processes

Article

Hybrid Memetic Algorithm to Solve Multiobjective Distributed Fuzzy Flexible Job Shop Scheduling Problem with Transfer

Jinfeng Yang and Hua Xu *

School of Artificial Intelligence and Computer Science, Jiangnan University, Wuxi 214122, China; 6213114028@stu.jiangnan.edu.cn
* Correspondence: xuhua@jiangnan.edu.cn

Abstract: Most studies on distributed flexible job shop scheduling problem (DFJSP) assume that both processing time and transmission time are crisp values. However, due to the complexity of the factory processing environment, the processing information is uncertain. Therefore, we consider the uncertainty of processing environment, and for the first time propose a multiobjective distributed fuzzy flexible job shop scheduling problem with transfer (MO-DFFJSPT). To solve the MO-DFFJSPT, a hybrid decomposition variable neighborhood memetic algorithm (HDVMA) is proposed with the objectives of minimizing the makespan, maximum factory load, and total workload. In the proposed HDVMA, the well-designed encoding/decoding method and four initialization rules are used to generate the initial population, and several effective evolutionary operators are designed to update populations. Additionally, a weight vector is introduced to design high quality individual selection rules and acceptance criteria. Then, three excellent local search operators are designed for variable neighborhood search (VNS) to enhance its exploitation capability. Finally, a Taguchi experiment is designed to adjust the important parameters. Fifteen benchmarks are constructed, and the HDVMA is compared with four other famous algorithms on three metrics. The experimental results show that HDVMA is superior to the other four algorithms in terms of convergence and uniformity of non-dominated solution set distribution.

Keywords: mutiobjective; distributed flexible job shop scheduling; fuzzy transfer time; fuzzy processing time; memetic algorithm; weight vector; variable neighborhood search

Citation: Yang, J.; Xu, H. Hybrid Memetic Algorithm to Solve Multiobjective Distributed Fuzzy Flexible Job Shop Scheduling Problem with Transfer. *Processes* **2022**, *10*, 1517. https://doi.org/10.3390/pr10081517

Academic Editor: Fabricio Napoles-Rivera

Received: 13 July 2022
Accepted: 29 July 2022
Published: 1 August 2022

1. Introduction

In recent years, flexible job shop scheduling problem (FJSP) has been widely studied [1–3]. The classic FJSP is to assign different workpiece processes to different machines for processing and to generate scheduling schemes based on process constraints after sorting the processes. All processes are completed in the same factory. However, to meet the needs of manufacturing globalization and industry 4.0, distributed manufacturing has attracted increasing attention. In distributed manufacturing, workpieces are assigned to different factories for processing. This manufacturing method can make better use of the production advantages of different factories, reduce production costs, and save production time. Compared with the single-factory scheduling problem, the distributed workshop scheduling problem is more complex, its environment is more changeable, and it is a more difficult problem to solve. However, as it fits the trend of industrial development, and is of great significance for collaborative production.

In the production process, the complex environment leads to great information uncertainty, such as uncertainty in workpiece processing times caused by the variable proficiency of workers and uncertainty in the workpiece transmission time caused by the wear degree of transportation tools. Therefore, we fuzzify the transmission time and processing time, replace the crisp time with triangular fuzzy numbers (TFNs), and for the first time propose a multiobjective distributed fuzzy flexible job shop scheduling model, referred to as the MO-DFFJSPT, with the objectives of minimizing the makespan, maximum factory load, and total workload. As this model is more in line with actual production, it is conducive to the sustainable development of factories while ensuring production benefits.

To solve the MO-DFFJSPT, we propose the HDVMA based on the framework of the non-dominated sorting genetic algorithm II (NSGA-II) [4]; furthermore, we design encoding and decoding methods and crossover and mutation operators for fuzzy numbers. The HDVMA combines an initialization strategy and a decomposition variable neighborhood search (DVNS) operator to achieve a balance between exploration and exploitation. Then, we adjust the important parameters in the algorithm through a Taguchi experiment. Finally, our results when running the algorithm on fifteen benchmarks prove its excellent performance.

The rest of the paper is organized as follows: Section 2 reviews the relevant research; Section 3 introduces the basic knowledge of fuzzy numbers and fuzzy operations; Section 4 describes the proposed MO-DFFJSPT model; Section 5 sketches the proposed algorithm; and Section 6 describes the results of running the algorithm on the benchmarks. Finally, Section 7 provides a summary and proposes future research directions.

2. Literature Review

This section consists of two subsections. Section 2.1 reviews the distributed flexible job shop scheduling problem, and Section 2.2 reviews the fuzzy flexible job shop scheduling problem.

2.1. Distributed Flexible Job Shop Scheduling Problem

DFJSP is becoming increasingly suitable for intelligent factory production. Naderi et al. [5] proposed a mixed integer model with low computational complexity; a new simulated annealing algorithm was used to solve the distributed problem, and good results were obtained. Chang et al. [6] proposed a genetic algorithm with multiple crossover and mutation operators to solve DFJSP. Wu et al. [7] proposed a DFJSP with assembly, and an improved differential evolution simulated annealing algorithm was proposed to solve the problem with the objectives of balancing earliness and tardiness and achieving a lower total cost. The results showed this method to have good performance and robustness. Tang et al. [8] proposed a hybrid teaching and learning algorithm with Tabu Search to solve DFJSP, and a production example proved its effectiveness. Ziaee et al. [9] developed a fast heuristic algorithm to solve DFJSP very quickly.

Although there are an increasing number of studies on DFJSP, most studies on DFJSP assume that a job can only be processed in one factory. However, in the real world, many workpieces must be processed in different factories. Luo et al. [10] proposed an efficient memetic algorithm to solve the DFJSP with transfer (DFJSPT). Sang et al. [11] proposed a high-dimensional multiobjective memetic algorithm to solve the DFJSPT. This method combines the improved NSGA-III and local search method and adds a dual-mode environment selection method, achieving a balance between diversity and convergence.

2.2. Fuzzy Flexible Job Shop Scheduling Problem

In recent years, there have been an increasing number of studies on fuzzy flexible job shop scheduling problem (FFJSP) [12]. Zheng et al. [13] proposed a multiobjective swarm-based neighborhood search method to solve FFJSP, with the objective of reducing the makespan and total workload. Lin et al. [14] integrated the path relinking technique into multiverse optimization and obtained a satisfactory makespan. Sun et al. [12] proposed an effective algorithm that combines particle swarm optimization with a genetic algorithm for improved convergence ability. Li et al. [15] proposed a hybrid self-adaptive multiobjective evolutionary algorithm based on decomposition with the objectives of fuzzy makespan and fuzzy total workload, using an effective solution selection method based on the Tchebycheff decomposition strategy to improve the convergence and diversity of the population. Gao et al. [16] proposed a discrete harmony search algorithm and proved its effectiveness. Lin et al. [17] proposed a biogeography-based optimization method combined with certain heuristics that balance exploration and exploitation. Gao et al. [18] proposed an improved artificial bee colony algorithm to solve the multiobjective FFJSP, and an effective heuristic rule was developed to initialize the population. The experimental results yielded a good Pareto solution.

From Sections 2.1 and 2.2, we can see that the DFJSP and FFJSP have become popular topics in the scheduling field. However, time uncertainty is not taken into account in the study of the DFJSP. In the FFJSP, the realistic scenario of collaboration among multiple factories is not considered. Therefore, we consider the constraints of both models at the same time and propose a new model (MO-DFFJSPT) to better solve scheduling problems in the real world. Additionally, due to the complexity of the MO-DFFJSPT, there is no optimization method as yet to solve the MO-DFFJSPT. The methods mentioned in the above literature cannot be directly applied to solve the MO-DFFJSPT, and most of the existing multiobjective evolutionary algorithms cannot achieve a good balance between exploration and exploitation. Thus, we propose a new algorithm, HDVMA, to generate diverse and improved Pareto-optimal solutions of the MO-DFFJSPT.

3. Fuzzy Set

3.1. Fuzzy Number

In real-life production, there is often no crisp time at which a machine processes workpieces, considering that production activities may be performed by more or less highly skilled workers and that the work environment changes frequently. The crudest representation of uncertain time frames is as a confidence interval; if certain values seem more plausible than others, then those values are expanded into fuzzy numbers.

TFNs are widely used in scheduling activities [19]. As shown in Figure 1, each triangular fuzzy number consists of three values, (a_1, a_2, a_3), with a_1 being the earliest processing time, a_2 the most likely processing time, and a_3 the latest processing time. The membership function formula for x is as follows:

$$\mu(x) = \begin{cases} 0, & x \le a_1 \\ \dfrac{x - a_1}{a_2 - a_1}, & a_1 < x \le a_2 \\ \dfrac{a_3 - x}{a_3 - a_2}, & a_2 < x < a_3 \\ 0, & x \ge a_3 \end{cases} \tag{1}$$

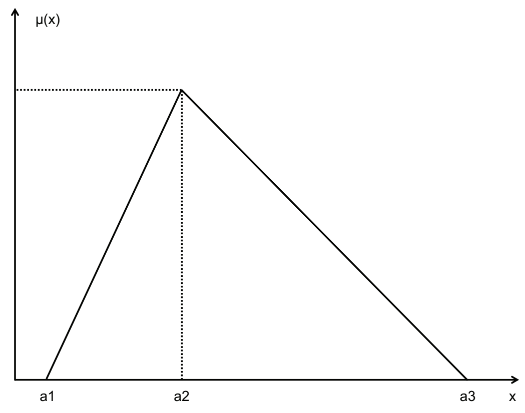


Figure 1. Fuzzy Number.
]

3.2. Fuzzy Operation

When generating a feasible job shop schedule, several different mathematical operations are needed. Therefore, the operations of fuzzy numbers need to be redefined. Sakawa et al. [20] first defined these operations in the 1990s.

Addition Operation: Given two fuzzy numbers, $X = (x_1, x_2, x_3)$ and $Y = (y_1, y_2, y_3)$, for any pair of TFNs X and Y it follows that

$$X + Y = (x_1 + y_1, x_2 + y_2, x_3 + y_3) \tag{2}$$

Ranking Operation: To ensure that scheduling conforms to constraints, three comparison criteria are defined to sort two TFNs.

Criterion 1: $F_1^{\sim}(X) = (x_1 + 2x_2 + x_3)/4$

Criterion 2: $F_2^{\sim}(X) = x_2$

Criterion 3: $F_3^{\sim}(X) = x_3 - x_1$

For any two TFNs, we calculate the Criterion 1 value $F_1^{\sim}(X)$ first, and the larger TFN has a larger criterion value. If the two TFNs values $F_1^{\sim}(X)$ are equal, we use Criterion 2. If the two TFNs values $F_2^{\sim}(X)$ are equal, we use Criterion 3.

Max Operation: We assume that with two TFNs $X = (x_1, x_2, x_3)$ and $Y = (y_1, y_2, y_3)$, Sakawa's criterion to obtain the maximum value is as follows:

$$Max(X, Y) = (\max(x_1, y_1), \max(x_2, y_2), \max(x_3, y_3)) \tag{3}$$

However, Lei [21] proposed a new max criterion in 2010 which has been proven to have better results. In this paper, Lei's criterion is adopted to determine the workpiece fuzzy processing start time, which is as follows:

$$If X > Y, Max(X, Y) = X; otherwise, Max(X, Y) = Y \tag{4}$$

4. Problem Description and Mathematical Modeling

4.1. MO-DFFJSPT Description

The MO-DFFJSPT consists of n jobs, f factories, and m machines belonging to different factories. The factories are located in several regions. There are $N_{J_i} (i \in \{1, 2, \ldots, n\})$ operations for each job, and each operation needs to be processed on a candidate machine of a factory. All the processing times for operations $O_{i,j}$ on a machine and the times for transporting operations between machines are TFNs, and all the TFNs are ascertained in advance. The assumptions of MO-DFFJSPT are as follows:

- All the processing times and transfer times are TFNs;
- All the machines and jobs are available at time zero;
- The processing time and transfer time of each operation are known in advance;
- All jobs can be transported within a factory or between different factories;
- The time for transferring any job between machines in a factory is the same;
- The time for transferring any job between machines in different factories is the same;
- The processing time includes the transfer time of the job.

4.2. MILP Model for the MO-DFFJSPT

In this section, a mixed-integer linear program (MILP) is constructed; the parameters and indices used in the model are as follows:

n: The total number of jobs

m: The total number of machines

f: The total number of factories

N_{J_i}: The total number of operations for job i

M_k: The k-th machine

F_l: The l-th factory

$O_{i,j}$: The j-th operation of the i-th job

$P_{i,j}^{k}$: The processing time for the j-th operation of the i-th job on machine k
T_M: The transfer time between machines
T_F: The transfer time between factories
W_l: The workload of factory l
C_i: The completion time of job i
$C_{i,j(u,v)}$: The completion time of $O_{i,j}(O_{u,v})$
L: A large positive number
Variables:
$X_{i,j}^{k}$: If $O_{i,j}$ is processed on machine M_k, this value is set to 1; otherwise, it is set to 0
$V_{i,j}^{l}$: If $O_{i,j}$ is processed in factory F_l, this value is set to 1; otherwise, it is set to 0
$Y_{i,j,u,v,k}^{l}$: If $O_{i,j}$ is processed directly after $O_{u,v}$ on M_k in F_l, this value is set to 1; otherwise, it is set to 0
$Z_{i,j}^{M}$: If there is a transfer between machines in a factory for $O_{i,j}$, this value is set to 1; otherwise, it is set to 0
$Z_{i,j}^{F}$: If there is a transfer between factories for $O_{i,j}$, this value is set to 1; otherwise, it is set to 0
Objectives:
(1) OF_1: Makespan

$$OF_1 = \max\{C_i \mid i = 1, 2, \ldots, n\} \tag{5}$$

(2) OF_2: Max factory load

$$OF_2 = \max\{W_l \mid l = 1, 2, \ldots, f\} \tag{6}$$

where W_l can be calculated by Equation (7):

$$W_l = \sum_{i=1}^{n} \sum_{j=1}^{N_{J_i}} \sum_{k=1}^{m} V_{i,j}^{l} * X_{i,j}^{k} * P_{i,j}^{k}, l = 1, 2, \ldots, f \tag{7}$$

(3) OF_3: Total workload

$$OF_3 = \sum_{i=1}^{n} \sum_{j=1}^{N_{J_i}} \sum_{l=1}^{f} \sum_{k=1}^{m} V_{i,j}^{l} * X_{i,j}^{k} * P_{i,j}^{k} \tag{8}$$

Subject to:

$$\sum_{l=1}^{f} \sum_{k=1}^{m} V_{i,j}^{l} * X_{i,j}^{k} = 1 \tag{9}$$

where $i = 1, 2, \ldots, n, j = 1, 2, \ldots, N_{j_i}$.

$$C_{i,j} - C_{i,j-1} \geq V_{i,j}^{l} * X_{i,j}^{k} * P_{i,j}^{k}, \forall i, j, l, k \tag{10}$$

$$C_{i,j} - C_{u,v} \geq Y_{i,j,u,v,k}^{l} * P_{i,j}^{k} + L * (Y_{i,j,u,v,k}^{l} - 1) \tag{11}$$

for all i, j, u, v, k, l.

$$Z_{i,j}^{M} + Z_{i,j}^{F} \in \{0, 1\}, \forall i, j \tag{12}$$

Equations (5), (6), and (8) are the three objective functions which include the makespan, max factory load, and total workload objectives; OF_1 determines the productivity of a factory, OF_2 determines the load balance between factories, and OF_3 determines the wear and tear on machines. Equation (9) guarantees that each job can only be processed by one machine at a time. Equation (10) guarantees the processing sequence constraints between previous and subsequent operations for the same job. Equation (11) guarantees that one machine can only process one job at a time. Equation (12) guarantees that an operation can only be transferred once between factories or machines.

5. The Proposed HDVMA

A memetic algorithm is a hybrid algorithm that combines a global search algorithm with a local search algorithm to improve the convergence performance. In our proposed HDVMA, we used the NSGA-II as the global optimization method and adopted VNS as the local optimization method. However, due to the high convergence of VNS, the population will converge to a few points too quickly. To maintain the diversity of the population, we introduce a set of weight vectors. The HDVMA can efficiently solve the MO-DFFJSPT.

5.1. Framework of the HDVMA

In this section, we introduce the framework of the whole algorithm in detail, including the initialization method, crossover strategy, mutation strategy, weight allocation strategy, and DVNS strategy. The framework of the HDVMA is described in Algorithm 1.

Algorithm 1 The framework of the HDVMA

```
Input: Setting the key parameters.
Output: Pareto solution set $POP$.
 1: Encode and initialize the population $POP$ according to the four rules.
 2: Decode $POP$ and calculate the three objective fitness values and crowding distances
    of $POP$.
 3: while the stopping criterion is not met do
 4:     Perform non-dominated sorting on $POP$ and execute 2 tournament selections on
        $POP$ to form the mating pool.
 5:     Population evolution (Sections 5.4 and 5.5).
 6:     Execute DVNS to generate $NEW\_POP$.
 7:     Merge $NEW\_POP$ and $POP$ to generate $ALL\_POP$.
 8:     Eliminate repeated individuals in $ALL\_POP$.
 9:     Calculate the crowding distance of $ALL\_POP$, perform non-dominated sorting for
        $ALL\_POP$, and select the best $N_p$ individuals to generate the next generation $POP$.
10: end while
```

5.2. Encoding and Decoding Method

The first step of the metaheuristic algorithm is to encode and decode the required problem. The encoding can transform the solution space to the search space, and the decoding can transform the search space into a scheduling scheme. In DFJSP, many researchers choose a three-layer encoding method [10,22] including factory encoding, machine encoding, and operation sequence encoding. However, the three-layer encoding method increases the complexity of the evolution operators. Therefore, we propose a new encoding method. Two vectors are used to represent the solution, and all the operation sequences, factory numbers, and machine numbers can be obtained from the two vectors.

The two vectors are the equipment allocation vector and operation sequence vector. The equipment allocation vector represents the allocation of machines and factories, and the operation sequence vector represents the operation process order. The feasibility of the machine allocation in flexible scheduling can be guaranteed by the index value of the available machine set for an operation. The operation sequence vector is constructed from the job number, and the procedures of a job are represented by the number of occurrences of the job. This encoding approach can ensure the feasibility of the solution, and there will be no infeasible solution.

A simple example of MO-DFFJSPT is constructed to illustrate the encoding method. The example consists of two factories (F_1, F_2); three machines (M_1, M_2, M_3) are assigned to F_1 and two machines (M_4, M_5) are assigned to F_2. The ten operations of three jobs are assigned to their respective candidate machines for processing; the detailed processing information of the example is shown in Table 1. Figure 2 shows an example of the encoding method; the total number of each vector is the total number of operations and the operation sequence is ($O_{3,1}$, $O_{1,1}$,. . ., $O_{3,3}$), as shown in Figure 2. The value of the equipment allocation vector represents the index of the available machine set. For instance, as the index of operation $O_{1,3}$ is 2, it is assigned the second index value among the available machines, i.e., machine number 3, while operation $O_{3,1}$ is assigned to machine number 5. It is notable that when the machine number is determined, the factory number is determined as well, because machine number 3 belongs to factory number 1, and the factory number of $O_{1,3}$ is set to 1. For the same reason, the factory number of $O_{3,1}$ is set to 2.

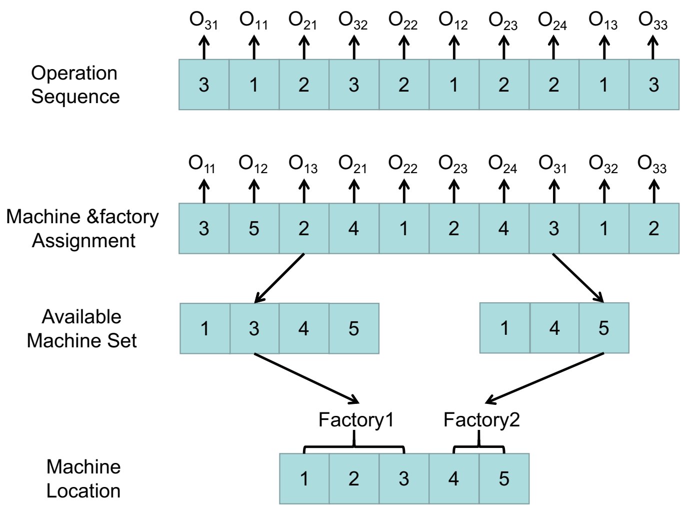


Figure 2. Solution representation.

Table 1. Example of MO-DFFJSPT.

Jobs	**Operations**	F_1			F_2	
		M_1	M_2	M_3	M_4	M_5
J_1	$O_{1,1}$	—	(1,2,3)	(2,3,4)	—	(1,3,5)
	$O_{1,2}$	(4,5,6)	(2,3,4)	(5,8,9)	(5,7,8)	(7,9,10)
	$O_{1,3}$	(4,5,7)	—	(2,3,4)	(1,2,3)	(8,10,12)
J_2	$O_{2,1}$	(4,7,8)	(5,7,10)	(1,2,3)	—	(1,3,5)
	$O_{2,2}$	(5,6,7)	(1,2,4)	(8,10,12)	—	(4,5,6)
	$O_{2,3}$	(1,3,4)	(4,5,8)	—	(2,3,4)	(5,6,7)
	$O_{2,4}$	(1,2,3)	(9,10,11)	—	(6,8,10)	(2,4,5)
J_3	$O_{3,1}$	(4,5,6)	—	—	(2,3,4)	(1,3,5)
	$O_{3,2}$	—	(12,13,17)	(10,11,13)	(14,15,16)	—
	$O_{3,3}$	(5,6,7)	—	(7,9,10)	(10,11,13)	—

Decoding yields a feasible scheduling scheme. In this section, we use a insert decoding method [23] that can process the operations as early as possible. The decoding steps are shown in Algorithm 2.

$$T_a = max\{C_{i,j-1} + T_T, T_s\} \tag{13}$$

$$T_a + P^k_{i,j} \leq T_e \tag{14}$$

Algorithm 2 Insert decoding method

```
Input: An operation sequence vector $OS$.
Output: A scheduling scheme.
1: while $OS$ is not empty do
2:     The fuzzy transfer time $T_T$ is set to [0,0,0].
3:     Extract the first gene from $OS$ as $CV$.
4:     Obtain $O_{i,j}$, $F_l$, $M_k$ and $P^k_{i,j}$ on $M_k$ of the $CV$.
5:     if $O_{i,j}$ is transferred from another factory then
6:         $T_T \Leftarrow T_F$
7:     end if
8:     if $O_{i,j}$ is transferred from another machine in the same factory then
9:         $T_T \Leftarrow T_M$
10:    end if
11:    Obtain all idle time periods of the machine.
12:    If the insertion criteria are met, $O_{i,j}$ is inserted and the idle time periods are updated.
13:    Remove $CV$ from $OS$.
14: end while
```

Equations (13) and (14) represent the insertion criteria. When Equation (14) is met, $O_{i,j}$ can be inserted into the current idle time period, where T_s and T_e denote the start time and end time, respectively, for each idle time period. T_a represents the start time of the current workpiece processing, and T_T represents the time taken for the workpiece to be transferred from other machines or factories to the current machine. A simple decoding example with fuzzy processing and transportation time is shown in Figure 3. The inverted triangle and upright triangle represent the start time and end time, respectively, the dotted triangle represents the transfer time, and the filled triangle represents the processing time.

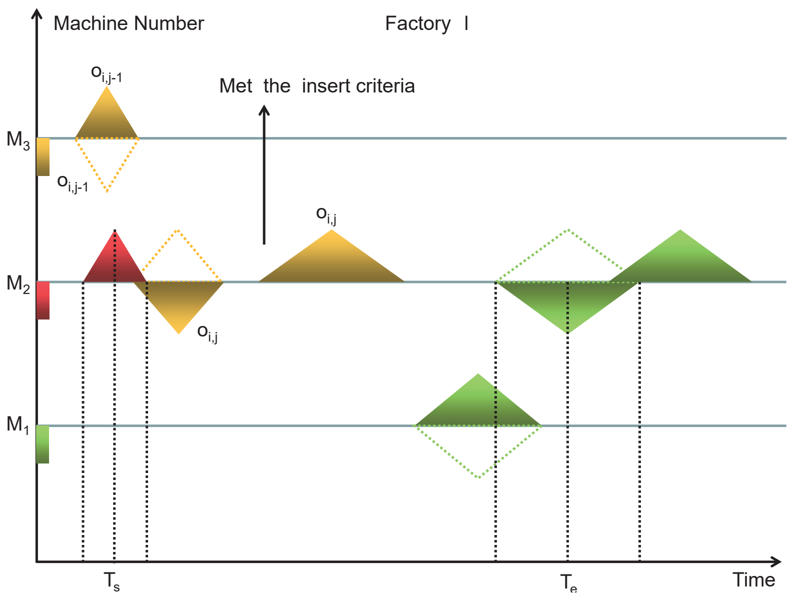


Figure 3. An example of the insert decoding method.

5.3. Population Initialization

In this section, four initialization rules are proposed to generate a high-quality population. To maintain the high quality and diversity of the initial population, we use different rules to design initialization strategies. The four rules are as follows.

Global rule: The global rule aims to balance the machine load and reduce the scheduling completion time (OF_1).

1. Generate a machine load array to store the total processing time of each machine. The length of the array is the total number of machines. The initial value of the array is set to 0.
2. Select a job randomly, and obtain the candidate machine number and fuzzy processing time of each operation of the job.
3. According to the operation sequence of the selected job, add the candidate processing time of each operation to the corresponding machine load array, find the lowest value in the array, and store the corresponding machine.
4. Update the machine load array, and repeat step 3 until all operations of the selected job are assigned to a machine.
5. Select other jobs in the same way until all jobs have been selected.

Factory load rule: The factory load rule aims to reduce the maximum factory load (OF_2). While it is similar to the global rule, the difference is that it generates a factory load array rather than a machine load array to balance the workload between factories.

Workload rule: The workload rule aims to reduce the total workload (OF_3). Each operation is allocated to the machine with the shortest processing time.

Random rule: The random rule randomly assigns each operation to a machine to ensure the diversity of the initial population.

The combination of the first three initialization rules improves the quality of the initial population. The random rule ensures the quality of the population and maintains the diversity of the population, preventing it from easily falling into a local optimum in the search process. Finally, the probabilities of these four initialization rules are determined to be 0.5, 0.1, 0.1, and 0.3, respectively.

5.4. Crossover

Crossover is an important part of the algorithm, as it determines the global search ability of the algorithm. Due to the different encoding methods of the two vectors, we adopt two different crossover methods: precedence operation crossover(POX) and multi-point crossover(MPX) [24].

Operation sequence crossover: POX can preserve the good characteristics of the two parent chromosomes to produce better operation sequence vectors for the offspring.

1. Randomly divide the job set into two nonempty sets (Job1,Job2).
2. Find the genes belonging to Job1 from Parent1 and copy them to New1 at the same position; find the genes belonging to Job2 from Parent2 and copy them to New2 at the same position.
3. Find the genes belonging to Job1 from Parent1 and copy them from left to right to the unassigned positions in New2; find the genes belonging to Job2 from Parent2 and copy them from left to right to the unassigned positions in New1. An example of this procedure is shown in Figure 4.

Equipment allocation crossover: MPX is adopted to generate feasible equipment allocation vectors.

1. Randomly generate several positions as crossover positions.
2. Exchange the gene values of Parent1 and Parent2 based on the generated crossover positions. An example of this procedure is shown in Figure 5.

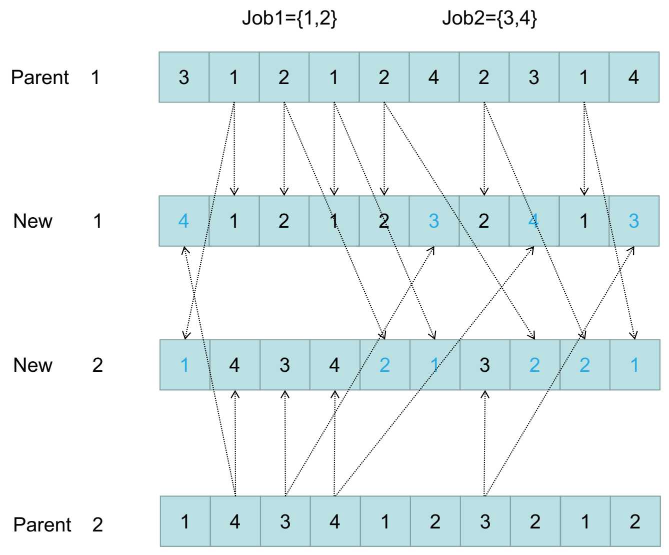


Figure 4. POX crossover.

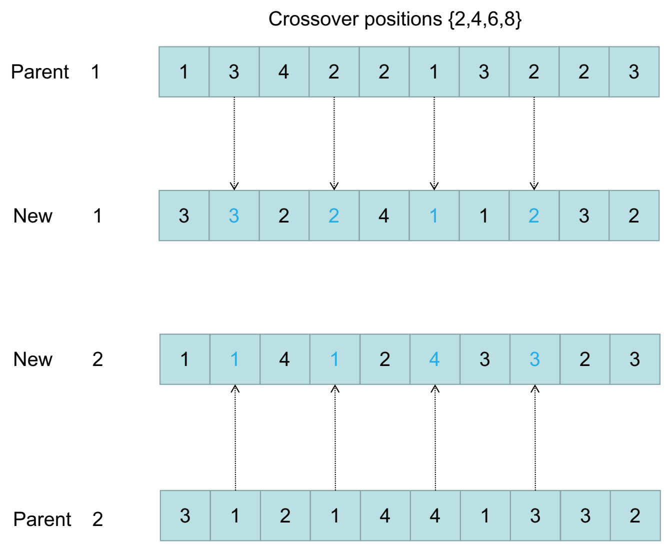


Figure 5. MPX crossover.

5.5. Mutation

The mutation operation maintains the diversity of the population during evolution and prevents a reduction in diversity from leading to convergence to a local optimum. In the process of mutation, we always expect the mutation to lead in a good direction; thus, we propose two mutation strategies to reduce the three objective values of the MO-DFFJSPT.

Operation sequence mutation:

1. Randomly select two positions, P_1 and P_2, in the operation sequence vector.
2. Exchange the values of P_1 and P_2.

Equipment allocation mutation:

1. Randomly select two positions, P_1 and P_2, in the equipment allocation vector.
2. Obtain $O_{i,j}$, the candidate machine number, and the corresponding fuzzy processing times of P_1 and P_2.
3. Change the current machine of P_1 and P_2 to the machine with the minimum fuzzy processing time other than the current machine.

For each individual among the offspring, we only perform one mutation operation on it. If the mutation conditions are met, a random number from 0 to 1 is generated. When the random number is less than 0.5, the operation sequence mutation is executed; otherwise, the equipment allocation mutation is executed.

5.6. DVNS

In this section, a VNS method combining three local search strategies is proposed, and a weight vector is introduced to evaluate the solution quality and design an acceptance criterion after performing VNS.

5.6.1. Individual Selection Criteria

There are two problems that need to be solved before conducting the local search of the population:

1. What is the probability of performing local search?
2. Which individuals are selected for local search?

The answer to the first question is determined in Section 6.3. For the purpose of selecting high-quality individuals, we introduce a set of weight functions to evaluate the quality of individuals. The corresponding equation is shown as Equation (15):

$$F(x,\lambda) = \lambda_1 OF_1(x) + \lambda_2 OF_2(x) + \lambda_3 OF_3(x) \tag{15}$$

where $OF_1(x), OF_2(x), OF_3(x)$ denotes the three objective function values after decoding vector x and $\lambda_1, \lambda_2, \lambda_3$ denotes uniformly distributed vectors. A set of weight vectors satisfies the following constraints:

$$\lambda_1 + \lambda_2 + \lambda_3 = I, \lambda_1, \lambda_2, \lambda_3 \in \{1, 2, \ldots, I\} \tag{16}$$

In Equation (16), I is set to 23 to generate 300 sets of weight vectors [25]. When a local search is performed, a set of weight vectors is selected randomly from among the 300 sets of uniformly distributed vectors as individual quality criteria. The value of $F(x,\lambda)$ is used to perform tournament selection for the current population in order to obtain high-quality individuals for the VNS. Algorithm 3 provides the detailed steps of individual selection.

Algorithm 3 High-quality individual selection

```
Input: Current population $POP$, local search probability $P_l$, population size $N_p$, 300 sets
    of weight vectors $\lambda$=($\lambda_1, \lambda_2, \lambda_3$).
Output: High-quality individuals $POP_h$, selected weight vector sets $\lambda_s$.
1: for $i = 1$ to $N_p * P_l$ do
2:     Randomly select a group of vectors from the set of weight vectors $\lambda$. Calculate $F(x,\lambda)$
       according to Equation (15) for each individual, and perform tournament selection to
       select a high-quality individual.
3:     Store the selected individual in $POP_h$ and store the selected weight vector set in $\lambda_s$.
4: end for
```

5.6.2. VNS Method

In this section, three neighborhood structures of VNS are introduced in detail. The combination of the three neighborhoods expands the search space of the neighborhood and improves the convergence performance of the algorithm. The three local search strategies are shown in Algorithms 4–6.

Algorithm 4 Strategy LS_1

```
Input: An individual vector $X$.
Output: Vector $X_{LS1}$ after performing local search $LS_1$.
1: Decode $X$ and obtain the number $F_m$ of the factory with the maximum load.
2: Obtain the operations processed in factory $F_m$ and store them in set $S_m$.
3: Randomly select an operation $O_r$ from $S_m$ and obtain its candidate fuzzy processing
   time array $A_c$, current gene value $G_c$ and candidate machine index array $I_a$ in $F_m$.
4: $A_c(G_c, 1:3) \Leftarrow [inf, inf, inf]$.
5: for i=1 to size($I_a$) do
6:    $A_c(I_a(i), 1:3) \Leftarrow [inf, inf, inf]$.
7: end for
8: Obtain the minimum value $V_m$ in fuzzy array $A_c$ and its index $I_m$ in $A_c$.
9: if $V_m == [inf, inf, inf]$ then
10:    Execute Algorithm 5.
11: else
12:    Change $G_c$ to $I_m$.
13: end if
```

Algorithm 5 Strategy LS_2

```
Input: An individual vector $X$.
Output: Vector $X_{LS2}$ after local search $LS_2$.
1: Randomly select an operation $O_r$ from among all operations.
2: Obtain the candidate fuzzy processing time array $A_c$ and current gene value $G_c$ of $O_r$.
3: $A_c(G_c, 1:3) \Leftarrow [inf, inf, inf]$.
4: Obtain the minimum value $V_m$ in fuzzy array $A_c$ and its index $I_m$ in $A_c$.
5: Change $G_c$ to $I_m$.
```

Algorithm 6 Strategy LS_3

```
Input: An individual vector $X$.
Output: Vector $X_{LS3}$ after local search $LS_3$.
1: Obtain the critical path of individual $X$.
2: Randomly select two positions $P_1$ and $P_2$ from the critical path.
3: Exchange the gene values of $P_1$ and $P_2$.
```

The first two local search strategies are designed for equipment allocation vectors. Algorithm 4 reduces the maximum factory load (OF_2) by changing the processing machine in the maximum-load factory to the machine with the minimum processing time among the other factories. The purpose of Algorithm 5 is to reduce the total workload (OF_3) by changing the current machine to a machine with the minimum processing time.

The third local search strategy is designed for the operation sequence vectors. As many previous studies [25,26] have proven that it is possible to reduce the makespan (OF_1) only by moving an operation on the critical path, we process the operations on the critical path to produce better scheduling results. Algorithm 6 exchanges the positions of two operations on the critical path to reduce the makespan of the MO-DFFJSPT. An example of Algorithm 4 is shown in Figure 6, and Algorithm 7 shows the steps of the entire VNS process.

Algorithm 7 VNS

```
Input: Neighborhood number $T$, high-quality individual set $POP_h$, selected weight vector
    sets $\lambda_s$.
Output: Individual sets $POP_v$ after the VNS.
1: for $i = 1$ to size$(POP_h)$ do
2:   $(\lambda_{c1}, \lambda_{c2}, \lambda_{c3}) \Leftarrow \lambda_s(i, 1:3)$.
3:   $POP_c \Leftarrow POP_h(i, :)$.
4:   Decode $POP_c$ and calculate $F(x, \lambda)(F_c)$ according to Equation (15).
5:   $k \Leftarrow 1$.
6:   while $k \leq 3$ do
7:     for $j = 1$ to $T$ do
8:       if $k == 1$ then
9:         Execute Algorithm 4 for $POP_c$ to generate $POP_{LS}$.
10:      end if
11:      if $k == 2$ then
12:        Execute Algorithm 5 for $POP_c$ to generate $POP_{LS}$.
13:      end if
14:      if $k == 3$ then
15:        Execute Algorithm 6 for $POP_c$ to generate $POP_{LS}$.
16:      end if
17:      Decode $POP_{LS}$ and calculate $F(x, \lambda)$.
18:      $F_{all}(j) \Leftarrow F(x, \lambda)$.
19:    end for
20:    Find the minimum value $F_{min}$ in $F_{all}$ and its corresponding vector $POP_{min}$.
21:    if $F_{min} \leq F_c$ then
22:      $k \Leftarrow 1, POP_c \Leftarrow POP_{min}, F_c \Leftarrow F_{min}$.
23:    else
24:      $k \Leftarrow k + 1$.
25:    end if
26:  end while
27:  $POP_v(i, :)$=$POP_c$.
28: end for
```

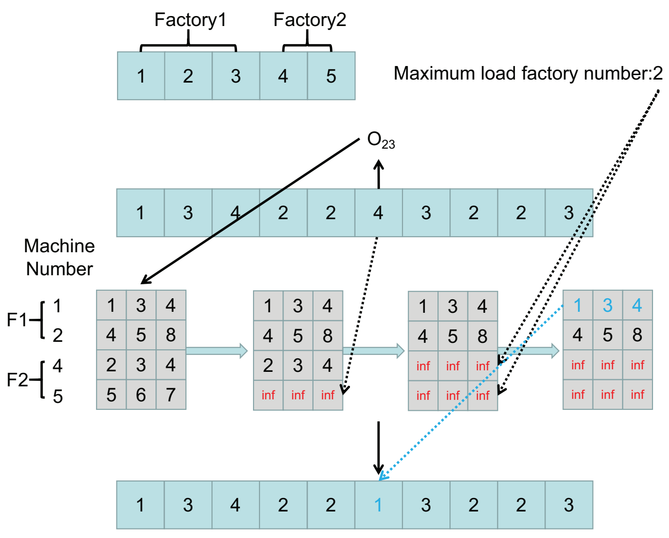


Figure 6. LS_1 example.

5.6.3. Acceptance Criteria

After performing VNS on the high-quality individuals, we need to judge whether the solution after the VNS is better than the original solution. In this section, in order to maintain the diversity of the population and prevent it from becoming trapped in local optima, we use Equation (15) to judge the quality of the solution. For each locally searched solution, the corresponding weight vector is extracted from the weight vector set λ_s, then the value $F(x, \lambda)$ is calculated. If the value is less than that of the original solution, the new solution is accepted. All the flows of DVNS are shown in Algorithm 8.

Algorithm 8 DVNS

```
Input: Population $POP$.
Output: Population $POP_D$ after DVNS.
1: Execute Algorithm 3 to select high-quality individuals $POP_h$.
2: Execute Algorithm 7 for $POP_h$ to generate individuals $POP_v$ after local search.
3: Replace $POP_h$ in $POP$ with $POP_v$ to generate $POP_D$.
```

6. Experimental Results and Discussion

The HDVMA was executed in MATLAB R2017a and computed on an Intel(R) Core i7 2.60 GHz processor with 16.0 GB RAM. To avoid the influence of randomness as much as possible, each experiment was carried out ten times and the average value was taken as the experimental result.

6.1. Benchmark Construction

Because there are no benchmarks for the MO-DFFJSPT, we extend fifteen benchmarks to test the effectiveness of the proposed algorithm. The first set of benchmarks is obtained from Lei [21,27]. We assign the machines to different factories to generate DFInstance01–DFInstance05. However, as Lei's benchmarks are all completely flexible, we expand another partially flexible benchmarks, which are obtained from the famous MK benchmarks [28]; we transform MK to fuzzy Mk (DFMK01-DFMK10) according to Lei's fuzzification method [29]. The distributed workshop is generated as follows:

1. The maximum number of factories is set to [2,3,4] according to the benchmark scale.
2. The difference in the number of machines between factories does not exceed 1.

The scheduling benchmark details are shown in Table 2. In addition, the fuzzy transfer time T_M between machines is set to (1,2,3) units, and the fuzzy transfer time T_F between factories is set to (8,10,12) units.

Table 2. Constructed benchmarks.

Benchmark	Size (n × f × m)	Source
DFMK01	10 × 2 × 6	MK01
DFMK02	10 × 2 × 6	MK02
DFMK03	15 × 2 × 4	MK05
DFMK04	20 × 2 × 5	MK07
DFMK05	15 × 3 × 8	MK03
DFMK06	15 × 3 × 8	MK04
DFMK07	20 × 3 × 10	MK08
DFMK08	20 × 3 × 10	MK09
DFMK09	10 × 4 × 15	MK06
DFMK10	20 × 4 × 15	MK10
DFInstance01	10 × 3 × 10	Instance01
DFInstance02	10 × 3 × 10	Instance02
DFInstance03	10 × 3 × 10	Instance03
DFInstance04	10 × 3 × 10	Instance04
DFInstance05	15 × 3 × 10	Instance05

6.2. Performance Metrics

To test the performance of different algorithms, the following three metrics are used to evaluate the algorithms.

1. HV metric:

$$HV(P,r) = \bigcup_{x \in P}^{P} v(x,r) \tag{17}$$

where P denotes the non-dominated solution set generated by the algorithm and $v(x,r)$ denotes the hypercube formed between a solution in the obtained Pareto front and the reference point r. r usually takes the maximum value after the normalization of the objective, i.e., (1, 1, 1), to obtain HV.

2. IGD metric:

$$IGD(P,P^*) = \frac{1}{\mid P^* \mid} \sum_{x \in P^*} \min_{y \in P} dis(x,y) \tag{18}$$

where P denotes the non-dominated solution set generated by the algorithm, P^* denotes the Pareto front, and $dis(x,y)$ denotes the minimum Euclidean distance between points x and y.

3. Spread metric:

$$Spread = \frac{d_l + d_f + \sum_{i=1}^{N-1} \mid d_i - \overline{d} \mid}{d_l + d_f + (N-1)\overline{d}} \tag{19}$$

where the parameters d_f and d_l are the Euclidean distances between the extreme solutions and the boundary solutions of the obtained non-dominated set, d_i denotes the Euclidean distance between consecutive solutions in the obtained non-dominated set of solutions, $\overline{d}$ denotes the average value of d_i, and N denotes the number of non-dominated solution sets.

In the above three metrics, HV can measure the comprehensive performance of the algorithm, IGD can measure the convergence and diversity of the algorithm, and Spread can measure the uniformity of the solution distribution in the algorithm. At the end of each algorithm run, we calculate three metrics values based on the Pareto frontier and the non-dominated set of solutions generated by the algorithm. The Pareto frontier for each benchmark is generated by the sets of non-dominated solutions for all algorithms. The lower the IGD value and Spread are, the better the algorithm performance, and the higher the HV value is, the better the algorithm's performance. Additionally, as all of the objective function values are TFNs, we need to calculate the expected value of each TFN and then standardize them to calculate the three metrics. The expected value calculation method [19] and the standardization method are shown in Equations (20) and (21), respectively:

$$E(a_{\xi}) = \frac{1}{4}(a_1 + 2a_2 + a_3) \tag{20}$$

$$OF_i = \frac{OF_i^o - OF_i^{\min}}{OF_i^{\max} - OF_i^{\min}}, i \in \{1,2,3\} \tag{21}$$

where OF_i^o denotes the original expected objective function value. $OF_i^{\max}$ and $OF_i^{\min}$ denote the maximum and minimum expected values, respectively, of the i-th objective for all solutions.

6.3. Parameter Settings

Different parameters affect the overall performance of the algorithm, and thus we need to adjust the key parameters of the algorithm. The invariant parameters include the population number N_p, number of iterations $MaxIt$, and neighborhood number T; N_p and $MaxIt$ are set to 100. Considering the high time complexity of the DVNS, T is set to 3.

In addition, there are three important variable parameters in the proposed algorithm: the mutation probability, P_m, local search probability, P_l, and number of tournament selections, N_t, in DVNS. The Taguchi design-of-experiment (DOE) approach is adopted here to determine the best parameter values. In this paper, the four levels for each parameter are as follows:

- N_t:$\{5, 10, 15, 20\}$
- P_l:$\{0.05, 0.10, 0.15, 0.20\}$
- P_m:$\{0.1, 0.2, 0.3, 0.4\}$

An orthogonal array $L_{16}(4^3)$ is adopted to measure the impact of each parameter level on a medium scale benchmark (DFInstance03), and we use a normalization value called the objective deviation sum (ODS) [10] as the metric, which is shown in Equation (22):

$$ODS(OF^c) = \sum_{i=1}^{3}\left(\frac{OF_i^c - \min\{OF_{all}^i\}}{\min\{OF_{all}^i\}} * 100\right) \tag{22}$$

where OF_i^c denotes the i-th average expected objective value of OF^c and $\{OF_{all}^i\}$ denotes the i-th average expected objective value set of all combinations; the smaller the ODS value is, the better the performance of the algorithm.

Table 3 shows the DOE results, and Figure 7 shows the influence of different levels of the three factors on the performance of the algorithm. From Figure 7, we find that the local search probability has the greatest impact on the ODS value, and the higher the local search probability is, the lower the ODS value, which shows the effectiveness of DVNS. However, the decrease in ODS from three to four levels is not a large difference. Considering the time complexity and algorithm performance, we choose 0.15 as the local search probability, and N_t and P_m are determined to be 10 and 0.1, respectively, according to Figure 7. To determine P_l change from level 3 to level 4 does not significantly affect the efficiency of HDVMA, we designed two comparative experiments (DFInstance02, DFMK05). The value of P_l is taken as 0.15 and 0.2 respectively, and the influence of P_l on the algorithm at level 3 and level 4 is observed by calculating three metrics.

Table 3. L_{16} Taguchi experiment for the ODS.

Number	Factor Levels			ODS
	N_t	P_l	P_m	
1	1	1	1	5.8898
2	1	2	2	6.0353
3	1	3	3	4.8123
4	1	4	4	5.3476
5	2	1	2	5.7420
6	2	2	1	5.6049
7	2	3	4	5.2423
8	2	4	3	5.0575
9	3	1	3	7.2676
10	3	2	4	5.8657
11	3	3	1	4.5849
12	3	4	2	5.1784
13	4	1	4	6.5863
14	4	2	3	5.7924
15	4	3	2	6.0990
16	4	4	1	4.2528

In Table 4, HDVMA-0.15 denotes the HDVMA with the P_l of 0.15, while HDVMA-0.20 denotes the HDVMA with the P_l of 0.20. It can be seen that the gap between HDVMA-0.20 and HDVMA-0.15 in the three metrics is very small, which proves the effectiveness of a P_l value of 0.15.

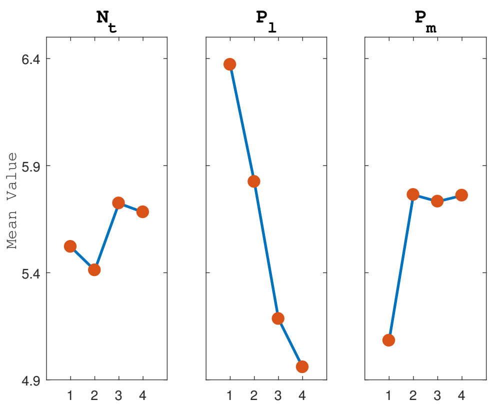


Figure 7. Factor levels.

Table 4. Comparison of P_l level.

Benchmarks	**HV**		**IGD**		**Spread**	
	HDVMA-0.15	**HDVMA-0.20**	**HDVMA-0.15**	**HDVMA-0.20**	**HDVMA-0.15**	**HDVMA-0.20**
DFMK05	0.7011	**0.7213**	0.0364	**0.0350**	0.6974	**0.6874**
DFInstance02	**0.6586**	0.6579	0.0515	**0.0505**	**0.4961**	0.5171

6.4. Effectiveness of the Initialization Strategy

In this section, we discuss the effectiveness of the initialization strategy. We use HDVMA-NIS to denote the HDVMA without an initialization strategy, meaning that the initial population is generated randomly. We tested the performance of the HDVMA-NIS and HDVMA on fifteen instances (DFMK01–DFMK10 and DFInstance01–DFInstance05) and calculated each HV, IGD, and Spread value on each benchmark. All the algorithms were run ten times, and the average values were taken as the final results shown in Table 5.

Table 5. Comparison of initialization strategy.

Benchmarks	**HV**		**IGD**		**Spread**	
	HDVMA-NIS	**HDVMA**	**HDVMA-NIS**	**HDVMA**	**HDVMA-NIS**	**HDVMA**
DFMK01	0.6471	**0.6624**	0.0807	**0.0797**	**0.4504**	0.4588
DFMK02	0.6486	**0.6591**	0.1588	**0.1442**	0.7462	**0.6997**
DFMK03	0.5682	**0.6644**	0.0569	**0.0317**	0.6723	**0.4440**
DFMK04	0.6196	**0.6299**	0.0572	**0.0531**	0.4306	**0.4301**
DFMK05	0.6517	**0.7011**	0.0508	**0.0364**	0.7625	**0.6974**
DFMK06	0.6578	**0.7470**	0.0526	**0.0341**	0.6023	**0.5506**
DFMK07	0.5490	**0.6344**	0.0813	**0.0622**	0.4220	**0.4200**
DFMK08	0.5386	**0.7093**	0.0768	**0.0436**	0.6828	**0.5330**
DFMK09	**0.5549**	0.5453	**0.0747**	0.0775	**0.5392**	0.5473
DFMK10	0.5360	**0.6527**	0.0757	**0.0664**	0.4899	**0.4407**
DFInstance01	**0.5413**	0.5284	**0.0844**	0.0867	0.5920	**0.5656**
DFInstance02	0.6306	**0.6586**	0.0643	**0.0515**	0.5057	**0.4961**
DFInstance03	**0.6609**	0.6273	0.0520	**0.0491**	0.5759	**0.5672**
DFInstance04	**0.6012**	0.5356	**0.0909**	0.1143	**0.6416**	0.6853
DFInstance05	**0.6592**	0.6293	0.0942	**0.0900**	0.6022	**0.5951**

In Table 5, the bold values in each metric represent better results. We observe that the three metrics of the HDVMA are better than those of the HDVMA-NIS on most DFMK benchmarks; however, the HV metrics of the HDVMA are slightly worse than those of the HDVMA-NIS on four DFInstance benchmarks. This is because on the completely flexible benchmarks there are too many candidate machines for each operation, and the initialization strategy may not obtain enough high-quality genes, thus losing points on the Pareto frontiers and resulting in a slight decrease in the HV value. However, this is acceptable, and the initialization strategy performs well on IGD and Spread, which proves that the initialization strategy can improve the convergence of the algorithm and the universality of the solution set distribution. In general, the results in Table 5 demonstrate that whether the HV, IGD, or Spread is used, the HDVMA achieves better results than the HDVMA-NIS on most benchmarks. Based on this, we can conclude that the proposed initialization strategy initializing the population with the four rules improves the quality of the initial population, effectively producing a high-quality initial population for the MO-DFFJSPT.

6.5. Effectiveness of the DVNS Strategy

In this section, we use HDVMA-NLS to denote the HDVMA algorithm without the DVNS strategy. The HDVMA-NLS and HDVMA were run ten times on each benchmark to demonstrate the superiority of the DVNS strategy. Better metric values are shown in bold.

As shown in Table 6, in terms of HV, the HDVMA outperforms the HDVMA-NLS on all fifteen benchmarks except for DFMK05. In terms of IGD, the performance of the HDVMA is slightly worse only on DFMK04 and is higher than that of the HDVMA-NLS on the other benchmarks. In the comparison of Spread, we find that the HDVMA performs worse than the HDVMA-NLS on only two benchmarks. This fully demonstrates the effectiveness of the proposed DVNS. The three well-designed local search strategies expand the solution space during the search, improve the convergence speed of the algorithm, and help it to avoid local optima. The individual selection strategy based on weight preference and acceptance criteria ensures the convergence of the algorithm and the diversity of solutions, and the solutions are more evenly distributed on the Pareto front.

Table 6. Comparison of DVNS strategy.

Benchmarks	**HV**		**IGD**		**Spread**	
	HDVMA-NLS	**HDVMA**	**HDVMA-NLS**	**HDVMA**	**HDVMA-NLS**	**HDVMA**
DFMK01	0.5983	**0.6624**	0.0960	**0.0797**	**0.4563**	0.4588
DFMK02	0.5638	**0.6591**	0.2009	**0.1442**	0.8341	**0.6997**
DFMK03	0.6214	**0.6644**	0.0422	**0.0317**	0.4900	**0.4440**
DFMK04	0.5923	**0.6299**	**0.0512**	0.0531	0.4662	**0.4301**
DFMK05	**0.7110**	0.7011	0.0378	**0.0364**	0.7177	**0.6974**
DFMK06	0.6873	**0.7470**	0.0444	**0.0341**	0.6076	**0.5506**
DFMK07	0.6269	**0.6344**	0.0698	**0.0622**	**0.3866**	0.4200
DFMK08	0.6968	**0.7093**	0.0510	**0.0436**	0.5650	**0.5330**
DFMK09	0.4200	**0.5453**	0.1341	**0.0775**	0.6358	**0.5473**
DFMK10	0.5898	**0.6527**	0.1087	**0.0664**	0.4608	**0.4407**
DFInstance01	0.3806	**0.5284**	0.1175	**0.0867**	0.6412	**0.5656**
DFInstance02	0.5010	**0.6586**	0.1572	**0.0515**	0.5368	**0.4961**
DFInstance03	0.5194	**0.6273**	0.0852	**0.0491**	0.6008	**0.5672**
DFInstance04	0.4135	**0.5356**	0.1553	**0.1143**	0.7287	**0.6853**
DFInstance05	0.5087	**0.6293**	0.1455	**0.0900**	0.6037	**0.5951**

Therefore, we can conclude that DVNS can enable the algorithm to obtain a better solution set and further improve the performance of the algorithm.

6.6. Comparison with Other Algorithms

In this section, to further evaluate the performance of the HDVMA, we choose four algorithms for comparison: NSGA-II, NSGA-III, MOEA/D, and DJAYA. Among them, NSGA-II, NSGA-III, and MOEA/D are all classical multiobjective optimization algorithms that are widely used in many fields. DJAYA [30] is a recently proposed algorithm for solving the multiobjective flexible job shop scheduling problem.

All algorithms use the same encoding and decoding methods and the same mutation operator. In order to ensure the fairness of the experiment, the population number, maximum number of iterations, and mutation probability of all algorithms are set to the same values, (100,100,0.1), except MOEA/D and NSGA-III. Due to the existence of weight vectors and reference points, the population numbers of MOEA/D and NSGA-III are set to the number closest to 100, i.e., 105. The number of neighborhoods in MOEA/D is set to 10. All the algorithms were run ten times independently in each benchmark, and the average value was taken as the final result.

Tables 7–9 show the HV, IGD and Spread values of the five algorithms, respectively. The HDVMA achieves better performance than all other algorithms in terms of every metric on almost all benchmarks. Regarding Spread, the HDVMA is inferior to NSGA-III and MOEA/D on only three benchmarks, performing well on the other benchmarks.

Table 7. HV value comparison.

Benchmarks	NSGA-II	NSGA-III	MOEA/D	DJAYA	HDVMA
DFMK01	0.5690	0.6106	0.4775	0.5037	**0.6624**
DFMK02	0.4793	0.5012	0.2522	0.2757	**0.6847**
DFMK03	0.4577	0.4981	0.2799	0.4788	**0.6644**
DFMK04	0.4748	0.4509	0.3378	0.2815	**0.6447**
DFMK05	0.2483	0.2801	0.0860	0.1620	**0.7011**
DFMK06	0.4946	0.4853	0.4239	0.4778	**0.7401**
DFMK07	0.4440	0.4370	0.3338	0.2564	**0.6273**
DFMK08	0.2654	0.3173	0.1439	0.1587	**0.7145**
DFMK09	0.1958	0.2890	0.0695	0.1337	**0.5883**
DFMK10	0.1862	0.2275	0.1009	0.1157	**0.6613**
DFInstance01	0.4604	0.4199	0.3707	0.4829	**0.5456**
DFInstance02	0.5426	0.4514	0.3451	0.5537	**0.6841**
DFInstance03	0.5516	0.4268	0.3691	0.5437	**0.6369**
DFInstance04	0.5217	0.4657	0.3623	0.4904	**0.6527**
DFInstance05	0.2718	0.2150	0.1356	0.3637	**0.6293**

Table 8. IGD value comparison.

Benchmarks	NSGA-II	NSGA-III	MOEA/D	DJAYA	HDVMA
DFMK01	0.0771	0.0792	0.1388	0.1746	**0.0565**
DFMK02	0.1516	0.1444	0.2408	0.2796	**0.0814**
DFMK03	0.0608	0.0717	0.1767	0.1817	**0.0228**
DFMK04	0.0640	0.0734	0.0982	0.3528	**0.0362**
DFMK05	0.1745	0.1653	0.2160	0.4757	**0.0165**
DFMK06	0.0860	0.1072	0.1256	0.1695	**0.0301**
DFMK07	0.0835	0.0805	0.1166	0.2508	**0.0620**
DFMK08	0.1891	0.2008	0.2828	0.5683	**0.0214**
DFMK09	0.1786	0.1929	0.2393	0.5937	**0.0266**
DFMK10	0.2233	0.2375	0.3141	0.5948	**0.0310**
DFInstance01	0.0862	0.1212	0.1341	0.1537	**0.0323**
DFInstance02	0.0962	0.1405	0.1421	0.1463	**0.0311**
DFInstance03	0.0714	0.1336	0.1322	0.1328	**0.0303**
DFInstance04	0.1161	0.2346	0.1589	0.1863	**0.0614**
DFInstance05	0.2242	0.4316	0.2589	0.2864	**0.0241**

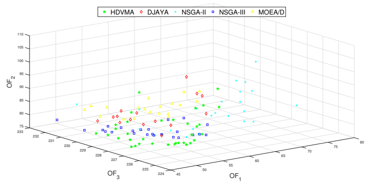
The HDVMA is superior to the other algorithms in convergence and more uniform in the distribution of solutions along the Pareto front. This excellent convergence performance is due to the fact that the initialization strategy produces a high-quality initialization population, which makes it easier for the algorithm to converge, while the three local search methods of DVNS further enhance the convergence performance of the algorithm. The more uniform distribution of the algorithm is due to the introduction of weight vectors.

Because the individual selection strategies and acceptance criteria are affected by different weights, the algorithm can accept new solutions with different weighting preferences and obtain solution sets with higher diversity and wider distributions.

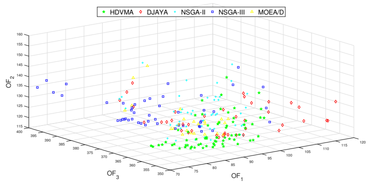
Table 9. Spread value comparison.

Benchmarks	NSGA-II	NSGA-III	MOEA/D	DJAYA	HDVMA
DFMK01	0.4735	0.4803	**0.4507**	0.4762	0.4770
DFMK02	0.7896	0.8015	1.0096	0.8362	**0.7210**
DFMK03	0.7414	0.7280	0.8923	0.6290	**0.4697**
DFMK04	0.4741	0.4727	0.5453	0.6106	**0.4195**
DFMK05	0.8243	0.7590	0.9287	0.7388	**0.6807**
DFMK06	0.6696	0.6749	0.7724	0.6337	**0.5532**
DFMK07	0.4775	0.4692	0.5123	0.6303	**0.4104**
DFMK08	0.7885	0.7316	0.7985	0.7186	**0.5616**
DFMK09	0.7554	**0.6602**	0.9057	0.7420	0.6903
DFMK10	0.6329	0.6540	0.8001	0.7662	**0.4421**
DFInstance01	0.7162	0.7481	0.8170	0.7309	**0.6127**
DFInstance02	0.6890	0.7571	0.7912	0.7023	**0.6523**
DFInstance03	0.7486	0.7310	0.8653	0.6954	**0.6608**
DFInstance04	0.7407	**0.6880**	0.8415	0.7038	0.6959
DFInstance05	0.7727	0.8001	0.9189	0.7725	**0.6572**

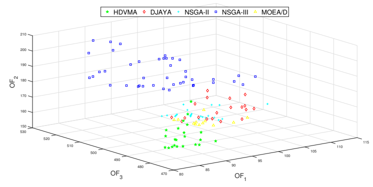
To observe the performance of different algorithms, we plot the non-dominated solution set graph generated by all algorithms on the six representative benchmarks from DFMK and DFInstance(DFInstance01, DFInstance03, DFInstance05, DFMK02, DFMK05, and DFMK08), which are shown in Figure 8. From these six benchmarks, we find that the HDVMA is closer to the bottom of the three-dimensional graph and more evenly distributed than the other algorithms, which indicates that the comprehensive performance of the HDVMA is better than that of the other algorithms. This is consistent with our analysis in Tables 7–9, and further validates the effectiveness of the HDVMA.



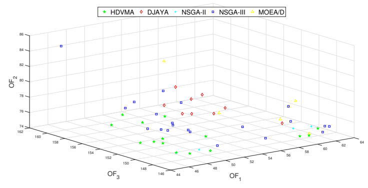
(**a**) DFInstance01



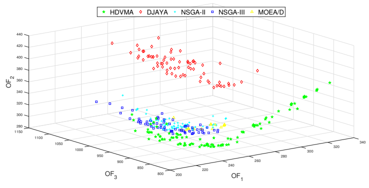
(**b**) DFInstance03



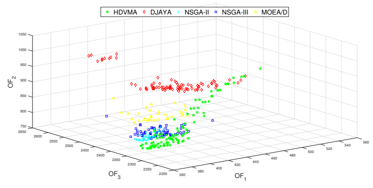
(**c**) DFInstance05



(**d**) DFMK02



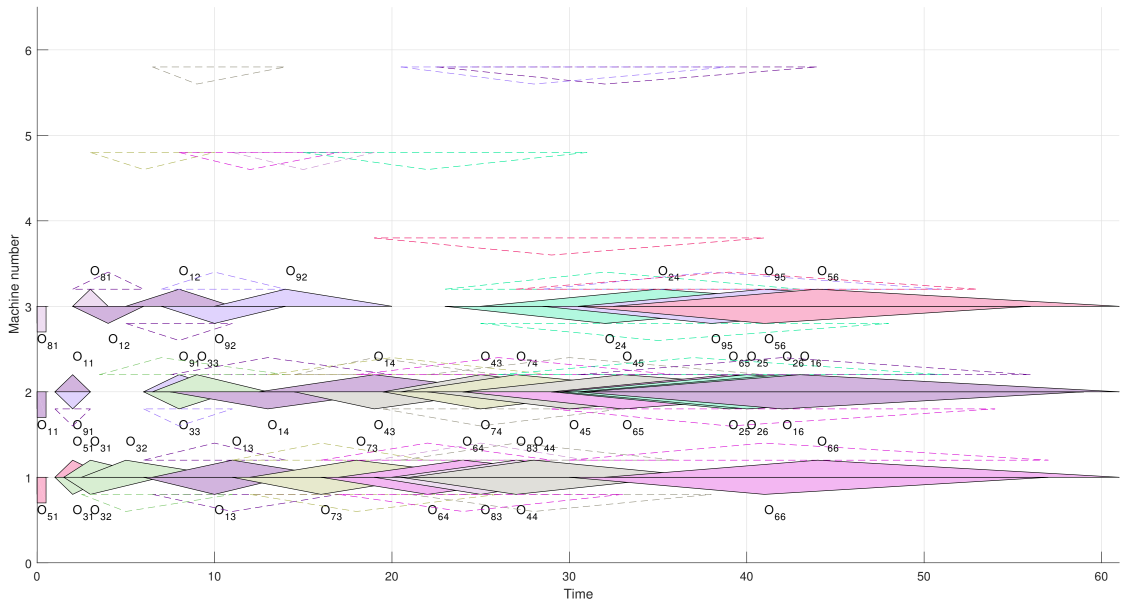
(**e**) DFMK05



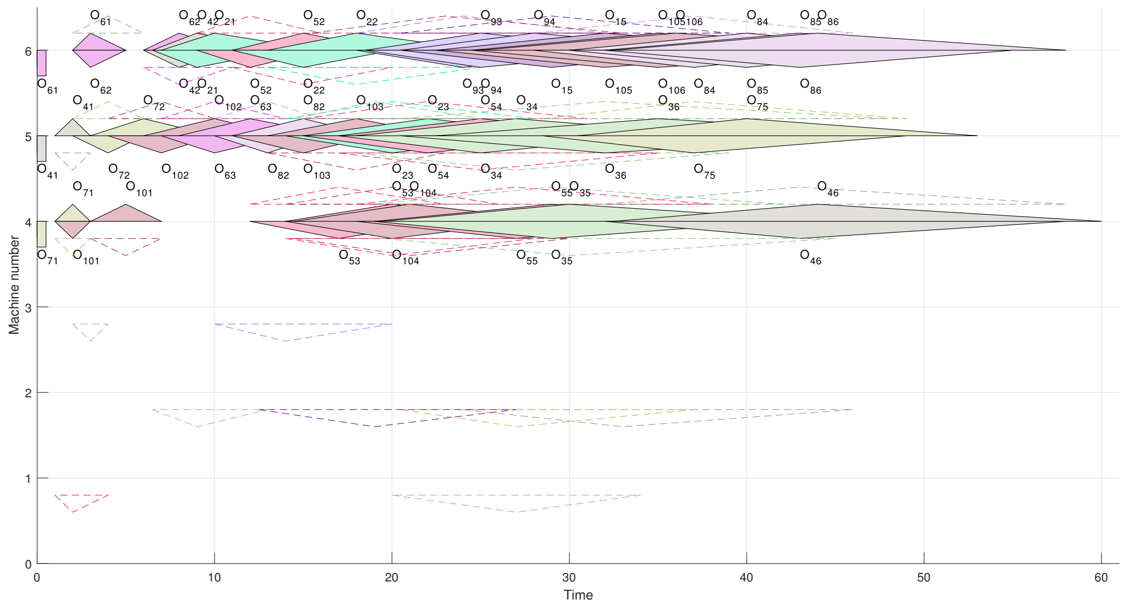
(**f**) DFMK08

Figure 8. PF chart of select benchmarks.

Figure 9 shows a two-workshop Gantt diagram example for DFMK02.The processing time and transfer time of different processes of the same workpiece have the same color. The workpiece is transferred from workshop 2 to workshop 1 and from workshop 1 to workshop 2 eight times to generate the final scheduling scheme. It can be seen that the distributed workshop considering workpiece transfer can make the scheduling more flexible, and the fuzziness of the processing time and transfer time is more in line with actual production situations.



(**a**) Workshop 1



(**b**) Workshop 2

Figure 9. DFMK02 Gantt chart.

7. Conclusions

In this paper, we construct a mathematical model of distributed flexible job shop scheduling problem with fuzzy transfer time and fuzzy processing time to better fit real-world production scheduling. The objectives are to minimize the fuzzy maximum completion time, fuzzy maximum factory load, and fuzzy total workload. Workpieces can be transferred between factories, and the factories work together to produce the final scheduling scheme. The fuzzy treatment of the transfer time and processing time makes it easier to express uncertainty in the real world.

To solve the MO-DFFJSPT, we meticulously design the encoding and decoding methods of this model. An efficient initialization heuristic that combines four different rules is designed to generate an initial population to produce high-quality initial individuals. Then, based on the algorithm framework of NSGA-II, crossover and mutation operators are designed to search the solution space and maintain the diversity of the population. We introduce a weight decomposition strategy and design a VNS method to further accelerate the convergence performance and ensure the uniform distribution of the solution set. Finally, we compare the HDVMA with four other excellent multiobjective algorithms, and the results show that the HDVMA is superior to the other algorithms in terms of convergence performance and the uniformity of the solution set distribution.

This work proposes a new scheduling problem model which is meaningful for real-world distributed production applications. At the same time, the proposed HDVMA takes into account the convergence and diversity, and can be used by other researchers to test the performance of the algorithm. In future work, we will further study complex situations of distributed production workshops, such as considering machine breakdown and new workpiece insertion. Additionally, one drawback of our proposed algorithm is too high a number of parameters, meaning that improper parameter settings that have a bad impact on the algorithm are too likely. Future research could use adaptive strategies or feedback mechanisms to dynamically adjust parameters, solving the problem of excessive parameters in the algorithm.

Author Contributions: J.Y. designed and performed the experiments; J.Y. provided analysis software; J.Y. analyzed the data; J.Y. organized the data and wrote the paper. H.X. perfected the details of the paper. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Data Availability Statement: The code in this article cannot be published due to privacy, and can be obtained from the corresponding author upon reasonable request.

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

References

1. Xu, Y.; Sahnoun, M.; Abdelaziz, F.B.; Baudry, D. A simulated multi-objective model for flexible job shop transportation scheduling. *Ann. Oper. Res.* **2022**, *311*, 899–920. [CrossRef]
2. Zhang, Y.; Zhu, H.; Tang, D. An improved hybrid particle swarm optimization for multi-objective flexible job-shop scheduling problem. *Kybernetes* **2020**, *49*, 2873–2892. [CrossRef]
3. Ziaee, M.; Mortazavi, J.; Amra, M. Flexible job shop scheduling problem considering machine and order acceptance, transportation costs, and setup times. *Soft Comput.* **2022**, *26*, 3527–3543. [CrossRef]
4. Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **2002**, *6*, 182–197. [CrossRef]
5. Naderi, B.; Azab, A. An improved model and novel simulated annealing for distributed job shop problems. *Int. J. Adv. Manuf. Technol.* **2015**, *81*, 693–703. [CrossRef]
6. Chang, H.C.; Liu, T.K. Optimisation of distributed manufacturing flexible job shop scheduling by using hybrid genetic algorithms. *J. Intell. Manuf.* **2017**, *28*, 1973–1986. [CrossRef]
7. Wu, X.; Liu, X.; Zhao, N. An improved differential evolution algorithm for solving a distributed assembly flexible job shop scheduling problem. *Memetic Comput.* **2019**, *11*, 335–355. [CrossRef]
8. Tang, H.; Fang, B.; Liu, R.; Li, Y.; Guo, S. A hybrid teaching and learning-based optimization algorithm for distributed sand casting job-shop scheduling problem. *Appl. Soft Comput.* **2022**, *120*, 108694. [CrossRef]
9. Ziaee, M. A heuristic algorithm for the distributed and flexible job-shop scheduling problem. *J. Supercomput.* **2014**, *67*, 69–83. [CrossRef]
10. Luo, Q.; Deng, Q.; Gong, G.; Zhang, L.; Han, W.; Li, K. An efficient memetic algorithm for distributed flexible job shop scheduling problem with transfers. *Expert Syst. Appl.* **2020**, *160*, 113721. [CrossRef]
11. Sang, Y.; Tan, J. Intelligent factory many-objective distributed flexible job shop collaborative scheduling method. *Comput. Ind. Eng.* **2022**, *164*, 107884. [CrossRef]

12. Sun, L.; Lin, L.; Gen, M.; Li, H. A hybrid cooperative coevolution algorithm for fuzzy flexible job shop scheduling. *IEEE Trans. Fuzzy Syst.* **2019**, *27*, 1008–1022. [CrossRef]
13. Zheng, Y.l.; Li, Y.x.; Lei, D.m. Multi-objective swarm-based neighborhood search for fuzzy flexible job shop scheduling. *Int. J. Adv. Manuf. Technol.* **2012**, *60*, 1063–1069. [CrossRef]
14. Lin, J.; Zhu, L.; Wang, Z.J. A hybrid multi-verse optimization for the fuzzy flexible job-shop scheduling problem. *Comput. Ind. Eng.* **2019**, *127*, 1089–1100. [CrossRef]
15. Li, R.; Gong, W.; Lu, C. Self-adaptive multi-objective evolutionary algorithm for flexible job shop scheduling with fuzzy processing time. *Comput. Ind. Eng.* **2022**, *168*, 108099. [CrossRef]
16. Gao, K.Z.; Suganthan, P.N.; Pan, Q.K.; Tasgetiren, M.F. An effective discrete harmony search algorithm for flexible job shop scheduling problem with fuzzy processing time. *Int. J. Prod. Res.* **2015**, *53*, 5896–5911. [CrossRef]
17. Lin, J. A hybrid biogeography-based optimization for the fuzzy flexible job-shop scheduling problem. *Knowl.-Based Syst.* **2015**, *78*, 59–74. [CrossRef]
18. Gao, K.Z.; Suganthan, P.N.; Pan, Q.K.; Chua, T.J.; Chong, C.S.; Cai, T.X. An improved artificial bee colony algorithm for flexible job-shop scheduling problem with fuzzy processing time. *Expert Syst. Appl.* **2016**, *65*, 52–67. [CrossRef]
19. Vela, C.R.; Afsar, S.; Palacios, J.J.; Gonzalez-Rodriguez, I.; Puente, J. Evolutionary tabu search for flexible due-date satisfaction in fuzzy job shop scheduling. *Comput. Oper. Res.* **2020**, *119*, 104931. [CrossRef]
20. Sakawa, M.; Mori, T. An efficient genetic algorithm for job-shop scheduling problems with fuzzy processing time and fuzzy duedate. *Comput. Ind. Eng.* **1999**, *36*, 325–341. [CrossRef]
21. Lei, D. A genetic algorithm for flexible job shop scheduling with fuzzy processing time. *Int. J. Prod. Res.* **2010**, *48*, 2995–3013. [CrossRef]
22. Xu, W.; Hu, Y.; Luo, W.; Wang, L.; Wu, R. A multi-objective scheduling method for distributed and flexible job shop based on hybrid genetic algorithm and tabu search considering operation outsourcing and carbon emission. *Comput. Ind. Eng.* **2021**, *157*, 107318. [CrossRef]
23. Wang, C.; Tian, N.; Ji, Z.; Wang, Y. Multi-objective fuzzy flexible job shop scheduling using memetic algorithm. *J. Stat. Comput. Simul.* **2017**, *87*, 2828–2846. [CrossRef]
24. Gao, K.Z.; Suganthan, P.N.; Chua, T.J.; Chong, C.S.; Cai, T.X.; Pan, Q.K. A two-stage artificial bee colony algorithm scheduling flexible job-shop scheduling problem with new job insertion. *Expert Syst. Appl.* **2015**, *42*, 7652–7663. [CrossRef]
25. Yuan, Y.; Xu, H. Multiobjective flexible job shop scheduling using memetic algorithms. *IEEE Trans. Autom. Sci. Eng.* **2013**, *12*, 336–353. [CrossRef]
26. Wang, L.; Zhou, G.; Xu, Y.; Wang, S.; Liu, M. An effective artificial bee colony algorithm for the flexible job-shop scheduling problem. *Int. J. Adv. Manuf. Technol.* **2012**, *60*, 303–315. [CrossRef]
27. Lei, D. Co-evolutionary genetic algorithm for fuzzy flexible job shop scheduling. *Appl. Soft Comput.* **2012**, *12*, 2237–2245. [CrossRef]
28. Brandimarte, P. Routing and scheduling in a flexible job shop by tabu search. *Ann. Oper. Res.* **1993**, *41*, 157–183. [CrossRef]
29. Lei, D.; Guo, X. Swarm-based neighbourhood search algorithm for fuzzy flexible job shop scheduling. *Int. J. Prod. Res.* **2012**, *50*, 1639–1649. [CrossRef]
30. 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]