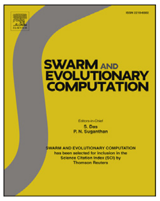
Swarm and Evolutionary Computation 54 (2020) 100664

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

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



An improved genetic algorithm for the flexible job shop scheduling problem with multiple time constraints

Guohui Zhang [a,*], Yifan Hu [a], Jinghe Sun [a], Wenqiang Zhang [b]

[a] School of Management Engineering, Zhengzhou University of Aeronautics, China
[b] College of Information Science and Engineering, Henan University of Technology, China

ARTICLE INFO

Keywords:
Flexible job shop scheduling
Setup time
Transportation time
Genetic algorithm

ABSTRACT

The flexible job shop scheduling problem is a very important problem in factory scheduling. Most of existing researches only consider the processing time of each operation, however, jobs often require transporting to another machine for the next operation while machines often require setup to process the next job. In addition, the times associated with these steps increase the complexity of this problem. In this paper, the flexible job scheduling problem is solved that incorporates not only processing time but setup time and transportation time as well. After presenting the problem, an improved genetic algorithm is proposed to solve the problem, with the aim of minimizing the makespan time, minimizing total setup time, and minimizing total transportation time. In the improved genetic algorithm, initial solutions are generated through three different methods to improve the quality and diversity of the initial population. Then, a crossover method with artificial pairing is adopted to preserve good solutions and improve poor solutions effectively. In addition, an adaptive weight mechanism is applied to alter mutation probability and search ranges dynamically for individuals in the population. By a series of experiments with standard datasets, we demonstrate the validity of our approach and its strong performance.

1. Introduction

The flexible job shop scheduling problem (FJSP) is a new scheduling problem that expands the traditional job shop scheduling problem (JSP). The traditional problem presumes that a job can be performed reasonably on a machine under specific objective constraints. These assumptions are as follows: each machine can be setup in zero time; only one job can be performed at the same time; a operation cannot be stopped; each job has a defined operation sequence on a known machine; and the processing time of each operation on each machine is known. The FSJP adds machine selection to the traditional JSP, that is, jobs can be processed on different machines. Solving the FJSP is equivalent to solving two subproblems. The first is a machine selection problem for choosing suitable machines for different operations. The second is an operation sequencing problem for finding the optimal order for processing the jobs.

With the growing need to support custom manufacturing requests, small batch production is gradually becoming an important aspect of modern manufacturing enterprises, however, this kind of variable production for different types of increases the transportation time and setup time. In a steel furniture discrete production job shop, the actual production time and the planned time error caused by the scheduling scheme only considering the processing time state are large. The reason is that there is a lot of non-processing time in the production process, including transportation time and setup time. Therefore, the processing time, transportation time, and setup time are considered as independent time factors for FJSP in this paper. Also, an improved genetic algorithm is proposed to solve this problem. In order to speed up the convergence and improve the quality of the solution, we adopt three different ways to generate the initial solution, which not only improves the quality of the initial solution but also preserves its diversity. At the genetic operator stage, we design more effective crossover and mutation operators, the former one is to carry out the crossover by means of artificial crossover pairing, and the latter one is to acquire mutation probability and neighborhood search ranges by using an adaptive weight method.

Brucker and Schile [1] were the first to address this problem in 1990. Since then, many researchers have studied large-scale combinatorial optimization problems and proposed many methods in the research process. For example, Pezzella et al. [2] and Zhang et al. [3] applied genetic algorithm (GA) to solve the FJSP. Gao et al. [4] and Yazdani et al. [5] used variable neighborhood search (VNS) algorithm to solve the question. Xu et al. [6] used an ant colony algorithm (ACO) to solve FJSP. Li et al. [7] and Ho et al. [8] applied a cultural algorithm (CA) to solve

* Corresponding author.
E-mail address: zgh09@zua.edu.cn (G. Zhang).

https://doi.org/10.1016/j.swevo.2020.100664
Received 31 March 2019; Received in revised form 3 February 2020; Accepted 3 February 2020
Available online 6 February 2020

FJSP. Karimi et al. [9] presented an efficient knowledge-based algorithm for solving FJSP.

Nonetheless, a single algorithm easily falls into a locally optimal solution or fails to converge completely when solving complex problems involving large-scale combinatorial optimization. Therefore, many scholars use hybrid algorithms to solve FJSP. Yuan et al. [10] proposed a novel hybrid harmony search (HHS) algorithm for solving FJSP. Yuan and Xu [11] proposed a hybrid differential evolution (HDE) algorithm for solving FJSP. Shao et al. [12] used discrete water wave optimization algorithm (IMMBO) for blocking flow-shop scheduling problem. Meng et al. [13] used migrating birds optimization to solve an integrated lot-streaming flow shop scheduling problem. Gao et al. [14] and Zhang et al. [15] combined a GA and a VNS. Li et al. [16] solved the FJSP using a GA and Tabu search (TS).

The problem can be divided into single or multiple objective problems, the optimization goal used to minimize the makespan. For instance, Teekeng et al. [17] used minimization of the total time as a single optimization objective. Ham et al. [18] considered the scheduling problem in parallel processing and took makespan as the optimization objective. Nevertheless, research on multi-objective problems has become more mainstream and valuable. The total completion time and resource consumption were regarded as multiple optimization objectives by Lu et al. [19]. Zhang et al. [20] combined total time, total machine workload, and total energy consumption as multiple objectives. Hu and Wang [21] minimized the overall completion time, total machine workload, and the greatest machine workload as the objective. Gao et al. [22,23] used the weighted combination of two minimization criteria, total completion time and the average time of early or late delivery as the optimization objective, he also used the makespan the mean of earliness and tardiness as two objectives to optimize.

Over time, researchers have taken more realistic factors into account when solving the FJSP. Karimi et al. [24] incorporated the time required for a job to move between machines. Zandieh et al. [25] taken condition-based maintenance (CBM) into account, recognizing that machines might be unavailable due to preventive maintenance, basic maintenance, or unforeseen breakdowns. Shen et al. [26] addressed the problem with sequence-dependent setup times. Wang et al. [27] regarded fuzzy processing time as a factor. Zhang et al. [28] regarded transportation time in flexible job shop scheduling problems.

The remainder of the paper is organized as follows. In Section 2, we describe the flexible job shop problem with multiple time constraints in detail and give an example. The improved genetic algorithm is described in Section 3, including encoding, decoding, generation of an initial solution and the chromosome genetic operator in detail. In Section 4, we present our experiments and results of our algorithm on standard dataset and compare them. Section 5 presents our conclusions and suggestions for future research.

2. Problem description

The flexible job shop scheduling problem is an NP-hard problem. Traditionally, researchers have only considered processing time. We consider not only the processing time of the job on different machines but also the time for transporting jobs between machines and the time for setting up machines before processing a job in the actual scheduling process.

The problem is described as follows. N jobs need to be processed on M machines, and each job can contain one or more operations. Each operation can only be processed by one machine, which is any machine in the operation optional machine set. When one operation of the job is completed, it will be transferred to the next operation. If the two adjacent operations for the job use the same machine, there is no need to consider the transportation time. If two adjacent operations of the processing machine are the same (i.e., doing the same thing), there is no need to consider the setup time. Choosing suitable machines for different processing of different jobs and reasonably arranging the order of jobs can reduce the maximum job completion time, the total job transportation time, and the total setup time.

We consider the FJSP problem with multiple time constraints assuming the following conditions.

Table 1
Processing and setup times for an example instance.

Job	Operation	Processing time/Setup time			
		M_1	M_2	M_3	M_4
J_1	$O_{1,1}$	3/2	10/3	–	3/2
	$O_{1,2}$	–	4/1	5/3	2/1
J_2	$O_{2,1}$	–	7/3	5/2	–
	$O_{2,2}$	6/3	2/1	–	8/6
	$O_{2,3}$	–	6/2	4/3	2/1
J_3	$O_{3,1}$	4/3	–	2/1	–
	$O_{3,2}$	–	3/2	6/1	–

Table 2
Transportation times between different machines.

Machine	M_1	M_2	M_3	M_4
M_1	0	1	2	1
M_2	4	0	1	2
M_3	3	2	0	4
M_4	2	3	2	0

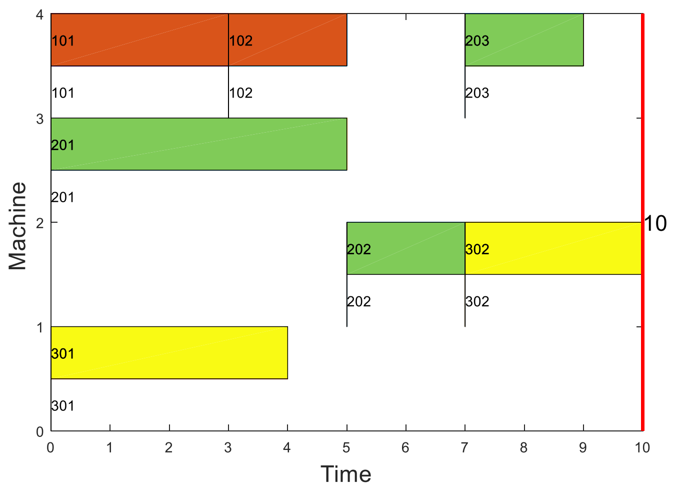


Fig. 1. The Gantt chart without transportation time and setup time.

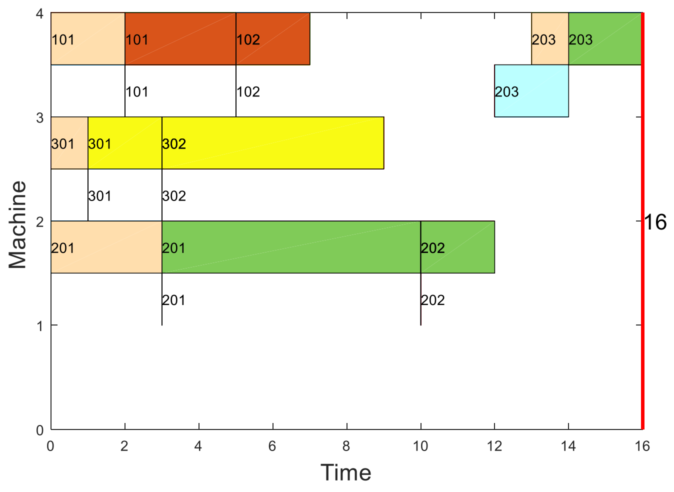


Fig. 2. The Gantt chart with transportation time and setup time.

(1) Each machine can process only one operation at a time.
(2) The same job can only be processed by one machine at the same time, and, once the job starts processing, the process cannot be interrupted.
(3) The processing sequence of each job has its own order. That is, after each operation is completed, it will be sent to the machine selected for the next operation.
(4) The processing time varies depending on the machine chosen. Processing times are known.
(5) When the same job travels between two adjacent processes on different machines, the transportation time varies with the machine used by the processes, and the transportation time is known. If two continuous processes of a job are processed on the same machine, the transportation time is 0.
(6) Different operations have different setup times because of the processing machines selected, with the machine setup times known. If two processes of a job are continuously processed on the same machine, the setup time is 0.

For convenience of description, the following symbols are defined:

$J = \{J_1, J_2, J_3, \ldots J_j, \ldots J_n\}$ represents the job set.
$M = \{M_1, M_2, M_3, \ldots M_i, \ldots M_m\}$ represents the machine set.
OM_{il} represents the l-th operation processed on machine i.
O_{jh} is the h-th operation of job j.
TJ_{ijh} denotes the time required for the h-th operation of job j to process on machine i.
SJ_{ijh} denotes the permissible start time of the h-th operation of job j on machine i.
SM_{il} denotes the permissible start time of the l-th operation of machine i.
SJM_{ijh} represents the actual processing start time of the h-th operation of job j on machine i.
CJ_{ijh} denotes the completion time of the h-th operation of job j on machine i.
CM_{il} denotes the end time of the l-th operation of machine i.
$setuptime_{jhm}$ denotes the setup time of the h-th operation of job j on machine m.
$transportation_{jhie}$ represents the transport time for job between machine i and machine e in the h-th process of job j.
CJ_j denotes the completion time of job j.

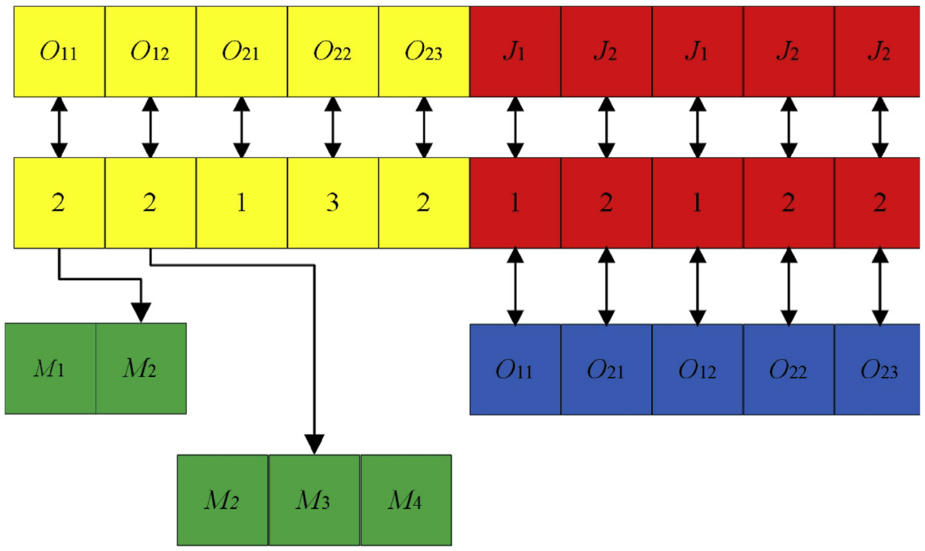


Fig. 3. A chromosome representation.

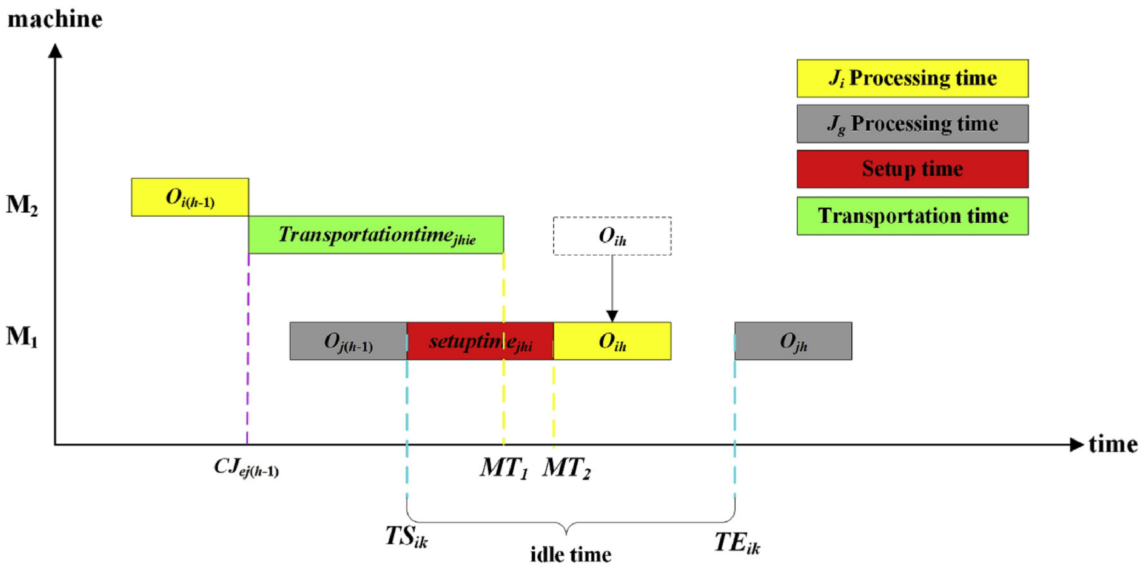


Fig. 4. Finding enough interval and inserting O_{ij}.

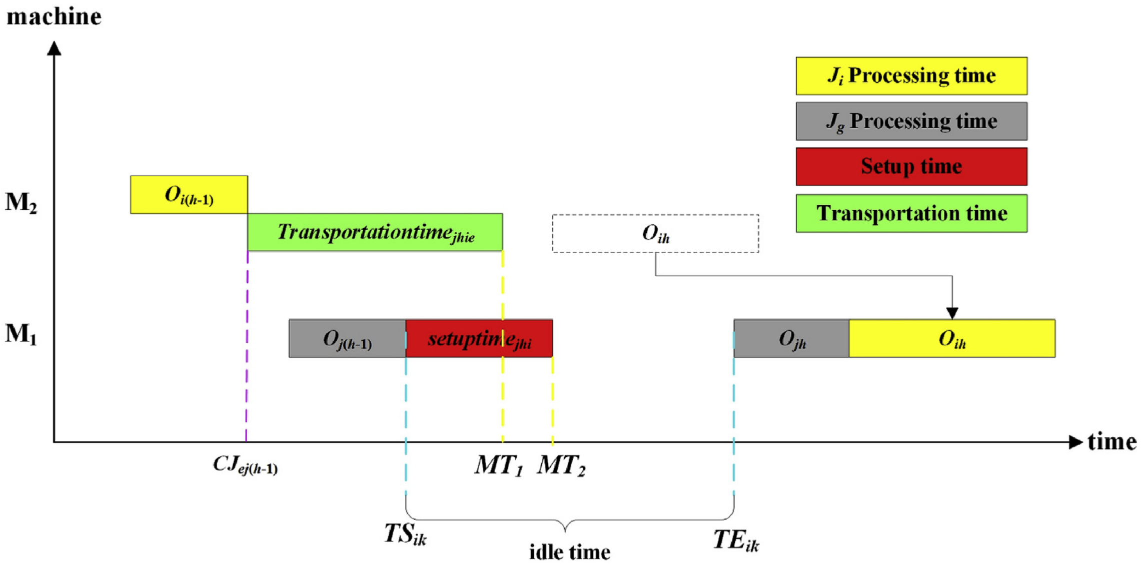


Fig. 5. Not finding enough interval and don't inserting O_{ij}.

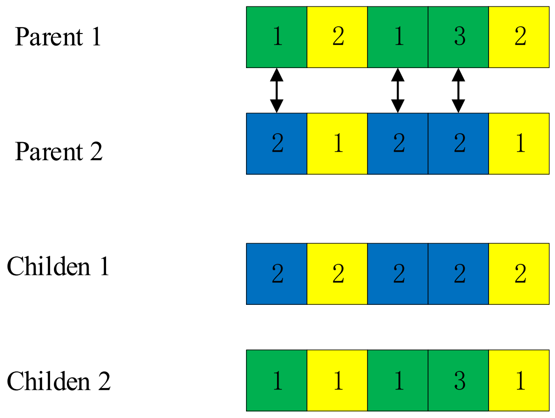


Fig. 6. Crossover operator for the machine selection part.

C denotes the Makespan time of all jobs.
TAT is the total setup time.
TMT is the total transportation time.

If the h-th operation of job j is the l-th operation of machine i, the optimization objectives are Eq. (1) minimizing the makespan, Eq. (2) minimizing the total transportation time, and Eq. (3) minimizing the total setup time:

$$C = \min\left(\max_{1 \le j \le n}(CJ_j)\right) \tag{1}$$

$$TAT = \min\left(\sum_{j=1}^{m}\sum_{h=1}^{m}\sum_{i=1}^{n} setuptime_{ijh}\right) \tag{2}$$

$$TMT = \min\left(\sum_{j=1}^{m}\sum_{h=1}^{m}\sum_{i=1}^{n} transportation_{jhie}\right) \tag{3}$$

For ease of understanding, we now provide an example of a partially flexible job shop scheduling problem. Table 1 represents the processing time and setup time of different jobs on different machines. "–" in the table indicates that the corresponding operation cannot be processed on the indicated machine.

Table 2 represents the transportation times for jobs between different machines, which are asymmetrical.

Fig. 1 shows a Gantt chart without the setup and transportation times, in which the yellow 301 indicates that the first operation job 2 requires 4 time units (x-axis) on machine 1 (y-axis). Fig. 2 shows another Gantt chart, this time including the setup and transportation times. The light pink 101 in front of the brown 101 requires 2 time units (x-axis) for the first operation of job 1 performed on machine 2 with a setup time of 3 time units. The green 203 has a light pink 203 in front and a light blue 203 below, representing the second operation of job 2 processed on machine 4. The setup time of the machine before processing is 1, and the time for transporting job 2 from machine 3 to machine 4 is 4.

Figs. 1 and 2 show that the Gantt chart with transportation time and setup time requires 6 more time units than the chart without transportation time and setup time. However, in the actual production process, the extra 6 is meaningful. If the transportation time and setup time are not taken into account, the plan reliability will be very low. If the transportation time and setup time are taken into account, the plan will

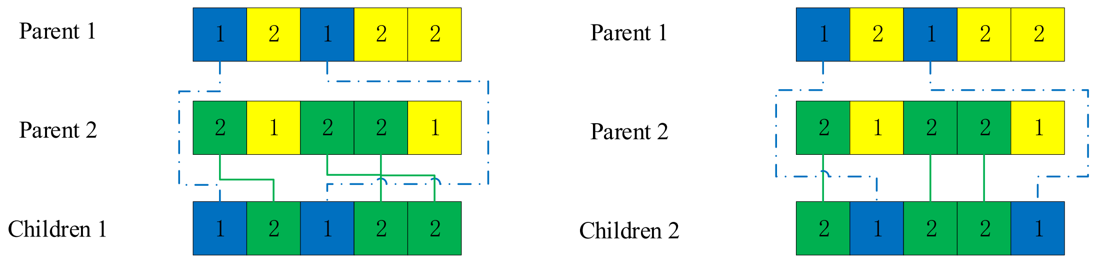


Fig. 7. Crossover operator for the operation sequence part.

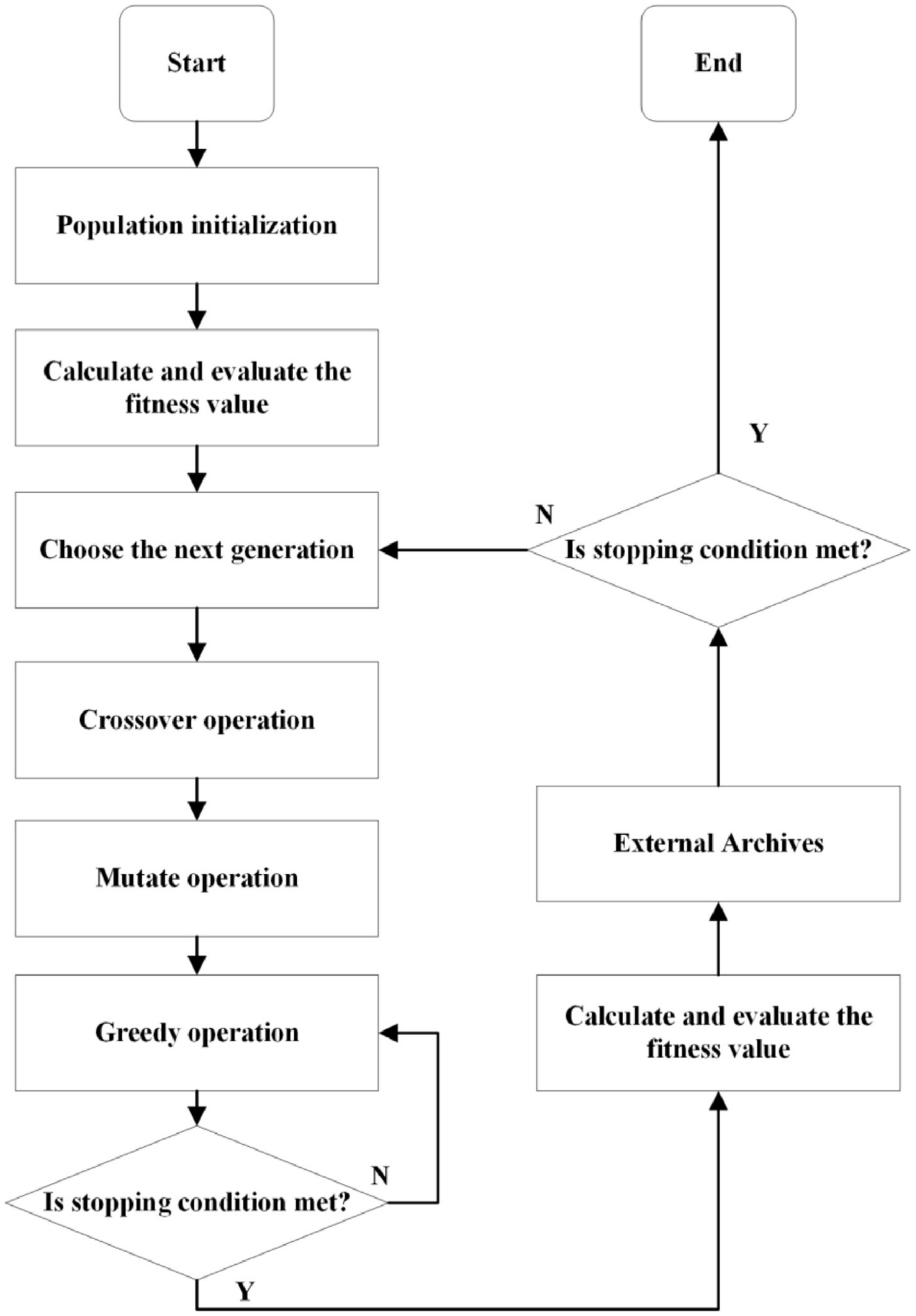

Start
End
Population initialization
Calculate and evaluate the fitness value
Y
N
Choose the next generation
Is stopping condition met?
Crossover operation
External Archives
Mutate operation
Greedy operation
Calculate and evaluate the fitness value
N
Is stopping condition met?
Y

Fig. 8. The framework of the proposed IGA algorithm.

Table 3
The parameter settings of IGA.

Parameter	Value
Number of individuals	100
Initial solution proportion of random generation method	0.4
Minimum time to select the proportion of initial solution	0.3
Initial solution proportion of maximum priority selection method for remaining processing time	0.3
Crossover probability	0.8
Minimum mutation probability	0.8
Maximum mutation probability	0.2

be an effective production guide.

3. Improved genetic algorithm for solving the FJSP with multiple time constraints

3.1. Chromosome coding

Coding is a method of transforming the feasible solutions of a given problem from the solution space to a search space that the algorithm can handle. Effective coding expresses the relationship between the individual and the feasible solution without producing illegal solutions. The FJSP contains two sub-problems: machine selection and sequencing. We use the machine selection sub-problem to identify which machine to perform each operation of the job. We use the operation sequencing sub-problem to determine the job processing order on the machine. In this paper, we code the two sub-problems on a chromosome to find a feasible FJSP solution.

(1) Machine selection: For machine selection, the chromosome length is the total number of operations. Gene positions are represented by integers, which are arranged from left to right according to the job processing sequence. Each integer represents the sequence number of the current operation of the processing job in the optional machine set. Using Table 1 and the left half of Fig. 3 as an example, the gene string is 1-2-1-3-2 meaning that operation O_{11} is processed on the first machine in the optional machine set, which corresponds to actual processing machine M_1. Continuing, operation O_{12} is processed on the second machine in the optional machine set (actual machine $M_{3)}$, and so on.
(2) Operation sequence: For operation sequencing, the chromosome length is the total number of operations. Each gene is coded by the job number. The number of times the job number appears indicates the number of operations required by job. As shown in the right half of Fig. 3, the gene strand is 2-1-1-2-2, and the corresponding processing sequence is O_{11}–O_{21}–O_{12}–O_{22}–O_{23}.

3.2. Chromosome decoding

An effective decoding method leads the individual solution to produce better results, so we use full insertion decoding. As noted, chromosomes contain two parts for the two sub-problems of machine selection and operation sequencing. We first decode the machine selection reading the chromosomes for the machine from left to right and converting them to machine sequence matrix J_m, job processing time matrix T_1, and machine setup time matrix T_2. $J_{m(j,h)}$ denotes the machine number of the h-th processing operation of the job j; $T_{1(j,h)}$ denotes the h-th processing time of the job j; $T_{2(j,h)}$ denotes the setup time of the h-th processing machine of the job j; and $T_{3(j,h)}$ denotes the transportation time for the h-th processing step of the job j to the machine; $J_{m(j,h)}$. $T_{1(j,h)}$, $T_{2(j,h)}$, and $T_{3(j,h)}$ correspond to each other. We decode these as shown in Eqs. (4)–(7):

$$J_m = \begin{bmatrix} 1 & 3 & \\ 2 & 4 & 3 \end{bmatrix} \tag{4}$$

$$T_1 = \begin{bmatrix} 3 & 5 & \\ 7 & 8 & 4 \end{bmatrix} \tag{5}$$

$$T_2 = \begin{bmatrix} 2 & 3 & \\ 3 & 6 & 3 \end{bmatrix} \tag{6}$$

$$T_3 = \begin{bmatrix} 0 & 2 & \\ 0 & 2 & 2 \end{bmatrix} \tag{7}$$

After decoding the machine part, we decode the operation sequence by reading the operation chromosomes from left to right. We obtain the scheduling result by sequencing the process according to the machine setup time and the job transportation time.

The sorting method is as follows:

- If operation O_{jh} is the first operation of job j and the first operation on machine i, it is processed immediately after the machine setup.
- If operation O_{jh} is not the first operation of job j, it takes place once operation $O_{j\ (h-1)}$ finishes, after O_{jh}'s arrival at machine i and machine i's setup.
- When O_{jh} is not the first operation of the job, and a job has already been processed on machine i, we must consider whether TSE_{ik} is sufficient for processing the next working operation, assuming that TS_{ik} and TE_{ik} mark the beginning and end of the idle time. When k jobs are processed on machine i, k-1 idle times result. At this time, it is necessary to determine whether the operation can be inserted forward. The main steps are as follows:

Step 1: Determine whether the earlier operation results in a sequence of two repeated operations of the same job, which would result in a setup time of 0.

Step 2: Compute MT_1 (the starting time TS_{ik} plus the setup time $Setuptime_{jhi}$) and MT_2 (the end time $CJ_{ej(h-1)}$ plus the time $Transportationtime_{jhie}$) and use the larger value. If the ending time is less than the idle time TE_{ik}, the insertion is satisfied (Fig. 4). and if not, the next idle period is considered. If no idle time satisfies the insertion condition, the operation O_{jh} is placed after the scheduled operation (Fig. 5).

In this paper, the individual fitness value is calculated by normalized summation. The specific implementation steps are as follows:

Table 4
Processing and setup times for the 4 × 5 instance.

Job	$O_{i,j}$	Processing time/Setup time				
		M_1	M_2	M_3	M_4	M_5
J_1	$O_{1,1}$	2/1	5/3	4/1	1/1	2/2
	$O_{1,2}$	5/3	4/1	5/2	7/6	5/1
	$O_{1,3}$	4/1	5/4	5/4	4/2	5/2
J_2	$O_{2,1}$	2/2	5/4	4/1	7/5	8/7
	$O_{2,2}$	5/1	6/6	9/7	8/4	5/3
	$O_{2,3}$	4/2	5/3	4/3	54/43	5/4
J_3	$O_{3,1}$	9/4	8/7	6/4	7/3	9/9
	$O_{3,2}$	6/6	1/1	2/2	5/3	4/1
	$O_{3,3}$	2/1	5/5	4/1	2/1	4/2
	$O_{3,4}$	4/4	5/1	2/2	1/1	5/3
J_4	$O_{4,1}$	1/1	5/1	2/1	4/8	12/8
	$O_{4,2}$	5/4	1/1	2/1	1/1	2/1

Table 5
Transportation times for the 4 × 5 instance.

Machine	M_1	M_2	M_3	M_4	M_5
M_1	0	4	2	1	5
M_2	1	0	5	1	5
M_3	1	3	0	2	3
M_4	3	1	3	0	1
M_5	4	2	5	3	0

Table 6
The non-dominated solutions for the 4 × 5 instance.

	Makespan	Total setup time	Total transportation time
Solution 1	18	8	4
Solution 2	21	10	3
Solution 3	20	11	3
Solution 4	18	12	2
Solution 5	21	11	2
Solution 6	22	7	1
Solution 7	25	7	0

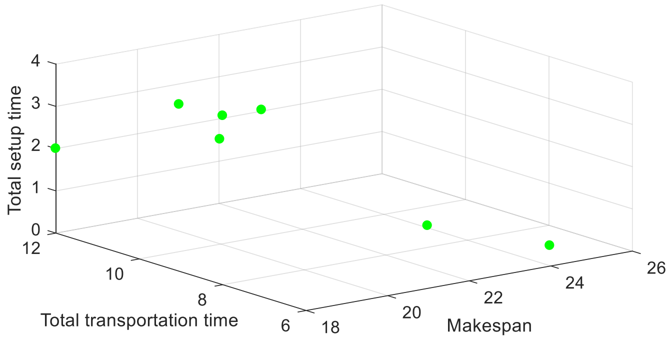


Fig. 9. Distribution of the non-dominant solutions for the 4 × 5 instance.

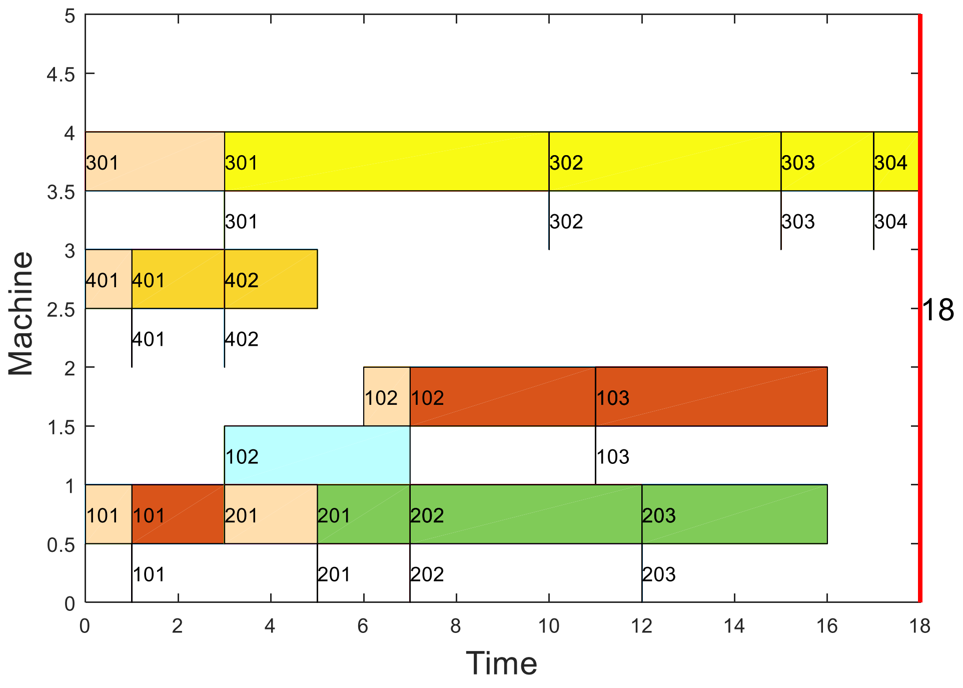


Fig. 10. The Gantt chart of Solution 1 for the 4 × 5 instance.

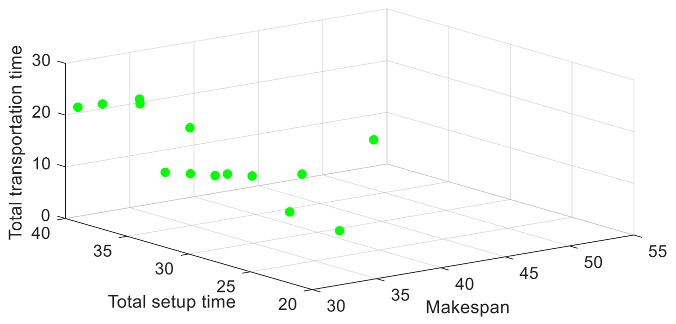


Fig. 11. Distribution of the non-dominant solutions for the 8 × 8 instance.

Table 7
The results for the 4 × 5 instance.

	Elitism GA	Enhanced GA	MOGWO	IGA
Number	2	3	5	7
BM	18	23	16	18
BTS	10	7	8	7
BTT	0	0	0	0

(1) Decode the population to obtain the makespan, the total transportation time and the total setup time of all individuals, and normalize these three indicators respectively;
(2) Each individual takes the sum of these three indicators as the value f_i. The smaller f_i, the better the individual quality;
(3) Rank all individuals in order of value f_i from smallest to largest, and the rank of individuals with the same value f_i is the same. The higher the rank n_i is, the higher the fitness value F_i is, the upper limit of fitness value a is equal to max (n) parts according to the rank max (n) of the worst individual, therefore, the individual fitness value $F_i = a - \frac{a}{\max(n)}(i-1)$. The value range of F_i is [0,a], and the higher the value of F_i, the higher the individual quality.

Table 8
Processing and setup times for the 8 × 8 instance.

Job	$O_{i,j}$	Processing time/Setup time							
		M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
J_1	$O_{1,1}$	5/1	3/1	5/1	3/3	3/3	–	10/1	9/7
	$O_{1,2}$	10/4	–	5/2	8/6	3/3	9/3	9/3	6/1
	$O_{1,3}$	–	10/2	–	5/2	6/3	2/1	4/1	5/4
J_2	$O_{2,1}$	5/4	7/2	3/1	9/5	8/1	–	9/5	–
	$O_{2,2}$	–	8/2	5/5	2/1	6/3	7/7	10/8	9/4
	$O_{2,3}$	–	10/1	–	5/2	6/3	4/3	1/1	7/2
	$O_{2,4}$	10/1	8/3	9/3	6/3	4/3	7/5	–	–
J_3	$O_{3,1}$	10/3	–	–	7/1	6/5	5/3	2/2	4/2
	$O_{3,2}$	–	10/1	6/2	4/4	8/6	9/3	10/8	–
	$O_{3,3}$	1/1	4/3	5/4	6/2	–	10/7	–	7/3
J_4	$O_{4,1}$	3/2	1/1	6/1	5/5	9/7	7/6	8/4	4/4
	$O_{4,2}$	12/6	11/7	7/1	8/3	10/6	5/5	6/5	9/3
	$O_{4,3}$	4/2	6/3	2/1	10/9	3/2	9/3	5/4	7/6
J_5	$O_{5,1}$	3/1	6/5	7/6	8/6	9/8	–	10/6	–
	$O_{5,2}$	10/7	–	7/1	4/2	9/1	8/5	6/2	–
	$O_{5,3}$	–	9/6	8/5	7/2	4/3	2/2	7/6	–
	$O_{5,4}$	11/1	9/4	–	6/6	7/2	5/4	3/2	6/6
J_6	$O_{6,1}$	6/2	7/6	1/1	4/3	6/1	9/2	–	10/3
	$O_{6,2}$	11/4	–	9/8	9/6	9/6	7/1	6/6	4/3
	$O_{6,3}$	10/5	5/5	9/2	10/2	11/8	–	10/4	–
J_7	$O_{7,1}$	5/3	4/4	2/1	6/4	7/7	–	10/9	–
	$O_{7,2}$	–	9/8	–	9/1	11/4	9/1	10/4	5/5
	$O_{7,3}$	–	8/1	9/6	3/1	8/4	6/1	–	10/8
J_8	$O_{8,1}$	2/1	8/3	5/5	9/3	–	4/2	–	10/8
	$O_{8,2}$	7/1	4/4	7/4	8/1	9/7	–	10/10	–
	$O_{8,3}$	9/2	9/8	–	8/2	5/4	6/3	7/2	1/1
	$O_{8,4}$	9/7	–	3/2	7/6	1/1	5/4	8/4	–

Table 9
Transportation times for the 8 × 8 instance.

Machine	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
M_1	0	4	2	1	5	1	2	1
M_2	1	0	5	1	5	2	5	5
M_3	1	3	0	2	3	4	5	5
M_4	3	1	3	0	1	4	3	3
M_5	4	2	5	3	0	1	4	1
M_6	5	5	1	1	2	0	3	2
M_7	1	1	2	5	2	5	0	3
M_8	3	5	1	5	5	4	2	0

Table 10
The non-dominated solutions for the 8 × 8 instance.

	Makespan	Total setup time	Total transportation time
Solution 1	31	40	21
Solution 2	54	40	5
Solution 3	33	33	21
Solution 4	32	36	24
Solution 5	31	35	26
Solution 6	31	38	23
Solution 7	34	32	12
Solution 8	34	31	13
Solution 9	33	35	11
Solution 10	34	34	11
Solution 11	35	26	16
Solution 12	34	29	14
Solution 13	36	24	6
Solution 14	35	27	8

Table 11
The results for the 8 × 8 instance.

	Elitism GA	Enhanced GA	MOGWO	IGA
Number	6	5	23	14
BM	39	38	31	31
BTS	36	33	28	24
BTT	16	7	7	5

Algorithm 1 provides the decoder's pseudocode.

Algorithm 1

```
input: Chromosome
output: Activity scheduling
1   Decoder machine selection step obtains $J_{m(j,h)}$, $T_{1(j,h)}$, $T_{2(j,h)}$, $T_{3(j,h)}$
2   for $j = 1: N$ (Decoder operation sequencing part)
3     for $i = 1: k$-1
4       if the operation before the idle and this operation are two continuous
        operations within the same job
5         $Setuptime_{jhm} = 0$;
6       end
7       $SJM_{ijh} = \max((TS_{ik} + Setuptime_{jhi}), (CJ_{ej(h-1)} + transportationtime_{jhie}))$
8       if $SJM_{ijh}TJ_{ijh} \le TE_{ik}$ (determine that insertion conditions are satisfied.)
9           Operation insertion can be performed
10          break
11      end
12    end
13  Not satisfying the insertion criteria, after the sorted jobs
14  end
```

(continued on next column)

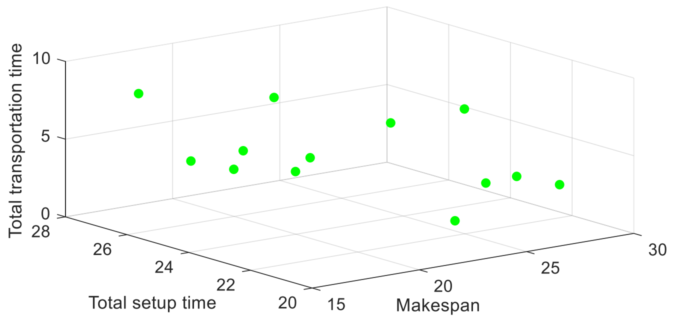


Fig. 13. Distribution of the non-dominant solutions for the 10 × 10 instance.

3.3. Initial population

Initialization of the population is an important problem in the algorithm as the initial solution plays an important role in the speed and quality of results. Therefore, when generating an initial solution, we should guarantee not only the diversity of solutions but also their quality. Diversity means that the solution space covered by the initial solution is large. Quality means that the initial solution generated is close to the optimal solution. In order to satisfy these two requirements, we generate the initial solution in three ways.

(1) Random Generation Method: For machine selection, we choose machines randomly from machines capable of performing the operation. For operation sequencing, we randomly generate the operations sequence.

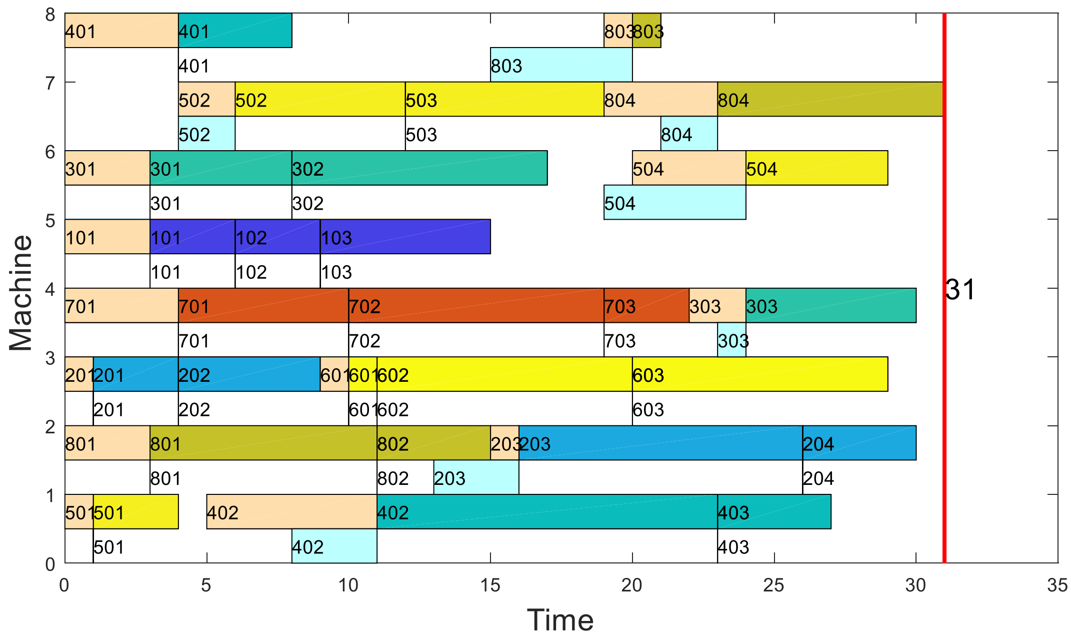


Fig. 12. The Gantt chart of Solution 1 for the 8 × 8 instance.

Table 12
Processing and setup times for the 10 × 10 instance.

Job	$O_{i,j}$	Processing time/Setup time									
		M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
J_1	$O_{1,1}$	1/1	4/2	6/1	9/6	3/1	5/3	2/2	8/5	9/9	5/5
	$O_{1,2}$	4/1	1/1	1/1	3/1	4/5	8/3	10/	4/1	11/11	4/3
	$O_{1,3}$	3/3	2/1	5/1	1/1	5/4	6/6	9/6	5/3	10/3	3/3
J_2	$O_{2,1}$	2/2	10/3	4/4	5/3	9/5	8/1	4/2	15/1	8/4	4/4
	$O_{2,2}$	4/2	8/8	7/6	1/1	9/9	6/5	1/1	10/6	7/3	1/1
	$O_{2,3}$	6/6	11/10	2/2	7/6	5/1	3/3	5/1	14/7	9/6	2/1
J_3	$O_{3,1}$	8/1	5/3	8/7	9/1	4/2	3/3	5/3	3/1	8/1	1/1
	$O_{3,2}$	9/7	3/3	6/3	1/1	2/1	6/1	4/1	1/1	7/7	2/2
	$O_{3,3}$	7/2	1/1	8/6	5/5	4/3	9/2	1/1	2/2	3/1	4/1
J_4	$O_{4,1}$	5/5	10/8	6/5	4/1	9/6	5/2	1/1	7/5	1/1	6/5
	$O_{4,2}$	4/1	2/2	3/1	8/7	7/5	4/3	6/1	9/1	8/4	4/2
	$O_{4,3}$	7/2	3/2	12/8	1/1	6/5	5/2	8/4	3/3	5/4	2/2
J_5	$O_{5,1}$	7/2	10/1	4/2	5/2	6/4	3/3	5/1	15/7	2/2	6/3
	$O_{5,2}$	5/2	6/5	3/2	9/9	8/2	2/2	8/7	6/6	1/1	7/1
	$O_{5,3}$	6/6	1/1	4/2	1/1	10/9	4/1	3/2	11/11	13/8	9/6
J_6	$O_{6,1}$	8/5	9/1	10/1	8/8	4/1	2/2	7/6	8/2	3/2	10/10
	$O_{6,2}$	7/3	3/3	12/3	5/5	4/1	3/2	6/3	9/6	2/1	15/13
	$O_{6,3}$	4/1	7/4	3/3	6/6	3/2	4/3	1/1	5/5	1/1	11/7
J_7	$O_{7,1}$	1/1	7/3	8/5	3/1	4/2	9/2	4/4	13/6	10/7	7/4
	$O_{7,2}$	3/2	8/8	1/1	2/2	3/2	6/2	11/8	2/2	13/1	3/1
	$O_{7,3}$	5/2	4/1	2/2	1/1	2/2	1/1	8/6	14/1	5/5	7/2
J_8	$O_{8,1}$	5/3	7/4	11/2	3/3	2/2	9/5	8/1	5/2	12/5	8/6
	$O_{8,2}$	8/2	3/1	10/3	7/2	5/4	13/4	4/2	6/6	8/4	4/4
	$O_{8,3}$	6/3	2/1	13/13	5/3	4/4	3/2	5/4	7/2	9/2	5/3
J_9	$O_{9,1}$	3/3	9/8	1/1	3/1	8/2	1/1	6/1	7/4	5/1	4/2
	$O_{9,2}$	4/4	6/6	2/2	5/2	7/1	3/3	1/1	9/6	6/2	7/6
	$O_{9,3}$	8/1	5/2	4/2	8/3	6/1	1/1	2/1	3/3	10/4	12/11
J_{10}	$O_{10,1}$	4/2	3/3	1/1	6/3	7/4	1/1	2/1	6/4	20/19	6/1
	$O_{10,2}$	3/2	1/1	8/2	1/1	9/8	4/2	1/1	4/2	17/8	15/1
	$O_{10,3}$	9/1	2/1	4/3	2/2	3/1	5/3	2/2	4/3	10/1	23/20

Table 13
Transportation times for the 10 × 10 instance.

Machine	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
M_1	0	4	2	1	5	1	2	1	5	5
M_2	1	0	5	1	5	2	5	5	1	5
M_3	1	3	0	2	3	4	5	5	5	4
M_4	3	1	3	0	1	4	3	3	4	1
M_5	4	2	5	3	0	1	4	1	3	2
M_6	5	5	1	1	2	0	3	2	3	2
M_7	1	1	2	5	2	5	0	3	2	4
M_8	3	5	1	5	5	4	2	0	3	1
M_9	3	3	1	3	1	3	2	3	0	4
M_{10}	1	5	5	3	1	3	3	2	5	0

Table 14
The non-dominated solutions for the 10 × 10 instance.

	Makespan	Total setup time	Total transportation time
Solution 1	17	27	8
Solution 2	19	24	9
Solution 3	23	23	7
Solution 4	19	25	5
Solution 5	25	22	8
Solution 6	26	21	4
Solution 7	20	24	4
Solution 8	26	22	3
Solution 9	20	26	3
Solution 10	18	26	4
Solution 11	28	21	3
Solution 12	25	27	2
Solution 13	26	23	0

(2) Maximum Priority Selection Method for Remaining Processing Time: For machine selection, we choose randomly from a central selection of appropriate machines. For the operation sequence, we select the job with the largest remaining total processing time of the current operation.

Table 15
The results for the 10 × 10 instance.

	Elitism GA	Enhanced GA	MOGWO	IGA
Number	9	4	11	13
BM	25	22	24	17
BTS	33	36	33	21
BTT	13	8	11	0

(3) Uniform Distribution Method: (a) machine selection part. Firstly, Processing data of current problems are analyzed and the *RMi*、*RLi and RS*$_i$ of each machine are calculated. Secondly, a job number is generated randomly, and the machine is selected for the first operation without sorting. The *WMi* and *WLi* of each machine are calculated for the machine that can be selected for this operation. Then, the operation are allocated. A random number is generated, if greater than the probability set, then use roulette form to select the machine, and update the data. If it is less than

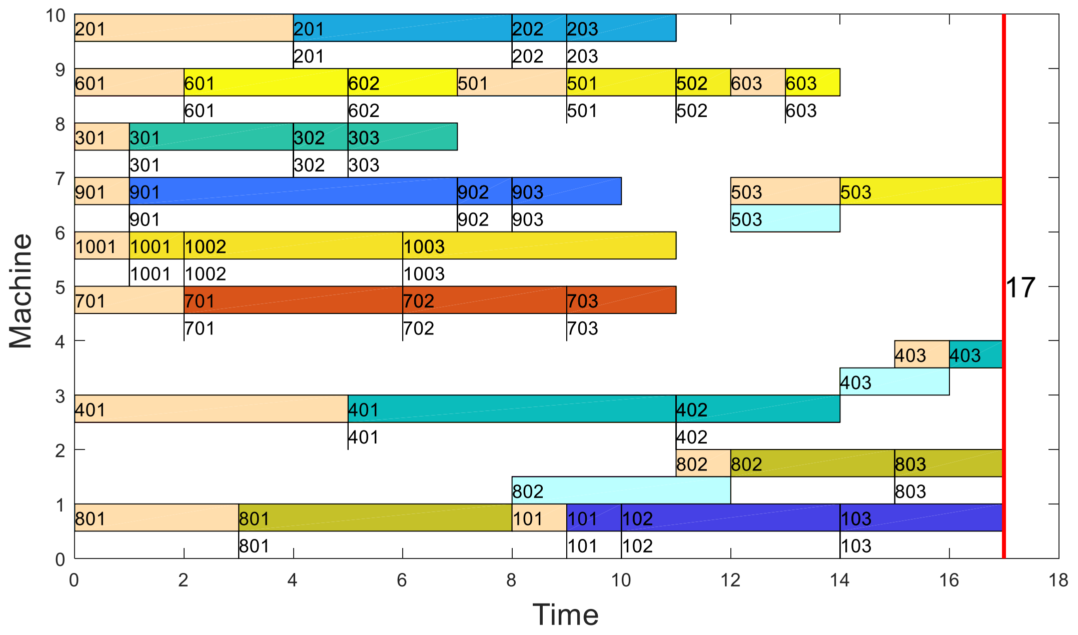


Fig. 14. The Gantt chart of Solution 1 for the 10 × 10 instance.

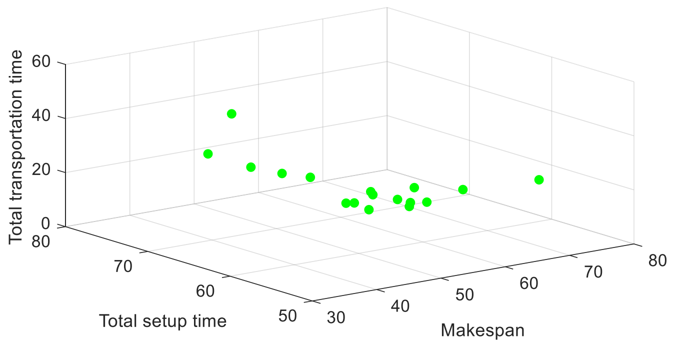


Fig. 15. Distribution of the non-dominant solutions for the 15 × 10 instance.

the set probability, choose the machine with the least *WLi.* If *WLi* is the same, choose the machine with the smallest *WMi.* (b) operation sequence part. Firstly, the *OPi* of each operation is calculated。Secondly, sort the operation of each machine from small to large according to the operation ratio.

Some of the variables are as follows:

RMi is maximum remaining processing time. The maximum remaining processing time is the sum of the time that the remaining operations can process on the machine.
RLi is minimum remaining processing time. The minimum remaining processing time is the sum of the remaining operations that can only be processed on the machine.
RS_i is the total time of the operations that have been selected for processing on this machine
WMi is processing time that $WMi = RMi + RS_i$
WLi is processing time that $WLi = RLi + RS_{i。}$
O_{jh} is the h-th operation of job j.
ORi is the operation ratio, $ORi = O_{jh}/O_{jn}$.

3.4. Crossover operator

The purpose of crossover is to preserve good information from the parent generation, generating new individuals through information exchange between the parents to reduce the probability of bad solutions and to search efficiently for new children.

Through many experiments, we have found that we obtain a better-quality individual by exchanging generations of two good individuals. Therefore, we introduce an artificial pairing mechanism as follows.

(1) Randomly select N individuals from the population.
(2) According to the individual's fitness, select two individuals from the N selected individuals by roulette wheel selection.
(3) Cross the selected two individuals.

Because chromosomes consist of two parts, we used different methods to cross the parts.

(1) Machine Selection: To ensure that the solution generated by this part is still feasible after crossover, we adopt a multi-point crossover operation. That is, we select multiple locations in the parent randomly and substitute (see Fig. 6).
(2) Operation Sequencing: We perform operation coding for the sequencing step. It is easy to generate infeasible solutions by crossing operations with traditional methods, so we use a precedence preserving order-based crossover (POX) to cross. That is, we perform cross-manipulation of multiple jobs in each chromosome to better integrate the best genes of the paternal individuals. The operation steps are as follows; Fig. 7 depicts the steps.
   1) Randomly divide the set of artifacts $J = \{J_1, J_2, J_3 \ldots, J_g \ldots J_n\}$ into two non-empty subsets, *Job01* and *Job02.*
   2) Then copy the genes of parental chromosomes P1 and P2 contained in the job set *Job01*/*Job02* to C1/C2 to ensure their position and sequence remain unchanged.
   3) Copy genes in P1/P2 not included in the job set *Job01*/*Job02* to C2/C1 to ensure their sequence remains unchanged.

3.5. Mutation operator

Mutation refers to altering part of an individual's genes to increase the diversity of solutions. Through many experiments, it could be found that mutation and neighborhood search for solutions with higher fitness often obtain better solutions. Therefore, an adaptive weights method is

Table 16
Processing and setup times for the 15 × 10 instance.

Job	$O_{i,j}$	Processing time/Setup time									
		M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
J_1	$O_{1,1}$	1/1	4/1	6/6	9/3	3/1	5/5	2/1	8/5	9/9	4/3
	$O_{1,2}$	1/1	1/1	3/3	4/1	8/6	10/.	4/2	11/7	4/3	3/2
	$O_{1,3}$	2/1	5/2	1/1	5/5	6/5	9/7	5/1	10/4	3/1	2/1
	$O_{1,4}$	10/1	4/3	5/1	9/7	8/1	4/3	15/7	8/6	4/1	4/3
J_2	$O_{2,1}$	4/3	8/6	7/5	1/1	9/1	6/2	1/1	10/10	7/5	1/1
	$O_{2,2}$	6/2	11/7	2/1	7/5	5/4	3/2	5/5	14/14	9/5	2/2
	$O_{2,3}$	8/1	5/2	8/4	9/1	4/3	3/3	5/2	3/3	8/1	1/1
	$O_{2,4}$	9/1	3/3	6/4	1/1	2/1	6/6	4/4	1/1	7/3	2/2
J_3	$O_{3,1}$	7/2	1/1	8/2	5/2	4/2	9/7	1/1	2/1	3/1	4/4
	$O_{3,2}$	5/1	10/9	6/4	4/4	9/8	5/5	1/1	7/7	1/1	6/2
	$O_{3,3}$	4/3	2/2	3/1	8/2	7/3	4/1	6/6	9/3	8/7	4/4
	$O_{3,4}$	7/7	3/3	12/8	1/1	6/5	5/5	8/5	3/2	5/2	2/2
J_4	$O_{4,1}$	6/6	2/2	5/4	4/2	1/1	2/1	3/1	6/2	5/5	4/2
	$O_{4,2}$	8/5	5/4	7/1	4/1	1/1	2/2	36/19	5/4	8/2	5/5
	$O_{4,3}$	9/3	6/6	2/1	4/1	5/3	1/1	3/2	6/2	5/4	2/1
	$O_{4,4}$	11/6	4/2	5/3	6/6	2/2	7/7	5/4	4/2	2/1	1/1
J_5	$O_{5,1}$	6/6	9/5	2/1	3/1	5/5	8/4	7/7	4/1	1/1	2/1
	$O_{5,2}$	5/5	4/3	6/1	3/1	5/1	2/1	28/1	7/4	4/1	5/1
	$O_{5,3}$	6/2	2/1	4/4	3/3	6/5	5/1	2/2	4/4	7/7	9/7
	$O_{5,4}$	6/5	5/4	4/2	2/1	3/3	2/2	5/4	4/2	7/3	5/2
J_6	$O_{6,1}$	4/3	1/1	3/3	2/2	6/1	9/9	8/7	5/4	4/1	2/2
	$O_{6,2}$	1/1	3/3	6/5	5/4	4/5	7/5	5/1	4/2	6/5	5/4
J_7	$O_{7,1}$	1/1	4/1	2/1	5/5	3/1	6/1	9/3	8/7	5/4	4/1
	$O_{7,2}$	2/2	1/1	4/3	5/2	2/2	3/1	5/4	4/1	2/1	5/3
J_8	$O_{8,1}$	2/1	3/1	6/6	2/1	5/2	4/3	1/1	5/5	8/7	7/6
	$O_{8,2}$	4/1	5/5	6/5	2/2	3/2	5/1	4/2	1/1	2/1	5/1
	$O_{8,3}$	3/3	5/5	4/1	2/2	5/1	49/40	8/5	5/5	4/1	5/1
	$O_{8,4}$	1/1	2/2	36/12	5/2	2/2	3/3	6/2	4/4	11/7	2/1
J_9	$O_{9,1}$	6/5	3/2	2/2	22/19	44/36	11/8	10/8	23/5	5/2	1/1
	$O_{9,2}$	2/2	3/2	2/1	12/6	15/11	10/8	12/5	14/10	18/2	16/5
	$O_{9,3}$	20/17	17/12	12/3	5/5	9/4	6/3	4/1	7/7	5/1	6/3
	$O_{9,4}$	9/3	8/1	7/1	4/3	5/2	8/8	7/3	4/4	56/47	2/2
J_{10}	$O_{10,1}$	5/2	8/6	7/2	4/3	56/29	3/3	2/2	5/1	4/2	1/1
	$O_{10,2}$	2/1	5/1	6/3	9/5	8/8	5/5	4/3	2/2	5/5	4/4
	$O_{10,3}$	6/2	3/2	2/2	5/5	4/3	7/3	4/4	5/4	2/1	1/1
	$O_{10,4}$	3/2	2/2	5/5	6/5	5/5	8/1	7/6	4/1	5/3	2/1
J_{11}	$O_{11,1}$	1/1	2/1	3/1	6/5	5/1	2/1	1/1	4/2	2/2	1/1
	$O_{11,2}$	2/1	3/2	6/4	3/1	2/2	1/1	4/1	10/5	12/5	1/1
	$O_{11,3}$	3/3	6/1	2/2	5/1	8/8	4/2	6/1	3/3	2/1	5/5
	$O_{11,4}$	4/4	1/1	45/16	6/6	2/2	4/3	1/1	25/14	2/2	4/3
J_{12}	$O_{12,1}$	9/8	8/5	5/2	6/4	3/2	6/1	5/5	2/2	4/2	2/2
	$O_{12,2}$	5/4	8/5	9/8	5/5	4/3	75/42	63/63	6/5	5/3	21/2
	$O_{12,3}$	12/11	5/4	4/1	6/1	3/2	2/2	5/1	4/1	2/1	5/5
	$O_{12,4}$	8/5	7/5	9/6	5/2	6/6	3/2	2/1	5/2	8/6	4/4
J_{13}	$O_{13,1}$	4/3	2/1	5/1	6/2	8/8	5/1	6/5	4/1	6/5	2/1
	$O_{13,2}$	3/2	5/5	4/1	7/6	5/4	8/4	6/5	6/5	3/2	2/1
	$O_{13,3}$	5/3	4/1	5/2	8/2	5/1	4/3	6/6	5/5	4/1	2/2
	$O_{13,4}$	3/1	2/2	5/3	6/2	5/5	4/2	8/2	5/4	6/5	4/4
J_{14}	$O_{14,1}$	2/1	3/2	5/3	4/2	6/5	5/3	4/3	85/59	4/2	5/2
	$O_{14,2}$	6/6	2/2	4/4	5/5	8/1	6/5	5/3	4/2	2/2	6/5
	$O_{14,3}$	3/1	25/9	4/2	8/2	5/5	6/2	3/2	2/1	5/1	4/3
	$O_{14,4}$	8/8	5/4	6/1	4/4	2/1	3/3	6/1	8/1	5/5	4/1
J_{15}	$O_{15,1}$	2/2	5/5	6/2	8/8	5/2	6/1	3/3	2/1	5/4	4/2
	$O_{15,2}$	5/4	6/5	2/1	5/4	4/1	2/1	5/4	3/2	2/2	5/5
	$O_{15,3}$	4/2	5/1	2/2	3/2	5/5	2/1	8/5	4/2	7/5	5/5
	$O_{15,4}$	6/4	2/2	11/2	14/2	2/1	3/2	6/5	5/5	4/4	8/8

Table 17
Transportation times for the 15 × 10 instance.

Machine	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
M_1	0	4	2	1	5	1	2	1	5	5
M_2	1	0	5	1	5	2	5	5	1	5
M_3	1	3	0	2	3	4	5	5	5	4
M_4	3	1	3	0	1	4	3	3	4	1
M_5	4	2	5	3	0	1	4	1	3	2
M_6	5	5	1	1	2	0	3	2	3	2
M_7	1	1	2	5	2	5	0	3	2	4
M_8	3	5	1	5	5	4	2	0	3	1
M_9	3	3	1	3	1	3	2	3	0	4
M_{10}	1	5	5	3	1	3	3	2	5	0

Table 18
The non-dominated solutions for the 15 × 10 instance.

	Makespan	Total setup time	Total transportation time
Solution 1	38	66	51
Solution 2	40	58	34
Solution 3	42	72	29
Solution 4	42	63	30
Solution 5	41	66	30
Solution 6	43	53	32
Solution 7	55	64	23
Solution 8	73	56	21
Solution 9	43	55	26
Solution 10	43	56	25
Solution 11	49	53	24
Solution 12	44	54	24
Solution 13	60	61	19
Solution 14	49	58	26
Solution 15	44	78	37
Solution 16	51	59	22
Solution 17	53	54	23
Solution 18	53	56	21
Solution 19	51	56	23
Solution 20	45	64	29
Solution 21	65	59	18

used to generate different mutation probabilities and neighborhood search ranges for different individuals. Specifically, we perform this according to Eq. (8):

$$P_m = \begin{cases} P_{\max} - \dfrac{(P_{\max} - P_{\min}) \times (F - F_{\min})}{F_{avg} - F_{\min}} & F < F_{avg} \\ P_{\max} & F > F_{avg} \end{cases} \tag{8}$$

where P_m denotes the individual's mutation probability, $P_{\max}$ and $P_{\min}$ denote the maximum and minimum crossover probabilities, respectively, F denotes the individual's fitness, $F_{\min}$ denotes the worst individual's fitness, and F_{avg} denotes the average fitness of the population. Better individual fitness increases the mutation probability of the individual, enlarges the search space of the individual, and avoids falling into a locally optimal solution. Worse individual fitness reduces the mutation probability of the individual and retains the better genes formed by the crossover between the individual and a better solution.

As with the crossover operation, we use different mutation methods depending on the component.

(1) Machine Selection: Using roulette wheel selection, we select machines with less processing time from a reduced set of suitable machines for each location of the machine part.
(2) Operation Sequencing: To improve the algorithm's efficiency, we allocate more search space to the better solution, giving the better solution a larger neighborhood search range, and adopt the method of adaptive neighborhood search to replace the process part. The steps of this operation are as follows.
  1) Sort individuals into five fitness levels. The higher the fitness, the higher the individual level. The neighborhood search range of the fifth level is 5, that is, 5 variation points. The neighborhood search range for level 4 individuals is 4, and so on.
  2) According to the generated neighborhood search range, we select the variation points randomly.
  3) Generate all permutations and combinations of neighborhood solutions and new neighborhood search solutions.
  4) Evaluate all neighborhood solutions and return the neighborhood solution with the highest fitness value to join the population.

3.6. *Greedy operation*

Based on the method in Ref. [29], a greedy operation is added to the algorithm to improve the individual quickly. The specific operation is as follows: first, we destroy the genes of the machine selection part and the corresponding process sequencing part of the individual, and then repair each gene according to the following rules. In the process of repair, each gene position is treated by different rules. Repair, and choose the best individual in the process of repair. Specific repair rules are as follows:

(1) Shortest processing time (SPT), selects the machine that does the operation within the shortest processing time.

Table 19
The results for the 15 × 10 instance.

	Elitism GA	Enhanced GA	MOGWO	IGA
Number	6	9	23	21
BM	47	47	45	38
BTS	94	84	73	53
BTT	55	44	46	18

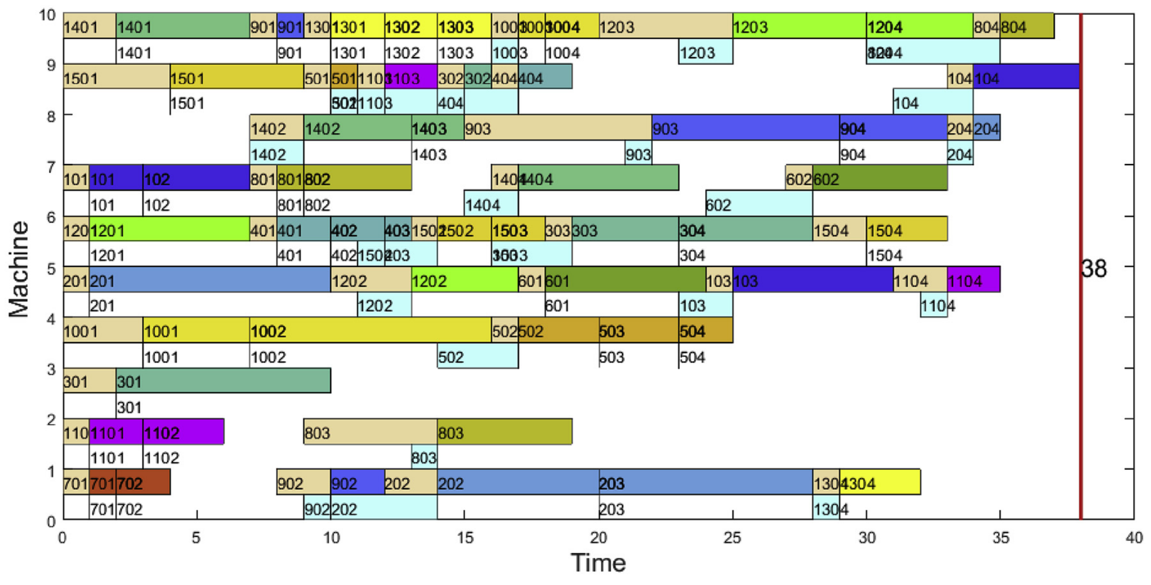


Fig. 16. The Gantt chart of Solution 1 for the 15 × 10 instance.

Table 20
The results for the Brandimarte dataset.

Problem	N*M	Elitism GA				Enhanced GA				MOGWO				IGA			
		Number	BM	BTS	BTT	Number	BM	BTS	BTT	Number	BM	BTS	BTT	Number	BM	BTS	BTT
MK01	10 × 6	11	82	81	72	22	68	73	67	35	70	69	57	27	65	75	63
MK02	10 × 6	15	63	78	54	8	57	56	41	25	61	65	40	12	52	62	42
MK03	15 × 8	16	350	543	259	21	330	463	224	36	322	496	214	20	313	462	220
MK04	15 × 8	13	120	149	125	21	112	147	119	47	106	157	102	17	107	147	110
MK05	15 × 4	14	314	304	101	8	284	279	90	35	287	284	87	25	276	255	83
MK06	10 × 15	21	174	285	255	16	158	262	257	46	159	255	217	39	129	248	245
MK07	20 × 5	13	311	370	124	19	265	292	115	40	255	263	68	50	267	265	83
MK08	20 × 10	21	805	1187	456	13	783	1182	444	53	792	1158	422	39	781	1138	446
MK09	20 × 10	13	615	1167	461	35	566	1014	458	30	548	1063	400	26	567	1094	433
MK10	20 × 15	28	474	1032	468	17	494	1063	467	24	457	990	400	59	424	1006	393

(2) Earliest start (ES), selects the machine that will start with this operation earlier.
(3) Earliest finish (EF), selects the machine that is able to finish the operation earlier.
(4) Least utilized machine (LUM), selects the machine that currently has the minimum workload.
(5) Minimum idle time (MIT), selects the machine that achieves the least idle time.
(6) Minimum gap per job (MGJ), selects the machine on which the gap between the operation and its preceding (the time that the job will be waiting for the machine) one is the smallest.

3.7. External archives

The addition of external archives records and preserves the non-dominant solutions obtained in the process of population evolution, avoiding the possibility of losing good individuals during cross-mutation. This is of great practical significance.

3.8. Selection operation

The purpose of the selection operation is to increase the probability of survival of the best individuals while preventing crossover, mutation, and other operations from damaging good genes and ensuring a constant population size to improve the computational efficiency and accelerate global convergence. The commonly used selection methods are rank-based selection, roulette wheel selection, seed selection, and tournament selection. The tournament selection method is used, which has better or equivalent convergence and computational complexity than the other selection operators. Each time, several individuals are selected from the population for fitness comparison, and those with high fitness are placed in the crossing pool until the crossing pool is filled.

3.9. Framework of the proposed IGA

The execution steps of the improved genetic algorithm for flexible job shop scheduling problem are as Fig. 8.

4. Experimental studies

The proposed algorithm was implemented in MATLAB on a computer with an Intel Core i3-8100 CPU at 3.6 GHz with 8 GB of RAM. In order to make the calculation results more meaningful, we ran each test question 10 times continuously when the algorithm was running. All the algorithm parameters are designed by orthogonal experiment to make the proposed algorithm more effective and obtain better solution. The parameters are shown in Table 3.

Our first set of experiments uses test data from Kacem et al. [30]. This set contains four test problems, which are tested in Experiment 1 (4 × 5), Experiment 2 (8 × 8), Experiment 3 (10 × 10), and Experiment 4 (15 × 10). The number of jobs in the problem varies from 4 to 15, and the number of machines varies from 5 to 10.

The second data set uses Brandimarte's data set [31], which consists of 10 problems. The number of jobs varies from 10 to 20, the number of machines varies from 4 to 15, and the number of operations for each job varies from 5 to 15.

These data only have processing time, so we randomly generated setup and transportation times according to processing time.

4.1. Experiment 1 (4 × 5)

First, we used small-scale data to test the effectiveness of the algorithm, as shown in Tables 4 and 5. The columns in Table 4 show the job, the operation, and the processing and setup times of the operation on each machine. For example, J_1 represents the first job, $O_{1,1}$ represents the first operation of the first job, $O_{2,1}$ represents the first operation of the second job, 2/1 represents the processing time of the first operation of the first job on machine 1 is 2, and the corresponding setup time is 1. The table reflects that, while different machines are suitable for the same operations, the corresponding processing and setup times can still vary.

Table 5 shows the transportation times for a job between different machines. For example, the transportation time between machine M_2 and machine M_1 is 8.

We found five non-dominated solutions in this experiment, as shown in Table 6.

Figs. 9, 11, 13 and 15 are non-dominant set graphs showing three-dimensional point graphs with X, Y, and Z indicating different targets. Each green point represents a non-dominated solution, and its corresponding value on the coordinate axis represents the value of the target.

Fig. 9 shows the distribution of the non-dominated solutions obtained for the 4 × 5 instance. Fig. 10 shows the Gantt chart for solution 1.

Tables 7, 11, 15, 19 and 20 show the results of our algorithm compared to other algorithms. Number denotes the number of non-dominant solutions found, BM denotes the best value of the makespan in the non-dominant solution set, BTS denotes the best value of the total setup time in the non-dominant solution set, and BTT denotes the best value of the total transparent time in the non-dominant solution set. The proposed IGA is compared with three algorithms reported in literature which are as follows: Elitism GA [32], Enhanced GA [33], MOGWO [34].

As can be seen from Table 7, Elitism GA finds two non-dominant solutions, Enhanced GA finds three non-dominant solutions, MOGWO finds five non-dominant solutions, and IGA finds seven non-dominant solutions. In the performance of finding non-dominant solutions, IGA is better than the other three algorithms. It can also be seen from the other three target values that our algorithm is superior to the Elitism GA and Enhanced GA. MOGWO is better in BM, but IGA is better in other targets.

4.2. Experiment 2 (8 × 8)

Experiment 2 involved medium-scale data with 8 jobs and 8 machines, as shown in Tables 8 and 9.

We found some non-dominated solutions in this experiment, as shown in Table 10.

Fig. 11 shows the distribution of the non-dominated solutions obtained for the 8 × 8 instance. Fig. 12 shows the Gantt diagram for Solution 1.

As can be seen from Table 10 and Fig. 11 the solution is distributed uniformly in space. From Table 11, we can see that the number of non-dominated solutions found by the four algorithms is more than that of previous experiments. That is, with the increase of the experimental data scale, the space of solutions becomes larger, and the number of non-dominated solutions is also increasing.

4.3. *Experiment 3 (10 × 10)*

We performed Experiment 3 with medium and large-scale data having 10 jobs and 10 machines, as shown in Tables 12 and 13.

We found 12 non-dominated solutions as shown in Table 14.

Fig. 13 shows the distribution of non-dominated solutions, and Fig. 14 shows the Gantt chart of Solution 1.

From Table 14 and Fig. 13, it is obvious that the non-dominant solutions are evenly distributed in space. The reason is that we adopt the normalization method to deal with the target and do not focus more on any target. From Table 15, we can see that the optimal value found by this algorithm on three objective values is better than the other three algorithms. The reason is that we add greedy operation, which makes the algorithm mining the neighborhood solution of the non-dominant solution more thoroughly. This makes the target value found better.

4.4. *Experiment 4 (15 × 10)*

We performed Experiment 4 with large-scale data including 15 jobs and 10 machines as shown in Tables 16 and 17.

After running the algorithm, we obtained some non-dominated solutions in Table 18.

The non-dominated sol1ution graph is shown in Fig. 15. The Gantt chart for Solution 1 is shown in Fig. 16.

From Table 19, we can see that IGA has found 21 non-dominant solutions on large data 15 × 10, which is more than the number of non-dominant solutions found in the previous three experiments. It proves the stability of the algorithm. That is, with the increasing of the solution space, the number of non-dominant solutions is also more. In the other three targets, the target values found by our algorithm are also better than those of the other three algorithms, which also prove the effectiveness of the algorithm.

4.5. *Experiment 5 (MK dataset)*

We then experimented with the Brandimarte dataset. This set contains 10 problems, with the number of jobs changing from 10 to 20, the number of machines changing from 6 to 15, and the number of jobs changing from 5 to 15, increasing the complexity of the problem. We used the processing times in the original dataset and randomly generated setup and transportation times according to the corresponding processing time.

In Table 20, the first column represents the problem in the MK series, the second column represents the number of jobs and machines in the test problem, and the remaining columns show grouped results for the Elitism GA, Enhanced GA, MOGWO, and IGA. Table 20 shows that, with the increasing complexity of the data, the number of non-dominated solutions found by the three algorithms increased overall.

The results from our proposed IGA are better than those of Elitism GA and Enhanced GA in most cases, and half of the three optimization targets are better than MOGWO. Although the number of non-dominated solutions found for some problems was less than that of the other three algorithms, the quality of the solutions found by IGA are better than that found by Elitism GA and Enhanced GA, and not weaker than that of MOGWO algorithm. That is to say, the solutions found by IGA could dominate the solutions found by Elitism GA and Enhanced GA to some extent. These results also confirm the validity of our proposed algorithm.

From these experiments, we can see that the algorithm proposed in this paper is very effective not only in finding the number of non-dominated solutions but also in terms of optimal solutions.

5. Conclusion and avenues for future research

In this paper, we have proposed an improved genetic algorithm for solving the flexible job shop scheduling model with multiple time constraints. In order to improve the quality of the initial solution, three different methods were proposed to generate the initial solution. To preserve good solutions and improve poor solutions quickly, we use a manual matching method to improve crossover operation. We apply an adaptive weight method to make different mutation probabilities of different individuals. A neighborhood search method with adaptive weight was presented to change the quality of the solution and make the search more efficient and faster. The computational results show that our proposed algorithm is effective when solving the FJSP with transportation time, setup time, and processing time constraints.

In the future, we would like to deeply explore the influence between the initial and final solutions and improve the effectiveness of crossover and mutation operations by adding individual fitness information into the genetic operator.

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.

Acknowledgements

This paper presents work funded by the National Natural Science Foundation of China (U1904167, 51705472, 71871204), Excellent Youth Foundation of Science & Technology Innovation of Henan Province (184100510001, 184100510004), Humanities and Social Sciences of Ministry of Education Planning Fund (18YJAZH125), Science and Technology Research Project of Henan Province (202102210131), Key Research Project in Universities of Henan Province (17A520030), and Postgraduate Education Innovation Program Fund of ZUA (2019CX17).

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.swevo.2020.100664.

References

[1] P. Brucker, R. Schile, Job-shop scheduling with multi-purpose machines, Computing 45 (4) (1990) 369–375.
[2] F. Pezzella, G. Morganti, G. Ciaschetti, A genetic algorithm for the flexible job-shop scheduling problem, Comput. Oper. Res. 35 (10) (2008) 3202–3212.
[3] G.H. Zhang, L. Gao, Y. Shi, An effective genetic algorithm for the flexible job-shop scheduling problem, Expert Syst. Appl. 38 (4) (2011) 3563–3573.
[4] 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.
[5] M. Yazdani, M. Amiri, M. Zandieh, Flexible job-shop scheduling with parallel variable neighborhood search algorithm, Expert Syst. Appl. 37 (1) (2010) 678–687.
[6] D.S. Xu, X.Y. Ai, L.N. Xing, An improved ant colony optimization for flexible job shop scheduling problems, J. Comput. Theor. Nanosci. 4 (6) (2009) 2127–2131.
[7] T.K. Li, W.L. Wang, W.X. Zhang, Solving flexible Job Shop scheduling problem based on cultural genetic algorithm, Comput. Integr. Manuf. Syst. 16 (4) (2010) 861–866.
[8] N.B. Ho, J.C. Tay, GENACE: an efficient cultural algorithm for solving the flexible job-shop problem, in: Proceedings of the 2004 Congress on Evolutionary Computation, vol. 2, OR, Portland, 2004, pp. 1759–1766. USA.
[9] H. Karimi, S.H.A. Rahmati, M. Zandieh, An efficient knowledge-based algorithm for the flexible job shop scheduling problem, Knowl. Base Syst. 36 (2012) 236–244.

[10] Y. Yuan, X. Hua, J. Yang, A hybrid harmony search algorithm for the flexible job shop scheduling problem, Appl. Soft Comput. J. 13 (7) (2013) 3259–3272.
[11] Y. Yuan, H. Xu, Flexible job shop scheduling using hybrid differential evolution algorithms, Comput. Ind. Eng. 65 (2) (2013) 246–260.
[12] Z. Shao, D. Pi, W. Shao, A novel discrete water wave optimization algorithm for blocking flow-shop scheduling problem with sequence-dependent setup times, Swarm Evol. Comput. 40 (2018) 53–75.
[13] T. Meng, Q.K. Pan, J.Q. Li, H.Y. Sang, An improved migrating birds optimization for an integrated lot-streaming flow shop scheduling problem, Swarm Evol. Comput. 38 (2018) 64–78.
[14] 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.
[15] G.H. Zhang, L.J. Zhang, X.H. Song, Y.C. Wang, C. Zhou, A variable neighborhood search based genetic algorithm for flexible job shop scheduling problem, Cluster Comput. (2018), https://doi.org/10.1007/s10586-017-1420-4.
[16] X.Y. Li, L. Gao, An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem, Int. J. Prod. Econ. 174 (2016) 93–110.
[17] W. Teekeng, A. Thammano, P. Unkaw, J. Kiatwuthiamorn, A new algorithm for flexible job-shop scheduling problem based on particle swarm optimization, Artif. Life Robot. 21 (1) (2016) 18–23.
[18] A. Ham, Flexible job shop scheduling problem for parallel batch processing machine with compatible job families, Appl. Math. Model. 45 (2017) 551–562.
[19] C. Lu, X. Li, L. Gao, W. Liao, J. Yi, An effective multi-objective discrete virus optimization algorithm for flexible job-shop scheduling problem with controllable processing times, Comput. Ind. Eng. 104 (2017) 156–174.
[20] Y. Zhang, J. Wang, Y. Liu, Game theory based real-time multi-objective flexible job shop scheduling considering environmental impact, J. Clean. Prod. 167 (2017) 665–679.
[21] N.P. Hu, P.L. Wang, An algorithm for solving flexible job shop scheduling problems based on multi-objective particle swarm optimization, in: Proceedings of the 2010 International Symposium on Information Science and Engineering, Shanghai, 2010, pp. 507–511.
[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. 289 (2014) 76–90.
[23] K.Z. Gao, P.N. Suganthan, Q.K. Pan, T.J. Chua, T.X. Cai, C.S. Chong, Discrete harmony search algorithm for flexible job shop scheduling problem with multiple objectives, J. Intell. Manuf. 27 (2) (2016) 363–374.
[24] S. Karimi, Z. Ardalan, B. Naderi, M. Mohammadi, Scheduling flexible job-shops with transportation times:Mathematical models and a hybrid imperialist competitive algorithm, Appl. Math. Model. 41 (2017) 667–682.
[25] M. Zandieh, A.R. Khatami, S.H.A. Rahmati, Flexible job shop scheduling under condition-based maintenance:Improved version of imperialist competitive algorithm, Appl. Soft Comput. 58 (2017) 449–464.
[26] L. Shen, Stéphane Dauzère-Pérès, J.S. Neufeld, Solving the flexible job shop scheduling problem with sequence-dependent setup times, Eur. J. Oper. Res. 265 (2018) 503–516.
[27] S. Wang, L. Wang, Y. Xu, M. Liu, An effective estimation of distribution algorithm for the flexible job-shop scheduling problem with fuzzy processing time, Int. J. Prod. Res. 51 (12) (2013) 3778–3793.
[28] G.H. Zhang, J.H. Sun, X. Liu, G.D. Wang, Y.Y. Yang, Solving flexible job shop scheduling problems with transportation time based on improved genetic algorithm, Math. Biosci. Eng. 16 (3) (2019) 1334–1347.
[29] G. Al Aqel, X.Y. Li, L. Gao, A modified iterated greedy algorithm for flexible job shop scheduling problem, Chin. J. Mech. Eng. 32 (2019) 1–11.
[30] I. Kacem, S. Hammadi, P. Borne, Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems, IEEE Trans. Syst. Man Cybern. 32 (1) (2002) 1–13.
[31] P. Brandimarte, Routing and scheduling in a flexible job shop by taboo search, Ann. Oper. Res. 41 (1993) 157–183.
[32] N.M. Xie, N.L. Chen, Flexible job shop scheduling problem with interval grey processing time, Appl. Soft Comput. 70 (2018) 513–524.
[33] M. Dai, D.B. Tang, A. Giret, M.A. Salido, Multi-objective optimization for energy-efficient flexible job shop scheduling problem with transportation constraints, Robot. Comput. Integrated Manuf. 59 (2019) 143–157.
[34] S. Luo, L.X. Zhang, Y.S. Fan, Energy-efficient scheduling for multi-objective flexible job shops with variable processing speeds by grey wolf optimization, J. Clean. Prod. 234 (2019) 1365–1384.