International Journal of Computational Intelligence Systems
Vol. **14(1)**, 2021, *pp.* 54–66
DOI: https://doi.org/10.2991/ijcis.d.201104.001; ISSN: 1875-6891; eISSN: 1875-6883
https://www.atlantis-press.com/journals/ijcis/

Research Article

An Adaptive Multi-Objective Evolutionary Algorithm with Two-Stage Local Search for Flexible Job-Shop Scheduling

Yingli Li, Jiahai Wang*, Zhengwei Liu

School of Mechanical Engineering, Tongji University, Shanghai, 201804, China

ARTICLE INFO

Article History
Received 18 Jul 2020
Accepted 29 Oct 2020

Keywords
Flexible job-shop scheduling
Multi-objective optimization
Evolutionary algorithm
Local search

ABSTRACT

An adaptive evolutionary algorithm with two-stage local search is proposed to solve the multi-objective flexible job-shop scheduling problem (MOFJSP). Adaptivity and efficient solving ability are the two main features. An autonomous selection mechanism of crossover operator is designed, which divides individuals into different levels and selects the appropriate one according to the both sides' levels to improve the self-adaptation. In parameter setting, the autonomous determination and adjustment mechanism is proposed, and parameters are adjusted autonomously according to the job scale and iteration number, so as to reduce the complexity of parameter setting and further improve the adaptivity. For improving solving ability, two-stage local search mechanism is designed. The first stage is performed before the evolution operation, so that each individual has more good genes to participate in the following operation. The second stage is performed after the evolution operation to further search the optimal solutions. Finally, a large number of comparative numerical tests are carried out, compared with other excellent algorithms, the proposed algorithm has fewer parameters to be set and stronger solving ability.

1. INTRODUCTION

Job-shop scheduling problem (JSP) is a branch of production scheduling and one of the most arduous combinatorial optimization problems [1]. In the classical JSP, a set of n jobs must be processed on m machines where each job i consists of n_i operations that should be performed on the predefined machines while satisfying precedence constraints [2]. However, in order to increase market competitiveness, flexible job-shop has become an inevitable choice for modern enterprises. This leads to a modified version of JSP called flexible JSP [3].

Flexible job-shop scheduling problem (FJSP) is an extension of the classical JSP. Unlike JSP issue, FJSP breaks the restriction of unique resources, operations are allowed to be processed on any among a set of available machines. Therefore, compared with JSP, FJSP is closer to the actual production situation. Because of the additional need to determine the assignment of operations on the machines, FJSP is more complex than JSP, and incorporates all the difficulty and complexity of JSP [4].

Brucker and Schlie [5] were the first to address the FJSP. They proposed a polynomial algorithm for solving the FJSP with two jobs, in which the machines capable of performing one operation have the same processing time. Commonly there are two methods for solving FJSP: hierarchical approach and integrated approach. Hierarchical approach was first proposed by Brandimarte [6]. Its basic idea is decomposing the complex problem into some sub-problems in order to decrease the complexity, i.e., it considered the assigning sub-problem and the sequencing sub-problem separately. However, in integrated approach, the assigning sub-problem and the sequencing sub-problem are solved simultaneously.

There have been many single-objective studies on FJSP, but the multi-objective flexible job-shop scheduling problem (MOFJSP) is more in line with the need of actual production. MOFJSP has been studied by many researchers in past decades. Xia and Wu [7] proposed a hierarchical solution approach by using a particle swarm optimization algorithm to assign operations on machines and a simulated annealing algorithm to schedule operations on each machine. Zhang *et al.* [8] proposed an effective hybrid approach for the MOFJSP, hybridizing the two optimization algorithms, particle swarm optimization and tabu search algorithm. Baykasoğlu *et al.* [9] presented a linguistic-based meta-heuristic modeling and multi-objective tabu search algorithm to solve the MOFJSP. Ho and Tay [10] studied a hybrid evolution algorithm combined with a guided local search and external Pareto archive set (AS).

Evolutionary algorithm (EA) is an intelligent optimization algorithm inspired by biology, which has been widely used in solving MOFJSP. The existing algorithms have poor self-adaptability. Specifically, the evolutionary operation does not consider the differences of individuals, and cannot make dynamic adjustment according to the individuals' differences. Also, the parameter setting process is complex and lacks flexibility. When the test case is replaced, parameters need to be reoptimized. In addition, how to

*Corresponding author. Email: jhwang@tongji.edu.cn

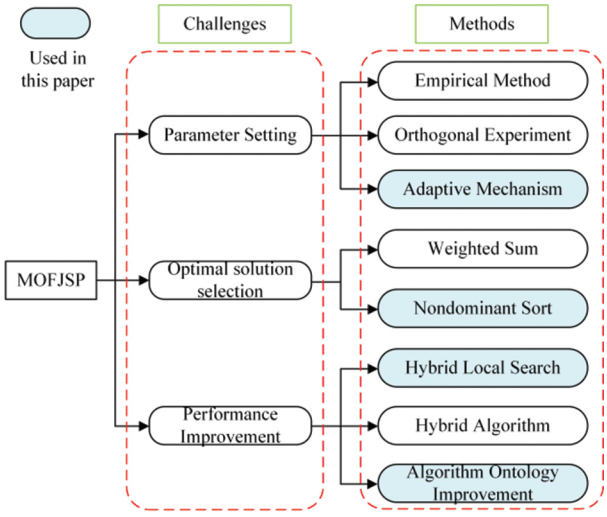

Figure 1 Challenges and overcome methods.

improve the algorithm's solution performance is also a current bottleneck. To solve the above problems, the EA is improved in many aspects in this paper, as shown in Figure 1.

In this paper, an adaptive evolutionary algorithm with two-stage local search is proposed. First, the individual coding scheme was improved. To adapt to the scheme, new methods of individual crossover, mutation and selection were designed. In terms of crossover operation, considering the differences between individuals, different crossover operators are designed. Individuals of different levels can select the appropriate crossover operator, thereby improving the algorithm's adaptive ability and solving ability. Second, an adaptive parameter adjustment mechanism is designed to further improve the adaptive ability. The algorithm can autonomously adjust the parameters according to the scale of the jobs and the number of iterations, thus reducing the complexity of parameter setting. Finally, a two-stage local search is embedded in the algorithm. The first stage is before the evolution operation to allow individuals to have more good genes, and the second stage is after the evolution operation to search for more superior solutions, consequently further improve the solving ability of the algorithm.

The remainder of the paper is organized as follows: In Section 2, the assumptions and formulations of MOFJSP are described in detail. In Section 3, the scheme of the proposed algorithm for MOFJSP is elaborated. In Section 4, the computational results and the comparison with other approaches are presented. The final conclusions are given in Section 5.

2. PROBLEM FORMULATION

2.1. MOFJSP Description

The MOFJSP is described as follows [11]: there are n jobs $(J_i, i \in \{1, 2, \dots, n\})$ and m machines $(M_k, k \in \{1, 2, \dots, m\})$. Job J_i consists of one or more operations $\left(O_{ij}, j \in \{1, 2, \dots, n_i\}\right)$, n_i is the total number of operations for J_i. Operation O_{ij} is allowed to be executed by one machine from a given set M_{ij}, which is consisted of all the capable machines of the jth operation for job J_i. If $M_{ij} = M_k$, it is called total flexible, otherwise it is partially flexible. The task of scheduling is to assign each operation to a suitable machine and to sequence the operations on all the machines, so as to optimize some objectives. In addition, some restrictions must be met:

- One machine can only process one job at a time, and one job can only be processed only on one machine at a time.
- There is a priority restriction between every two different operations of the same job, i.e., only after the previous operation is completed, the next one can be processed.
- The operations of different jobs do not have precedence constraints.
- All jobs have equal priorities, all machines are independent and each machine is ready at zero time.
- Once one operation is started, it will not be interrupted until the process is finished.
- Moving time between operations and setting up time of machines are negligible.

The notation used in this paper is summarized in the follows:

- i, h	Index of jobs
- j, g	Index of operations
- k	Index of machines
- n	Total number of jobs
- m	Total number of machines
- n_i	Total number of operations of job i
- O_{ij}	The jth operation of job i
- O_{hg}	The gth operation of job h
- M_{ij}	The set of available machines for the operation O_{ij}
- F_1	Makespan (the maximal completion time)
- F_2	Critical machine workload (the machine with the biggest workload)
- F_3	Total workload of machines (the total working time of all machines)
- p_{ijk}	Processing time of O_{ij} on machine k
- p_{hjk}	Processing time of O_{hg} on machine k
- x_{ijk}	Decision variable
- x_{hgk}	Decision variable
- C_i	Completion time of job i
- C_{ij}	Completion time of operation O_{ij}
- C_{hg}	Completion time of operation O_{hg}

In this paper, three objectives will be optimized simultaneously, as follows [12]:

$$F_1 = \min\left(\max\{C_i | i = 1, 2, \dots, n\}\right). \tag{1}$$

$$F_2 = \min\left[\max\left\{\sum_{i=1}^{n}\sum_{j=1}^{n_i} p_{ijk}x_{ijk} | k = 1, 2, \dots, m\right\}\right]. \tag{2}$$

$$F_3 = \min\left(\sum_{i=1}^{n}\sum_{j=1}^{n_i}\sum_{k=1}^{m} p_{ijk}x_{ijk}\right). \tag{3}$$

Subject to:

$$x_{ijk} = \begin{cases} 1, & \text{if } O_{ij} \text{ is processed on machine } k \\ 0, & \text{otherwise} \end{cases}. \tag{4}$$

$$x_{hgk} = \begin{cases} 1, & \text{if } O_{hg} \text{ is processed on machine } k \\ 0, & \text{otherwise} \end{cases}. \tag{5}$$

$$\left[C_{ij} - C_{i(j-1)} \geq p_{ijk} x_{ijk}\right] \vee \left[C_{hg} - C_{h(g-1)} \geq p_{hgk} x_{hgk}\right], \forall i, j, k. \tag{6}$$

$$\left[\left(C_{hg} - C_{ij} - p_{hgk}\right) x_{hgk} x_{ijk} \geq 0\right] \vee \left[\left(C_{ij} - C_{hg} - p_{ijk}\right) x_{hgk} x_{ijk} \geq 0\right], \forall i, j, h, g, k. \tag{7}$$

$$\sum_{k=1}^{m} x_{ijk} = 1, \forall i, j. \tag{8}$$

$$\sum_{k=1}^{m} x_{hgk} = 1, \forall h, g. \tag{9}$$

where Eqs. (1–3) are the three optimization objectives. Eq. (1) is minimizing the makespan. Eq. (2) is minimizing the critical machine workload. Eq. (3) is minimizing the total workload of machines. These objectives are in conflict to some extent and have been widely used in many literatures [13]. Eq. (6) is the operation priority restriction. Eq. (7) is the conflict constraint ensuring that each machine can process only one operation at a time. Eqs. (8) and (9) ensure that each operation can be processed only one time.

2.2. Basic Concepts of Multi-Objective Optimization

Multi-objective optimization problem is usually defined in the following form [14]:

$$\text{Minimize } f(x) = \left(f_1(x), f_2(x), \ldots, f_q(x)\right). \tag{10}$$

s.t.

$$x \in X. \tag{11}$$

where x is a decision vector, X is the decision space and q is the number of sub-objectives. The objective vector $f(x)$ is composed of multiple sub-objectives $f_i(x)$.

Existing research approaches for multi-objective optimization problem can be categorized into three: Pareto dominance-based, aggregation-based and lexicographical order-based, and most of the research are on the first two. Due to the superiority of the Pareto dominance-based approach in the number of solutions, and no need to consider the weight assignment problem among multiple sub-objectives, this paper chooses Pareto dominance-based approach to solve the multi-objective optimization problem.

The goal of Pareto dominance-based approach is to obtain all the nondominated solutions, which are referred to as Pareto-optimal solutions. The definition of Pareto dominance and Pareto optimality are given as follows:

i. Nondominated solutions
Solution a (it is also called nondominated solution) is said to dominate solution b, denoted by $a \prec b$, if and only if

$$\begin{cases} f_i(a) \leq f_i(b), \forall i \in \{1, 2, \ldots, q\} \\ f_i(a) < f_i(b), \exists i \in \{1, 2, \ldots, q\} \end{cases}. \tag{12}$$

ii. Pareto optimality
A solution x^* is called Pareto optimal if and only if

$$\nexists x \in X, x \prec x^*. \tag{13}$$

3. THE PROPOSED EA

3.1. Encoding and Decoding Scheme

The proposed algorithm optimizes assigning sub-problem and sequencing sub-problem simultaneously, and thus the information of operations and related machines should be encoded in the same chromosome. Two encoding schemes are commonly used to solve MOFJSP [15]: A-B string [8,16,17] and 3-tuple scheme [18–20]. Both schemes represent the same information, and the difference is only in the implementation. In 3-tuple scheme, a gene includes three elements which are the job number, the operation number and the assigned machine. However, in A-B string, two genes are needed to represent the same information. So, 3-tuple scheme is simpler and more intuitive than A-B string in the manner of representation, and the chromosome length of 3-tuple is only half of A-B strings. In this paper, a new encoding scheme, called 2-tuple, is designed for MOFJSP, which is improved on the basis of 3-tuple, and it is more concise than 3-tuple because a gene goes from three elements to two.

Each chromosome is a sequence of genes. A gene is a 2-tuple (i, k), in which i and k denote job number and assigned machine separately. The jth occurrence of i, from left to right of chromosome, represents the jth operation of job i. Taking Figure 2 as an example, this chromosome encodes the sequence of six operations, three of job1 and three of job2. The first gene (2, 1) of chromosome represents that the 1st operation of job2 will be executed on machine M_1, the third gene (2, 3) denotes that the 2nd operation of job2 will be processed on machine M_3, and the rest can be interpreted in the same manner.

Decoding is the process of converting each chromosome to scheduling solution. Schedules are grouped into three types by [21]: non-delay schedule, semi-schedule and active schedule. It has been proved that active schedule contains the optimal schedule. In an active schedule, no operation can be processed in advance except putting off another operation's start time or changing the order of operations [22]. In order to obtain active schedule, this paper introduces the left-shift greedy decoding scheme [23]. The advantage of this scheme is that it has a self-repairing mechanism, always searching a sufficient and suitable idle time interval for each operation on the processing machine without delaying any other operations, so as to shift the operation to the left as compact as possible. The detailed procedures are shown as follows:

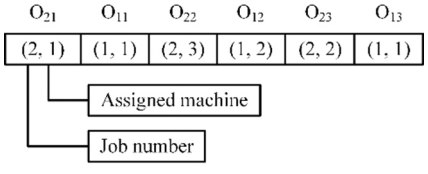

Figure 2 Chromosome encoding.

Step 1: Read gene sequence of a chromosome from left to right successively. Assuming a gene (i, k) is currently obtained, it denotes operation O_{ij} will be executed on machine k.

Step 2: Search the first idle time interval $\left[t_k^S, t_k^E\right]$ on machine k, where t_k^S and t_k^E represents the beginning time and the ending time of the interval separately.

Step 3: Identify the beginning time of operation O_{ij}. Due to the precedence constrains among operations, operation O_{ij} can be started only after its immediate job predecessor $O_{i(j-1)}$ being completed. The starting time S_{ij} of O_{ij} can be calculated through Eq.(14), where $C_{i(j-1)}$ is the completion time of $O_{i(j-1)}$. If the operation has no predecessor, $C_{i(j-1)}$ is replaced with 0.

$$S_{ij} = \max\left\{t_k^S, C_{i(j-1)}\right\}. \tag{14}$$

Step 4: Evaluate the condition of left-shift. If there is enough time from S_{ij} to t_k^E, i.e., $S_{ij} + p_{ijk} \leq t_k^E$, where p_{ijk} is the processing time of O_{ij} on machine k, then the interval $\left[t_k^S, t_k^E\right]$ is available for O_{ij}. That is, O_{ij} can be left shifted, and the completion time of O_{ij} is $S_{ij} + p_{ijk}$. Otherwise, search the next idle time interval. If there is no interval satisfy the condition of left-shift, then O_{ij} will be appended in the rear of the sequence of operations that has already been scheduled before O_{ij} on machine k.

Step 5: Update chromosome. Due to the employ of the left-shift greedy decoding scheme, operation r may be processed earlier than another operation v, which appears before r in the chromosome. Hence an update operation should be performed according to the result of decoding instead of the original chromosome, so as to keep the uniformity of encoding and decoding.

3.2. Population Initialization

The quality of initial population is very important for accelerating convergence speed and avoiding premature convergence. It has been proved that utilizing a hybrid multi-approach to generate initial population is better than using single one in the performance [20]. Thus, in this paper, an approach of integrating strategies is employed for generating initial population. It includes five assignment rules and four sequencing rules.

i. Assignment rules
   - Random rule: It assigns each operation to a capable machine randomly.
   - Minimum processing time rule [20]: For every operation, the machine with the minimum processing time will be selected in the corresponding operation's candidate machine set. If there are multi-machines with the same minimum processing time, select one from them randomly, then assign the operation to the selected machine.
   - Global minimum processing time rule [20]: From the processing time table, find the minimum processing time among all operations, fix the assignment, then add the selected processing time to every other entry in the same column, and conduct the next time selecting, until all operations are assigned. If there are multi-global minimum processing time, select one randomly.
   - Local minimum processing time rule [23]: Find the minimum processing time from the first operation of the first job to the last operation of the last job successively. For each operation, the selected time will be added to other entry in the same column for conducting next time selecting, and each operation will be assigned to the corresponding selected machine. If there are multi-local minimum processing time, select one randomly.
   - Permutation rule: Shuffle the order of operations in processing time table randomly. Take the operation in the first row as the starting operation to select the machine with minimum processing time, fix the assignment, then add the selected processing time to every other entry in the same column for conducting next time's selecting. If there are multi-machines have the same minimum processing time, randomly select one.

ii. Sequencing rules
   - Random rule: It places the operations into a chromosome in a random order.
   - Most work remaining rule [6]: One operation is selected according to the remaining work time of all jobs and be placed into a chromosome. The larger the remaining work time is, the earlier the operation of corresponding job can be selected. If there are multi-jobs have the same remaining work time, select one randomly.
   - Most number of operations remaining rule [20]: This rule select operation according to the remaining number of operations, the more the remaining number of operations is, the earlier the operation of corresponding job can be selected. If there are multi-jobs have the same remaining number of operations, select one randomly.
   - Shortest processing time rule [6]: The first operations of the remaining operations from each job are compared, the operation with the minimum processing time will be selected preferentially.

In the researches that utilizing hybrid multi-strategies to generate initial population, commonly a percentage-based approach, allocating different percentages for each approach to generate the required initial population, is adopted. In this paper, a new loop-based method is designed to generate initial population, and the difference is percentage allocation is not needed for each strategy compared with some other excellent algorithm, e.g., EPABC proposed in [13], AIA proposed in [18], etc. The utilization of loop-based method helps to increase the diversity of population and to extend the search space. The detailed procedure is described as follows:

Step 1: Place 20 (5×4) combinations of the assignment and sequencing rules in the set CH.

Step 2: A combination is successively selected in the manner of loop from CH to generate an individual. Assuming that the combination selected is (A, S). First, employ strategy A to complete operation assignment. Second, utilize strategy S to complete operation sequencing. Finally, combine the result of operation assignment and operation sequencing to generate a chromosome.

Step 3: Repeat perform step 2 until the number of initial individuals reaching N_{pop}.

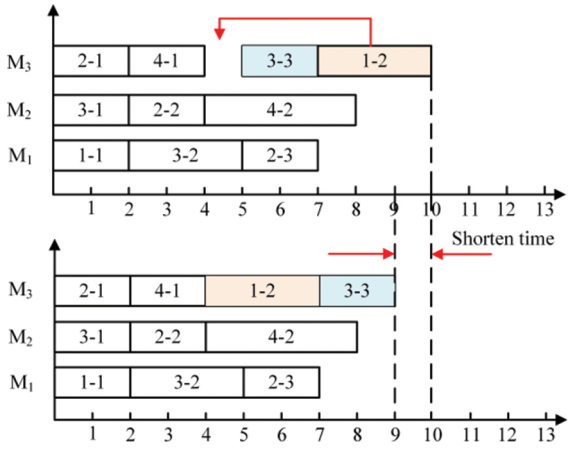

Figure 3 | Local search before evolution operation.

3.3. Local Search I (LS-I)

Local search is an effective method to exploit better solution around a current solution. It has been widely used in MOFJSP. Inspired by the idea of left-shift greedy decoding scheme (mentioned in Section 3.1), also for keeping initial population's distribution characteristics, a new local search method is proposed. The performing process of LS-I is described as follows:

Step 1: Take out one chromosome from population, decode the chromosome, obtain three objective's value (F_1, F_2 and F_3) and the terminal completion time ($C_k, k \in \{1, 2, \dots, m\}$) of each machine.

Step 2: Decode again. Due to different operations of the same job have precedence constrains and parent population commonly is not the optimal solutions, there are some idle time interval existing in a scheduling solution. Read a gene q from chromosome, find the first interval $\left[T_u^E, T_v^S\right]$ which is located between adjacent operations u and v on the same machine, where T_u^E is the end processing time of u, T_v^S is the start processing time of v. Let T_q^{eS} represents the earliest start processing time of operation q after moving q to the interval $\left[T_u^E, T_v^S\right]$. If $T_q^{eS} = \max\left\{T_{q-1}^E, T_u^E\right\}$ and $T_q^{eS} < T_v^S$, q has qualification to be moved to the interval $\left[T_u^E, T_v^S\right]$ and step 3 is performed. If q cannot be moved, next interval is examined. If all intervals do not satisfy the condition, decode the next gene. In case q has no job predecessor, T_{q-1}^E is replaced with "0".

Step 3: Judgment. If the moving of operation q results the solution become better, i.e., the new solution dominance the old one, update the chromosome and decode next gene. If the new solution is equal to the old one, but at least one machine's maximum completion time C_k become short, update the chromosome and decode next gene, so as to make the scheduling as compact as possible. Otherwise, q cannot be moved.

Step 4: Repeat step 2 to step 3, until all genes on the chromosome are decoded. Repeat step 1 to step 3, until all the individuals in initial population are optimized.

In order to better illustrate LS-I, taking Figure 3 as a simple example, moving operation O_{12} to the interval between O_{41} and O_{33} will shorten the processing time of machine M_3. If it is critical operation, the maximum completion time in all machines will be shorten, if it is not critical operation, the maximum completion time of current machine will be shortened and the scheduling will be more compact.

3.4. Crossover Operators

The goal of crossover is to obtain two offsprings which inherit the gene information from their parent's chromosomes and generate new superior gene sequences. In this algorithm, crossover includes two parts: crossover for operation sequence and crossover for machine assignment. These two parts are implemented separately, and the terminal condition is that the amount of offspring equals N_{pop}, where N_{pop} is the amount of the initial population.

i. Crossover for operation sequence
Many effective approaches are developed for operation sequence's crossover in the past decades, such as order crossover employed by [16], HCO proposed by [12], IPOX designed by [11]. In this paper, considering the chromosome structure, IPOX is adopted. In addition, a new crossover approach, called RPOX, is proposed on the basis of IPOX, so the proposed algorithm can adaptively select one from the two crossover approaches according to the quality of the parent's chromosomes to generate good offspring. The details of IPOX and RPOX are shown as follows:

Step 1: Fast nondominated sorting [24] is performed for initial population which is divided into different ranks R_α ($\alpha \in \{1, 2, \dots, l, \dots, r\}$). If $\sum_{\alpha=1}^{l-1} N_{R_\alpha} < 0.7N_{pop}$ and $\sum_{\alpha=1}^{l} N_{R_\alpha} \geq 0.7N_{pop}$, the individuals in R_1 to R_l are called excellent population and the remained individuals are called inferior population, where N_{R_α} represents the amount of individuals in rank R_α.

Step 2: Randomly select two individuals (P_1 and P_2) from initial population as parent's chromosomes, and let C_1 and C_2 be their children's chromosomes by crossing. All the jobs are randomly divided into two sets J_1 and J_2.

Step 3: Copy the elements of P_1 that are included in J_1 to C_1 in the same position and copy the elements of P_2 that are included in J_2 to C_2 in the same position.

Step 4: If the elements of P_2 that are included in J_2's rest position in the same order and the elements of P_1 that are included in J_1's rest position in the same order, this method is called IPOX. If the elements of P_2 that are included in J_2's rest position in the reverse order and the elements of P_1 that are included in J_1's rest position in the reverse order, this method is called RPOX. At this moment, keep the machine associated with operation is empty. A simple example about these two crossover methods is shown in Figure 4.

Step 5: If the parent's chromosomes are in the inferior population, RPOX is adopted. Otherwise, IPOX is adopted. Because the chromosomes selected from the inferior population always possess few good gene information, only with the large change, the offspring can possibly generate useful gene sequence. On the contrary, the chromosomes selected from the excellent population possess good gene sequence, so small change should be made to keep the good gene information to the next generation.

ii. Crossover machine assignment
After operation sequence's crossover is completed, crossover for machine assignment should be performed to determine the machine for each operation in children's chromosomes C_1 and C_2. Two-point crossover method, called ITP, is used, and the details are described as follows:

Step 1: Two integers I_1 and I_2 are selected from the range of $[2, L-1]$, where L is the length of one chromosome.

Step 2: The genes of C_1 between position I_1 and I_2 inherit the operation's machine information from P_1, the remined genes inherit from P_2. The genes of C_2 between position I_1 and I_2 inherit the operation's machine information from P_2, the remined genes inherit from P_1. A simple example is shown in Figure 5.

3.5. Mutation Operators

Mutation is one of the key technologies for avoiding premature and increasing the diversity of population. A substitution method is adopted in this paper. For operation sequence's mutation, two genes that do not belong to the same job are selected randomly, then their position will be swapped and the corresponding machine information will be updated.

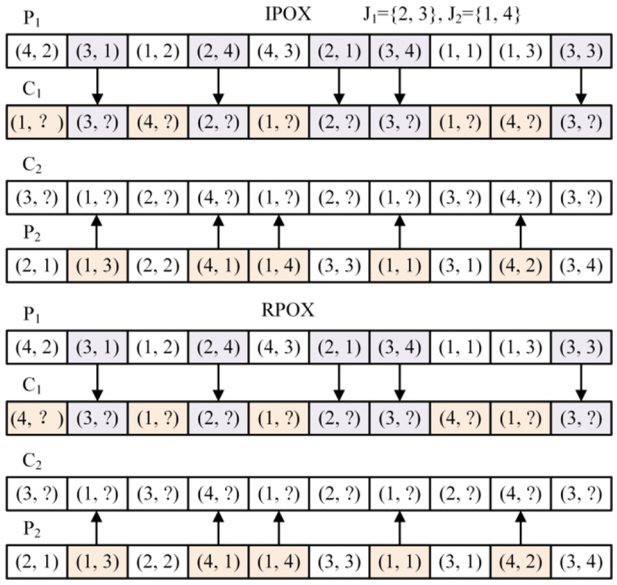

Figure 4 | IPOX and RPOX crossover operation for the operation sequence.

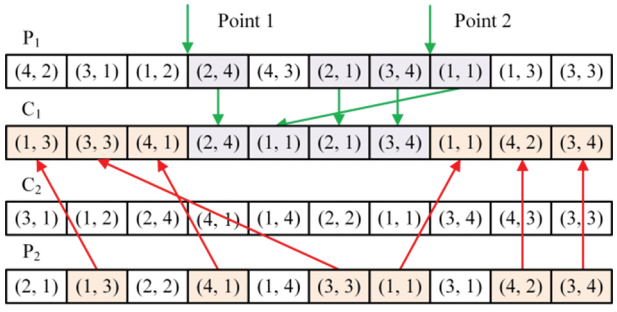

Figure 5 | ITP crossover operation for the machine assignment.

For machine assignment's mutation, the machine M_i in a randomly selected gene is replaced by a new one that is reselected from its candidate machine set. This reselection must meet a rule. If a subset, consisted by the machines with the processing time shorter than M_i, exists, select the new machine from this subset. Otherwise, select randomly from the candidate set but except M_i. The terminal time is controlled by mutation rate β.

3.6. Environment Selection

Environment selection is to select certain amounts of individuals, called new initial population or survived individuals, from the population of parent and children. The survived individuals will be participated in the next time's genetic operation. For environment selection, fitness-based [7] and dominance-based [25] selection methods have been developed and widely used in scheduling problem. Due to the advantage of dominance-based method in term of considering the tradeoff among multi-objectives, it is adopted in this paper.

In dominance-based method, crowding distance [25] is commonly used to select certain individuals from the subpopulation with the same rank. It is effective to the optimization problem with two objectives, but with three objectives it is insufficient to some extent. In order to keep the diversity of the new initial population, a partition selection method is designed to replace crowding distance. The details of environment selection are described as follows:

Step 1: Fast nondominated sorting is performed for all individuals and divided them into different levels R_α ($\alpha \in \{1, 2, \dots, l, \dots r\}$).

Step 2: If $\sum_{\alpha=1}^{l-1} N_{R_\alpha} < N_{pop}$ and $\sum_{\alpha=1}^{l} N_{R_\alpha} > N_{pop}$, go to step 3. If $\sum_{\alpha=1}^{l} N_{R_\alpha} = N_{pop}$, all individuals in R_1 to R_l will survive to the next generation, and go to step 4. Here N_{R_α} is the number of individuals in rank R_α.

Step 3: All individuals in R_1 to R_{l-1} survive to the next generation directly. The task of environment selection is to determine the survived individuals in R_l. Let N_R^l represents the number of individuals need to be selected from N_{R_l}, where $N_R^l = N_{pop} - \sum_{\alpha=1}^{l-1} N_{R_\alpha}$. First, if two objectives are called a sort operator, six sort operators can be consisted by three objectives (F_1, F_2 and F_3), and it is denoted by number 1 to 6 separately. For example, (F_1, F_2) is denoted by number 1. Second, a number is selected randomly from 1 to 6. Assuming the selected number is 1 and it represents the sort operator (F_1, F_2). Sort all individuals of R_l in ascending order based on F_1, if F_1 of multi-individuals are the same, sort the multi-individuals based on F_2. Third, according to the equation $n_l = \wedge\left(N_{R_l}/N_R^l\right)$, divide N_{R_l} individuals into N_R^l parts, each part except the last one has n_l individuals, where $\wedge$ presents taking lower bound value. For example, if $N_{R_l}/N_R^l = 2.6$, n_l is 2. The individual number of the last part may be larger than n_l. The individual in the first position of each part will be selected to the next generation.

Step 4: An external AS is imbedded in the proposed algorithm. The individuals in R_1 will be added into the AS when the environment selection procedure is performed at first time. Otherwise, this step can be negligible.

3.7. Local Search II (LS-II)

The purpose of LS-I mentioned in Section 3.3 is to make the chromosome carry more superior genes in a short time. However, a new local search method, called LS-II, based on the critical path is purposed to find more promising solutions. The emphasis of the two methods are different, LS-I pursues speed, LS-II pursues quality.

In this paper, a concept, called full sequence, is proposed. If no time interval exists between every two adjacent operations on the same machine, this state is called a full sequence, e.g., M_2 is under a full sequence in Figure 3. If a machine is under a full sequence and makespan belongs to this machine, it cannot reduce the makespan no matter how to shift the critical operation, unless reassign some operations to other machines.

In order to optimize three objectives simultaneous, the local search method based on critical path is extended. It includes two parts: reassignment and shifting critical operations. Compared with some methods that only shifting critical operations, LS-II not only reduce the makespan (F_1), but also balance the critical machine workload and the total workload (F_2&F_3). The details of LS-II are described as follows:

Step 1: According to the result of fast nondominated sorting in Section 3.6, the individuals in R_1 will be local searched. The goal of this is to reduce the load of calculation.

Step 2: Take out a chromosome c, exam whether the chromosome has the full sequence and the full sequence belongs to the machine with makespan. If no, go to step 3, else, go to step 4.

Step 3: Shifting critical operation. First, identify the critical paths and the critical blocks on the chromosome. Let O_{ij} represents one critical operation of a critical block, and O_{ij} cannot be the head operation of this critical block. Left-shift O_{ij} to the front of every operation in the critical block successively and calculate the corresponding objective value. If the solution is better, the chromosome after shift and this time's shift will be recorded. Repeat this process, until every critical operation except the head one in every critical block are performed. Multi-critical paths may possess some public critical blocks, so if recorded shift appears, it will be negligible. After shifting critical operations is performed, a new population N_s based on c is formed, the best one in N_s will be selected to replace c. This step can realize reducing the makespan (F_1).

Step 4: Reassignment. In all full sequences, if an operation has at least one candidate machine and the candidate machine's processing time is not larger than the current one, this operation is called reassignment operation and these candidate machines compose the replacement machine subset. Select one reassignment operation randomly, select the machine that has the minimum workload from the replacement machine subset, and move the selected operation to the selected machine. If some machines have the same minimum total workload, select one randomly. Repeat performing reassignment until the number of performing equals to the total sum of reassignment operations. If the new chromosome is better than the old one, the chromosome will be recorded. The best one in all new chromosomes will replace the old chromosome. This step can realize reducing the makespan (F_1), balancing the critical machine workload and the total workload (F_2&F_3)).

Step 5: Repeat step 1 to step 4, until all the individuals in R_1 are performed. Compare the individuals in R_1 to AS. If some individuals in AS are dominated by one individual in R_1, they will be removed and the better individual will be added. If one individual in R_1 and individuals in AS are not in the dominance relationship, the individual in R_1 will be added into AS.

3.8. Main Algorithm

The framework of the proposed algorithm for MOFJSP can be shown in Figure 6.

Step 1: Initialization. Loop-based method (Section 3.2) is used to generate initial population N_{pop}.

Step 2: Apply LS-I (Section 3.3) to optimize all individuals in N_{pop}. Update N_{pop} to New_{pop}.

Step 3: Perform crossover operators (Section 3.4) and mutation operators (Section 3.5) to generate offspring populations Nc_{pop} and Nm_{pop}.

Step 4: Merge New_{pop}, Nc_{pop} and Nm_{pop}. Remove duplicate individuals. Perform environment selection (Section 3.6) to generate the survive individuals Ns_{pop}.

Step 5: Apply LS-II (Section 3.7) to search more promising solutions, update Ns_{pop} and AS.

Step 6: If the terminating condition is satisfied, the algorithm ends. else, go to step 2.

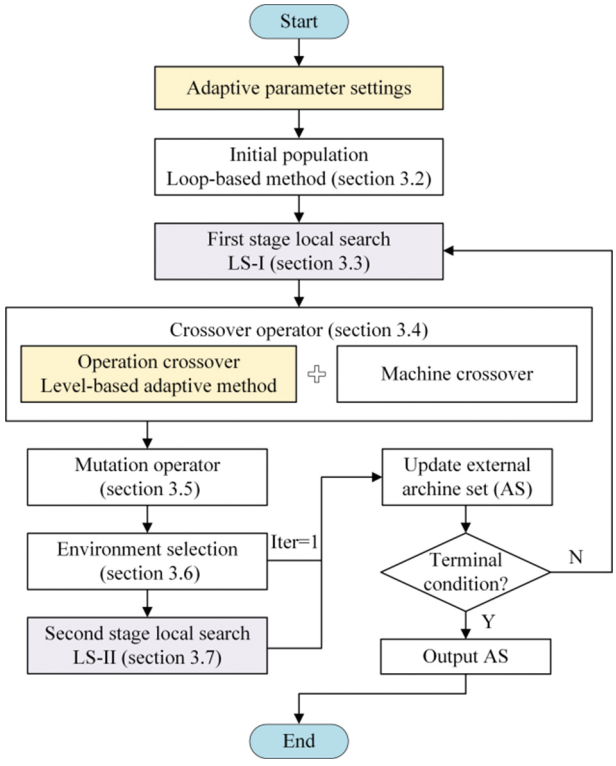

Figure 6 The flowchart of the proposed algorithm.

4. EXPERIMENTAL RESULTS

4.1. Parameter Setup

The process of determining the feasible parameter is complex in some researches. To simplify the process, in our algorithm, very few parameters are needed, and an adaptive parameter setup method is designed. The amount of initial population N_{pop} is determined by Eq. (15), where n is the amount of jobs. The mutation rate β is calculated by Eq. (16), where *iter* is the number of iterations. The terminal condition is that the AS members remain unchanged after 30 consecutive times iteration.

$$N_{pop} = \begin{cases} 100, \text{if } n \leq 10 \\ 15 \times n, \text{if } n > 10 \end{cases}. \tag{15}$$

$$\beta = 1.5 - \frac{1}{e^{\left(iter/N_{pop}\right)}}. \tag{16}$$

4.2. Result and Comparison

To test the performance of the proposed algorithm, the algorithm procedure was implemented in python and run through the PyCharm software on a PC with AMD A-8 6410 2.0 GHz CPU 4.0 G RAM and Windows 10 operating system. Four representative instance sets based on the practical data have been selected. The first one is taken from [26], which includes 10 small size problem instances (SFJS01:10) and 10 medium and large size problem instances (MFJS01:10). The second one is taken from [27] without release time, which includes five problem instances (problem 4×5, 8×8, 10×7, 10×10, 15×10), where problem 4×5 denotes four jobs and five machines, other four problem instances are interpreted in the same manner. The third one is taken from [6], which includes ten problem instances (Mk01-10). The last one is taken from [28], which includes eighteen problem instances (01a–18a). These four instance sets are the most commonly adopted benchmark instances in the researches on MOFJSP.

The algorithm's advantage is that very few parameters setup is needed. Table 1 gives a comparison of several state-of-the-art algorithm's number of parameters, including Li *et al.* [17,29], Wang *et al.* [11] and Xing *et al.* [30]. From this table, the number of parameters needed in the proposed algoithom is the least, which is three.

In order to test the effect of improved coding, crossover, mutation, population generation and environment selection methods on the performance of the proposed algorithm, the two-stage local search was removed from the proposed algorithm, and the remaining part (IEEA) was tested together with the traditional EA through the same case. The proposed algorithm is called IEA. The test case set uses SFJS01-10 and MFJS01-10. The EA is selected from the literature [15], which uses a variety of combination strategies, and has proved to be more superior than other classic EAs. In this section, metric C is used. $C(A, B)$ measures the fractions of members of B that are dominated by members of A, and Eq. (17) is the calculating equation of $C(A, B)$. $C(A, B) = 0$ means that all solutions in B are not dominated by any solution in A, $C(A, B) = 1$ means that each solution in B is dominated by some solutions in A. The test results are shown in Table 2.

$$C(A, B) = \frac{|\{b \in B | \exists a \in A : a \succ b\}|}{|B|}. \tag{17}$$

It can be seen from Table 2 that in the 20 problem instances, IEEA is superior to EA except MFJS03 and MFJS07. Compared with IEEA, IEA does not show obvious advantage in 10 small size problem instances, indicating that for small size problem instances IEA can find the optimal solution without local search, while for 10 medium and large size problem instances, IEA has shown obvious advantage. In addition, the frontier of EA, IEEA and IEA on MFJS02, MFJS03, MFJS05 and MFJS10 are given, as shown in Figure 7. It can also be seen from the figure that IEEA is better than EA, and IEA is better than IEEA.

In order to test the overall performance of the proposed algorithm, it was tested on other three instance sets with several state-of-the-art algorithms.

For five problem instances, four state-of-the-art algorithms in Table 1 are also used to compare with the proposed algorithm. The reason of doing this is that the four algorithms are powerful, and it has been proved that it is better than other well-known algorithms (e.g., PSO+SA proposed by [7], AL+CGA proposed by [31], etc.). Hence, if the proposed algorithm is better than the four powerful algorithms, it is also better than PSO+SA and AL+CGA, etc.

Before comparison, some terms should be clarified. Optimal solutions in this paper refer to the nondominated solutions obtained

Table 1 Compare in the number of parameters.

Author	Framework	Fitness Assignment	Number of Parameters
Li and Pan	ABC	Dominance	4
Li	TS	Weighted sum	6
Wang	GA	Dominance	4
Xing	LS	Weighted sum	13
Proposed	GA + LS	Dominance	3

Table 2 The test result of SFJS01-10 and MFJS01-10.

Item	EA and IEEA		IEEA and IEA	
	C (EA, IEEA)	*C* (IEEA, EA)	*C* (IEEA, IEA)	*C* (IEA, IEEA)
SFJS01	0.00	1.00	0.00	0.00
SFJS02	0.00	1.00	0.00	0.00
SFJS03	0.00	1.00	0.00	0.00
SFJS04	0.00	1.00	0.00	0.00
SFJS05	0.00	1.00	0.00	0.00
SFJS06	0.00	1.00	0.00	0.00
SFJS07	0.00	1.00	0.00	0.00
SFJS08	0.00	1.00	0.00	0.00
SFJS09	0.00	0.78	0.00	0.22
SFJS10	0.00	1.00	0.00	0.40
MFJS01	0.00	0.88	0.00	0.14
MFJS02	0.07	0.44	0.00	0.79
MFJS03	0.50	0.10	0.00	0.86
MFJS04	0.00	0.24	0.00	0.22
MFJS05	0.00	0.73	0.00	0.63
MFJS06	0.00	0.94	0.00	0.33
MFJS07	0.68	0.08	0.14	0.53
MFJS08	0.00	1.00	0.00	0.00
MFJS09	0.00	1.00	0.00	0.30
MFJS10	0.00	1.00	0.00	0.81

by each algorithm. Pareto front is constructed by the optimal solutions of all compared algorithms and Pareto solutions refer to the solutions in Pareto front. All solutions in the Pareto front are called overall Pareto solutions. If some optimal solutions in a compared algorithm's optimal solutions are the Pareto solutions simultaneously, they are called partial Pareto solutions.

First, compare the amount of optimal solutions. Table 3 is the result of optimal solutions' amount. From this table, the proposed algorithm can obtain the best results for all problem instances. In problem 4×5, the amount of optimal solutions obtained by the proposed algorithm is more than other four. In problem 8 × 8, the proposed algorithm obtains more optimal solutions than others, and it equals to Xing. However, one sloution (16, 13, 73) in the proposed algorithm's result dominates one solution (17, 13, 73) in Xing's. In problem 10 × 7, the proposed algorihtm obtains more optimal solutions only than Li, equals to other two. But one solution (11, 11, 61) in the proposed algortithm dominates two solutions (12, 11, 61) and (11, 11, 63) of Li and Pan. In problem 10×10, Li and Pan, Li and Xing all obtain three optimal solutions, the proposed algorithm can achieve four solutions. Compared with Wang, though the optimal solutions' amount is the same, one solution (7, 5, 45) in Wang is dominated by one solution (7, 5, 43) in the proposed algorithm. In problem

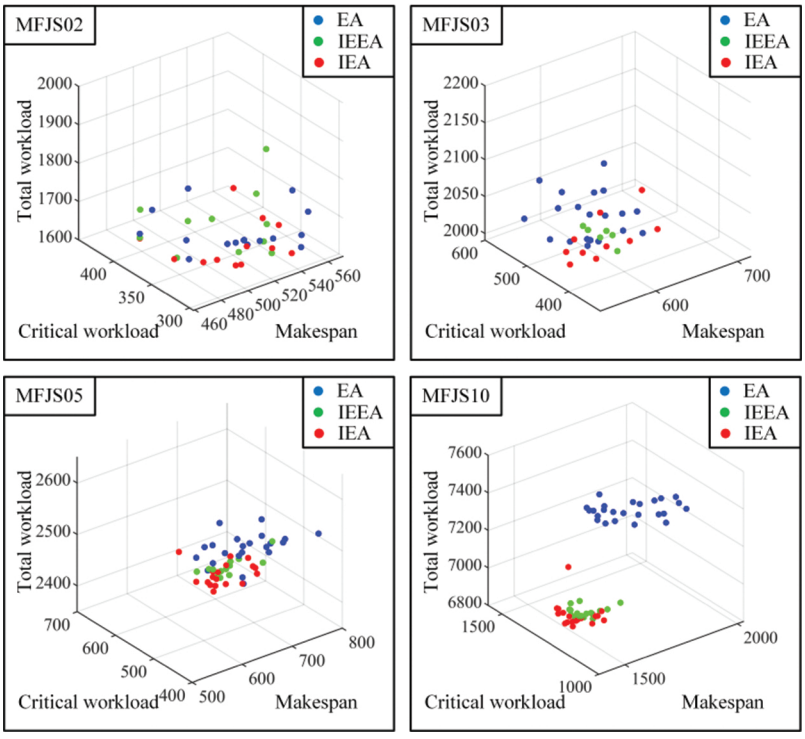

Figure 7 The frontier of EA, IEEA and IEA on problem instances MFJS02, MFJS03, MFJS05 and MFJS10.

Table 3 The optimal solutions of each algorithm for five problem instances.

Size	Method	Li and Pan			Li			Wang				Xing				Proposed			
	Objects	1	2	3	1	2	3	1	2	3	4	1	2	3	4	1	2	3	4
4 × 5	f_1	11	12	13	11	12	–	11	11	12	–	11	11	12	–	11	11	12	13
	f_2	10	8	7	10	8	–	10	9	8	–	10	9	8	–	10	9	8	7
	f_3	32	32	33	32	32	–	32	34	32	–	32	34	32	–	32	34	32	33
8 × 8	f_1	14	15	16	14	15	–	15	15	16	–	14	15	16	17	14	15	16	16
	f_2	12	12	13	12	12	–	11	12	13	–	12	12	11	13	12	12	11	13
	f_3	77	75	73	77	75	–	81	75	73	–	77	75	77	73	77	75	77	73
10 × 7	f_1	12	11	12	11	11	–	–	–	–	–	11	11	12	–	11	11	12	–
	f_2	11	11	12	11	10	–	–	–	–	–	11	10	12	–	10	11	12	–
	f_3	61	63	60	61	62	–	–	–	–	–	61	62	60	–	62	61	60	–
10 × 10	f_1	8	7	8	7	7	8	8	7	8	7	7	8	8	–	7	7	8	8
	f_2	7	5	5	5	6	5	5	6	7	5	6	7	5	–	6	5	7	5
	f_3	41	43	42	43	42	42	42	42	41	45	42	41	42	–	42	43	41	42
15 × 10	f_1	12	11	–	11	11	–	11	12	11	–	11	–	–	–	11	11	–	–
	f_2	11	11	–	11	10	–	11	10	10	–	11	–	–	–	11	10	–	–
	f_3	91	93	–	91	93	–	91	95	98	–	91	–	–	–	91	93	–	–

15×10, the proposed algorithm's optimal solutions amount is only more than Xing, is fewer than Wang, and equals to Li and Pan and Li. But one solution (11, 11, 91) in the proposed algorithm dominates all solutions in Li and Pan, and one solution (11, 10, 93) dominates two solutions (12, 10, 95) and (11, 10, 98) in Wang. Therefore, it can be concluded that our algorithm has a strong search ability on MOFJSP.

Second, compare the quality of optimal solutions. Table 4 gives an overall perspective of optimal solutions' quality. In this table, H denotes the number of the overall Pareto solutions, N represents the number of optimal solutions, M represents the number of partial Pareto solutions. Q, calculated by Eq. (18), represents the percentage of partial Pareto solutions M in optimal solutions N. R, calculated by Eq. (19), represents the percentage of partial Pareto solutions M in overall Pareto solutions H. For example, in problem 8×8, the proposed algorithm obtains four optimal solutions (i.e., $N = 4$), and the Pareto front includes five solutions (i.e., $H = 5$), all optimal solutions are Pareto solutions (i.e., $M = N = 4$), the percentage Q of partial Pareto solutions M in the optimal solutions N is 100%, and the percentage R of partial Pareto solutions M in overall Pareto solutions H is 80%.

$$Q = \frac{count\{H \cap N\}}{count\{N\}} \times 100\%. \tag{18}$$

$$R = \frac{count\{H \cap N\}}{count\{H\}} \times 100\%. \tag{19}$$

From Table 4, all the optimal solutions obtained by the proposed algorithm in the five problem instances are Pareto solutions, and, in each problem instance except problem 8×8, the number of the optimal solutions are equal to the number of overall Pareto solutions. Compared with other four algorithms, only Xing in problem 10×7 and Li in problem 15×10 achieve the same effectiveness. The last row gives a statistic result in the term of H, N, M, Q and R. In order to best show the statistics result, corresponding value is made into Figure 8. From Figure 8 the proposed algorithm can obtain the best result both in solutions' amount and solutions' quality.

Table 5 gives a result from a local perspective, i.e., taking every one in the four state-of-the-art algorithms to compare with our algorithm. In this table, metric C is used. The proposed algorithm is denoted by B.

From this table, all optimal solutions obtained by the proposed algorithm are not dominated by any optimal solution obtained by other compared algorithms in the five problem instances. However, at least one optimal solution obtained by the proposed algorithm can dominate some optimal solutions of Li and Pan (A1), Wang (A3) and Xing (A4) in at least one problem instance. Though, the proposed algorithm's optimal solutions are nondominated with Li's,

Table 4 The quality of optimal solutions I.

Size	H	Li and Pan			Li			Wang			Xing			Proposed		
		N/M	Q	R	N/M	Q	R	N/M	Q	R	N/M	Q	R	N/M	Q	R
4×5	4	3/3	100	75	2/2	100	50	3/3	100	75	3/3	100	75	4/4	100	100
8×8	5	3/3	100	75	2/2	100	40	3/3	100	60	4/3	75	60	4/4	100	80
10×7	3	3/1	33.3	33.3	2/2	100	66.7	–	–	–	3/3	100	100	3/3	100	100
10×10	4	3/3	100	75	3/3	100	75	4/3	75	75	3/3	100	75	4/4	100	100
15×10	2	2/0	0	0	2/2	100	100	3/1	33.3	50	1/1	100	50	2/2	100	100
Sum	18	14/10	71.4	55.6	11/11	100	61.1	13/10	76.9	55.6	14/13	92.9	72.2	17/17	100	94.4

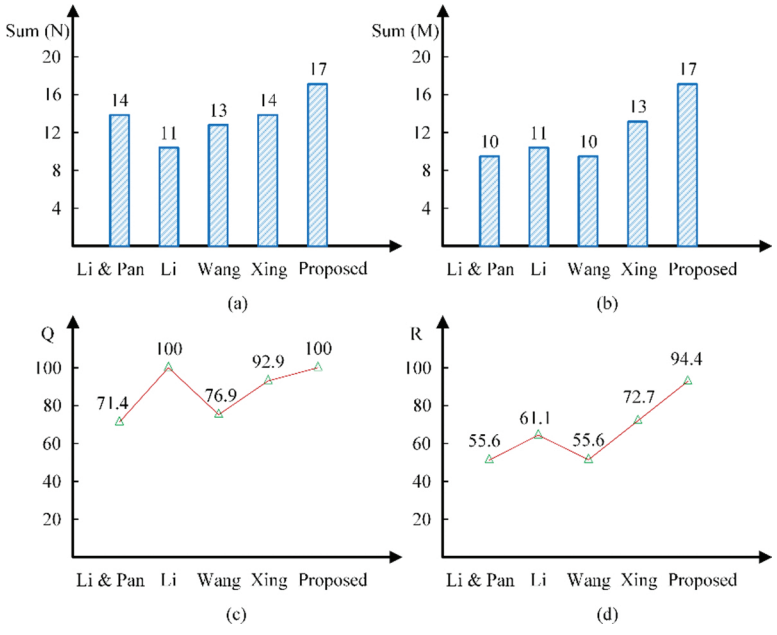

Figure 8 Statistic result on N, M, Q and R.

Table 5 The quality of optimal solutions II.

Size	Li and Pan (A1)		Li (A2)		Wang (A3)		Xing (A4)	
	C (A1, B)	*C* (B, A1)	*C* (A2, B)	*C* (B, A2)	*C* (A3, B)	*C* (B, A3)	*C* (A4, B)	*C* (B, A4)
4 × 5	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
8 × 8	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.250000
10 × 7	0.000000	0.666667	0.000000	0.000000	–	–	0.000000	0.000000
10 × 10	0.000000	0.000000	0.000000	0.000000	0.000000	0.250000	0.000000	0.000000
15 × 10	0.000000	1.000000	0.000000	0.000000	0.000000	0.666667	0.000000	0.000000

but from Tables 3 and 4, the proposed algorithm can achieve more optimal solutions. So, it can be also concluded that the proposed algorithm is more superior than others compared algorithm.

In addition, the distribution and convergence of optimal solutions is analyzed. One metric called inverted generational distance (*IGD*) is used. It is calculated by Eq. (20), where P^* denotes the Pareto front, which consists of optimal solutions, $|P^*|$ denotes the number of Pareto solutions, $d(x, y)$ is the Euclidean distance between points x and y. The smaller $IGD(A, P^*)$ is, the closer A is to the Pareto front. Besides that, the objectives of all chromosomes are normalized by Eq. (21), where $f_i^{\min}$ and $f_i^{\max}$ are the minimum and maximal values of the ith objective among all solutions. Table 6 is the calculating result. From this table, the proposed algorithm is closer to Pareto front than other compared algorithms.

$$IGD(A, P^*) = \frac{1}{|P^*|} \sum_{x \in P^*} \min_{y \in A} d(x, y). \tag{20}$$

$$f(x) = \frac{f_i(x) - f_i^{\min}}{f_i^{\max} - f_i^{\min}}, i = 1, 2, 3. \tag{21}$$

For MK01-10, two state-of-the-art algorithms, Bagheri *et al.* [18] and Xing *et al.* [30] are selected to compare with the proposed algorithm. The reason of selecting these two algorithms is that they have strong searching capability and give three objective values rather than other popular algorithms' only giving one value. Because the number of nondominated solutions obtained by each algorithm is usually large, it is difficult to judge the algorithms' superiority by the number of solutions. In order to verify the superiority of the proposed algorithm, like the methods used in other papers, this article only compares optimal solution with minimum makespan.

Table 7 lists the optimal solutions with the minimum makespan of each algorithm. In this table, *LB* is the best-known lower bound of makespan [32]. *Dev* is the relative deviation, calculated by Eq. (22), where C_{pro} is the best makespan obtained by the proposed algorithm, C_{par} is the best makespan of the algorithm that compares to the proposed algorithm.

According to *Dev* of this table, the proposed algorithm's result is better than Bagheri *et al.*'s in five problem instances (MK05, MK06, MK07, MK09 and MK10) and is better than Xing *et al.*'s in eight problem instances (M01-MK10 except MK03 and MK08). For all optimal solutions of each algorithm in ten problem instances, metric *C* is used to evaluate the performance of the proposed algorithm, and the calculating result is shown in Table 8, where *P* represents the proposed algorithm.

$$Dev = \left[\left(C_{par} - C_{pro} \right) / C_{par} \right] \times 100\%. \tag{22}$$

Table 6 The calculating results of *IGD*.

Size	*IGD*				
	Li and Pan	Li	Wang	Xing	Proposed
4 × 5	0.263523	0.458957	0.195434	0.195434	0.000000
8 × 8	0.286518	0.416689	0.203518	0.186852	0.120185
10 × 7	0.422531	0.388889	–	0.000000	0.000000
10 × 10	0.139754	0.257694	0.125000	0.139754	0.000000
15 × 10	0.642857	0.000000	0.357143	0.520008	0.000000

From Table 8, eight optimal solutions obtained by Bagheri *et al.* are dominated by some optimal solutions obtained by the proposed algorithm, and no optimal solution of the proposed algorithm is dominated by Bagheri *et al.*'s. Three optimal solutions obtained by Xing *et al.* are dominated by the proposed algorithm's, but only two of the proposed algortihm's are dominated by Xing *et al.*'s.

In order to further test the performance of the proposed algorithm, the 01a–18a problem instances were used. Two state-of-the-art algorithms, MG designed in [11] and BEG proposed in [33], are selected to compare with the proposed algorithm, because these two algorithms list all solutions in their literature. Metric *C* is used to evaluate the performance of the proposed algorithm and the calculation results are shown in Table 9.

Compared with MG, IEA has 10 instances better than MG, and MG has 3 better ones in all 18 instances. Compared with BEG, IEA has 16 better ones, while BEG has none. Therefore, it is also concluded that the proposed algorithm has superior solution performance.

5. CONCLUSIONS

MOFJSP is a kind of NP-hard problem. There are many excellent intelligent algorithms can solve the problem. For example, monarch butterfly optimization (MBO) [34,35], earthworm optimization algorithm (EWA) [36], elephant herding optimization (EHO) [37,38] and moth search (MS) [39,40]. In this paper, we put forward an adaptive multi-objective EA with two-stage local search for solving MOFJSP. We use a new encoding scheme, improve the initial population generating method, design effective crossover and mutation operators to adapt the special chromosome structure, propose a new environment selection approach. In the selection of crossover operators, individual differences are considered, and different crossover operators are selected according to individual differences, which improve the adaptive ability of the algorithm. Meanwhile, the parameter setting adopts an adaptive mechanism, which reduces the complexity of parameter setting and further improve the algorithm adaptive capabilities. In order to improve the solution performance of the algorithm, a two-stage local search is

Table 7 The solving result for MK01-10.

Instance	n × m	LB	Bagheri *et al.*				Xing *et al.*				Proposed		
			F_1	*Dev*	F_2	F_3	F_1	*Dev*	F_2	F_3	F_1	F_2	F_3
MK01	10 × 6	36	40	0	36	171	42	+4.8	42	162	40	36	167
MK02	10 × 6	24	26	0	26	154	28	+7.1	28	155	26	26	154
MK03	15 × 8	204	204	0	204	1207	204	0	204	852	204	204	1092
MK04	15 × 8	48	60	0	60	403	68	+11.8	67	352	60	60	390
MK05	15 × 4	168	173	+0.6	173	686	177	+2.8	177	702	172	172	687
MK06	10 × 15	33	63	+9.5	56	470	75	+24	67	431	57	56	446
MK07	20 × 5	133	140	+0.7	140	695	150	+7.3	150	717	139	139	693
MK08	20 × 10	523	523	0	523	2723	523	0	523	2524	523	523	2629
MK09	20 × 10	299	312	+1.6	306	2591	311	+1.3	299	2374	307	301	2560
MK10	20 × 15	165	214	+0.9	206	2121	227	+6.6	221	1989	212	199	1992

Table 8 The number of being dominated solutions.

Item	Bagheri *et al.* (Ba)		Xing *et al.* (X)	
	C (P, Ba)	*C* (Ba, P)	*C* (P, X)	*C* (X, P)
Value	0.8	0.0	0.3	0.2

Table 9 The calculating results of metric C for 01a–18a.

Item	*C*			
	(MG, P)	(P, MG)	(BEG, P)	(P, BEG)
01a	0.000000	0.666667	0.000000	0.666667
02a	0.000000	0.000000	0.000000	0.333333
03a	0.000000	0.000000	0.428571	0.500000
04a	0.000000	0.250000	0.000000	0.888889
05a	0.000000	0.200000	0.000000	1.000000
06a	0.000000	0.428571	0.013699	0.600000
07a	0.000000	0.000000	0.000000	0.600000
08a	0.333333	0.000000	0.000000	0.666667
09a	0.750000	0.000000	0.000000	0.333333
10a	0.000000	0.125000	0.000000	0.833333
11a	0.000000	0.428571	0.000000	0.000000
12a	0.000000	0.100000	0.000000	0.000000
13a	0.000000	0.000000	0.000001	1.000000
14a	0.000000	0.000000	0.000000	0.500000
15a	0.600000	0.000000	0.100000	0.833333
16a	0.000000	0.222222	0.000000	1.000000
17a	0.000000	0.666667	0.000000	0.615385
18a	0.000000	0.666667	0.000000	1.000000

designed. The numerical experiments and comparison indicate that the proposed algorithm is effective and a few parameters need to be set.

For future work, we will focus on adaptive scheduling based on machine learning and big data mining to increase the intelligent of algorithm.

CONFLICTS OF INTEREST

The authors declare that they have no competing interests.

AUTHORS' CONTRIBUTIONS

The study was conceived and designed by Yingli Li and Jiahai Wang and experiments performed by Zhengwei Liu. All authors read and approved the manuscript.

ACKNOWLEDGMENTS

This research is supported by the National Key R&D Program of China—Construction, Reference Implementation and Verification Platform of Reconfigurable Intelligent Production System (Grant No.2017YFE0101400).

REFERENCES

[1] H.J. Ding, X.S. Gu, Improved particle swarm optimization algorithm based novel encoding and decoding schemes for flexible job shop scheduling problem, Comput. Oper. Res. 121 (2020), 104951.

[2] D. Gao, G.G. Wang, W. Pedrycz, Solving fuzzy job-shop scheduling problem using DE algorithm improved by a selection mechanism, IEEE Trans. Fuzzy Syst. (2020).

[3] G.H. Zhang, Y.F. Hu, J.H. Sun, W.Q. Zhang, An improved genetic algorithm for the flexible job shop scheduling problem with multiple time constraints, Swarm Evol. Comput. 54 (2020), 100664.

[4] C. Soto, B. Dorronsoro, H. Fraire, L. Cruz-Reyes, C. Gomez-Santillan, N. Rangel, Solving the multi-objective flexible job shop scheduling problem with a novel parallel branch and bound algorithm, Swarm Evol. Comput. 53 (2020), 100632.

[5] P. Brucker, R. Schlie, Job-shop scheduling with multi-purpose machines, Computing. 45 (1990), 369–375.

[6] P. Brandimarte, Routing and scheduling in a flexible job shop by tabu search, Ann. Oper. Res. 41 (1993), 157–183.

[7] W.J. Xia, Z.M. Wu, An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems, Comput. Ind. Eng. 48 (2005), 409–425.

[8] G.H. Zhang, X.Y. Shao, P.G. Li, L. Gao, An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem, Comput. Ind. Eng. 56 (2009), 1309–1318.

[9] A. Baykasoğlu, L. Özbakir, A. Sönmez, Using multiple objective tabu search and grammars to model and solve multi-objective flexible job shop scheduling problems, J. Intell. Manuf. 15 (2004), 777–785.

[10] N.B. Ho, J.C. Tay, Solving multiple-objective flexible job shop problems by evolution and local search, IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. 38 (2008), 674–685.

[11] X.J. Wang, Gao L, C.Y. Zhang, X.Y. Shao, A multi-objective genetic algorithm based on immune and entropy principle for flexible

job-shop scheduling problem, Int. J. Adv. Manuf. Technol. 51 (2010), 757–767.
[12] G.L. Gong, Q.W. Deng, R.M. Chiong, X.R. Gong, H.Z.Y. Huang, An effective memetic algorithm for multi-objective job-shop scheduling, Knowl. Based Syst. 182 (2019), 104840.
[13] L. Wang, G. Zhou, Y. Xu, M. Liu, An enhanced Pareto-based artificial bee colony algorithm for the multi-objective flexible job-shop scheduling, Int. J. Adv. Manuf. Technol. 60 (2012), 1111–1123.
[14] H.C. Cheng, T.C. Chiang, L.C. Fu, A two-stage hybrid memetic algorithm for multiobjective job shop scheduling, Expert Syst. Appl. 38 (2011), 10983–10998.
[15] T.C. Chiang, H.J. Lin, A simple and effective evolutionary algorithm for multiobjective flexible job shop scheduling, Int. J. Prod. Econ. 141 (2013), 87–98.
[16] J. Gao, L.Y. Sun, M. Gen, A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems, Comput. Oper. Res. 35 (2008), 2892–2907.
[17] 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 (2011), 1159–1169.
[18] A. Bagheri, M. Zandieh, I. Mahdavi, M. Yazdani, An artificial immune algorithm for the flexible job-shop scheduling problem, Futur. Gener. Comput. Syst. 26 (2010), 533–541.
[19] F.M. Defersha, M.Y. Chen, A parallel genetic algorithm for a flexible job-shop scheduling problem with sequence dependent setups, Int. J. Adv. Manuf. Technol. 49 (2010), 263–279.
[20] F. Pezzella, G. Morganti, G. Ciaschetti, A genetic algorithm for the flexible job-shop scheduling problem, Comput. Oper. Res. 35 (2008), 3202–3212.
[21] L.P. Michael, Scheduling: Theory, Algorithms, and Systems, Springer, New York, NY, USA, 2018.
[22] 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, Inf. Sci. (Ny). 289 (2014), 76–90.
[23] J.Q. Li, Q.K. Pan, P.N. Suganthan, T.J. Chua, A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem, Int. J. Adv. Manuf. Technol. 52 (2011), 683–697.
[24] K. Deb, S. Agrawal, A. Pratap, T. Meyarivan, A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II, in: M. Schoenauer, *et al.* (Eds.), Parallel Problem Solving from Nature PPSN VI, PPSN 2000, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, Germany, 2000, pp. 849–858.
[25] A. Teymourifar, G. Ozturk, O. Bahadir, A Comparison between two modified NSGA-II algorithms for solving the multi-objective flexible job shop scheduling problem, Univers. J. Appl. Math. 6 (2018), 79–93.
[26] P. Fattahi, M. Saidi Mehrabad, F. Jolai, Mathematical modeling and heuristic approaches to flexible job shop scheduling problems, J. Intell. Manuf. 18 (2007), 331–342.
[27] I. Kacem, S. Hammadi, P. Borne, Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic, Math. Comput. Simul. 60 (2002), 245–276.
[28] S. Dauzère-Pérès, J. Paulli, An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search, Ann. Oper. Res. 70 (1997), 281–306.
[29] J.Q. Li, Q.K. Pan, Y.C. Liang, An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems, Comput. Ind. Eng. 59 (2010), 647–662.
[30] L.N. Xing, Y.W. Chen, K.W. Yang, An efficient search method for multi-objective flexible job shop scheduling problems, J. Intell. Manuf. 20 (2009), 283–293.
[31] 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 (2002), 1–13.
[32] M. Mastrolilli, L.M. Gambardella, Effective neighbourhood functions for the flexible job shop problem, J. Sched. 3 (2000), 3–20.
[33] Q.W. Deng, G.L. Gong, X.R. Gong, L.K. Zhang, W. Liu, Q.H. Ren, A bee evolutionary guiding nondominated sorting genetic algorithm ii for multiobjective flexible job-shop scheduling, Comput. Intell. Neurosci. 2017 (2017), 1–20.
[34] Y.H. Feng, X. Yu, G.G. Wang, A novel monarch butterfly optimization with global position updating operator for large-scale 0-1 Knapsack problems, Mathematics. 7 (2019), 1056.
[35] Y.H. Feng, G.G. Wang, W.B. Li, N. Li, Multi-strategy monarch butterfly optimization algorithm for discounted {0-1} Knapsack problem, Neural Comput. Appl. 30 (2018), 3019–3036.
[36] G.G. Wang, S. Deb, L.D.S. Coelho, Earthworm optimization algorithm: a bio-inspired metaheuristic algorithm for global optimization problems, Int. J. Bio-Inspired Comput. 12 (2018), 1–22.
[37] G.G. Wang, S. Deb, L.D.S. Coelho, Elephant herding optimization, in 2015 3rd International Symposium on Computational and Business Intelligence (ISCBI), Bali, Indonesia, 2015, pp. 1–5.
[38] G.-G. Wang, S. Deb, X.-Z. Gao, L.D.S. Coelho, A new metaheuristic optimisation algorithm motivated by elephant herding behaviour, Int. J. Bio-Inspired Comput. 8 (2016), 394–409.
[39] G.G. Wang, Moth search algorithm: a bio-inspired metaheuristic algorithm for global optimization problems, Memetic Comput. 10 (2018), 151–164.
[40] Y.H. Feng, G.G. Wang, Binary moth search algorithm for discounted {0-1} Knapsack problem, IEEE Access. 6 (2018), 10708–10719.