexible job shop scheduling problem (FJSP) [1], as a complex version of the scheduling problem, has been investigated for many years. In realistic production systems such as the steelmaking system, a FJSP problem can be modeled in several phases. Fig. 1 gives the illustration of the typical steelmaking system, where there are three main phases, i.e., the molten iron scheduling, the steelmaking casting, and the hot-rolling phases. In each phase, charges or jobs have different routes, and the machine assignment for processing these charges is also flexible for industrial constrains. However, the flexibility makes the problem more complex than the canonical scheduling problem.

In a classical FJSP, there are n jobs to be processed on m machines. Each job has n_i ($i = 1, 2,\ldots, n$) number of operations. Each operation should select one available machine from a set of candidate machines, such that each operation can be processed on only one machine at a time, and one machine can process only one operation. Preemption is not permitted; that is, the machine cannot be occupied until the assigned operation is completed. Consequently, the addition to select the available machine causes the FJSP to be more difficult compared to the flowshop, hybrid flowshop, and job shop scheduling problems [1,2]. To solve the FJSPs, many researchers have utilized different types of algorithms. Kacem et al. applied the genetic algorithm (GA), where the solution quality is enhanced by genetic manipulations [2]. Jensen considered the robust or flexible solutions by using the GA algorithm [3]. Ho et al. proposed a learnable genetic architecture by considering the evolution and learning interaction [4]. Combining the features of the GA and the variable neighborhood descent algorithm, Gao designed a hybrid algorithm for the problem [5]. Zhang et al. designed a hybrid algorithm based on the particle swarm optimization algorithm for the multi-objective FJSPs [6]. The other efficient algorithms for the FJSPs include the knowledge-based ant colony optimization (ACO) algorithm [7], the artificial bee colony (ABC) algorithm [8,9], the memetic algorithm [10], the harmony search algorithm [11], and the particle swarm optimization (PSO) [12]. In addition to the population-based algorithm, the local search-based algorithm also exhibits an efficient performance for solving the FJSPs. Cruz-Chávez et al. presented an accelerated simulated annealing (ASA) algorithm with a partial scheduling mechanism [13]. Aqel et al. proposed

* Corresponding author at: College of Computer Science, Liaocheng University, Liaocheng 252059, PR China.
E-mail address: lijunqing@lcu-cs.com (J.-q. Li).

https://doi.org/10.1016/j.knosys.2020.106032
0950-7051/

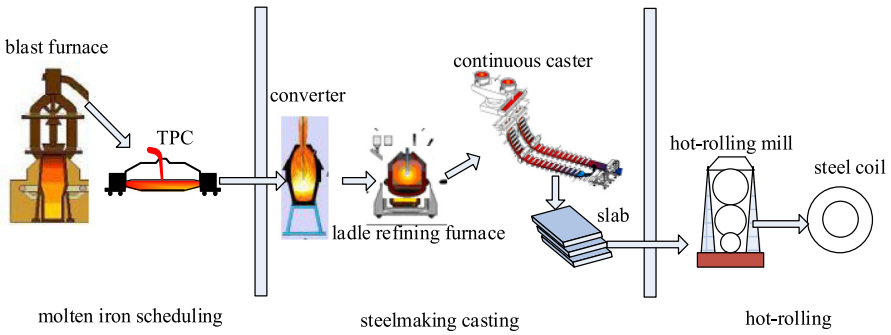


Fig. 1. Scheduling problem in a typical steelmaking system.

a modified iterated greedy (IG) algorithm, which includes two phases, and each phase is used to solve a subproblem of the FJSP [14]. From the outlined literature on FJSPs, it is evident that many meta-heuristics have been utilized to solve the complex optimization scheduling problem. However, most of the current literature have not considered the realistic constraints of the problem.

The most practical realistic constrains in the scheduling problem include machine breakdown, fuzzy processing time, new job arrival, resource constraint, transportation time, and the operation sequence setup time. The machine breakdown constraint has been researched for other typical scheduling problems, such as flowshop and job shop. For the FJSPs, Li et al. designed a discrete ABC algorithm with maintenance activities [15]. Ahmadi et al. considered FJSPs under random machine breakdown by evolutionary algorithms [16]. For the fuzzy processing time constraint, Xu et al. designed an effective teaching–learning-based optimization (TLBO) algorithm [17]. Jamrus et al. developed a hybrid PSO combined with genetic operators for semiconductor manufacturing [18]. For the new job arrival constraints, Gao et al. utilized a discrete Jaya (DJaya) algorithm to optimize a multi-objective FJSPs [19]. Gao and Pan investigated a shuffled multi-swarm micro-migrating birds optimizer for a multi-resource-constrained FJSP [20].

For the transportation time constraint, Zhang et al. developed a hybrid algorithm combining GA with tabu search (TS) procedure for flexible job shop scheduling with transportation constraints and bounded processing times [21]. Dai et al. utilized a multi-objective optimization for energy-efficient FJSP with transportation constraints [22]. Liu et al. designed an integrated genetic algorithm and glowworm swarm optimization (GA-GSO) algorithm for the FJSPs of the crane transportation process [23]. For the setup time constraint, Shen et al. designed a TS algorithm with specific neighborhood functions and a diversification structure [24]. Rossi et al. studied the FJSP with routing flexibility and separable setup times utilizing the ACO algorithm [25]. On the other hand, the energy-efficient objective has also been studied for the scheduling problem. Lei et al. studied the FJSPs with energy consumption minimization, employing a two-phase meta-heuristic (TPM) method, based on the imperialist competitive algorithm and the variable neighborhood search (VNS) [26]. Wang et al. considered a two-stage optimization method for energy-saving FJSPs [27]. Mokhtari and Hasani minimized the total energy cost of both production and maintenance operations in the FJSP by using the evolutionary algorithm [28]. Wu and Sun developed a green scheduling heuristic [29]. The mathematical models for the energy-aware FJSPs are investigated in [30,31]. In a nutshell, the FJSP is a complex optimization problem, and realistic constraints will make it more complex; however, currently there are only a few publications which consider these realistic constraints simultaneously.

Very recently, the Jaya algorithm was proposed by Rao to solve the constrained and unconstrained optimization problems [32]. Ever since, it has been applied to solve engineering optimization [33], maximum power point tracking [34], thermal devices optimization problem [35], multi-area interconnected power system [36], truss structure layout optimization problem [37], power quality improvement problem [38], and parameters identification [39,40]. Compared to other optimization algorithms such as ABC [41–43], IG [44,45] and Chemical-reaction optimization (CRO) algorithm [46], the Jaya algorithm is efficient and easy to be implemented. Its first step is to form a random initial population; afterward, the best and worst solutions are extracted from the initial population and are employed to update other solutions for the next iteration. The best and worst solutions have to be updated at each iteration. In the course of optimization, only the population size and the maximum number of function evaluations are predefined.

From the analysis of the state art of the typical constraints in FJSP, it can be observed that the transportation time and setup time constraints exist in many types of realistic industrial application, such as the steelmaking casting, and the hot-rolling in steel process system. However, currently there are only a few publications which consider one or two realistic constraints simultaneously. Based on the efficiency performance of the Jaya algorithm and the problem features of the realistic FJSPs, we propose an improved Jaya (IJaya) that targets the transportation and setup time FJSPs. The main contributions of this study are as follows:

- The FJSP is defined wherein the transportation and sequence-based setup time constraints are considered.
- The energy consumption and makespan objectives are optimized simultaneously.
- Seven different types of local search approaches are proposed to enhance the exploitation abilities.
- SA-based acceptation criterion is embedded in the proposed algorithm to enhance the exploration abilities.

The rest of this paper is organized as follows. Section 2 briefly describes the problem. Next, Section 3 illustrates the framework of the canonical Jaya algorithm. The proposed algorithm, which embeds the problem-specific heuristics, is presented in Section 4. Section 5 illustrates the experimental results and makes comparisons to the performance results of other algorithms from the existing literature to demonstrate the superiority of the proposed algorithm. Finally, the last section presents the concluding remarks and future research directions.

2. Problem description

2.1. Problem description

We consider the realistic production procedure in a typical steelmaking industry, which commonly has five stages or processes. The first process is to load the molten iron into the torpedo car (TPC), and then the TPC should be processed through the second and third refining stages. The fourth stage is to dump the molten iron, and the last stage is the continuous casting process.

2.2. Problem formulation

In this study, we consider a FJSP with a sequence-based setup time and transportation constraints; the detailed assumptions are given as follows:

(1) *Assumptions*

- The predefined processing time of a positive integer and the operation is deterministic.
- All the machines are available at time zero and remain continuously available during the entire production.
- The transportation time for each job from one machine to the consecutive machine is taken into account.
- The effect of the setup time for each pair of consecutive processing operations on the same machine is considered.
- The processing of one job should be started after the completion time of the predecessor job while considering the setup time on the same machine.
- The processing of each operation should be started after the completion time of the predecessor operation of the same job while considering the transportation time between the two processing machines.
- Sufficient buffers are always available between any two continuous machines.

(2) *Notation*

Indices and parameters

- n: Number of jobs.
- m: Number of machines.
- g: Number of operations.
- j, h: Index of jobs.
- i, k: Index of the machines.
- u, z: Index of operations.
- r: Index of processing positions on the machine.
- n_j: Number of operations belonging to job j.
- $O_{j,u}$: The uth operation of job j.
- $p_{j,u,i}$: The processing time of the uth operation of job j on the machine i.
- $t_{j,k,i}$: Transportation time of job j from machine k to i.
- $s_{j,h,i}$: Setup time of job j and h on the machine i.
- $e_{j,u,i}$: A binary value which is set to 1 if $O_{j,u}$ can be processed on the machine i.
- E^i_{busy}: The unit energy consumption for the processing task of the machine i.
- E^i_{idle}: The unit energy consumption the machine i staying at the idle state [47].
- E_{busy}: Total energy consumptions of the machine processing.
- E_{idle}: Total energy consumptions of machine staying at the idle state.
- L: A large positive number.
- ω: The weight value for the two objective values.

Decision variables:

- $x_{j,u,i,r}$: A binary value that is set to 1 if $O_{j,u}$ is processed at the rth position of the machine i; otherwise, $x_{j,u,i,r}$ is set to 0.
- $y_{j,u,i,k}$: A binary value that is set to 1 if $O_{j,u}$ is processed on machine i and $O_{j,u-1}$ machine k; otherwise, $y_{j,u,i,k}$ is set to 0.
- $C_{j,u}$: The completion time of $O_{j,u}$.
- $MC_{i,r}$: The completion time of the operation processing at the rth position of the machine i.

The main mathematical model for the considered problem is given as follows:

$$\min\ \omega \times C_{\max} + (1-\omega) \times (E_{busy} + E_{idle})$$

$$C_{\max} \geq C_{j,n_j} \qquad j \in \{1, 2, \ldots, n\} \tag{1}$$

$$E_{busy} = \sum_{i=1}^{m} (E^i_{busy} \times \sum_{j=1}^{n} \sum_{u=1}^{n_j} \sum_{k=1}^{m} (y_{j,u,i,k} \cdot p_{j,u,i})) \tag{2}$$

$$E_{idle} = \sum_{i=1}^{m} (E^i_{idle} \times (\sum_{j=1}^{n} \sum_{k=1}^{m} y_{j,n_j,i,k} \times C_{j,n_j} - \sum_{j=1}^{n} \sum_{k=1}^{m} y_{j,1,i,k} \times C_{j,1} - p_{j,1,i} - \sum_{j=1}^{n} \sum_{u=1}^{n_j} \sum_{k=1}^{m} y_{j,u,i,k} \times p_{j,u,i})) \tag{3}$$

$$\sum_{i=1}^{m} \sum_{k=1}^{m} y_{j,u,i,k} = 1 \qquad j \in \{1, 2, \ldots, n\};\ u \in \{2, \ldots, n_j\} \tag{4}$$

$$\sum_{i=1}^{m} y_{j,1,i,0} = 1 \qquad j \in \{1, 2, \ldots, n\} \tag{5}$$

$$\sum_{k=1}^{m} y_{j,u,i,k} \leq e_{j,u,i} \qquad j \in \{1, 2, \ldots, n\};\ u \in \{2, \ldots, n_j\};\ i \in \{1, 2, \ldots, m\} \tag{6}$$

$$y_{j,1,i,0} \leq e_{j,1,i} \qquad j \in \{1, 2, \ldots, n\};\ i \in \{1, 2, \ldots, m\} \tag{7}$$

$$y_{j,u,i,k} \leq \sum_{f=1}^{m} y_{j,u-1,k,f} \qquad j \in \{1, 2, \ldots, n\};\ u \in \{3, \ldots, n_j\};\ i \in \{1, 2, \ldots, m\};\ k \in \{1, 2, \ldots, m\} \tag{8}$$

$$y_{j,2,i,k} \leq y_{j,1,k,0}, \qquad j \in \{1, 2, \ldots, n\};\ i \in \{1, 2, \ldots, m\};\ k \in \{1, 2, \ldots, m\} \tag{9}$$

$$\sum_{i=1}^{m} \sum_{r=1}^{g} x_{j,u,i,r} = 1, \qquad j \in \{1, 2, \ldots, n\};\ u \in \{1, 2, \ldots, n_j\} \tag{10}$$

$$\sum_{j=1}^{n} \sum_{u=1}^{n_j} x_{j,u,i,r} \leq 1 \qquad i \in \{1, 2, \ldots, m\};\ r \in \{1, 2, \ldots, g\} \tag{11}$$

$$MC_{i,r} \leq C_{j,u} + L \cdot (1 - x_{j,u,i,r}) \qquad j \in \{1, 2, \ldots, n\};\ u \in \{1, 2, \ldots, n_j\};\ i \in \{1, 2, \ldots, m\};\ r \in \{1, 2, \ldots, g\} \tag{12}$$

$$C_{j,u} \leq MC_{i,r} + L \cdot (1 - x_{j,u,i,r}) \qquad j \in \{1, 2, \ldots, n\};\ u \in \{1, 2, \ldots, n_j\};\ i \in \{1, 2, \ldots, m\};\ r \in \{1, 2, \ldots, g\} \tag{13}$$

$$C_{j,u} + t_{j,k,i} \leq C_{j,u+1} - p_{j,u,i} + L \cdot (1 - y_{j,u,i,k}) \qquad j \in \{1, 2, \ldots, n\};\ u \in \{1, 2, \ldots, n_j - 1\};\ i \in \{1, 2, \ldots, m\};\ k \in \{1, 2, \ldots, m\} \tag{14}$$

$$\sum_{j=1}^{n} \sum_{u=1}^{n_j} x_{j,u,i,r} \geq \sum_{h=1}^{n} \sum_{z=1}^{n_h} x_{h,z,i,r+1}\ i \in \{1, 2, \ldots, m\};\ r \in \{1, 2, \ldots, g-1\} \tag{15}$$

$$MC_{i,r} + \sum_{h=1,h\neq j}^{n}\sum_{z=1}^{n_h} s_{h,j,i}\cdot x_{h,z,i,r} \le MC_{i,r+1} - p_{j,u,i} + L\cdot(1-x_{j,u,i,r+1}) \quad (16)$$
$$j\in\{1,2,\ldots,n\};\ u\in\{1,2,\ldots,n_j\};\ i\in\{1,2,\ldots,m\};\ r\in\{1,2,\ldots,g-1\}$$

$$MC_{i,r} \ge \sum_{j=1}^{n}\sum_{u=1}^{n_j} p_{j,u,i}\cdot x_{j,u,i,r} \qquad i\in\{1,2,\ldots,m\};\ r\in\{1,2,\ldots,g\} \quad (17)$$

$$x_{j,u,i,r}, y_{j,u,i,k}\in\{0,1\} \quad (18)$$

The objective is to minimize the weighted sum of two objectives, i.e., the maximum completion time of all operations and the sum of energy consumptions, including the machine processing energy and the energy of machine staying at the idle state.

Constraints (4) and (5) ensure that each operation of each job should be assigned to only one available machine. Constraints (6) and (7) guarantee that the assigned machine for each operation must be selected from the given eligible set of machines. Constraints (8) and (9) enforce the processing relation of each operation between the current machine and the immediate successive machine. Constraint (10) ensures that each operation can select only one processing positions on the candidate machines. Constraint (11) enforces that each processing position on any machine can be occupied by only one operation at a time, that is, it is not permitted to assign more than one operation to one processing position at a time. Constraints (12) and (13) create the relationship between the completion times of the operations and the assigned machine positions. Constraint (14) ensures that the start time of the current operation must be greater than the sum of the following values, i.e., the completion time of the predecessor operation, and the transportation time between the consecutive machines. Constraint (15) enforces that the processing positions on each machine should be assigned from left to right, that is, the machine processing position skip is not permitted. Constraint (16) ensures that start time of the current operation must be greater than the sum of the following values, i.e., the completion time of the operation at the predecessor processing position on the same machine, and the setup time between the two jobs. Constraint (17) guarantees that the completion time of each operation should not be less than its processing time. Constraint (18) enforces the range of the decision variables.

2.3. Problem illustration

To aid comprehension of our target problem, an example of FJSPs is selected where there are three jobs to be processed on three machines. Table 1 gives the processing times of each operation on each machine. Tables 2 and 3 list the transportation times and the sequence-based setup times, respectively. Fig. 2 gives the corresponding Gantt chart, where the processing time for each job is represented by a rectangle labeled with the job number and the operation number. For example, on machine M_3, the first operation to be processed is the $O_{1,1}$, and the last operation to be processed is $O_{2,3}$.

The setup time between two jobs is represented by a rectangle filled with green colors. For example, on machine M_3, the setup time between $O_{1,2}$ and $O_{2,2}$ is labeled with "1-2", which illustrates the setup time between the two consecutive jobs J_1 and J_2. The transportation time is represented by a rectangle with blue colors. For example, after completing the first operation $O_{2,1}$, the job J_2 should be transported from M_2 to M_3, which is illustrated by a rectangle labeled with "2" between M_2 and M_3.

Table 1
The processing times of each operation on each machine.

Jobs	Operations	Machines		
		M_1	M_2	M_3
Job 1	$O_{1,1}$	11	16	17
	$O_{1,2}$	15	9	6
	$O_{1,3}$	18	13	–
Job 2	$O_{2,1}$	–	11	15
	$O_{2,2}$	16	–	19
	$O_{2,3}$	15	12	13
Job 3	$O_{3,1}$	15	–	23
	$O_{3,2}$	–	9	9
	$O_{3,3}$	13	12	–

Table 2
The transportation times between the machines.

Machines	Machines		
	M_1	M_2	M_3
M_1	0	12	20
M_2	12	0	8
M_3	20	8	0

Table 3
The setup times between the operations on each machine.

Machines	Jobs	Jobs		
		J_1	J_2	J_3
M_1	J_1	0	6	12
	J_2	9	0	5
	J_3	14	9	0
M_2	J_1	0	13	13
	J_2	7	0	9
	J_3	10	10	0
M_3	J_1	0	6	12
	J_2	6	0	6
	J_3	10	7	0

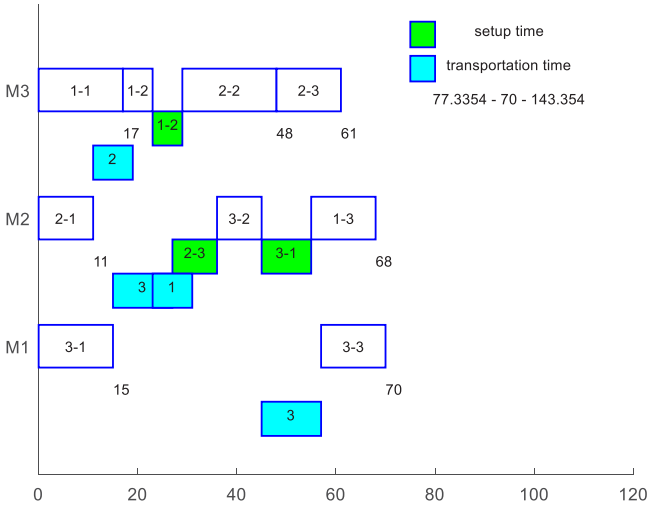


Fig. 2. The Gantt chart of the FJSP with the transportation times and setup times. . (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3. The canonical Jaya algorithm

The Jaya algorithm, which can be seen as an extension of the TLBO algorithm, was proposed by Rao et al. in 2016. In the canonical Jaya algorithm, the detailed steps are as follows:

Step 1. The initialization phase: Randomly generates an initial population.

Step 2. Perform the steps 3–4 until the stop condition is satisfied.

Step 3. Improve each solution by using the following formulation:

$$x_{j,k,t}{}' = x_{j,k.t} + rand_{1,j,t} \times (x_{j,best,t} - |x_{j,k.t}|) - rand_{2,j,t} \times (x_{j,worst,t} - |x_{j,k.t}|) \quad (19)$$

where $x_{j,k.t}$ is the jth variable for the kth solution during the tth iteration; x_{best} and x_{worst} are the best and worst solutions found so far; $rand_{1,j,t}$ and $rand_{2,j,t}$ are the uniformly distributed random number in the range [0,1], $|x_{j,k.t}|$ is the absolute value of the solution $x_{j,k.t}$. The component $+rand_{1,j,t} \times (x_{j,best,t} - |x_{j,k.t}|)$ let each solution learn from the best solution, while the second component $-rand_{2,j,t} \times (x_{j,worst,t} - |x_{j,k.t}|)$ let the solution far from the worst solution.

Step 4. Evaluate $x_{j,k,t}{}'$, and replace $x_{j,k.t}$ if $x_{j,k,t}{}'$ is better than $x_{j,k.t}$.

Step 5. Output the best solution.

4. The proposed algorithm

In this section, a stepwise procedure is given for the implementation of the IJaya algorithm. A detailed description is provided in Algorithm 1.

4.1. Solution representation and decoding strategy

To record the processing sequence and the machine assignment information, each solution is represented by two vectors, i.e., the routing vector and the scheduling vector. The scheduling vector reports the processing sequence for each operation, where each operation is represented by the job number, while the routing vector is employed to record the machine assignment for the corresponding positions of operations. It should be noted that each job j will appear n_j times in each of the two vectors, and the length of the two vectors are the same and equal to $\sum_{j=1}^{n} n_j$.

Fig. 3 gives a solution representation example, where there are three jobs and three machines. The total number of operations of these three jobs is {3,3,3}, respectively. The scheduling vector indicates that the first to be scheduled is the first operation of J_1 and then the first operation of J_2, and the last one is the last operation of J_3. The routing vector indicates the assigned machine for the corresponding position of operations. For example, the first operation of J_1 is assigned to the machine M_3, and the last operation of J_3 is scheduled on M_1.

The decoding procedure for each solution should complete the following tasks: (1) decide the starting time for each operation taking into account the completion time of the previous operation and the machine's idle time, while the transportation time and the setup time should also be considered, and (2) compute the makespan and the energy consumptions, i.e., the objective values. Fig. 2 shows the Gantt chart for the solution representation defined in Fig. 3. The makespan of the solution is 70, and the total energy consumption is 143.354.

4.2. Initialization strategy

To initialize a group of solutions, this study utilizes a simple and efficient way. The time complexity of the initialization strategy is $O(nm)$. The detailed steps are as follows:

Step 1. While the initial population size is less than P_{size}, perform the following steps.

Step 2. For the scheduling vector: (1) Initialize an empty scheduling vector, and add the number for each job j n_j times, and (2) rearrange the sequence for all the numbers in the scheduling vector in a random way.

Step 3. For the routing vector: (1) Initialize an empty routing vector, and (2) select each operation $O_{i,j}$ in the scheduling vector, randomly select one available machine $M_{i,j}$ for $O_{i,j}$, and store $M_{i,j}$ in the routing vector.

4.3. Local search for the routing vector

In this study, we propose a simple but efficient local search method for the routing vector, which utilized seven types of local search methods randomly to address our target problem features and objectives. Firstly, the function to obtain all critical operations is described in Algorithm 2, afterward, the proposed seven types of local search heuristics are described in LS1 to LS7 respectively. It can be observed that: (1) the time complexity of Algorithm 2 is $O(n^2m)$; and (2) the time complexity of the seven local search approaches are $O(n^2m)$, respectively.

Algorithm 2: Get all critical operations

Step 1. Find all the operations with the completion time equal to the makespan, and store them into a vector named $CO_{\max}$.
Step 2. Perform the following steps until $CO_{\max}$ is empty.
Step 3. Fetch and delete the first operation $O_{i,j}$ in $CO_{\max}$, and perform the following steps 4–5.
Step 4. Let $C_{h,k}$ be the completion time of the immediate predecessor operation $O_{h,k}$ on the same machine as $O_{i,j}$, and $S_{i,j}$ be the starting time of $O_{i,j}$. If $S_{i,j} = C_{h,k}$, then store the operation $O_{h,k}$ in $CO_{\max}$.
Step 5. If $S_{i,j} = C_{i,j-1}$, then store the operation $O_{i,j-1}$ in $CO_{\max}$.

LS1: Local search for critical operations

Step1. For a solution X, find all the critical operations by using Algorithm 2.
Step2. Randomly select one critical operation r_c,
Step2.1 If it has more than one candidate machine, then randomly change another machine for it.
Step2.2 Otherwise, loop until a previous critical operation r_{cc}, which has more than one candidate machines is found.
Step2.3 Randomly change another machine for r_{cc}.
Step 3. Replace the current solution if a better value has been obtained.

LS2: Local search for a random critical operation

Step1. For a solution X, find all the critical operations by using Algorithm 2.
Step2. Randomly select one critical operation r_c
Step 3. If r_c has more than one candidate machines, then randomly change another machine for it.
Step 4. Evaluate the newly generated neighboring solution, and replace the current solution if the latter is better.

LS3: Local search for a random critical machine

Step1. For a solution X, find all the critical operations.
Step2. Randomly select one critical operation r_c, such that the machine assigned for r_c is M_c.
Step 3. Randomly select one operation r_{cc} on M_c. If r_{cc} has more than one candidate machine, then randomly change another machine for it.
Step 4. Evaluate the newly generated neighboring solution, and replace the current solution.

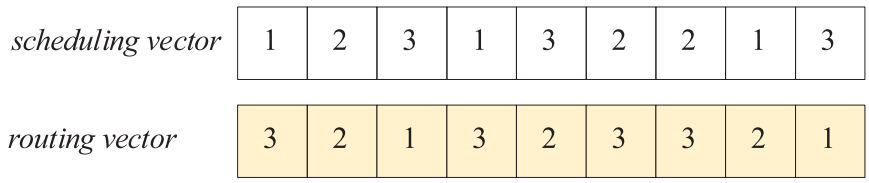


Fig. 3. Encoding of an example.

Algorithm 1: General Framework of IJAYA

Input: system parameters, including the predefined population size P_{size} and the stop criterion,
Output: the best solution obtained thus far

```
Generate the initial population by using the initialize strategy (c.f. Subsection 4.2)
while stop criterion is not satisfied do
    Routing local search phase
    for i ←N do
        Let s_i be the i^th solution in the current population; s_b is the best solution obtained thus far.
        Randomly select one solution from s_i and s_b as the current solution s_c.
        If s_c ≺ s_i then
            Replace s_i with s_c
        else
            Perform the local search strategy (c.f. Subsection 4.3)
            Evaluate the newly-generated solution s_c'.
            Update the best solution s_b with s_c' if the latter is better than the former.
        end
    end
    Scheduling local search phase
        for i=1 to 4/n do
            Perform the three types of mutation heuristics in a random way for the best individual obtained
              thus far (cf. Subsection 4.4).
            Update the best individual by using the newly generated neighboring solution
        end
    end
    Exploration phase: Perform the SA-based acceptance criterion (cf. Subsection 4.5)
    Output the best solution found.
end
```

LS4: Local search for a random critical machine

Step 1. For a solution X, find all the critical operations.
Step 2. Randomly select one critical operation r_c, such that the machine assigned for r_c is M_c.
Step 3. Randomly select one operation r_{cc} on M_c. If r_{cc} has more than one candidate machine, then randomly change another machine for it.
Step 4. Evaluate the newly generated neighboring solution, and replace the current solution if the latter is better.

LS5: Local search for a busiest machine

Step 1. For a solution X, compute workloads for all machines
Step 2. Select machine M with the maximum workload.
Step 3. Randomly select an operation $O_{i,j}$ on machine M. If $O_{i,j}$ has more than one candidate machine, then randomly change another machine for it.
Step 4. Evaluate the newly generated neighboring solution, and replace the current solution.

LS6: Local search for a random operation

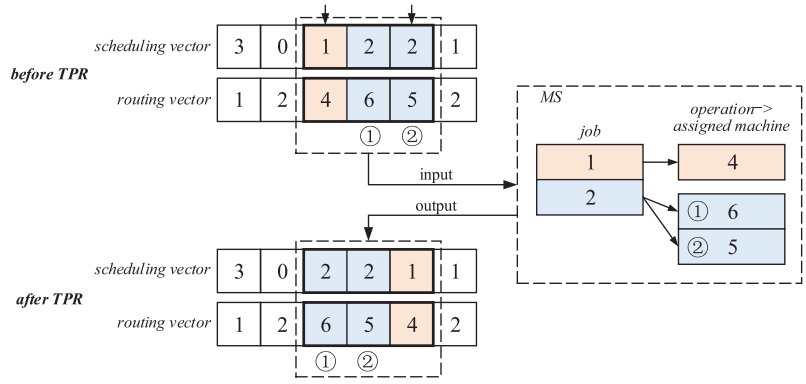
Step 1. For a solution X, randomly select one operation $O_{i,j}$ on the scheduling vector.
Step 2. If $O_{i,j}$ has more than one candidate machine, then randomly change another machine for it.
Step 3. Evaluate the newly generated neighboring solution, and replace the current solution if the latter is better.

LS7: Local search for a random operation with minimum processing machine

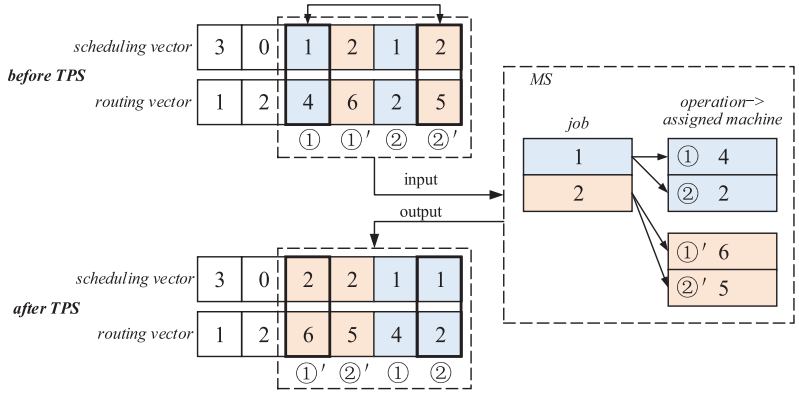
Step 1. For a solution X, randomly select one operation $O_{i,j}$ on the scheduling vector.
Step 2. If $O_{i,j}$ has more than one candidate machine, then change the machine with the minimum processing time for it.
Step 3. Evaluate the newly generated neighboring solution, and replace the current solution if the latter is better.

4.4. Local search for the scheduling vector

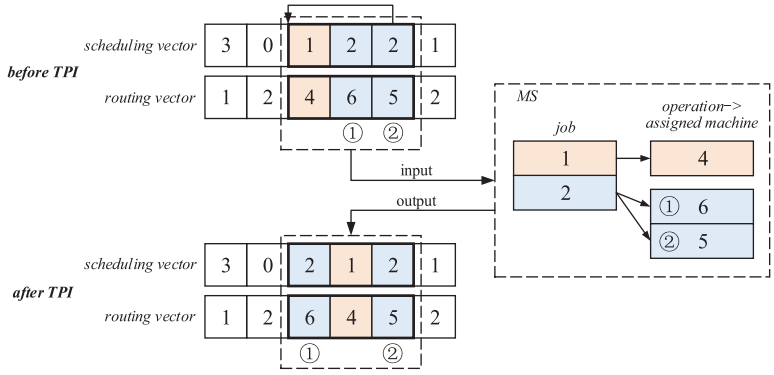
For the considered problem, we include the following three types of mutation operators:



(a) TPR operator



(b) TPS operator



(c) TPI operator

Fig. 4. Mutation operators.

(1) The two-point reverse (TPR) operator

The TPR operator is to generate a neighboring solution by reversing a selected segment, i.e., to reverse all of the elements between two randomly selected positions in the scheduling vector. Fig. 4(a) shows the procedure of the TPR operator.

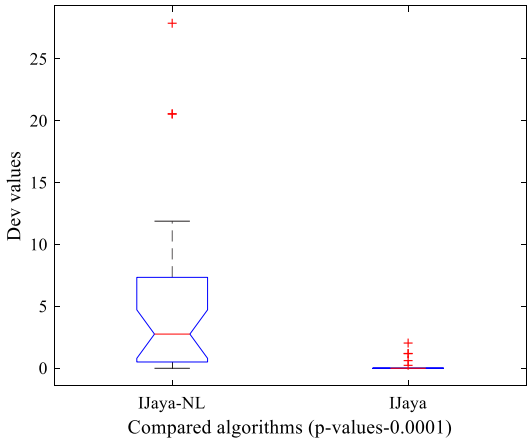
(2) The two-point swap (TPS) operator

The TPS operator is to generate a neighboring solution by swapping two selected jobs. Fig. 4(b) shows the procedure of the TPS operator.

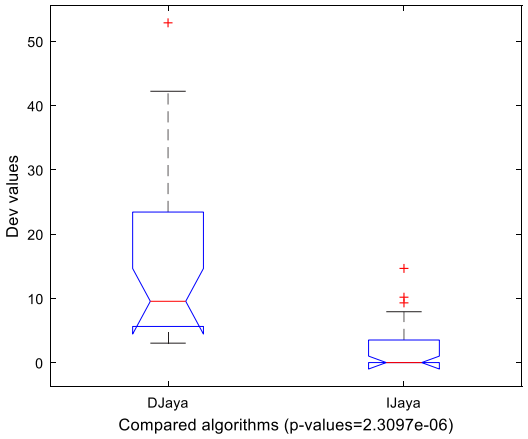
(3) Two-point insertion (TPI) operator

The TPI operator is to generate a neighboring solution by inserting one job before the position of another selected job. Fig. 4(c) shows the procedure of the TPI operator.

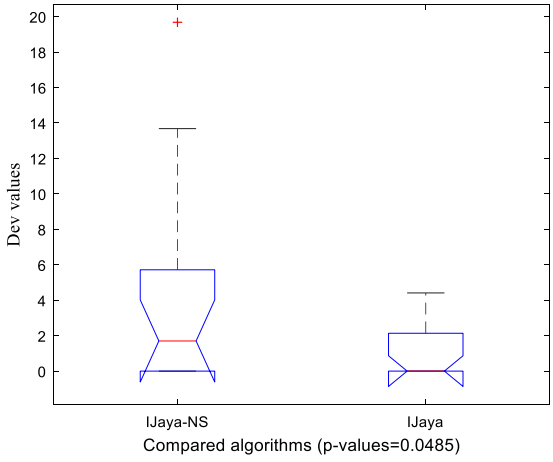
After the mutation of the scheduling vector, the other task is to apply mutation in the machine assignment vector as well. Here we investigate a simple method, which initially retains the assigned machines for all of the affected operations, as shown in Fig. 4(a)–(c), then, randomly selects a position in the machine assignment vector, and replaces another available machine for the corresponding operation. It can be easily observed that the time complexity is $O(n)$ for TPR, TPS, and TPI, respectively.



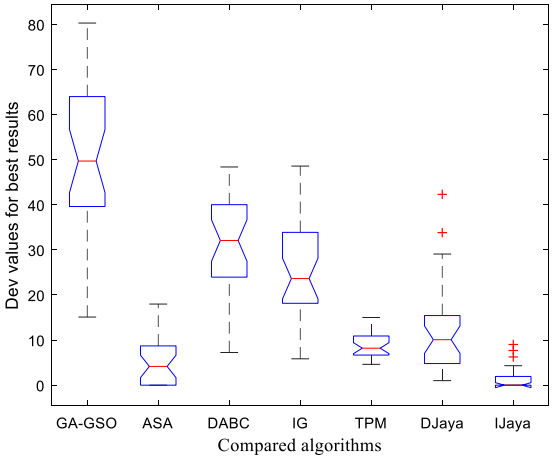
(a) LSD for local search heuristic



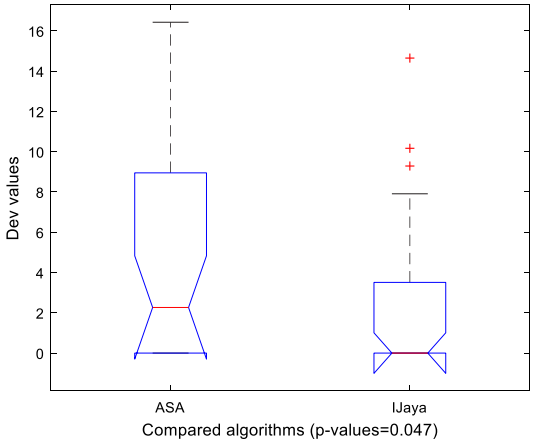
(d) LSD for IJaya and DJaya



(b) LSD for exploration heuristic

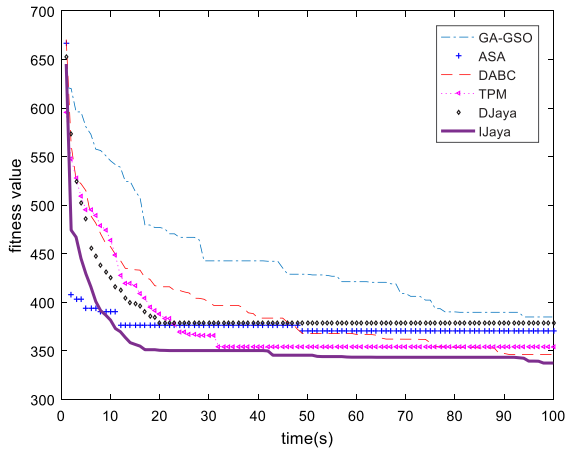


(e) LSD for different algorithms

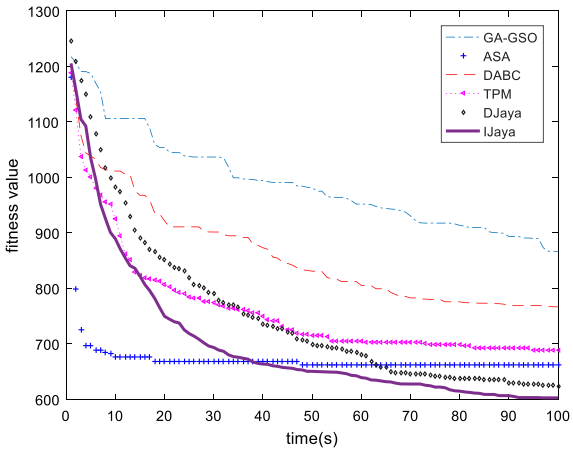


(c) LSD for IJaya and ASA

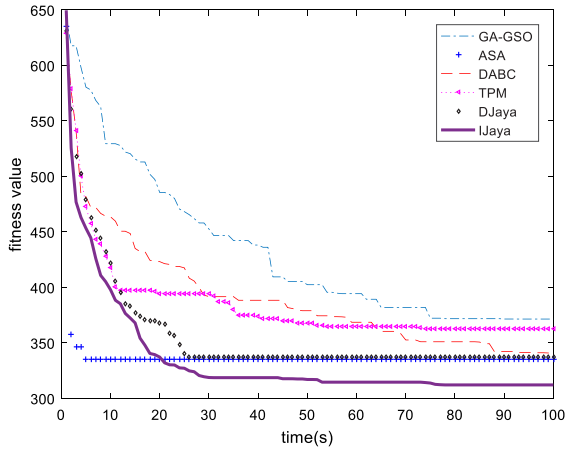
Fig. 5. ANOVA comparison results.



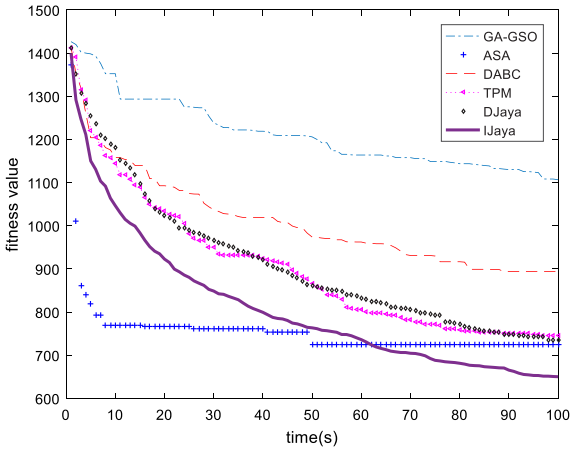
(a) Convergence curve for Inst1



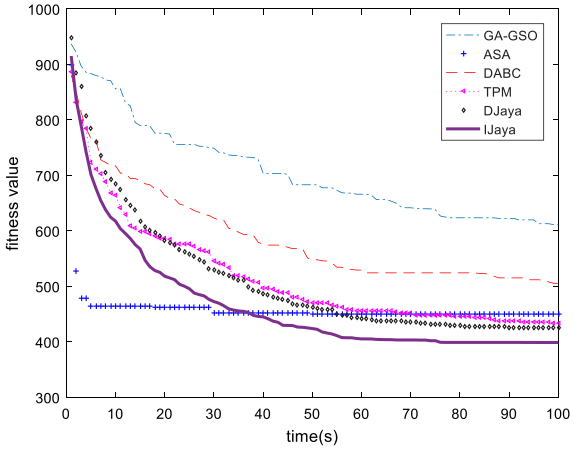
(d) Convergence curve for Inst8



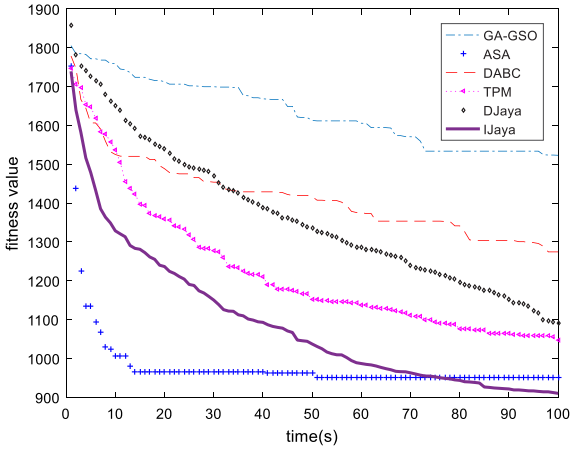
(b) Convergence curve for Inst2



(e) Convergence curve for Inst10



(c) Convergence curve for Inst5



(f) Convergence curve for Inst15

Fig. 6. Convergence curves for different types of instances.

4.5. Exploration approaches

Similar to Ruiz and Stutzle (2007) [45], in the proposed IJaya algorithm, the SA-based heuristic is also used for the acceptance criterion to enhance the algorithm with the ability to escape from the local optimal. In this study, we adopt a simple constant temperature acceptance, where the *Temperature* = $\frac{T\cdot\sum_{k=1}^{m}\sum_{j=1}^{n}\sum_{i=1}^{\theta_j}T_{i,j,k}}{n\cdot m\cdot 10}$ and T is a calibrated parameter. It can be easily observed that the time complexity of the SA-based exploration approach is $O(n^2m)$.

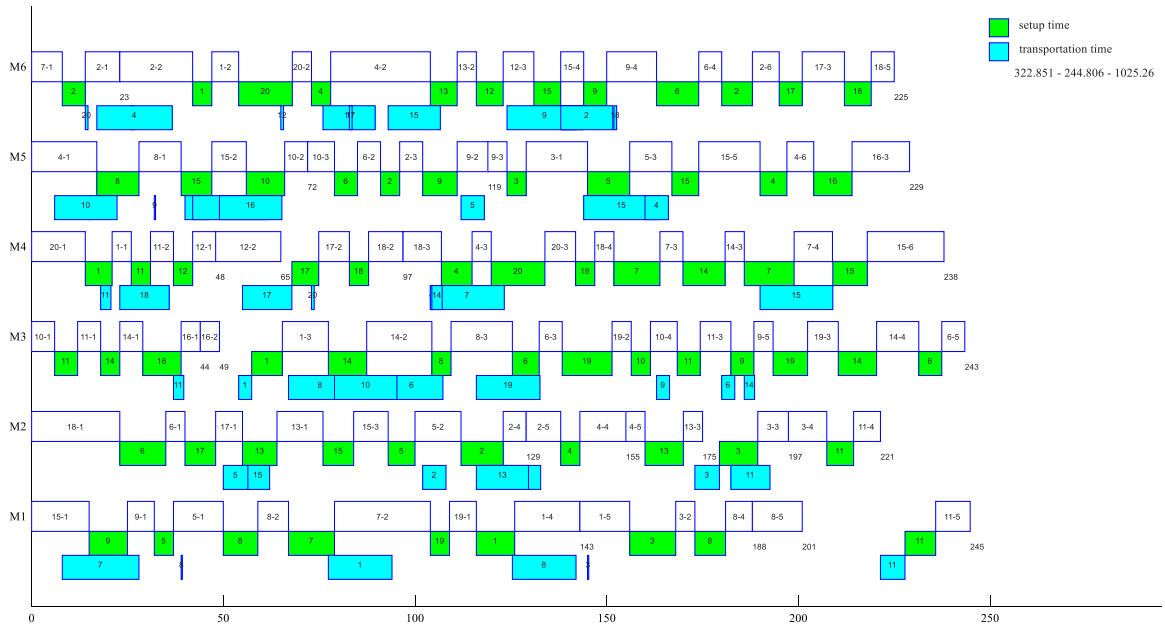


Fig. 7. A Gantt chart for the best solution for Inst1.

5. Numerical analysis

This section discusses the computational experiments used to evaluate the performance of the proposed algorithm. Our algorithm was implemented in C++ on an Intel Core *i*7 3.4-GHz PC with 16 GB of memory. It should be noted that 30 independent runs for each instance is commonly used to make fair comparisons in many references [12,19,41]. Therefore, to verify the effectiveness and efficiency of the proposed algorithm, after 30 independent runs, the resulting best solutions were collected for performance comparisons.

The compared algorithms include the GA-GSO (Liu et al. 2019) [23], ASA (Cruz-Chávez et al. 2017) [13], IG (Aqel et al. 2019) [14], DABC (Li et al. 2014) [15], TPM (Lei et al. 2018) [26], and DJaya (Gao et al. 2018) [19]. The main reasons to select the above compared algorithm are as follows: (1) the GA-GSO algorithm is designed for the similar problems in this study, except that the setup time is not considered in GA-GSO algorithm; and (2) the other five algorithms, i.e., ASA, IG, DABC, TPM, and DJaya are mainly designed for the FJSP without considering the crane transportation and setup time constraints. However, almost all the components of the six algorithms can be easily embedded to solve the considered problem. It should be noted there is few literature considering the FJSP with crane transportation and setup time constraints, and therefore, we select the above six algorithms to make fair comparisons. All of the compared algorithms were adjusted to adapt to solving the considered problem, where the parameters were defined according to the considered literature. The selected performance measure was the relative percentage increase (RPI), which is calculated as follows:

$$RPI\ (C) = \frac{f_c - f_b}{f_b} \times 100 \tag{20}$$

where f_b is the best fitness value collected by all the compared algorithms and f_c is the minimum fitness value found by a given algorithm.

5.1. Experimental instances

We test the performance of the proposed algorithms based on the realistic instances defined in (Liu et al. 2019 [23]). Thirty different scales of instances were randomly generated, where the number of jobs $n = \{20,30,40,50,80,100\}$ and the number of machines $m = \{6,7,8,9,10\}$. Furthermore, the total number of operations of each job is distributed uniformly in the interval $[m/2, m]$. The instances can be found on the website https://www.researchgate.net/publication/338108832_FJSP_setup_transportation.

5.2. Experimental parameters

The population size is the only predefined parameter of the proposed algorithm. Based on the existing literature and our experimental observation, 100 was selected.

5.3. Efficiency of the proposed local search for the routing vector

To investigate the effectiveness of the proposed local search heuristic discussed in Section 4.3, we implemented the two different types of IJaya algorithms, i.e., the IJaya-NL without the local search heuristic and the IJaya with all the components discussed in Section 4. All the other components of the two compared algorithms were left unchanged.

The comparison results are given in Table 4. In the comparison table, the first column contains the instance name entries, and the second column contains the instance scale entries. The two numbers in the instance scale entries represent the number of jobs and machines, respectively. For example, the first instance (Ins1) is a 20-job-6-machine problem, while the last instance (Ins30) is a 100-job-10-machine problem. The third column represents the best fitness value obtained out of the two compared algorithms, while the following two columns represent the fitness values obtained by the two compared algorithms, respectively. The last two columns display the deviation or RPI values by the two algorithms.

It can be observed from Table 4 that: (1) out of the 30 instances with different problem scales, IJaya obtained better values 25 times, although the values were slightly better than the IJaya-NL method in each case; (2) from the last two columns, the RPI values also verify the performance of the IJaya method, which apparently outperforms the IJaya-NL method; and (3) from the last row in the table, the average performance also verifies the superior performance of the IJaya method with an average RPI

Table 4
Comparison results for the proposed local search heuristic.

Instance	Scale	Best	Fitness		Dev	
			IJaya-NL	IJaya	IJaya-NL	IJaya
Ins1	20-6	322.851	322.85	340.147	5.36	**0.00**
Ins2	20-7	291.81	291.81	321.21	10.08	**0.00**
Ins3	20-8	367.59	367.59	381.09	3.67	**0.00**
Ins4	20-9	353.62	353.62	360.08	1.83	**0.00**
Ins5	20-10	390.22	390.22	395.39	1.32	**0.00**
Ins6	30-6	583.87	583.87	611.28	4.69	**0.00**
Ins7	30-7	440.04	440.04	459.31	4.38	**0.00**
Ins8	30-8	573.73	573.73	583.8	1.76	**0.00**
Ins9	30-9	510.18	508.97	508.97	**0.00**	0.24
Ins10	30-10	632.23	628.32	628.32	**0.00**	0.62
Ins11	40-6	724.99	724.99	758.51	4.62	**0.00**
Ins12	40-7	633.85	633.85	638.65	0.76	**0.00**
Ins13	40-8	772.51	772.51	775.55	0.39	**0.00**
Ins14	40-9	715.9	701.61	701.61	**0.00**	2.04
Ins15	40-10	886.79	876.47	876.47	**0.00**	1.18
Ins16	50-6	924.7	924.70	942.48	1.92	**0.00**
Ins17	50-7	855.97	855.97	860.32	0.51	**0.00**
Ins18	50-8	1020.11	1008.00	1008	**0.00**	1.20
Ins19	50-9	1053.62	1053.62	1055.35	0.16	**0.00**
Ins20	50-10	1133.09	1133.09	1160.92	2.46	**0.00**
Ins21	80-6	1429.33	1429.33	1466.05	2.57	**0.00**
Ins22	80-7	1215.75	1215.75	1271.66	4.60	**0.00**
Ins23	80-8	1691.4	1691.40	1815.72	7.35	**0.00**
Ins24	80-9	1506.52	1506.52	1646.71	9.31	**0.00**
Ins25	80-10	1882.84	1882.84	2082.72	10.62	**0.00**
Ins26	100-6	2072.19	2072.19	2133.55	2.96	**0.00**
Ins27	100-7	1648.95	1648.95	1844.95	11.89	**0.00**
Ins28	100-8	2029.09	2029.09	2446.77	20.58	**0.00**
Ins29	100-9	1850.17	1850.17	2230.16	20.54	**0.00**
Ins30	100-10	2299.46	2299.46	2940.69	27.89	**0.00**
Mean		1027.11	1025.72	1108.21	5.41	0.18

Table 5
Comparison results for the proposed SA-based exploration search heuristic.

Instance	Scale	Best	Fitness		Dev	
			IJaya-NS	IJaya	IJaya-NS	IJaya
Ins1	20-6	317.90	317.90	322.85	**0.00**	1.56
Ins2	20-7	291.81	295.42	291.81	1.24	**0.00**
Ins3	20-8	367.59	369.65	367.59	0.56	**0.00**
Ins4	20-9	353.62	357.37	353.62	1.06	**0.00**
Ins5	20-10	381.57	381.57	390.22	**0.00**	2.27
Ins6	30-6	575.54	575.54	583.87	**0.00**	1.45
Ins7	30-7	430.53	430.53	440.04	**0.00**	2.21
Ins8	30-8	558.19	558.19	573.73	**0.00**	2.78
Ins9	30-9	489.04	489.04	510.18	**0.00**	4.32
Ins10	30-10	620.75	620.75	632.23	**0.00**	1.85
Ins11	40-6	714.89	714.89	724.99	**0.00**	1.41
Ins12	40-7	623.37	623.37	633.85	**0.00**	1.68
Ins13	40-8	752.72	752.72	772.51	**0.00**	2.63
Ins14	40-9	708.60	708.60	715.90	**0.00**	1.03
Ins15	40-10	854.62	854.62	886.79	**0.00**	3.76
Ins16	50-6	924.70	932.45	924.70	0.84	**0.00**
Ins17	50-7	834.22	834.22	855.97	**0.00**	2.61
Ins18	50-8	977.01	977.01	1020.11	**0.00**	4.41
Ins19	50-9	1013.87	1013.87	1053.62	**0.00**	3.92
Ins20	50-10	1096.52	1096.52	1133.09	**0.00**	3.34
Ins21	80-6	1419.32	1419.32	1429.33	**0.00**	0.71
Ins22	80-7	1215.75	1218.42	1215.75	0.22	**0.00**
Ins23	80-8	1691.40	1727.95	1691.40	2.16	**0.00**
Ins24	80-9	1506.52	1560.81	1506.52	3.60	**0.00**
Ins25	80-10	1882.84	1992.20	1882.84	5.81	**0.00**
Ins26	100-6	2072.19	2107.39	2072.19	1.70	**0.00**
Ins27	100-7	1648.95	1738.42	1648.95	5.43	**0.00**
Ins28	100-8	2029.09	2306.59	2029.09	13.68	**0.00**
Ins29	100-9	1850.17	2096.86	1850.17	13.33	**0.00**
Ins30	100-10	2299.46	2752.10	2299.46	19.68	**0.00**
Mean		1016.76	1060.81	1027.11	2.31	1.40

value of 0.18, while the IJaya-NL method obtained an average RPI value of 5.41.

We perform a multifactor analysis of variance (ANOVA) to evaluate the significance of the difference between the two methods. Fig. 5(a) shows the means and the 95% LSD (least significant difference) intervals for the fitness values of the two compared methods. The *p*-value is close to zero; hence, there are significant differences between the compared methods. Hence, it can be concluded that the proposed local search heuristic improves the performance significantly. The main reason for this effect is that by applying the local search heuristic, the proposed IJaya algorithm enhances the exploitation abilities.

5.4. Efficiency of the SA-based exploration method

To investigate the effectiveness of the exploration heuristic discussed in Section 4.5, we implemented the two different types of IJaya algorithms, i.e., the IJaya-NS without the exploration heuristic and the IJaya with all the components discussed in Section 4. All the other components of the two compared algorithms were left unchanged. The comparison results are given in Table 5.

It can be observed from Table 5 that: (1) out of the 30 instances with different problem scales, IJaya obtained better values 13 times, which is slightly worse than the IJaya-NS method. However, the proposed IJaya method shows better performance for solving the relatively large scale problems, i.e., from "Inst22" to "Inst30"; (2) from the last two columns, the RPI values also verify the performance of the IJaya method, which apparently outperforms the IJaya-NS method; and (3) from the last row in the table, the average performance also verifies the superior performance of the IJaya method with an average RPI value of 1.40, while the IJaya-NS method obtained an average RPI value of 2.31. In addition, Fig. 5(b) shows the ANOVA result obtained by the two heuristics on the last 15 instances with relatively large scales. It can be concluded from Fig. 6(b) that the proposed SA-based exploration heuristic improves the performance of proposed approach significantly, especially for the instances with relatively large scales.

5.5. Comparison result with other types of efficient algorithm

We further design experiments to compare the existing state-of-the-art algorithms with the proposed IJaya on the same problem. The compared algorithms include GA-GSO (Liu et al. 2019) [23], PSO-SA (Tang et al. 2019) [12], ASA (Cruz-Chávez et al. 2017) [13], IG (Aqel et al. 2019) [14], DABC (Li et al. 2014) [15], TPM (Lei et al. 2018) [26], and DJaya (Gao et al. 2018) [19]. The results are given in Table 6, where the RPI values of all the compared algorithms are recorded.

It can be concluded from the table that: (1) IJaya obtained 20 better solutions among the 30 given instances, which is better than the second best (ASA algorithm), and (2) from the last row in the table, the proposed IJaya algorithm noticeably outperforms the other algorithms with an average RPI value of 1.36.

To verify the statistical efficiency, we also perform the ANOVA on the compared methods. Fig. 5(c) indicates that the difference between the IJaya algorithm and the second-best ASA algorithm is significant. Fig. 5(d) indicates that the IJaya method outperforms the DJaya algorithm, while Fig. 5(e) reports that IJaya shows competitive performance compared with the other efficient algorithms. Fig. 6 shows the convergence curve comparisons for the different types of instances. It can be observed from these convergence curves that the proposed IJaya method exhibits better convergence capacity for problems with different scales.

Fig. 7 gives a Gantt chart for the best solution for Inst1, where each operation is represented by a rectangle labeled with the job number and the operation number. The setup time and the transportation time are labeled with rectangles filled with different colors. It can be observed from Fig. 7 that the solution obtained by the proposed algorithm is effective and efficient.

Table 6
Comparison results for the seven heuristics.

Instance	Best	Fitness							Dev						
		GA-GSO	ASA	DABC	IG	TPM	DJaya	IJaya	GA-GSO	ASA	DABC	IG	TPM	DJaya	IJaya
Ins1	322.85	371.64	355.82	346.25	341.73	345.02	348.22	322.85	15.11	10.21	7.25	5.85	6.87	7.86	**0.00**
Ins2	291.81	356.59	322.89	340.17	319.97	315.57	299.53	291.81	22.20	10.65	16.57	9.65	8.14	2.64	**0.00**
Ins3	367.59	503.43	424.97	447.71	454.99	414.01	388.84	367.59	36.95	15.61	21.80	23.78	12.63	5.78	**0.00**
Ins4	353.62	475.30	417.22	436.94	431.70	393.78	362.32	353.62	34.41	17.99	23.56	22.08	11.36	2.46	**0.00**
Ins5	390.22	546.98	448.99	505.07	492.01	422.50	394.17	390.22	40.17	15.06	29.43	26.09	8.27	1.01	**0.00**
Ins6	583.87	732.40	634.74	689.40	641.30	623.85	611.50	583.87	25.44	8.71	18.08	9.84	6.85	4.73	**0.00**
Ins7	440.04	583.32	474.63	544.17	488.10	491.06	468.06	440.04	32.56	7.86	23.66	10.92	11.59	6.37	**0.00**
Ins8	573.73	848.65	617.52	748.27	708.62	637.24	598.67	573.73	47.92	7.63	30.42	23.51	11.07	4.35	**0.00**
Ins9	510.18	764.49	556.70	674.61	626.04	553.79	519.79	510.18	49.85	9.12	32.23	22.71	8.55	1.89	**0.00**
Ins10	632.23	1040.56	696.10	893.87	870.64	692.80	677.45	632.23	64.59	10.10	41.39	37.71	9.58	7.15	**0.00**
Ins11	724.99	964.52	764.53	893.14	820.30	785.66	742.78	724.99	33.04	5.45	23.19	13.15	8.37	2.45	**0.00**
Ins12	633.85	926.40	688.46	843.65	764.11	693.98	672.79	633.85	46.15	8.61	33.10	20.55	9.49	6.14	**0.00**
Ins13	772.51	1227.98	811.40	1081.73	991.88	851.07	849.57	772.51	58.96	5.03	40.03	28.40	10.17	9.97	**0.00**
Ins14	715.90	1167.75	753.53	1003.02	934.25	758.13	802.08	715.90	63.12	5.26	40.11	30.50	5.90	12.04	**0.00**
Ins15	886.79	1478.98	928.63	1261.47	1211.44	946.24	1010.47	886.79	66.78	4.72	42.25	36.61	6.70	13.95	**0.00**
Ins16	924.70	1291.11	958.02	1177.83	1045.53	999.50	1000.38	924.70	39.62	3.60	27.37	13.07	8.09	8.18	**0.00**
Ins17	855.97	1219.00	883.06	1089.95	1011.40	895.51	897.07	855.97	42.41	3.17	27.34	18.16	4.62	4.80	**0.00**
Ins18	1000.72	1574.12	1000.72	1380.21	1252.40	1062.97	1135.77	1020.11	57.30	**0.00**	37.92	25.15	6.22	13.50	1.94
Ins19	1027.39	1662.45	1027.39	1435.86	1325.01	1098.24	1220.12	1053.62	61.81	**0.00**	39.76	28.97	6.90	18.76	2.55
Ins20	1086.12	1845.32	1086.12	1541.38	1456.61	1167.59	1363.63	1133.09	69.90	**0.00**	41.92	34.11	7.50	25.55	4.32
Ins21	1429.33	2020.81	1440.65	1771.86	1673.35	1496.55	1575.33	1429.33	41.38	0.79	23.96	17.07	4.70	10.21	**0.00**
Ins22	1215.75	1862.79	1227.35	1604.02	1474.81	1292.76	1402.90	1215.75	53.22	0.95	31.94	21.31	6.33	15.39	**0.00**
Ins23	1625.00	2665.01	1625.00	2241.71	2201.83	1779.03	1941.31	1691.40	64.00	**0.00**	37.95	35.50	9.48	19.47	4.09
Ins24	1504.32	2421.21	1504.32	2053.50	1977.08	1579.87	1712.58	1506.52	60.95	**0.00**	36.51	31.43	5.02	13.84	0.15
Ins25	1727.06	3020.80	1727.06	2505.62	2448.10	1915.47	2229.30	1882.84	74.91	**0.00**	45.08	41.75	10.91	29.08	9.02
Ins26	2000.99	2869.46	2000.99	2490.79	2407.70	2133.05	2232.00	2072.19	43.40	**0.00**	24.48	20.33	6.60	11.54	3.56
Ins27	1648.95	2466.24	1652.30	2142.91	2034.92	1759.10	1903.60	1648.95	49.56	0.20	29.96	23.41	6.68	15.44	**0.00**
Ins28	2005.73	3361.27	2005.73	2800.73	2685.44	2243.71	2578.45	2029.09	67.58	**0.00**	39.64	33.89	11.87	28.55	1.16
Ins29	1741.11	3021.29	1741.11	2563.83	2461.77	1990.56	2330.16	1850.17	73.53	**0.00**	47.25	41.39	14.33	33.83	6.26
Ins30	2135.35	3850.21	2135.35	3168.63	3172.89	2455.95	3039.00	2299.46	80.31	**0.00**	48.39	48.59	15.01	42.32	7.69
mean	1004.29	1571.34	1030.38	1355.94	1290.86	1093.15	1176.93	1027.11	50.57	5.02	32.08	25.18	8.66	12.64	**1.36**

6. Conclusions

In this study, to solve the FJSP with setup time and transportation time constraints, we developed an improved Jaya algorithm. The main contributions of this study are as follows: (1) the problem was modeled by employing an integer programming method; (2) an improved Jaya algorithm was proposed to solve the target problem; (3) each solution is represented by a two-dimensional vector; (4) several problem-specific local search operators are developed to perform the exploitation tasks; (5) to enhance the exploration ability, a SA-based heuristic is embedded in the algorithm; and (6) 30 instances with different scales were generated and used for simulation tests. Six efficient algorithms were selected for detailed comparisons. The simulation results confirmed that the proposed algorithm can solve the considered problem with superior effectiveness.

Future work will mainly focus on the following tasks: (1) apply the proposed algorithm to solve FJSPs with multiple cranes; (2) consider multi-objective optimization methods [48] that generate a list of non-dominated solutions such as the Pareto-based method and multi-objective evolutionary algorithm based on decomposition; (3) combine other efficient heuristics, such as the species and memory-driven method, deep-learning algorithms, to enhance the performance of the proposed algorithm; (4) consider applying the proposed algorithm to solve other realistic applications, such as the prefabricated systems [49], flexible open shop scheduling problems [50], and Cloud computing scheduling problems [51]; and (5) adapt other realistic constraints and objectives into the considered problem.

CRediT authorship contribution statement

Jun-qing Li: Conceptualization, Methodology, Writing - original draft, Supervision. **Jia-wen Deng:** Software, Validation. **Cheng-you Li:** Visualization, Investigation. **Yu-yan Han:** Formal analysis, Resources. **Jie Tian:** Data curation, Writing - review & editing. **Biao Zhang:** Validation, Data curation. **Cun-gang Wang:** Formal analysis, Writing - original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research is partially supported by National Science Foundation of China under Grant 61773192, 61803192, 61773246, Shandong Province Higher Educational Science and Technology Program, China (J17KZ005), and major Program of Shandong Province Natural Science Foundation, China (ZR2018ZB0419).

References

[1] P. Brandimarte, Routing and scheduling in a flexible job shop by tabu search, Ann. Oper. Res. 41 (3) (1993) 157–183.
[2] I. Kacem, S. Hammadi, P. Borne, Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems, IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.) 32 (1) (2002) 1–13.
[3] M.T. Jensen, Generating robust and flexible job shop schedules using genetic algorithms, IEEE Trans. Evol. Comput. 7 (3) (2003) 275–288.
[4] N.B. Ho, J.C. Tay, E.M.K. Lai, An effective architecture for learning and evolving flexible job-shop schedules, European J. Oper. Res. 179 (2) (2007) 316–333.
[5] J. Gao, L. Sun, M. Gen, A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems, Comput. Oper. Res. 35 (9) (2008) 2892–2907.
[6] G. Zhang, X. Shao, P. Li, L. Gao, An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem, Comput. Ind. Eng. 56 (4) (2009) 1309–1318.
[7] L.N. Xing, Y.W. Chen, P. Wang, Q.S. Zhao, J. Xiong, A knowledge-based ant colony optimization for flexible job shop scheduling problems, Appl. Soft Comput. 10 (3) (2010) 888–896.

[8] J.Q. Li, Q.K. Pan, K.Z. Gao, Pareto-based discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling problems, Int. J. Adv. Manuf. Technol. 55 (9–12) (2011) 1159–1169.

[9] L. Wang, G. Zhou, Y. Xu, S. Wang, M. Liu, An effective artificial bee colony algorithm for the flexible job-shop scheduling problem, Int. J. Adv. Manuf. Technol. 60 (1–4) (2012) 303–315.

[10] Y. Yuan, H. Xu, Multiobjective flexible job shop scheduling using memetic algorithms, IEEE Trans. Autom. Sci. Eng. 12 (1) (2013) 336–353.

[11] K.Z. Gao, P.N. Suganthan, Q.K. Pan, T.J. Chua, T.X. Cai, C.S. Chong, Pareto-based grouping discrete harmony search algorithm for multi-objective flexible job shop scheduling, Inform. Sci. 289 (2014) 76–90.

[12] H. Tang, R. Chen, Y. Li, Z. Peng, S. Guo, Y. Du, Flexible job-shop scheduling with tolerated time interval and limited starting time interval based on hybrid discrete PSO-SA: An application from a casting workshop, Appl. Soft Comput. 78 (2019) 176–194.

[13] M.A. Cruz-Chávez, M.G. Martínez-Rangel, M.H. Cruz-Rosales, Accelerated simulated annealing algorithm applied to the flexible job shop scheduling problem, Int. Trans. Oper. Res. 24 (5) (2017) 1119–1137.

[14] G. Al Aqel, X. Li, L. Gao, A modified iterated greedy algorithm for flexible job shop scheduling problem, Chin. J. Mech. Eng. 32 (1) (2019) 21.

[15] J.Q. Li, Q.K. Pan, M.F. Tasgetiren, A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities, Appl. Math. Model. 38 (3) (2014) 1111–1132.

[16] E. Ahmadi, M. Zandieh, M. Farrokh, S.M. Emami, A multi objective optimization approach for flexible job shop scheduling problem under random machine breakdown by evolutionary algorithms, Comput. Oper. Res. 73 (2016) 56–66.

[17] Y. Xu, L. Wang, S.Y. Wang, M. Liu, An effective teaching–learning-based optimization algorithm for the flexible job-shop scheduling problem with fuzzy processing time, Neurocomputing 148 (2015) 260–268.

[18] T. Jamrus, C.F. Chien, M. Gen, K. Sethanan, Hybrid particle swarm optimization combined with genetic operators for flexible job-shop scheduling under uncertain processing time for semiconductor manufacturing, IEEE Trans. Semicond. Manuf. 31 (1) (2017) 32–41.

[19] K. Gao, F. Yang, M. Zhou, Q. Pan, P.N. Suganthan, Flexible job-shop rescheduling for new job insertion by using discrete jaya algorithm, IEEE Trans. Cybern. 49 (5) (2018) 1944–1955.

[20] L. Gao, Q.K. Pan, A shuffled multi-swarm micro-migrating birds optimizer for a multi-resource-constrained flexible job shop scheduling problem, Inform. Sci. 372 (2016) 655–676.

[21] Q. Zhang, H. Manier, M.A. Manier, A genetic algorithm with tabu search procedure for flexible job shop scheduling with transportation constraints and bounded processing times, Comput. Oper. Res. 39 (7) (2012) 1713–1723.

[22] M. Dai, D. Tang, A. Giret, M.A. Salido, Multi-objective optimization for energy-efficient flexible job shop scheduling problem with transportation constraints, Robot. Comput.-Integr. Manuf. 59 (2019) 143–157.

[23] Z. Liu, S. Guo, L. Wang, Integrated green scheduling optimization of flexible job shop and crane transportation considering comprehensive energy consumption, J. Cleaner Prod. 211 (2019) 765–786.

[24] L. Shen, S. Dauzère-Pérès, J.S. Neufeld, Solving the flexible job shop scheduling problem with sequence-dependent setup times, European J. Oper. Res. 265 (2) (2018) 503–516.

[25] A. Rossi, G. Dini, Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method, Robot. Comput.-Integr. Manuf. 23 (5) (2007) 503–516.

[26] 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) (2018) 1097–1109.

[27] H. Wang, Z. Jiang, Y. Wang, H. Zhang, Y. Wang, A two-stage optimization method for energy-saving flexible job-shop scheduling based on energy dynamic characterization, J. Cleaner Prod. 188 (2018) 575–588.

[28] H. Mokhtari, A. Hasani, An energy-efficient multi-objective optimization for flexible job-shop scheduling problem, Comput. Chem. Eng. 104 (2017) 339–352.

[29] X. Wu, Y. Sun, A green scheduling algorithm for flexible job shop with energy-saving measures, J. Clean. Prod. 172 (2018) 3249–3264.

[30] L. Zhang, Q. Tang, Z. Wu, F. Wang, Mathematical modeling and evolutionary generation of rule sets for energy-efficient flexible job shops, Energy 138 (2017) 210–227.

[31] L. Meng, C. Zhang, X. Shao, Y. Ren, MILP models for energy-aware flexible job shop scheduling problem, J. Clean. Prod. 210 (2019) 710–723.

[32] R. Rao, Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems, Int. J. Ind. Eng. Comput. 7 (1) (2016) 19–34.

[33] R.V. Rao, A. Saroj, A self-adaptive multi-population based jaya algorithm for engineering optimization, Swarm Evol. Comput. 37 (2017) 1–26.

[34] C. Huang, L. Wang, R.S.C. Yeung, Z. Zhang, H.S.H. Chung, A. Bensoussan, A prediction model-guided jaya algorithm for the PV system maximum power point tracking, IEEE Trans. Sustain. Energy 9 (1) (2017) 45–55.

[35] R.V. Rao, K.C. More, Design optimization and analysis of selected thermal devices using self-adaptive jaya algorithm, Energy Convers. Manage. 140 (2017) 24–35.

[36] S.P. Singh, T. Prakash, V.P. Singh, M.G. Babu, Analytic hierarchy process based automatic generation control of multi-area interconnected power system using jaya algorithm, Eng. Appl. Artif. Intell. 60 (2017) 35–44.

[37] S.O. Degertekin, L. Lamberti, I.B. Ugur, Sizing, layout and topology design optimization of truss structures using the jaya algorithm, Appl. Soft Comput. 70 (2018) 903–928.

[38] S. Mishra, P.K. Ray, Power quality improvement using photovoltaic fed DSTATCOM based on JAYA optimization, IEEE Trans. Sustain. Energy 7 (4) (2016) 1672–1680.

[39] K. Yu, B. Qu, C. Yue, S. Ge, X. Chen, J. Liang, A performance-guided JAYA algorithm for parameters identification of photovoltaic cell and module, Appl. Energy 237 (2019) 241–257.

[40] N.N. Son, C. Van Kien, H.P.H. Anh, Parameters identification of bouc–wen hysteresis model for piezoelectric actuators using hybrid adaptive differential evolution and jaya algorithm, Eng. Appl. Artif. Intell. 87 (2020) 103317.

[41] J.Q. Li, X.R. Tao, B.X. Jia, Y.Y. Han, C. Liu, P. Duan, Z.X. Zheng, H.Y. Sang, Efficient multi-objective algorithm for the lot-streaming hybrid flowshop with variable sub-lots, Swarm Evol. Comput. (2020) http://dx.doi.org/10.1016/j.swevo.2019.100600.

[42] J.Q. Li, M.X. Song, L. Wang, P.Y. Duan, Y.Y. Han, H.Y. Sang, Q.K. Pan, Hybrid artificial bee colony algorithm for a parallel batching distributed flow shop problem with deteriorating jobs, IEEE Trans. Cybern. 50 (6) (2020) 2425–2439.

[43] W. Gao, H. Sheng, J. Wang, S. Wang, Artificial bee colony algorithm based on novel mechanism for fuzzy portfolio selection, IEEE Trans. Fuzzy Syst. 27 (2018) 966–978.

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

[45] R. Ruiz, T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop with scheduling problem, European J. Oper. Res. 177 (3) (2007) 2033–2049.

[46] J.Q. Li, Q.K. Pan, H.Y. Duan, Solving multi-area environmental/economic dispatch by a pareto-based chemical-reaction optimization algorithm, IEEE/CAA J. Autom. Sin. 6 (5) (2019) 1240–1250.

[47] B. Zhang, Q. Pan, L. Gao, L. Meng, X. Li, K. Peng, A three-stage multi-objective approach based on decomposition for an energy-efficient hybrid flowshop scheduling problem, IEEE Trans. Syst. Man Cybern.: Syst. (2019) http://dx.doi.org/10.1109/TSMC.2019.2916088.

[48] Y.Y. Han, J.Q. Li, H.Y. Sang, Y.P. Liu, K.Z. Gao, Q.K. Pan, Discrete evolutionary multi-objective optimization for energy-efficient blocking flow shop scheduling with setup time, Appl. Soft Comput. 2020 (2020) http://dx.doi.org/10.1016/j.asoc.2020.106343.

[49] J.Q. Li, Y.Q. Han, P.Y. Duan, Y.Y. Han, B. Niu, C.D. Li, Z.X. Zheng, Y.P. Liu, Meta-heuristic algorithm for solving vehicle routing problems with time windows and synchronized visit constraints in prefabricated systems, J. Clean. Prod. (2020) http://dx.doi.org/10.1016/j.jclepro.2019.119464.

[50] D. Bai, Z. Zhang, Q. Zhang, Flexible open shop scheduling problem to minimize makespan, Comput. Oper. Res. 67 (2016) 207–215.

[51] J.Q. Li, Y. Han, A hybrid multi-objective artificial bee colony algorithm for flexible task scheduling problems in cloud computing system, Cluster Comput. (2020) http://dx.doi.org/10.1007/s10586-019-03022-z.