mathematics

MDPI

Article

Improved Self-Learning Genetic Algorithm for Solving Flexible Job Shop Scheduling

Ming Jiang [1,*], Haihan Yu [1] and Jiaqing Chen [2]

[1] School of Internet Economics and Business, Fujian University of Technology, Fuzhou 350014, China; 2218908004@mail.fjut.edu.cn
[2] School of Economics and Finance, Xi'an Jiaotong University, Xi'an 710061, China; xiamenvip@139.com
* Correspondence: 19842158@fjut.edu.cn

Abstract: The flexible job shop scheduling problem (FJSP), one of the core problems in the field of generative manufacturing process planning, has become a hotspot and a challenge in manufacturing production research. In this study, an improved self-learning genetic algorithm is proposed. The single mutation approach of the genetic algorithm was improved, while four mutation operators were designed on the basis of process coding and machine coding; their weights were updated and their selection mutation operators were adjusted according to the performance in the iterative process. Combined with the improved population initialization method and the optimized crossover strategy, the local search capability was enhanced, and the convergence speed was accelerated. The effectiveness and feasibility of the algorithm were verified by testing the benchmark arithmetic examples and numerical experiments.

Keywords: self-learning genetic algorithm; flexible job shop scheduling; self-learning variational strategy

MSC: 68T20

Citation: Jiang, M.; Yu, H.; Chen, J. Improved Self-Learning Genetic Algorithm for Solving Flexible Job Shop Scheduling. *Mathematics* **2023**, *11*, 4700. https://doi.org/10.3390/math11224700

Academic Editor: Jiuqiang Liu

Received: 7 October 2023
Revised: 6 November 2023
Accepted: 9 November 2023
Published: 20 November 2023

1. Introduction

The flexible job shop problem is an extension of the traditional job shop scheduling problem [1]. It allows an operation to be handled by any processing machine of a given set. The flexible shop floor scheduling problem was first proposed by Brucker [2]. The flexible job shop problem is one of the most important decision-making processes on the manufacturing floor. The flexible job shop scheduling problem (FJSP) is a hot issue in the current research in the field of intelligent manufacturing and industrial automation, which is widely used in discrete manufacturing and process industries. Solving the flexible workshop scheduling problem can help the enterprise to command, control, and regulate the effective resources reasonably and improve economic efficiency [3]. In conclusion, it is meaningful to explore and solve the FJSP.

Intelligent optimization algorithms currently in common use for solving FJSPs include the genetic algorithm, the ant colony optimization algorithm, the particle swarm optimization algorithm, the gray wolf algorithm, the taboo search algorithm, etc. With the introduction of the genetic algorithm (GA), the use of the GA for solving FJSPs has evolved. A study by Amjad et al. [4] showed that a large number of studies on the FJSP problem in recent years have used the GA for solving the problem, and in particular, its excellent global optimization capability and strong robustness make the GA one of the best algorithms for dealing with global problems. Xiaoyu Gao et al. proposed a genetic algorithm based on reinforcement learning that used Q-learning to self-learn crossover probabilities and improve the generalization ability of the genetic algorithm to achieve a solution to the large-scale flexible job shop scheduling problem [5].

Most of the current optimization algorithms are uncertain and random, and the FJSP features are weakly connected and have poor stability. Therefore, in this paper, using

the improved genetic algorithm to solve the flexible workshop scheduling problem, a self-learning mutation operator was designed, and four mutation schemes were designed for the characteristics of the flexible job shop scheduling problem. The feasibility and effectiveness of the proposed algorithm were optimized through algorithm testing and numerical experiments.

The rest of the paper is organized as follows. Section 2 presents the literature review. Section 3 describes the definition of the problem. In Section 4, the improved self-learning genetic algorithm is proposed. Section 5 analyzes the solution results of the improved self-learning genetic algorithm on various international benchmark instances of the FJSP. Section 6 summarizes the conclusions of this paper and presents the research directions for future work.

2. Literature Review

To solve the FJSP, scholars have proposed various algorithms from different perspectives. However, these algorithms have some common drawbacks, such as the large search space and inability to solve quickly.

Zhang Guohui et al. designed an initialization method combining global search, local search, and random generation to improve the quality of the initial solutions of populations and accelerate the convergence of genetic algorithms [6]; Taillard E D applied the forbidden search global optimization technique to the job shop scheduling problem, which provides an optimal solution in polynomial mean time in experiments but requires an exponential computational time in specific problems [7]; Holsapple C W et al. proposed a hybrid scheduling production algorithm based on artificial intelligence using genetic algorithms and domain knowledge fused with filtering algorithms for optimal searching [8]; Nouiri M et al. proposed an efficient particle swarm optimization algorithm to distribute PSOs into a multi-agent system (MAS) to decentralize the decision-making and involve each entity in the overall problem solution [9]. Caldeira R H et al. investigated a Pareto-based discrete Jaya algorithm to solve the flexible shop floor scheduling problem, incorporating problem-specific domain local search techniques into the approach and proposing a dynamic mutation operator and an improved congestion distance metric to enhance diversity in the search process [10]. Wagner J et al. combined the policy gradient algorithm with a participant–evaluator architecture and interpreted the production system as a multi-agent system, which experimentally generated plans that performed better than scheduling rules when tested in different production environments [11]. Guo Q. et al. designed a matrix coding-based approach to solve the flexible shop floor scheduling problem to improve the genetic algorithm [12]. Tang et al. proposed a hybrid algorithm by combining chaotic particle swarm optimization with the genetic algorithm in their paper [13]. Zhang Xiaoxing et al. proposed an improved hybrid frog jumping algorithm, which avoided illegal solution generation and the trimming of the algorithm and accelerated the algorithm's optimization rate by designing a local updating strategy based on the MPX (Maximum Preserved Crossover) operation and the gene shifting operation of the uniparental genetic algorithm [14].

Tengfei Zhang et al. discussed an improved genetic algorithm for solving the flexible job shop scheduling problem, which greatly improved the quality of the solution and the speed of obtaining the optimal solution by improving the intersection and mutation methods of generating machine and process codes, and they adopted a special method to prevent the algorithm from falling into local optimization in multiple stages [15]. Wang Jia-yi et al. improved the crossover operator and optimized the variation operator on the basis of the process from multiple parents and the crossover of the machine layer, and they confirmed the effectiveness of the algorithm after comparison [16]. Wang Su et al. combined the POS algorithm with the GA algorithm, borrowed the rewarming strategy in the SA algorithm, and replaced the population of a specific number of generations in the GA algorithm with the individual extreme value population generated by the PSO algorithm, which introduced the adaptive control factor and the permutation operation,

thus effectively solving the problem caused by the random generation of the initial population. Abd El-Wahed Khalifa et al. proposed a novel method called multistage fuzzy binding to solve the problem of job processing time, weight, and break-down machine characterized by piecewise quadratic fuzzy numbers [17]. Majumder et al. proposed a varying population genetic algorithm with indeterminate crossover (VPGAwIC). A new strategy for determining chromosome lifespan was defined. The age of each chromosome increases with the generations. A chromosome was considered dead and removed from the population if its age was greater than its lifespan [18].

Since the traditional genetic algorithm has low search efficiency and is not easy to converge when solving the flexible workshop scheduling problem, this paper constructed a flexible workshop scheduling model with the goal of minimizing the maximum completion time on the basis of the above research, combined with the actual situation and its characteristics in the scheduling process of the actual flexible production workshop. A combination of global selection, local selection, and random selection was adopted in the process of population initialization to generate high-quality initial populations in the initialization stage. A double-layer integer-coded chromosome was adopted; a process code and a machine code based on process arrangement constraints and machine selection were designed, respectively; a selection operation combining the roulette selection strategy and elite retention strategy was designed; a self-learning mutation strategy based on a scoring mechanism was designed; and four kinds of mutation schemes were designed for the mutation operation to improve the population searching ability; and finally, the updated optimal fitness values of the offspring and the parent generation were selected for the next generation. In this way, the solution space was searched effectively.

3. Problem Description

3.1. Flexible Shop Floor Scheduling Model

Flexible job shop scheduling is defined as N workpieces $J = \{J_1, J_2, J_3, \ldots\ldots, J_n\}$, which are processed sequentially on M machines $M = \{M_1, M_2, M_3, \ldots\ldots, M_m\}$; each workpiece J_i contains $O_{ij}(j = 1, 2, \ldots\ldots, N)$ processes, and the processing time of each process of each workpiece on each machine is known. The task of flexible job shop scheduling is to determine the order in which the workpieces are to be processed on each machine in order to optimize the specified performance metrics. The symbols are defined as shown in Table 1.

Table 1. Definition of symbols.

n	Total number of workpieces
m	Total number of machines
i, h	Workpiece number
j, k	Work sequence number
z	Machine number
O_{ij}	Workpiece i's j-th process
M_{ij}	Optional machines for process O_{ij}
S_{ij}	Start time of process O_{ij}
C_{ij}	Completion time of process O_i and process O_{ij}
makespani	Completion time of workpiece i
Makespan	Maximum completion time
X_{ijz}	Whether process O_{ij} is machined on machine z is a 0–1 variable, and machining on z is 1
G_{ijhk}	Sequence of process O_{ij} and process O_{hk}, as 0–1 variables, with ij preceded by 1
T_{ijz}	Machining time of process O_{ij} on machine z

3.2. Assumptions of the Problem

The basic assumptions for flexible job shop scheduling are as follows:

1. In the same workpiece process, there is a sequential constraint relationship.
2. The workpieces at the beginning of the moment can all be processed.

3. The workpiece can only be processed on one machine at the same time.
4. Each machine can be added to process any process, but at the same time, a machine can only process one process.
5. Until the workpiece processing is complete, the machine cannot be interrupted.
6. Each workpiece processing order is known.
7. The processing time of each workpiece process is determined by the corresponding processing machine.
8. Random factors, such as machine breakdowns, are not considered.

Because the flexible shop scheduling problem design's objective function is to minimize the maximum completion time, its flexible shop scheduling model is as follows:

$$\text{Makespan} = \min\{\max\{\text{makespan}_i\}\}$$

s.t.

$$C_{ij} - S_{ij} = T_{ijz} \tag{1}$$

$$\sum_{z=1}^{n} X_{ijz} = 1 \tag{2}$$

$$S_{ij} + X_{ijz} T_{ijz} < C_{ij} \tag{3}$$

$$C_{ij} \leq makespan \tag{4}$$

$$X_{hkz} C_{hk} + M(G_{hkij} - 1) \leq X_{ijz} S_{ij} \tag{5}$$

$$C_{ij} \leq S_{i(j+1)} \tag{6}$$

$$S_{ij} \geq 0, T_{ijz} \geq 0, C_{ij} \geq 0 \tag{7}$$

Equation (1) means that the process cannot be interrupted after the start of the process; Equation (2) means that any process is processed only once on the machine; Equation (3) means that the start time of any process is not greater than the completion time; Equation (4) means that the completion time of any process is not greater than the maximum completion time; Equation (5) means that only one process can be processed by each machine at the same time; Equation (6) means that the completion time of the previous process for any workpiece is not greater than the start time of the next process; and Equation (7) means that the start, processing, and completion times for any process are all non-negative.

4. Improving Self-Learning Genetic Algorithms

4.1. Traditional Genetic Algorithms

4.1.1. The Basic Idea of Traditional Genetic Algorithm

The genetic algorithms originated from computer simulation studies performed on biological systems. It is a stochastic global search and optimization method developed to mimic the mechanism of biological evolution in nature, drawing on Darwin's theory of evolution and Mendel's doctrine of genetics. Its essence is an efficient, parallel, global search method that automatically acquires and accumulates knowledge about the search space during the search process and adaptively controls the search process to find the best solution.

4.1.2. Traditional Genetic Algorithm Process

Its specific algorithmic flow is shown in Figure 1:

→ Step 1: Set the maximum number of iterations (Max_Itertions) to 0, set the maximum number of evolutionary generations to Max, and randomly generate Pop_size individuals as the initial population P.
→ Step 2: Calculate the fitness of each individual in population P.
→ Step 3: Apply the selection operator to the population. The purpose of selection is to pass optimized individuals directly to the next generation or produce new individuals by pairwise crossover and then pass them to the next generation. Selection operation is based on the evaluation of the fitness of the individuals in the population.
→ Step 4: Apply the crossover operator to the population. The crossover operator plays a central role in genetic algorithms.
→ Step 5: Apply the variation operator to the population. That is, it is a change in the gene values on certain loci of the individual strings in the population. The population P is selected and crossed over and the mutation operator is applied to obtain the next generation's population P(Max_Itertions + 1).
→ Step 6: Determine whether the conditions for the end of the algorithm are met; if they are not satisfied, continue to execute steps 2–5, and if they are satisfied, end the algorithm and output the results.

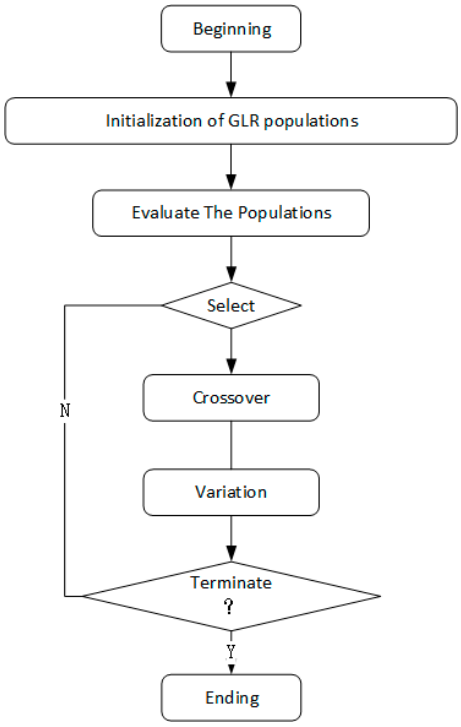

Figure 1. Flowchart of traditional genetic algorithm.

4.2. Improved Self-Learning Genetic Algorithm

4.2.1. Chromosome Coding and Decoding

Transforming the solution of a scheduling problem into a solution of a chromosome capable of genetic manipulation is called encoding. There are various methods of encoding chromosomes for genetic algorithms, such as floating-point encoding, binary encoding, integer encoding, symbolic encoding, matrix encoding, and so on [19]. The method of encoding the chromosomes directly affects the overall efficiency of the algorithm and is the key to the genetic algorithm.

The flexible shop scheduling problem contains two word problems: machine selection and process sequencing. In this paper, we adopted a two-layer integer coding approach based on process ordering constraints and machine selection constraints. These were machine code MS and process code OS; machine code determines the machine for the

process selection, and process code determines the sequential processing order. Combining the two encodings is a feasible solution to the flexible shop scheduling problem and excludes the problem of chromosomes that are not feasible solutions. Machine coding and process coding are shown in Figure 2.

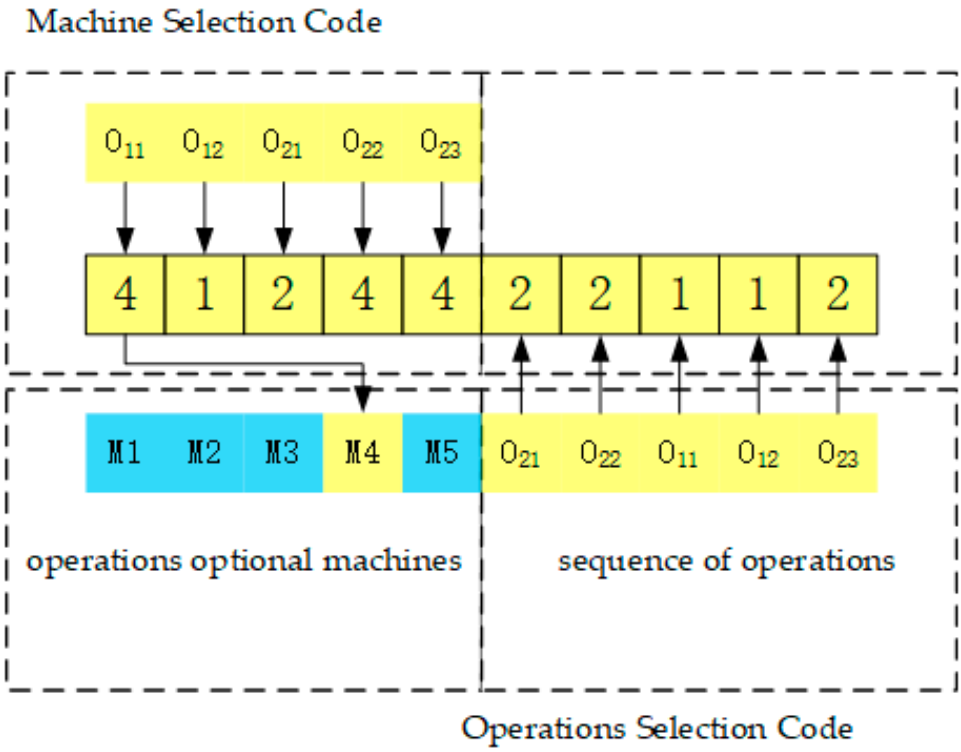

Figure 2. FJSP chromosome coding example diagram.

Machine Selection Code (MS): We used integer values to represent the selection of machines, which were arranged in the order of the workpieces' operations. In Figure 2, O_{11} pointing to 4 means that the first operation of workpiece 1 was machined on the fourth machine in the set of selectable machines, and O_{12} pointing to 1 means that the second operation of workpiece 1 was machined on the first machine in the set of selectable machines. The length of the machine code was determined by the number of processes of all the workpieces and was the sum of all the processes, so the length was $L = \sum O_{ij}$.

Operation Selection Code (OS): We used the integer value to indicate the workpiece operation's processing order according to the number of times the value was indicated for the current workpiece operations number, as shown in the figure; the operation code in the first occurrence of 2 was O_{21}, and the operation code in the number 2 s occurrence was O_{22}. The length of the operation code length by the number of all the workpieces of the process was determined by the length of the machine code for the same length, given as $L = \sum O_{ij}$.

OS encoding and MS encoding are of equal length, both being L. This encoding ensures that the solutions generated by subsequent strategies, such as crossover, mutation, and local search, are still feasible and are simple and flexible without any requirements on the length of the process or the number of artifacts. On the other hand, individual operations on one layer do not affect the other layer, providing strong parallel performance.

The solution is a combination of machine coding and process coding, and the solution can be decoded as semi-active, active, non-delayed, or hybrid scheduling. The goal of this paper was to minimize the maximum completion time, so active scheduling was used. The specific steps for decoding were as follows:

→ Step 1: Generate machining machines for each operation based on machine codes;
→ Step 2: Determine the set of workpieces for each machining machine;
→ Step 3: Determine the set of machining machines for each workpiece;

→ Step 4: Determine an allowable start time for each operation;
→ Step 5: Check the idle time of the machining machines for each operation;
→ Step 6: Generate the set of start times and finish times for each workpiece.

4.2.2. Fitness Function

The fitness function is an important criterion for judging chromosomes and is used to compare the advantages and disadvantages of individual chromosomes; it is an important part of the genetic algorithm. In the workshop scheduling problem, with the goal of minimizing the maximum completion time, the genetic algorithm usually adopts the inverse of the chromosome decoding value as the fitness value, but the fitness values of the chromosomes in the method are not very different; the individual chromosomes' differences are not clearly expressed, which is not conducive to the subsequent application of the fitness value. In this paper, the quotient of the function difference was used to solve the fitness value. The formula is as follows:

$$F(i) = \frac{f_{\max} - f_i}{f_{\max} - f_{avg}} \tag{8}$$

$f_{\max}$ is the large maximum value of the completion time, f_i is the completion time of the current chromosome, and f_{avg} is the average chromosome completion time.

4.2.3. Population Initialization

In genetic algorithms, population initialization is a decisive step, and the quality of the initial solution directly affects the distribution of the population in the whole search space and the convergence speed of the algorithm. Previous studies have usually used random initialization methods, which makes the solution quality low and often requires more iterations to obtain a better-quality population; in addition, the program running time is too long in the case of increasing problem size. Therefore, in this paper, the MS codes were initialized by GLR, which is a combination of global selection, local selection, and random selection [20] in a ratio of 3:1:1, and the OS codes were randomly generated. Finally, the MS codes and OS codes were combined into chromosomes. Global selection aims to ensure that all processing machines have the shortest processing time from a global perspective; local selection aims to ensure that all processing machines have the shortest processing time from the perspective of the independence of each workpiece; and random selection aims to make the initial population distributed as much as possible throughout the entire solution space to improve the diversity of the population. Generating individuals in the initial population according to a certain ratio can significantly improve the quality of chromosomes in the initial population.

4.2.4. Select of Operator

In genetic algorithms, selection operators select individuals on the basis of fitness. In this paper, two selection operators were used. The first one was a selection method based on the roulette algorithm, which selects each individual by the ratio of the fitness value of each individual in the population to the total fitness of the population; this method can better retain the individuals with higher fitness value, and at the same time, it also retains a certain amount of diversity in the population. The other operation was a selection method based on elite retention. Using this method, it is possible to retain individuals with good fitness values in the parent generation in the offspring.

4.2.5. Crossover Operator

Crossover operations are a key step in biological evolution and can be used to model the genetic manipulation of the genome. Specifically, a crossover operation is a random combination of genetic information from two or more parents in a genetic algorithm to produce a new offspring. The purpose of the crossover operation is to obtain an optimal solution in a genetic algorithm, which allows the offspring to obtain the advantages of

the parents and helps improve the performance of the offspring. The spatial search ability of the genetic algorithm mainly relies on crossover operation, while traditional genetic algorithms mostly rely on two parents to generate offspring, and there are situations in which the final offspring has inferior advantages and disadvantages compared with the parents. Therefore, in this paper, we improved on the POX crossover by comparing the parent as well as the generated offspring after the crossover operation to produce the best individual.

In this paper, for the process coding (OS), priority cross operation (POX) was used [21]. The POX cross-flow is shown in Figure 3. The basic flow of POX is shown in Algorithm 1:

Algorithm 1 Pseudo-code of the POX

1: The set of jobs is randomly divided into two groups, JOB1 and JOB2.
2: Any element of J1 belonging to P1 is appended to the same position of C1 and deleted in P1, and any element belonging to P2 is appended to the same position of C2 and deleted in P2.
3: The remaining elements in P2 are sequentially appended to the remaining empty spaces in O1, and the remaining elements in P1 are sequentially appended to the remaining empty spaces in the remaining empty space in O2.
4: Evaluate populations P1, P2, C1, and C2.
5: Return the best individual.

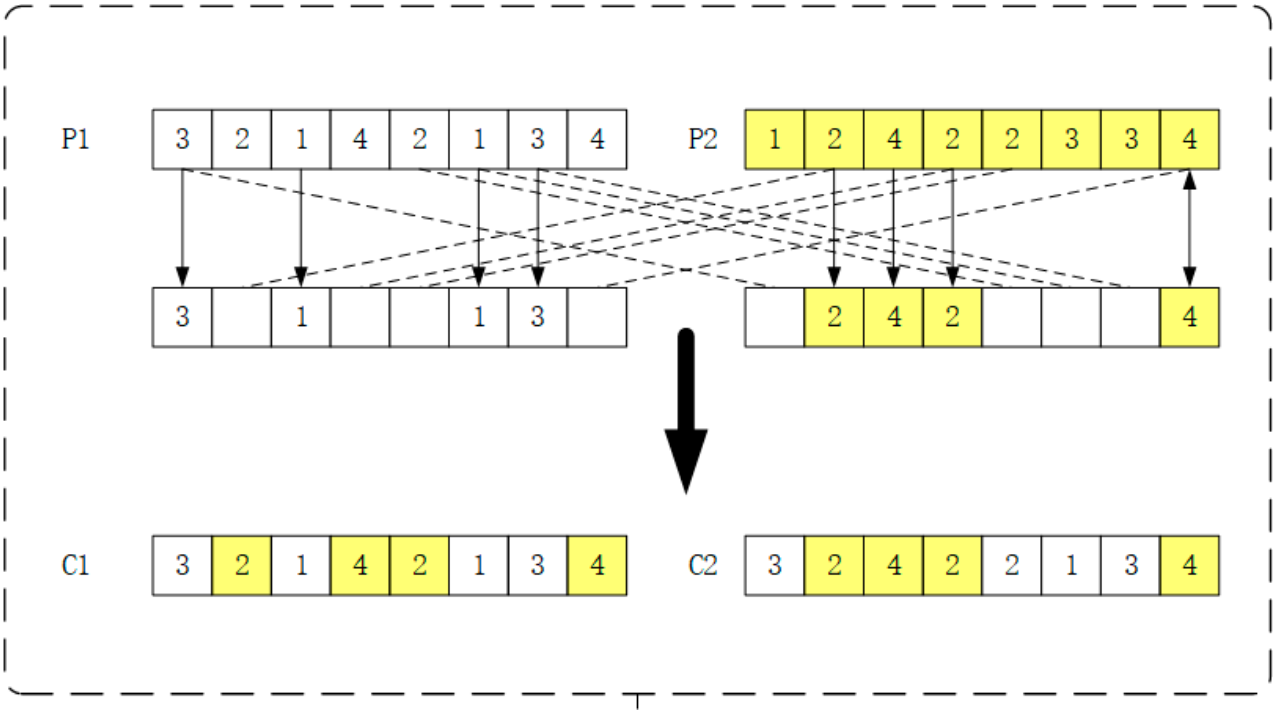

Figure 3. POX crossover.

For the machine-coded MS, multipoint crossover is used as the crossover operator. In this operator, r(r < L) positions are first selected. Then, two offspring codes are generated by exchanging the positional elements of multiple-parent machine codes. Because each gene in these two codes represents a machine selected by a fixed operation, the number of operations does not change throughout the search. Simultaneously, the selected parents are viable, and this crossover ensures that viable offspring are generated. The specific operation is shown in Figure 4:

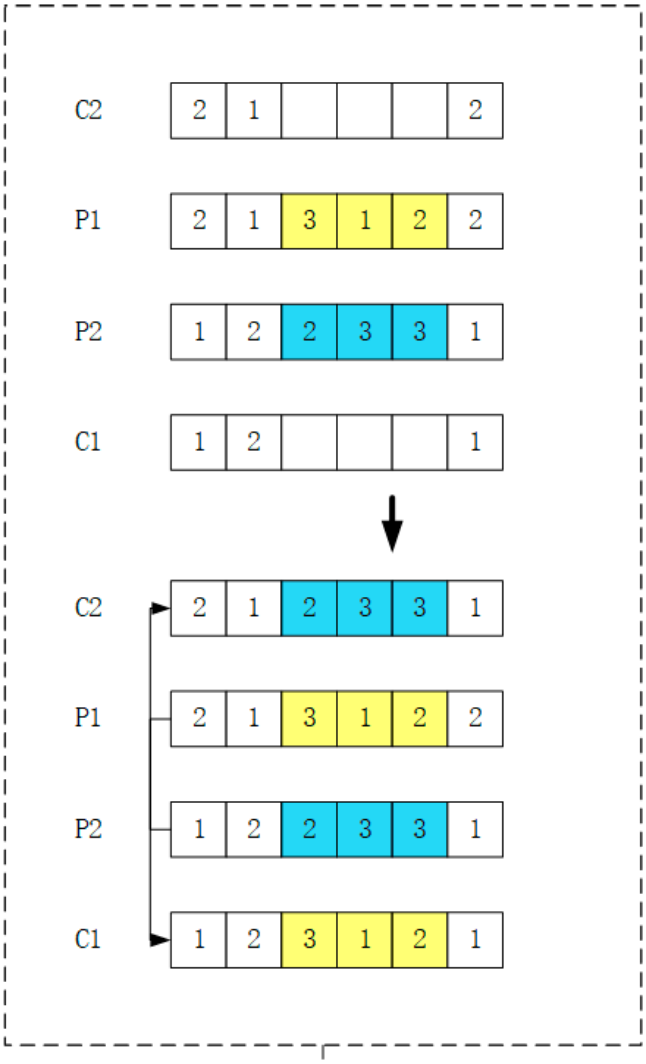

Figure 4. Machine coding crossover.

4.2.6. Improving Self-Learning Variants

The mutation operation of genetic algorithms is a stochastic change technique in which the values of some of the attributes of one or more individuals are altered in a low-probability manner. The mutation operation can introduce new genetic traits to a population, which can make the population differ from one individual to another, thus serving to facilitate the population's search for a better solution. Usually, mutation takes the form of a swap mutation or an insertion mutation, but both of these methods drastically change the order of the artifact process, resulting in the need to fix illegal loci during the run. Therefore, this paper proposes a self-learning mutation operation and designed multiple mutations to enhance the local search capability of the genetic algorithm, while the search capability for the global optimal solution was also improved because of the existence of multiple mutation operations, which change the previous method of single mutation. The self-learning mutation operations were also designed. Four variant operations were designed: the machine greedy variant [22], the machine random selection variant, the process random multi-switching variant, and the process two-point variant. At the beginning of the algorithm, the same weights were set for each of the four variations.

The choice of mutation mode is introduced by the following equation:

$$P_i = \frac{w_i}{\sum_{i=1} w_i} \tag{9}$$

w_i is the weight of the mutation mode, and P_i is the probability of choosing the mutation mode i. The variant method is determined by the weights; the higher the weight, the higher the probability of choosing that variant method.

The mutation operator was designed as follows:

Machine greedy variant: Randomly select r positions from the machine code, and select the machine with the shortest processing time for the current operation among the selected positions.

Machine random selection variant: Randomly select r positions from the machine code, and randomly select the machine in the selected position for which the current operation is selectable for processing.

Process randomized multi-swap mutation: Select r positions from the process code, disrupt the genes in the r positions, and then place the process code sequentially into the process code again.

Process two-point mutation: Two genes with two digits different from the process code are selected for exchange.

Whenever a mutation operation was performed, we made an assessment of the goodness of the mutation method; if the mutation pair was improved, the weight of the mutation method was increased, and vice versa, if the mutation pair was worsened, the weight of the mutation method is decreased; and at the same time, the upper and lower limits of the weight were set to prevent the mutation method from converging to a single method. The formula for adjusting the weights is as follows:

$$w_i = \begin{cases} w_i + \frac{f_i - f_{\min}}{f_{\min}} & if(w_i < w_{\max}) \\ w_i + \frac{f_i}{f_{\min}} & if(w_i > w_{\min}) \end{cases}$$

w_i is the weight of the mutation mode, f_i is the fitness value of the chromosome after the mutation, $f_{\min}$ is the global chromosome optimal solution, and $w_{\max}$, $w_{\min}$ are the upper and lower limits of the w_i weights.

4.3. Algorithmic Process

In this paper, the algorithm flow chart for the improvement of the genetic algorithm combined with the flexible workshop scheduling problem, is shown in Figure 5. The pseudo-code is given by Algorithm 2.

Algorithm 2 Pseudo-code of the improved self-learning genetic algorithm for the FJSP

```
1: Set up the initial improved self-learning genetic algorithm parameter
2: GRL population initialization
3: For i = 0,. . .. . ., Max_Itertions do
4:   Evaluate the populations
5:   Elite selection strategy
6:   For j = 1,. . .., population size do
7:    If random() < Pc
8:       Crossover(j,random(1, population size))
9:    If random() < Pm
10:       Self-learning mutation(j)
11:       Select the mutation mode
12:       Weighting update
13:    End
14:    Decode()
14: End
15 Output: the best solution
```

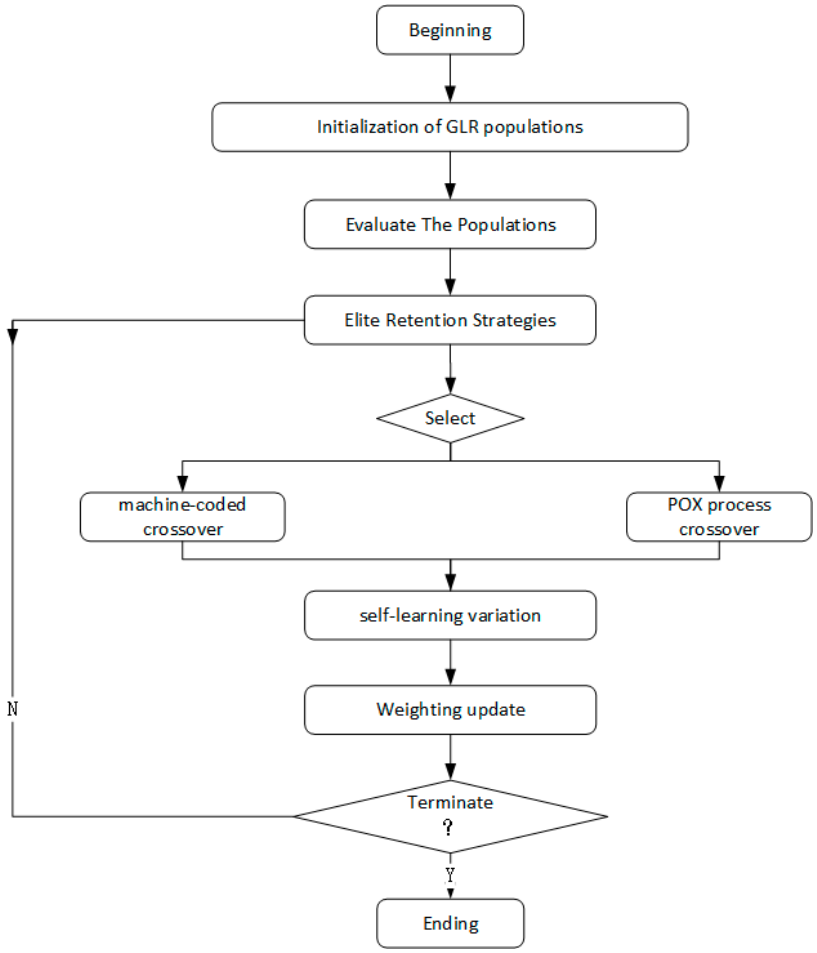

Figure 5. Improved self-learning genetic algorithm.

5. Experimental Analyses

5.1. Parameterization

In order to verify the effectiveness of the improved self-learning genetic algorithm in solving the flexible workshop scheduling problem, the program was programmed in Python language under pycharm, and the program processor was Intel(R) Core(TM) i5-6300HQ; the main frequency was 2.30 Hz, the memory was 8 GB, and the operating system was a 64-bit Windows 10 running on a personal computer. The specific parameter settings of the algorithm are shown in Table 2:

Table 2. The specific parameters of the algorithm.

Parameters	
Population size: Pop_size	200
Crossover probability: Pc	0.9
Probability of mutation: Pm	0.1
Number of elite reserves: len	1
Maximum number of iterations: Max_Itertions	100
Percentage of population initialization: GS:LS:RS	3:1:1
Self-learning variant weights: W1; W2; W3; W4	W1 = W2 = W3 = W4 = 10

5.2. Experiment 1

To illustrate the performance of the self-learning genetic algorithm proposed in this paper for solving FJSP problems, this paper's experiments were compared with the simulation experiments in the literature [23], and the experimental data were consistent with the

paper of [23], as shown in Table 3. Figures 6 and 7 show the scheduling schemes obtained by the algorithms in the literature [23] and in this paper, respectively. In the table, $J_{i,j}$ is the jth process of the ith workpiece, and M1–M8 are individual processing machines.

Table 3. Arithmetic source data.

Workpiece Number	Working Procedure $J_{i,j}$	Processing Time for Each Workpiece and Each Process on Selectable Processing Machines							
		*M*1	*M*2	*M*3	*M*4	*M*5	*M*6	*M*7	*M*8
1	$J_{1,1}$	5	3	5	3	3	-	10	9
	$J_{1,2}$	10	-	5	8	3	9	9	6
	$J_{1,3}$	-	10	-	5	6	2	4	5
2	$J_{2,1}$	5	7	3	9	8	-	9	-
	$J_{2,2}$	-	8	5	2	6	7	10	9
	$J_{2,3}$	-	10	-	5	6	4	1	7
	$J_{2,4}$	10	8	9	6	4	7	-	-
3	$J_{3,1}$	10	-	-	7	6	5	2	4
	$J_{3,2}$	-	10	6	4	8	9	10	-
	$J_{3,3}$	1	4	5	6	-	10	-	7
4	$J_{4,1}$	3	1	6	5	9	7	8	4
	$J_{4,2}$	12	11	7	8	10	5	6	9
	$J_{4,3}$	4	6	2	10	3	9	5	7
5	$J_{5,1}$	3	6	7	8	9	-	10	-
	$J_{5,2}$	10	7	-	4	9	8	6	-
	$J_{5,3}$	-	9	8	7	4	2	7	-
	$J_{5,4}$	11	9	-	6	7	5	3	6
6	$J_{6,1}$	6	7	1	4	6	9	-	10
	$J_{6,2}$	11	-	9	9	9	7	8	4
	$J_{6,3}$	10	5	9	10	11	-	10	-
7	$J_{7,1}$	5	4	2	6	7	-	10	-
	$J_{7,2}$	-	9	-	9	11	9	10	5
	$J_{7,3}$	-	8	9	3	8	6	-	10
8	$J_{8,1}$	2	8	5	9	-	4	-	8
	$J_{8,2}$	7	4	7	8	9	-	10	-
	$J_{8,3}$	9	9	-	8	5	6	7	1
	$J_{8,4}$	9	-	3	7	1	5	8	-

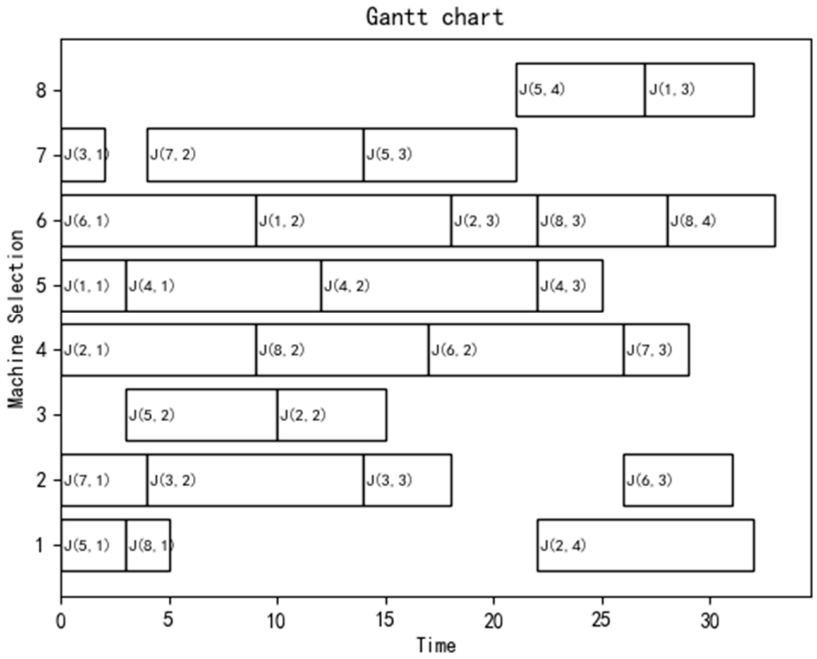

Figure 6. Reference 23's experimental results.

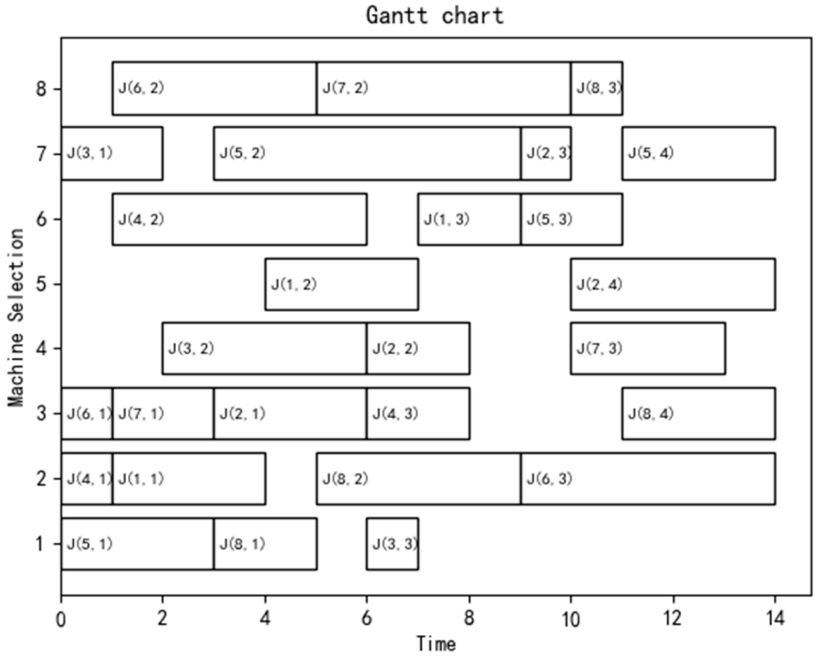

Figure 7. Experimental results of the algorithm in this paper.

The experimental results are shown in Figures 6 and 7. The maximum completion time in reference [23] was 33. The scheduling scheme obtained by the algorithm in this paper had a maximum completion time of 14, which is better than that in reference [23], and it is more reasonable in terms of machine allocation and process sequence. Comparing the data calculations in Table 3, the algorithm proposed in this paper improved the efficiency by 57.5%.

5.3. Experiment 2

In order to illustrate the performance of the self-learning genetic algorithm proposed in this paper in solving the FJSP problem, it was applied to the benchmark cases MK01–MK10 in the benchmark calculus, which were run 20 times to obtain the optimal solution, and the results of the runs are shown in Table 4.

Table 4. Comparison of benchmark samples.

MK Algorithms	N × m	GWO	IDE	HQPSO	ALPS-GA	ILGA	This Paper
MK01	10 × 6	42	40	42	40	40	40
MK02	10 × 6	31	28	28	29	28	**27**
MK03	15 × 8	204	204	204	207	204	204
MK04	15 × 8	68	71	70	65	64	64
MK05	15 × 4	179	179	179	181	175	**173**
MK06	10 × 15	68	73	68	81	71	**63**
MK07	20 × 5	155	146	149	149	145	**140**
MK08	20 × 10	523	528	523	555	523	523
MK09	20 × 10	—	321	342	342	315	**307**
MK10	20 × 15	—	235	246	255	260	**218**

The following experimental results of MK algorithms are for the grey wolf optimization (GWO) algorithm proposed by Jiang Tian-hua [24], the improved differential evolution (IDE) algorithm proposed by Prasert et al. [25], the hybrid quantum particle swarm optimization (HQPSO) proposed by Zhang et al. [26], the ALPS-GA algorithm proposed by

Jiang Hou-min et al. [27], and the improved learning genetic algorithm (ILGA) proposed by Zhang Liang et al. [28].

As can be seen in Table 4, with the improved self-learning genetic algorithm proposed in this paper, the MK algorithms in MK02, MK05, MK06, MK07, MK09, and MK10 were better than those of the results of the other papers and have been bolded. The comparison shows that in terms of solution depth, the algorithms in this paper achieved better solutions than the listed algorithms, except for the MK01, MK03, MK04, and MK08 algorithms.

In order to prove the local search ability of the improved self-learning genetic algorithm proposed in this paper, experiments were carried out using MK05 in Brandimarte's ten examples, which were solved by the genetic algorithm with the self-learning mutation operator, the improved genetic algorithm with the removal of the self-learning mutation operator, and the traditional genetic algorithm, respectively, and the optimal individuals' objective function values in the solution sets of the three algorithms were recorded by repeating the experiments 50 times. The results of the experiments are shown in Figure 8. In the 50 experiments, the optimal individual Cmax obtained by the genetic algorithm with the addition of the self-learning mutation operator was mainly concentrated below the optimal solution of 142, while the optimal individual Cmax obtained by the improved genetic algorithm with the removal of the self-learning mutation operator was mainly concentrated in the suboptimal solutions of 144 and 145; the optimal individual Cmax obtained by the traditional genetic algorithm was mainly concentrated in the suboptimal solution of 146 or so. It was proven that the improved genetic algorithm with the addition of the self-learning mutation operator is significantly better than the improved genetic algorithm with the removal of the self-learning mutation operator and the traditional genetic algorithm in terms of the stability of the solution and the depth of the search. When the size of the problem increased and the topology of the solution space was complex, the self-learning mutation operator proposed in this paper could quickly and stably search for a better solution on the basis of the global optimization of the genetic algorithm.

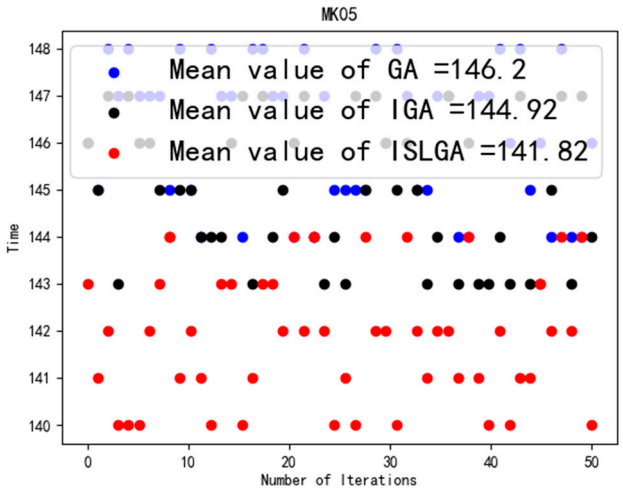

Figure 8. Example of the result of MK05 repeated 50 times.

5.4. Experiment 3

In order to illustrate the performance of the self-learning genetic algorithm proposed in this paper in comparison with the traditional genetic algorithm and the improved genetic algorithm in solving the FJSP problem, it was applied to the MK01–MK10 benchmark examples and was run to output its maximum completion time with the number of population iterations.

Figure 9 shows the MK05, MK07, MK09, and MK10 examples of the MK05, MK07, MK09, and MK10 algorithms, respectively. In Figure 9, it can be seen that the self-learning genetic algorithm's pre-convergence speed was very fast, which indicates that the self-learning genetic algorithm in the crossover process of the pox crossover operator played a

key role in searching for the solution space, and it searched for the optimal solution relatively quickly. The comparison curve shows that the adaptive mutation operator enhanced the algorithm's ability to find the optimal solution; compared with the traditional genetic algorithm, its convergence speed is faster, and its search ability is greater. Meanwhile, as the self-learning mutation strategy of the algorithm changes with the number of iterations, the w-weights in its self-learning mutation operator are constantly updated, which enhances the self-learning mutation operator in the later stages of the local search efficiency. It can be seen that the variable self-learning mutation strategy proposed in this paper results in a greater enhancement of the algorithm's local optimization ability, and at the same time, compared with the traditional genetic algorithm, the algorithm proposed in this paper has a faster convergence speed and a stronger search ability.

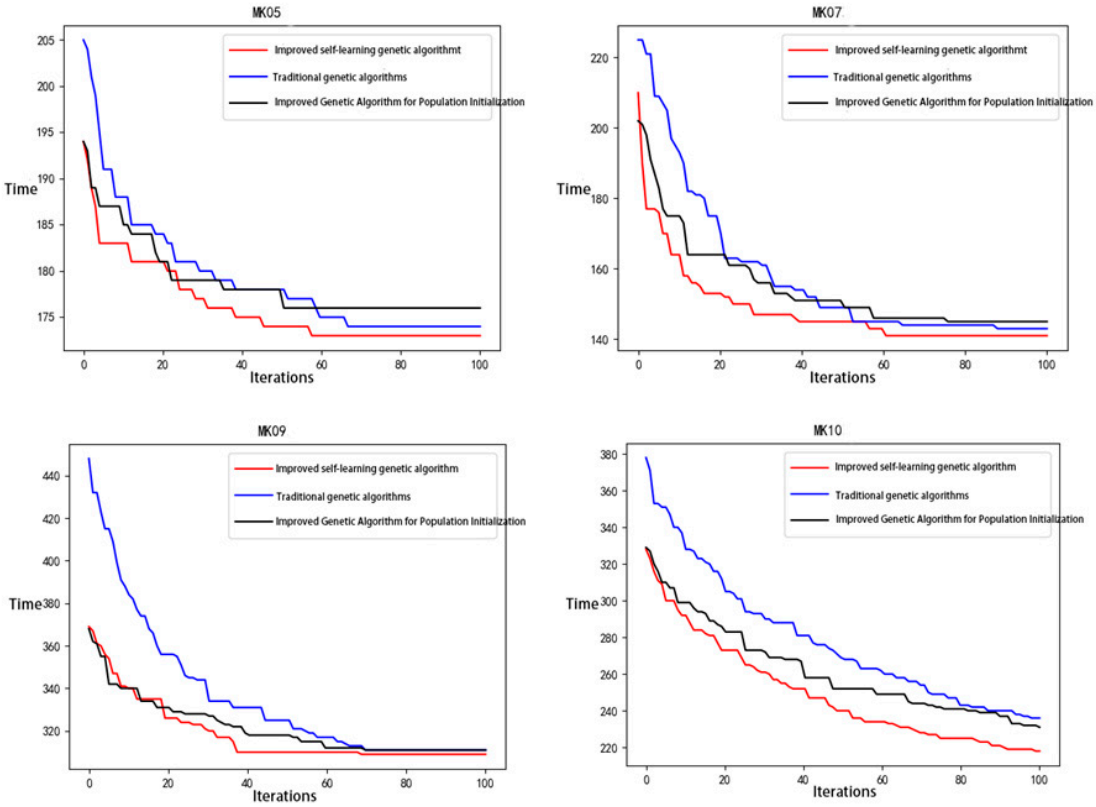

Figure 9. Ablation experiments on MK05, MK07, MK09, and MK10.

In contrast to the genetic algorithm and the improved genetic algorithm, it was shown that the initial population in the improved genetic algorithm generates quality solutions significantly better than the traditional genetic algorithm, and its convergence speed and traditional genetic algorithm to the solution space of the search ability was also stronger than the traditional genetic algorithm. In addition, the improved genetic algorithm in this paper added the self-learning mutation strategy and designed four kinds of mutation methods; as can be seen in the convergence curve in Figure 9, the convergence speed of this paper's algorithm and the ability to search for the solution space are stronger than those of the two algorithms. The feasibility of this paper's algorithm is illustrated by the experimental results, and the algorithm has a strong search ability and can solve the FJSP effectively.

6. Discussion

The FJSP is a combinatorial optimization problem that requires assigning different production tasks to different machines or workers in a limited amount of time in order to maximize customer satisfaction and production efficiency. Solving the FJSP can help

enterprises achieve multiple goals, such as refined management, improved productivity, cost reduction, and product quality. The government can encourage enterprises to increase research and investment in the FJSP by formulating relevant policies to promote technological innovation and industrial upgrading in related fields. In this paper, the Brandimarte examples verify the validity of the proposed algorithm, but no other examples are available. Variational operators may behave differently in different examples, so it is essential to identify how to find more universal variational operators. Meanwhile, the process of adjusting the weights may fall into the local exploration problem, which can be considered to increase the perturbation of the weights.

7. Conclusions

In this paper, an improved self-learning genetic algorithm is proposed; four variant operators were designed according to the characteristics of the FJSP, which strengthened the local search ability of the genetic algorithm and expanded the ability to find better individuals. At the same time, weights were designed for the mutation operators, and the weights were updated according to their local search of flexible workshop scheduling at different stages to strengthen the search ability for the solution space where the global optimal solution might exist. The results showed good performance compared with the algorithms in the references, especially in the Brandimarte examples. In future research, it is recommended to consider designing more variational operators to combine with other intelligent optimization algorithms and to find a more appropriate way to update the weights. In addition, multi-objective FJSP and dynamic scheduling problems can be explored in depth to make the algorithms for solving job shop scheduling problems more efficient, complete, and adaptable.

Author Contributions: Software, H.Y.; validation, H.Y.; formal analysis, H.Y.; investigation, J.C.; resources, M.J.; data curation, J.C.; writing—original draft, H.Y.; writing—review & editing, M.J.; supervision, M.J.; funding acquisition, M.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Research Project of the Science and Technology Innovation Think Tank of the Fujian Society of Science and Technology under Grant No. FJKX-2022XKB023; the National Social Science Foundation of China under Grant No. 22BGL007; Fujian Social Sciences; the Federation Planning Project under Grant No. FJ2021XZB089 and No. FJ2021Z006; the Project of the Science and Technology Innovation Think Tank of the Fujian Society of Science and under Grant No. FJKX-A2113; and Fujian University of Technology under Grant No. GY-S20042.

Data Availability Statement: The data presented in this study are available on request from the first or corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Brandimarte, P. Routing and scheduling in a flexible job shop by tabu search. *Ann. Oper. Res.* **1993**, *41*, 157–183. [CrossRef]
2. Brucker, P.; Schlie, R. Job-shop scheduling with multi-purpose machines. *Computing* **1990**, *45*, 369–375. [CrossRef]
3. Luo, Z.; Zhu, G. Research status and development trend of workshop scheduling problems. *Technol. Innov. Appl.* **2020**, *23*, 123–124.
4. Amjad, M.K.; Butt, S.I.; Kousar, R.; Ahmad, R.; Agha, M.H.; Faping, Z.; Anjum, N.; Asgher, U. Recent Research Trends in Genetic Algorithm Based Flexible Job Shop Scheduling Problems. *Math. Probl. Eng.* **2018**, *2018*, 9270802. [CrossRef]
5. Gao, X.; Yang, S.; Li, L. Optimization of flow shop scheduling based on genetic algorithm with reinforcement learning. *J. Phys. Conf. Ser.* **2022**, *2258*, 012019. [CrossRef]
6. Guohui, Z.; Liang, G.; Peigen, L.; Chaoyong, Z. Improved genetic algorithm for the flexible job-shop scheduling problem. *J. Mech. Eng.* **2009**, *45*, 145–151.
7. Taillard, E.D. Parallel Taboo Search Techniques for the Job Shop Scheduling Problem. *Informs J. Comput.* **1994**, *6*, 108–117. [CrossRef]
8. Holsapple, C.W.; Jacob, V.; Pakath, R.; Zaveri, J. A genetics-based hybrid scheduler for generating static schedules in flexible manufacturing contexts. *IEEE Trans. Syst. Man Cybern.* **1993**, *23*, 953–972. [CrossRef]

9. Nouiri, M.; Bekrar, A.; Jemai, A.; Niar, S.; Ammari, A.C. An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem. *J. Intell. Manuf.* **2018**, *29*, 603–615. [CrossRef]
10. Caldeira, R.H.; Gnanavelbabu, A. A Pareto based discrete Jaya algorithm for multi-objective flexible job shop scheduling problem. *Expert Syst. Appl.* **2021**, *170*, 114567. [CrossRef]
11. Burggräf, P.; Wagner, J.; Saßmannshausen, T.; Ohrndorf, D.; Subramani, K. Multi-agent-based deep reinforcement learning for dynamic flexible job shop scheduling. *Procedia CIRP* **2022**, *112*, 57–62. [CrossRef]
12. Yu, Q.; Zhao, L.; Pan, S. Scheduling Optimization of Flexible Job-shop Using Genetic Algorithm. *Sci. Technol. Eng.* **2020**, *20*, 11931–11936.
13. Tang, J.; Zhang, G.; Lin, B.; Zhang, B. A Hybrid Algorithm for Flexible Job-Shop Scheduling Problem. *Procedia Eng.* **2011**, *15*, 3678–3683. [CrossRef]
14. Zhang, X.; Yan, W.; Yan, D.; Ji, Z. Improved Shuffled Frog-Leaping Algorithm for Solving Flexible Job Shop Scheduling Problem. *J. Syst. Simul.* **2017**, *29*, 2093.
15. Liu, D.M.; Wei-Ping, F.U.; Lai, C.W.; Wang, W.; Bao, Y.T. Improved Genetic Algorithm for Flexible Job Shop Scheduling Problem. *J. Chin. Comput. Syst.* **2017**, *38*, 129–132.
16. Qiong, L.; Chaoyong, Z.; Yunqing, R.; Xinyu, S. Improved genetic algorithm to solve flexible job-shop scheduling problem. *Manuf. Autom.* **2022**, *44*, 91–94+106.
17. Abd El-Wahed Khalifa, H.; Alodhaibi, S.S.; Kumar, P. Solving constrained flow-shop scheduling problem through mul-tistage fuzzy binding approach with fuzzy due dates. *Adv. Fuzzy Syst.* **2021**, *2021*, 6697060.
18. Majumder, S. Some network optimization models under diverse uncertain environments. *arXiv* **2021**, arXiv:2103.08327.
19. Lei, W.; Jingcao, C.; Ming, L. Parameter Optimization of Genetic Algorithm Based on Orthogonal Experiment. *J. Nanjing Norm. Univ.* **2016**, *16*, 81–85.
20. Zhang, G.; Gao, L.; Shi, Y. An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Syst. Appl.* **2011**, *38*, 3563–3573. [CrossRef]
21. Kacem, I. Genetic algorithm for the flexible job-shop scheduling problem. In Proceedings of the 2003 IEEE International Conference on Systems, Man and Cybernetics (SMC'03), Conference Theme-System Security and Assurance (Cat. No. 03CH37483), Washington, DC, USA, 8 October 2003.
22. Xiong, L.; Qian, Q.; Yunfa, F. Review of Application of Genetic Algorithms for Solving Flexible Job Shop Scheduling Problems. *Comput. Eng. Appl.* **2019**, *55*, 15–21.
23. Jie, Z.; Li, B.; Jiang, J.; Gu, H. Flexible job-shop scheduling problem based on coyote optimization algorithm. *Mod. Manuf. Eng.* **2020**, *10*, 39–44.
24. Tian-Hua, J. Flexible job shop scheduling problem with hybrid grey wolf optimization algorithm. *Control Decis.* **2018**, *33*, 503–508.
25. Sriboonchandr, P.; Kriengkorakot, N.; Kriengkorakot, P. Improved Differential Evolution Algorithm for Flexible Job Shop Scheduling Problems. *Math. Comput. Appl.* **2019**, *24*, 80. [CrossRef]
26. Zhang, Q.; Hu, S. An improved hybrid quantum particle swarm optimization algorithm for fjsp. In Proceedings of the 2019 11th International Conference on Machine Learning and Computing, Zhuhai, China, 22–24 February 2019.
27. Jiang, H.M.; Li, S.B.; Wang, J.X.; Bian, X.X. Solving Flexible Job-Shop Scheduling Problem Using ALPS-GA. *Comput. Simul.* **2019**, *5*, 390–394.
28. Zhang, L.; Mao, J.; Wang, N.; Li, R. Learning Genetic Algorithm Based on Key Machines and Neighborhood Search to Solve FlexibleShop Scheduling Problems. *Modul. Mach. Tool Autom. Manuf. Tech.* **2023**, *2*, 183–186+192.