


Article

A Dual-Population Genetic Algorithm with Q-Learning for Multi-Objective Distributed Hybrid Flow Shop Scheduling Problem

Jidong Zhang ¹ and Jingcao Cai ^{2,3,*} 

¹ School of Mechanical Engineering, Anhui Institute of Information Technology, Wuhu 241000, China; 2014030@aait.edu.cn

² School of Mechanical Engineering, Anhui Polytechnic University, Wuhu 241000, China

³ Anhui Key Laboratory of Detection Technology and Energy Saving Devices, Anhui Polytechnic University, Wuhu 241000, China

* Correspondence: caijingcao@foxmail.com

Abstract: In real-world production processes, the same enterprise often has multiple factories or one factory has multiple production lines, and multiple objectives need to be considered in the production process. A dual-population genetic algorithm with Q-learning is proposed to minimize the maximum completion time and the number of tardy jobs for distributed hybrid flow shop scheduling problems, which have some symmetries in machines. Multiple crossover and mutation operators are proposed, and only one search strategy combination, including one crossover operator and one mutation operator, is selected in each iteration. A population assessment method is provided to evaluate the evolutionary state of the population at the initial state and after each iteration. Two populations adopt different search strategies, in which the best search strategy is selected for the first population and the search strategy of the second population is selected under the guidance of Q-learning. Experimental results show that the dual-population genetic algorithm with Q-learning is competitive for solving multi-objective distributed hybrid flow shop scheduling problems.

Keywords: genetic algorithm; Q-learning; distributed scheduling; hybrid flow shop; multi-objective optimization



Citation: Zhang, J.; Cai, J. A Dual-Population Genetic Algorithm with Q-Learning for Multi-Objective Distributed Hybrid Flow Shop Scheduling Problem. *Symmetry* **2023**, *15*, 836. <https://doi.org/10.3390/sym15040836>

Academic Editors: Chong Wang and Zine El Abidine Fellah

Received: 9 March 2023

Revised: 27 March 2023

Accepted: 28 March 2023

Published: 30 March 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The hybrid flow shop scheduling problem (HFSP) is a complex combinatorial optimization problem that has been extensively studied due to its practical relevance in a wide range of manufacturing and production environments [1,2]. In HFSP, the production process involves multiple stages or workstations, and each stage may have multiple machines that can process different types of jobs. The problem is to determine the processing sequence of jobs and the allocation of machines to each stage to minimize a specific objective function [3].

The distributed shop scheduling problem (DSSP) is a complex optimization problem that has gained significant attention in the literature due to its relevance to a wide range of manufacturing and production environments [4–6]. In DSSP, there are multiple production shops, each with its set of machines and jobs to be scheduled. The objective is to determine an optimal schedule for all the jobs and machines in the shops, subject to various constraints, such as machine availability, processing times, and due dates. DSSP has been studied extensively in the literature, and various algorithms have been proposed to solve this problem. In recent years, there has been a growing interest in developing efficient and effective algorithms to address DSSP with additional constraints such as uncertain processing times [7], energy consumption [8], and the presence of multiple objectives [9].

The distributed hybrid flow shop scheduling problem (DHFSP) is an extension of HFSP, which has attracted significant research attention due to its importance in production

environments [10–34]. The objective of DHFSP is to minimize the maximum completion time for all production processes to enhance production efficiency. Several studies have been conducted to address this problem using different approaches. Reference [26] proposed a mixed-integer linear programming formulation and a self-tuning iterated greedy (SIG) algorithm with an adaptive cocktail decoding mechanism to solve DHFSP with multiprocessor tasks. Reference [18] proposed a dynamic shuffled frog-leaping algorithm to solve the same problem and provided a lower bound. References [32–34] developed a hybrid brain storm optimization algorithm. References [10,16,22] designed many kinds of artificial bee colony algorithms. Reference [11] proposed a bi-population cooperative memetic algorithm. Reference [30] presented a novel shuffled frog-leaping algorithm with reinforcement learning for DHFSP with assembly. Reference [29] formulated three novel mixed-integer linear programming models and a constraint programming model for DHFSP with sequence-dependent setup times. Reference [24] was the only one who considered optimizing the minimization of the sum of earliness, tardiness, and delivery costs in the single-objective optimization study of DHFSP. They proposed an adaptive human-learning-based genetic algorithm. In summary, DHFSP is a significant problem in production environments and several studies have been conducted to address it using different optimization techniques. These include mixed-integer linear programming, shuffled frog-leaping algorithm, hybrid brain storm optimization algorithm, artificial bee colony algorithm, and bi-population cooperative memetic algorithm, among others. The proposed algorithms have shown promising results, but further research is needed to develop more effective and efficient algorithms for more complex DHFSP variants.

2. Logical Scheme to DHFSP

Recent research has explored the use of Q-learning, a type of reinforcement learning, in combination with intelligent optimization algorithms to improve algorithm performance [35,36]. Reference [37] demonstrated the use of Q-learning to adjust key parameters of a genetic algorithm, while Reference [38] used Q-learning to select a search operator dynamically. Reference [30] embedded Q-learning in a memplex search strategy to select a search strategy dynamically. However, current research has only applied this approach to solve single-objective shop scheduling problems. To apply Q-learning and intelligent optimization algorithms to solve multi-objective optimization problems, further exploration is needed to design appropriate actions and evaluate populations. Therefore, there is a need for future research to investigate the combination of Q-learning and intelligent optimization algorithms for multi-objective optimization problems. Such research could contribute to developing effective algorithms for complex optimization problems in diverse fields. Results from such studies would be of interest to researchers and practitioners working in the area of intelligent optimization.

Optimization problems, such as the distributed hybrid flow shop scheduling problem, are often solved using intelligent optimization algorithms. Among these, genetic algorithms have been widely applied due to their effectiveness in solving complex optimization problems. Recently, the combination of intelligent optimization algorithms with reinforcement learning algorithms, such as Q-learning, has received attention as a means to improve algorithm performance. In this paper, we propose a dual-population genetic algorithm with Q-learning to minimize the maximum completion time and the number of tardy jobs for the distributed hybrid flow shop scheduling problem. The algorithm combines genetic algorithm (GA) with Q-learning to guide the selection of crossover and mutation operators. Multiple crossover and mutation operators are proposed, and only one search strategy combination is selected in each iteration. To evaluate the evolutionary state of the population, we provide a population assessment method at the initial state and after each iteration. Two populations adopt different search strategies, in which the best search strategy is selected for one population and the search strategy of the other population is selected under the guidance of Q-learning. The approach proposed in this paper could be applied to other optimization problems and could be of interest to researchers and

practitioners working in the field of intelligent optimization. Figure 1 gives the logical scheme to DHFSP.

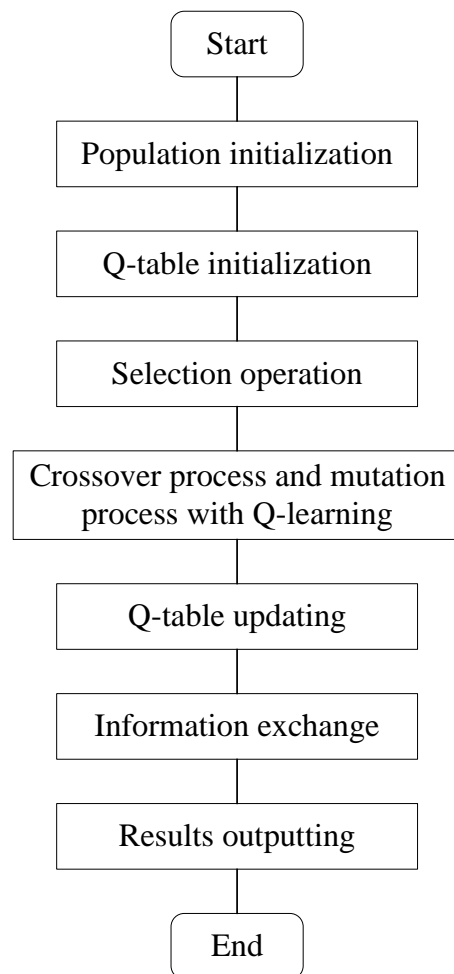


Figure 1. The logical scheme to DHFSP.

The remainder of the paper is organized as follows. The distributed hybrid flow shop scheduling problem is described in Section 2, and an introduction to GA and Q-learning is provided in Section 3. Section 4 presents a dual-population genetic algorithm with Q-learning and numerical experiments on the performance of the algorithm are reported in Section 5. In conclusion, our approach demonstrates promising results for solving the distributed hybrid flow shop scheduling problem.

3. Description of Distributed Hybrid Flow Shop Scheduling Problem

DHFSP is described as follows. There are n jobs, J_1, J_2, \dots, J_n , that need to be processed. There are F factories, each of which is a hybrid flow shop. In the process of processing, each job has S processes. Each job in different processes needs to be processed on a different machine and the performance and function of these machines are identical. There are m_{fs} identical parallel machines, $M_{fs1}, M_{fs2}, \dots, M_{fsl}$, at stage s in factory f . The processing time of J_i on M_{fsl} is p_{is} . d_i is the due date of J_i . The machine distribution in different factories has symmetry.

The goal of DHFSP is to minimize the maximum completion time and the number of tardy jobs simultaneously through reasonable scheduling.

$$C_{\max} = \max_{i \in \{1,2,\dots,n\}} \{C_i\} \tag{1}$$

$$\bar{U} = \sum_{i=1}^n W_i \tag{2}$$

where C_{\max} is the maximum completion time of all jobs, C_i is the completion time of J_i , \bar{U} is the number of tardy jobs, W_i represents whether J_i is delivered on time, $W_i = 1$, if $C_i > d_i$, and $W_i = 0$, if $C_i \leq d_i$.

DHFSP has three sub-problems, factory assignment, machine assignment, and processing sequence assignment. Obviously, an acceptable solution can be obtained only when all three of these assignments are reasonable.

Table 1 describes an illustrative example of DHFSP with 20 jobs, 2 stages and 2 factories. For example, the processing time of J_1 in the first stage is 39. There are four parallel machines in the first stage and five parallel machines in the second stage. Figure 2 shows a schedule of the example which is described in Table 1. As can be seen from Figure 2, job J_1 is processed in machine $M_{2,1,2}$, which is the second machine in the first stage of factory 1, and the processing time is $p_{11} = 39$.

Table 1. An illustrative example.

Job/i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
p_{i1}	39	81	88	43	61	36	48	100	45	61	30	98	85	99	62	89	39	69	61	98
p_{i2}	53	41	81	41	85	98	47	45	84	42	92	75	69	74	52	86	38	80	45	92

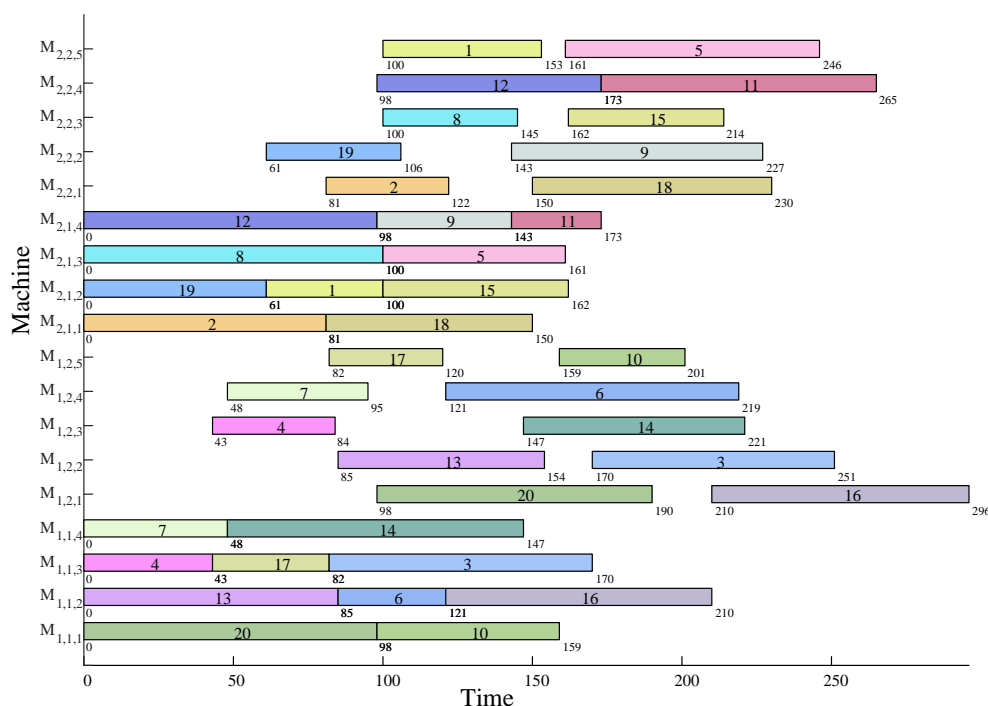


Figure 2. Gantt chart of the case.

4. Introduction to GA and Q-Learning

The main steps of GA are as follows: initialization, evaluation, selection, crossover, and mutation. (1) Initialization: the initial population contains many candidate solutions, usually generated using random numbers. (2) Evaluation: evaluate the fitness function

value for each individual. (3) Selection: select individuals based on their fitness function values to be preserved in later evolution. (4) Crossover: perform crossover between two selected individuals to generate new offspring. (5) Mutation: randomly select some individuals and mutate them to increase diversity in the population. (6) Repeat steps (2)–(5) until a stopping condition is met. (7) Output the final best solution.

The main steps of Q-learning are as follows: (1) Initialize the Q-table: First, a table that stores Q values (Q-table) needs to be created, with each element representing the Q value corresponding to the current state and action. (2) Select an action: Under the current state, select an action. The action can be selected randomly or based on the current Q value (for example, selecting the action with the highest current Q value). (3) Perform the action: Perform the selected action, thereby updating the state of the environment. (4) Calculate the reward: based on the new state, calculate the reward obtained. (5) Update the Q value: Use the Q-learning formula to update the Q value corresponding to the current state and action. (6) Repeat steps (2)–(5) until the environment reaches the terminal state or reaches the maximum number of steps set. (7) Output the result: Output the learned optimal strategy, that is, the action corresponding to the maximum Q value in the Q-table. Table 2 gives an example of a Q-table, and the Q-table will be updated as the algorithms runs.

Table 2. An example of a Q-table.

	Action 1	Action 2	Action 3	Action 4
State 1	0.029	0.783	0.069	0.684
State 2	0.320	0.381	0.577	0.732
State 3	0.548	0.486	0.006	0.316
State 4	0.636	0.416	0.615	0.630
State 5	0.451	0.005	0.735	0.469
State 6	0.429	0.639	0.647	0.512
State 7	0.742	0.209	0.386	0.442
State 8	0.583	0.691	0.347	0.209
State 9	0.674	0.181	0.164	0.346
State 10	0.104	0.811	0.135	0.224

5. A Dual-Population Genetic Algorithm with Q-Learning

5.1. Coding and Decoding

DHFSP encompasses three sub-problems: factory assignment, machine assignment, and processing sequence assignment. A reasonable coding method is the foundation for solving this problem using intelligent optimization algorithms. A double-string coding method is adopted to encode the solution of the problem, where the encoding string contains machine assignment and processing sequence assignment.

An encoding of DEHFSP is represented as a factory string $[\alpha_1, \alpha_2, \dots, \alpha_n]$ and a sequence string $[\beta_1, \beta_2, \dots, \beta_n]$, and $\alpha_i \in \{1, 2, \dots, F\}$, $\beta_i \in \{1, 2, \dots, n\}$, $\beta_i \neq \beta_j, \forall i, j, i \neq j$. The decoding procedure is described as follows: Assign all jobs to factories according to the factory string, where J_i is assigned to factory α_i . In each factory, the order in which all jobs are processed is determined by a sequence string. If two jobs, J_{β_i} and J_{β_j} , are such that $i < j$, then J_{β_i} is given priority and assigned to the machines. When selecting a machine for a job, the set of machines that can be selected is first determined and then the machine is chosen that will minimize the completion time of the job.

To further explain the decoding process, a solution is provided in Section 2. This solution consists of a factory string [2,2,1,1,2,1,1,2,2,1,2,2,1,1,2,1,1,2,2,1] and a sequence string [2,19,8,20,13,4,12,7,17,1,14,18,3,9,15,6,10,5,11,16]. Based on the factory string, jobs $J_3, J_4, J_6, J_7, J_{10}, J_{13}, J_{14}, J_{16}, J_{17}$, and J_{20} are assigned to factory 1, and their process sequence is determined by the sequence string to be $J_{20}, J_{13}, J_4, J_7, J_{17}, J_{14}, J_3, J_6, J_{10}$, and J_{16} . The scheduling scheme can be obtained and the Gantt chart is shown in Figure 2. The final value of C_{max} is 296 and the number of tardy jobs is 13.

5.2. Crossover Operator

The crossover operator is a way of generating new solutions in genetic algorithms. Its purpose is to combine the genetic information of two parents to produce a new individual, with the new solution containing some features of both parents.

A crossover operator $CO(x, y)$ is designed for DHFSP, where x and y represent two parents. The process of $CO(x, y)$ is as follows: (1) randomly select a set A of jobs; (2) swap the values of the elements corresponding to jobs in A in x with those in y for the factory string; (3) rearrange the order of jobs in x belonging to A to match the order of these jobs in y for the sequence string. Figure 3 gives the process of global search and it can be observed that two offspring are significantly different from the previous parents, indicating a large variation in the search process during global search.

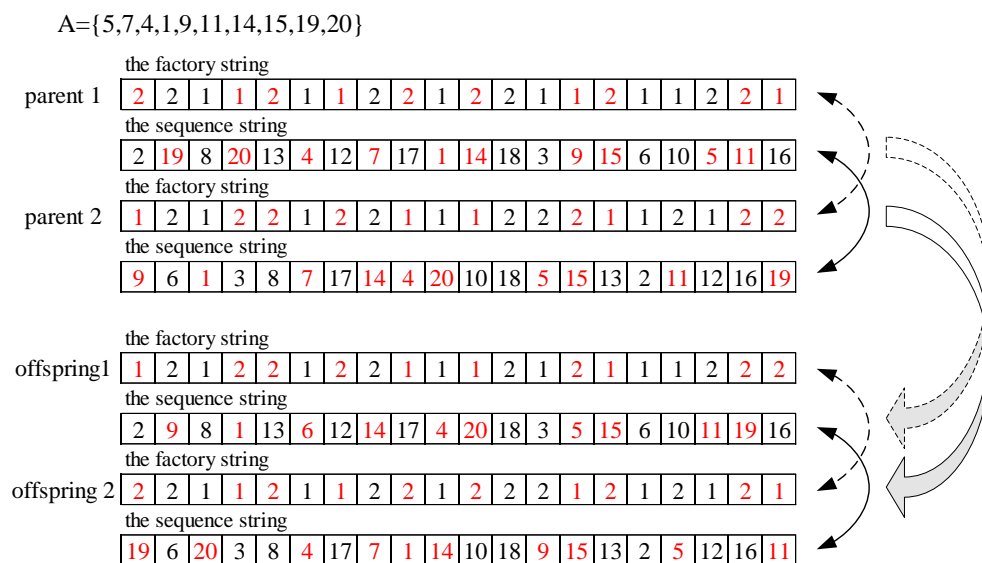


Figure 3. The process of global search.

5.3. Mutation Operator

Mutation is a common operator used in evolutionary algorithms to generate new individuals in a population, increasing the diversity of the algorithm. Mutation is typically performed by introducing random changes to the gene values of an individual at some gene locus, either by randomizing or by applying some transformation rule. Mutation is usually applied to an individual in the current population to generate a new individual with slight variations. Unlike crossover, mutation does not require the pairing of individuals and can explore the search space independently. Mutation is often used in combination with crossover to increase the diversity of the population and facilitate better exploration of the search space.

Two mutation operators, MO1 based on insertion and MO2 based on exchange, are designed to generate new solutions by changing the factory string and the sequence string. In MO1, randomly select jobs J_i and J_j such that $i < j$, and let $\pi_{pos_1} = i$ and $\pi_{pos_2} = j$. Then, insert $\pi_{pos_2} = j$ into the position of π_{pos_1} in the sequence string and let $\theta_j = \theta_i$. In MO2, randomly choose jobs J_i and J_j such that $i < j$, and let $\pi_{pos_1} = i$ and $\pi_{pos_2} = j$. Then, swap the values of π_{pos_1} and π_{pos_2} in the sequence string and swap the values of θ_j and θ_i in the factory string. The process of generating new solutions in MO1 and MO2 are illustrated in Figures 4 and 5.

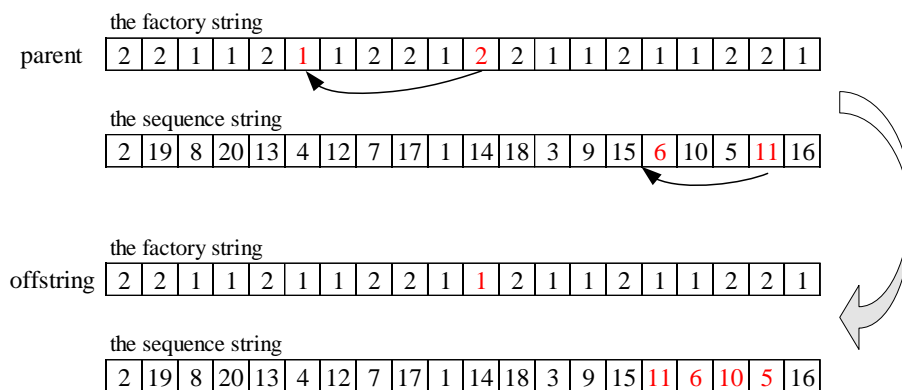


Figure 4. The process of local search MO1.

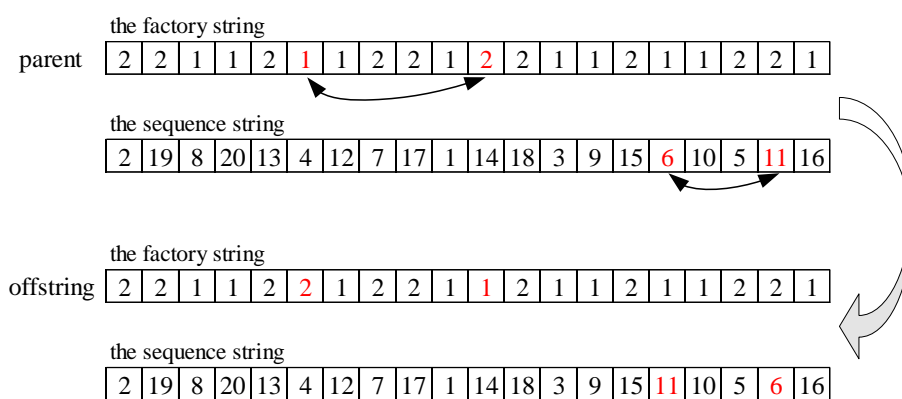


Figure 5. The process of local search MO2.

5.4. Q-Learning Process

The Q-learning algorithm consists of a state set s_t , an action set a_t , a reward function, and an action selection strategy. The environment state is determined based on population evaluation, while actions are represented by various search strategies. A novel reward function is designed and the action selection strategy adopts the ϵ -greedy strategy.

A new method is provided for evaluating the state of the population. The evaluation value of the population in generation t is calculated as

$$E_t = \frac{\bar{N}_t}{N} \tag{3}$$

where \bar{N}_t represents the number of solutions in the t -th generation population that are not dominated by any solutions in the $(t-1)$ -th generation population and E_t represents the evaluation value of the population in generation t .

The evaluation value E_t of t -th generation is bounded by $[E^1, E^2]$ with $E^1 = 0$ and $E^2 = 2$. The state set $S = 1, 2, \dots, 10$ is defined by dividing the interval $[E^1, E^2]$ into 10 equal parts, where the state value $s_t = k$ if $E_t \in [0.2 \times (k - 1), 0.2 \times k]$ for $1 \leq k \leq 10$, and $a_t = 10$ if $E_t \geq 2$.

Two variants of CO have been created, namely CO_1 and CO_2 . Each of these variants modifies only one string, with CO_1 changing the factory string and CO_2 altering the sequence string.

The set of actions, denoted by A , has been constructed by combining various crossover operators and mutation operators. Specifically, there are two crossover operators and two mutation operators, resulting in a total of four possible combinations. Table 3 illustrates the corresponding relationships between these combinations and the resulting actions in the set A .

Table 3. The corresponding relationship between action and search method.

Action	Search Method
1	CO ₁ + MO ₁
2	CO ₁ + MO ₂
3	CO ₂ + MO ₁
4	CO ₂ + MO ₂

Since the current metric s_t is relatively small, indicating that the population is closer to the set of Pareto front in the solution space, we define r_{t+1} as the difference between s_{t+1} and s_t . If we take action a_t in state s_t to improve the population's state, we will receive a positive gain. Conversely, we will be penalized.

To select an action, the ϵ -greed method is employed. Typically, all entries in the initial Q-table are initialized to 0.

5.5. Dual-Population Genetic Algorithm with Q-Learning

QDGA is an algorithm that combines GA and Q-learning, and Q-learning is used in the process of crossover process and mutation process to choose a suitable crossover operator and mutation operator. The main steps of QDGA include population initialization and Q-table initialization, selection operation, crossover process and mutation process with Q-learning, Q-table updating, the information exchange of two populations, and outputting results.

In the population initialization, the two initial populations are randomly generated. The initial Q-table is initialized to 0. The two populations perform crossover, mutation, and selection operations independently. The two solutions are chosen according to the roulette wheel, which belongs to the parent population, then crossover operator and mutation operator corresponding to the action are selected according to the Q-table. The Q-table is updated according to the performance of action in the state. Then non-dominant sorting is performed on all individuals in both populations and a portion of individuals from both populations are randomly exchanged, accounting for 50% of the population. The result is output when the termination condition is satisfied.

6. Computation Experiments

All experiments are programmed using Visual Studio 2022 C++ and run on a computer with 16.0G RAM 12th Gen Intel(R) Core(TM) i7-12700H 2.70GHz.

6.1. Instances

To evaluate the effectiveness of the algorithm proposed in this study, 140 instances have been made available for download. These instances comprise varying numbers of jobs, factories, and stages, and can be obtained from <https://gitee.com/caijingcao/DHFSP002> (30 March 2022). Each instance is represented by the notation $n \times F \times S$, where n denotes the number of jobs, F denotes the number of factories, and S denotes the number of stages.

6.2. Computational Metrics

Metric ρ is the proportion of solutions that an algorithm A can provide for the reference solution set Ω^* . ρ is calculated as

$$\rho = \frac{|\{x \in \Omega_A \mid x \in \Omega^*\}|}{|\Omega^*|} \quad (4)$$

where Ω_A is the set of non-dominated solutions obtained by algorithm A , Ω^* represents the reference solution set, and $|\Omega^*|$ indicates the number of solutions in Ω^* .

Metric \mathcal{C} is applied to compare the approximate Pareto optimal set obtained by algorithms. $\mathcal{C}(L, B)$ measures the fraction of members of B that are dominated by members of L .

$$\mathcal{C}(L, B) = \frac{|\{b \in B : \exists h \in L, h \succ b\}|}{|B|} \quad (5)$$

where $x \succ y$ indicates that x dominates y and $|B|$ indicates the number of solutions in B . The Pareto optimal solution refers to a set of solutions in multi-objective optimization problems, in which improving one objective further would result in a degradation of at least one other objective.

Metric IGD (inverted generational distance) is a comprehensive performance indicator to evaluate algorithms. The smaller the value of IGD , the better the algorithm's overall performance of A . IGD is calculated as

$$IGD(\Omega_A, \Omega^*) = \frac{1}{|\Omega^*|} \sum_{x \in \Omega^*} \min_{y \in \Omega_A} d(x, y) \quad (6)$$

where $d(x, y)$ is the Euclidean distance between solution x and y by normalized objectives.

6.3. Comparison Algorithms

To assess the effectiveness of QDGA, two widely recognized multi-objective evolutionary algorithms were selected for comparison: non-dominated sorting genetic algorithm-II (NSGA-II [39]) and strength Pareto evolutionary algorithm2 (SPEA2 [40]). These algorithms were chosen based on their reputation and their demonstrated ability to effectively solve complex multi-objective optimization problems. By comparing the performance of QDGA against these established algorithms, we can gain a better understanding of the strengths and weaknesses of QDGA in tackling multi-objective optimization challenges. In order to assess the impact of Q-learning on the algorithm, a variant of the QDGA algorithm was implemented in which the crossover and mutation operators were randomly selected. This variant is referred to as GA. By introducing this variant, we aim to determine whether the incorporation of Q-learning leads to improvements in the algorithm's performance. The GA variant serves as a control group against which the performance of the original QDGA algorithm can be compared and the results can be used to evaluate the effectiveness of Q-learning in the algorithm.

6.4. Parameter Settings

QDGA relies on $N, P_c, P_m, \alpha, \gamma, \epsilon$ and stopping condition. Although longer algorithm runs could potentially result in better outcomes, experiments have demonstrated that both QSFLA and its comparison algorithm tend to converge or experience only minor improvements after running for $0.1 \times n \times S$ seconds. To ensure a fair comparison, we chose $0.1 \times n \times S$ seconds as the stopping condition for all algorithms, which is consistent with similar studies [30].

The Taguchi method [31] is a powerful statistical approach to optimize the performance of a product or process by identifying the best combination of controllable factors or parameters. This method has been widely used in various fields including engineering, manufacturing, and healthcare. The Taguchi method was utilized to determine the optimal settings for the other parameters, with several instances featuring varying numbers of jobs, factories, or stages used in parameter experiments.

By conducting Taguchi experiments on examples of different scales, a set of optimal parameters can be obtained that is best for all examples. Based on the results, it can be concluded that, among different combinations of parameters for QDGA, the setting with $N = 100, P_c = 0.8, P_m = 0.05, \alpha = 0.1, \gamma = 0.9$, and $\epsilon = 0.2$ achieves the best performance. Therefore, we adopt these settings for QDGA in our further experiments. All parameters of NSGAI, SPEA2, and GA are determined by the same way.

6.5. Results and Analyses

The computational results of four algorithms, which were run 10 times for each instance, were reported. Tables 4–6 display the performance metrics of the algorithms. The reference set Ω^* was created by aggregating the non-dominated solutions obtained by the four algorithms.

Table 4. Computational results of four algorithms on metric ρ .

Instance	QDGA	GA	NSGAI	SPEA2	Instance	QDGA	GA	NSGAI	SPEA2
1	0.833	0.000	0.167	0.000	43	0.800	0.000	0.200	0.000
2	0.667	0.000	0.333	0.000	44	0.400	0.000	0.200	0.400
3	0.500	0.000	0.500	0.500	45	1.000	0.000	0.000	0.000
4	0.000	0.500	0.500	0.000	46	1.000	0.000	0.000	0.000
5	0.667	0.667	0.333	0.000	47	1.000	0.000	0.000	0.000
6	0.400	0.400	0.200	0.000	48	0.333	0.667	0.000	0.000
7	1.000	1.000	1.000	1.000	49	0.333	0.000	0.333	0.667
8	0.667	0.333	0.000	0.000	50	0.333	0.000	0.000	0.667
9	1.000	1.000	1.000	1.000	51	0.667	0.000	0.333	0.000
10	1.000	0.000	0.000	0.000	52	0.250	0.000	0.750	0.000
11	1.000	0.000	0.000	0.000	53	0.500	0.000	0.500	0.000
12	0.800	0.200	0.200	0.000	54	1.000	0.000	0.000	0.000
13	1.000	0.000	0.000	0.000	55	0.000	0.000	1.000	0.000
14	1.000	0.000	0.000	0.000	56	0.500	0.000	0.000	0.500
15	0.500	0.000	0.000	0.500	57	0.667	0.333	0.000	0.000
16	0.500	0.250	0.250	0.000	58	1.000	0.000	0.000	0.000
17	0.667	0.333	0.000	0.000	59	0.667	0.000	0.000	0.333
18	0.000	0.333	0.333	0.333	60	1.000	0.000	0.000	0.000
19	0.000	0.000	1.000	0.000	61	0.000	0.000	0.000	1.000
20	0.000	0.333	0.000	0.667	62	0.000	0.000	0.500	0.500
21	0.167	0.333	0.500	0.000	63	0.250	0.250	0.000	0.500
22	0.429	0.429	0.000	0.143	64	0.000	0.000	0.000	1.000
23	0.250	0.750	0.000	0.250	65	0.333	0.333	0.333	0.000
24	1.000	0.000	0.000	0.000	66	0.000	0.000	1.000	0.000
25	0.500	0.500	0.000	0.000	67	0.000	1.000	0.000	0.000
26	1.000	0.000	0.000	0.000	68	0.000	0.000	0.500	0.500
27	0.500	0.000	0.000	0.500	69	0.000	0.000	0.200	0.800
28	0.000	0.000	0.000	1.000	70	0.000	0.250	0.750	0.000
29	1.000	0.000	0.000	0.000	71	0.667	0.000	0.000	0.667
30	0.500	0.500	0.000	0.000	72	0.667	0.000	0.000	0.333
31	0.333	0.333	0.333	0.000	73	0.667	0.000	0.000	0.333
32	1.000	0.000	0.000	0.000	74	0.500	0.500	0.000	0.000
33	0.000	0.333	0.333	0.333	75	0.667	0.000	0.000	0.333
34	1.000	0.000	0.000	0.000	76	0.333	0.000	0.333	0.333
35	0.667	0.000	0.333	0.000	77	0.500	0.000	0.000	0.500
36	0.000	0.333	0.667	0.000	78	0.000	0.333	0.000	0.667
37	0.500	0.500	0.000	0.000	79	1.000	0.000	0.000	0.000
38	0.000	1.000	0.000	0.000	80	0.333	0.167	0.500	0.000
39	1.000	0.000	0.000	0.000	81	0.000	0.500	0.500	0.000
40	1.000	0.000	0.000	0.000	82	1.000	0.000	0.000	0.000
41	0.500	0.000	0.500	0.000	83	0.667	0.333	0.000	0.000
42	0.500	0.500	0.250	0.000	84	1.000	0.000	0.000	0.000

The computational results presented in Table 4 indicate that QDGA outperforms GA, NSGAI, and SPEA2 in most instances in terms of the metric ρ . Specifically, the metric ρ_I of QDGA is higher than that of its comparative algorithms in 45 instances and it equals 1 in 22 instances. This implies that all members of the reference set Ω^* are generated by CVS. Moreover, the statistical results displayed in Figure 6 support the advantage of QDGA over the other algorithms. These findings suggest that QDGA is a more effective method for DHFSP.

Table 5 shows the computational results of four algorithms on metric \mathcal{C} where ‘Ins’, ‘Q’, ‘G’, ‘N’, and ‘S’ represent ‘Instance’, ‘QDGA’, ‘GA’, ‘NSGAI’, and ‘SPEA2’, respectively. The results presented in Table 5 demonstrate that, in 41 instances, the non-dominated solutions are not dominated by any solutions of GA, as $\mathcal{C}(G, Q)$ equals 0. $\mathcal{C}(Q, N)$ is smaller than $\mathcal{C}(N, Q)$ in 58 instances and $\mathcal{C}(Q, N)$ equals 1 in 38 instances indicating that all solutions

of NSGAI are dominated by the non-dominated solutions of QDGA. Compared with SPEA2, QDGA also has obvious advantages in terms of metric \mathcal{C} . The statistical results presented in Figure 7 illustrate the differences in \mathcal{C} among QDGA and its three comparative algorithms. It can be concluded that QDGA can generate better results compared to the comparative algorithms.

Table 5. Computational results of four algorithms on metric \mathcal{C} .

Ins	$\mathcal{C}(Q,G)$	$\mathcal{C}(G,Q)$	$\mathcal{C}(Q,N)$	$\mathcal{C}(N,Q)$	$\mathcal{C}(Q,S)$	$\mathcal{C}(S,Q)$	Ins	$\mathcal{C}(Q,G)$	$\mathcal{C}(G,Q)$	$\mathcal{C}(Q,N)$	$\mathcal{C}(N,Q)$	$\mathcal{C}(Q,S)$	$\mathcal{C}(S,Q)$
1	1.000	0.000	0.857	0.000	0.833	0.000	43	1.000	0.000	0.667	0.000	1.000	0.000
2	1.000	0.000	0.750	0.333	1.000	0.000	44	1.000	0.000	0.750	0.000	0.500	0.000
3	1.000	0.000	0.333	0.333	0.333	0.333	45	1.000	0.000	1.000	0.000	1.000	0.000
4	0.667	0.000	0.000	1.000	1.000	0.000	46	1.000	0.000	1.000	0.000	1.000	0.000
5	0.333	0.333	0.667	0.000	1.000	0.000	47	1.000	0.000	1.000	0.000	1.000	0.000
6	0.600	0.500	0.667	0.500	0.833	0.000	48	0.333	0.750	0.250	0.500	1.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	49	1.000	0.000	0.667	0.000	0.000	0.500
8	0.500	0.333	1.000	0.000	1.000	0.000	50	0.000	0.600	1.000	0.000	0.000	0.800
9	0.000	0.000	0.000	0.000	0.000	0.000	51	1.000	0.000	0.500	0.333	1.000	0.000
10	1.000	0.000	1.000	0.000	1.000	0.000	52	0.500	0.333	0.000	0.667	1.000	0.000
11	1.000	0.000	1.000	0.000	1.000	0.000	53	1.000	0.000	0.667	0.500	0.500	0.500
12	0.750	0.000	0.750	0.000	1.000	0.000	54	1.000	0.000	1.000	0.000	1.000	0.000
13	1.000	0.000	1.000	0.000	1.000	0.000	55	0.000	0.333	0.000	1.000	0.000	1.000
14	1.000	0.000	1.000	0.000	1.000	0.000	56	1.000	0.000	1.000	0.000	0.000	0.500
15	1.000	0.000	0.750	0.200	0.000	0.600	57	0.500	0.333	1.000	0.000	1.000	0.000
16	0.500	0.333	0.667	0.000	1.000	0.000	58	1.000	0.000	1.000	0.000	1.000	0.000
17	0.667	0.333	0.667	0.333	1.000	0.000	59	1.000	0.000	1.000	0.000	0.750	0.333
18	0.500	0.500	0.667	0.000	0.500	0.500	60	1.000	0.000	1.000	0.000	1.000	0.000
19	1.000	0.000	0.000	1.000	1.000	0.000	61	0.000	1.000	0.000	1.000	0.000	1.000
20	0.667	0.333	1.000	0.000	0.000	1.000	62	0.500	0.000	0.000	1.000	0.000	1.000
21	0.400	0.333	0.250	0.667	1.000	0.000	63	0.667	0.000	0.400	0.333	0.000	0.667
22	0.500	0.571	1.000	0.000	0.750	0.000	64	0.333	0.667	0.000	1.000	0.000	1.000
23	0.000	0.750	1.000	0.000	0.000	0.500	65	0.750	0.000	0.667	0.750	1.000	0.000
24	1.000	0.000	1.000	0.000	1.000	0.000	66	0.500	0.500	0.000	1.000	0.667	0.000
25	0.000	0.500	1.000	0.000	0.667	0.000	67	0.000	1.000	0.500	0.500	1.000	0.000
26	1.000	0.000	1.000	0.000	1.000	0.000	68	1.000	0.000	0.000	0.333	0.000	1.000
27	0.500	0.200	0.667	0.200	0.333	0.600	69	0.000	1.000	0.000	1.000	0.000	1.000
28	1.000	0.000	1.000	0.000	0.000	1.000	70	0.333	0.500	0.000	1.000	1.000	0.000
29	1.000	0.000	1.000	0.000	1.000	0.000	71	1.000	0.000	1.000	0.000	0.333	0.333
30	0.000	0.750	1.000	0.000	0.250	0.625	72	1.000	0.000	1.000	0.000	0.500	0.500
31	0.500	0.000	0.500	0.500	0.750	0.000	73	1.000	0.000	1.000	0.000	0.667	0.333
32	1.000	0.000	1.000	0.000	1.000	0.000	74	0.667	0.667	1.000	0.000	0.750	0.667
33	0.500	0.250	0.000	0.750	0.000	0.750	75	1.000	0.000	0.500	0.250	0.500	0.500
34	1.000	0.000	1.000	0.000	1.000	0.000	76	1.000	0.000	0.667	0.000	0.750	0.000
35	1.000	0.000	0.667	0.333	1.000	0.000	77	0.000	0.200	1.000	0.000	0.000	0.600
36	0.000	1.000	0.000	1.000	0.667	0.333	78	0.000	1.000	0.000	1.000	0.000	1.000
37	0.000	0.500	1.000	0.000	1.000	0.000	79	1.000	0.000	1.000	0.000	1.000	0.000
38	0.000	1.000	0.000	0.500	0.667	0.250	80	0.333	0.333	0.000	0.667	1.000	0.000
39	1.000	0.000	1.000	0.000	1.000	0.000	81	0.000	1.000	0.000	1.000	0.000	0.667
40	1.000	0.000	1.000	0.000	1.000	0.000	82	1.000	0.000	1.000	0.000	1.000	0.000
41	1.000	0.000	0.000	0.667	1.000	0.000	83	0.500	0.000	1.000	0.000	1.000	0.000
42	0.333	0.000	0.500	0.000	0.667	0.000	84	1.000	0.000	1.000	0.000	1.000	0.000

Based on the comprehensive analysis presented in Table 6, it is discernible that QDGA surpasses GA, NSGA-II, and SPEA2 with regard to convergence in the majority of instances examined. Specifically, QDGA demonstrates a significantly lower diversity indicator IGD compared to the three comparative algorithms in 50 instances and it attains a value of 0 in 22 instances, which unequivocally suggests that all members of the reference set Ω^* are generated solely by QDGA. The statistical results reported in Figure 8 further validate the superior convergence performance of QDGA.

The performance of QDGA is superior, primarily due to the significant role played by Q-learning in the search process. The results presented in Tables 4–6 demonstrate that QDGA achieves outstanding performance. The statistical results shown in Figures 6–8 indicate, with statistical significance, that QDGA outperforms the compared algorithms.

Q-learning is capable of effectively determining which of the various designed crossover and mutation operators should be used to achieve better algorithm performance.

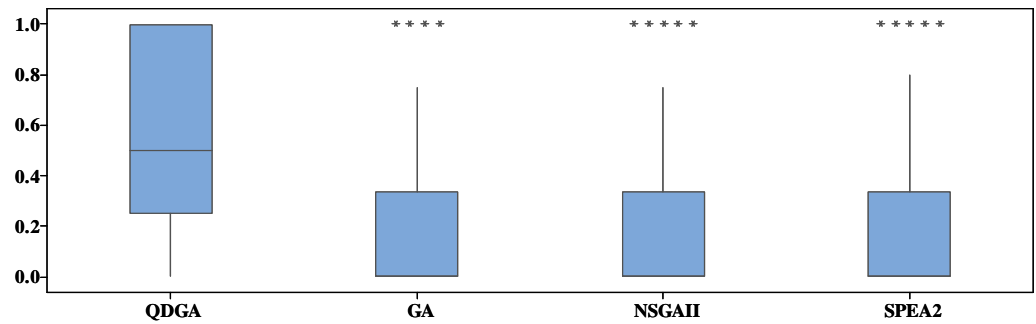


Figure 6. The box plots of metric ρ for four algorithms.

Table 6. Computational results of four algorithms on metric *IGD*.

Instance	QDGA	GA	NSGAI	SPEA2	Instance	QDGA	GA	NSGAI	SPEA2
1	0.021	0.020	0.014	0.015	43	0.011	0.165	0.146	0.437
2	0.002	0.037	0.003	0.235	44	0.092	0.252	0.027	0.023
3	0.029	0.221	0.155	0.142	45	0.000	0.065	0.723	0.453
4	0.445	0.018	0.531	0.179	46	0.000	0.029	0.037	0.167
5	0.001	0.027	0.027	0.083	47	0.000	0.248	0.091	0.057
6	0.161	0.098	0.008	0.087	48	0.040	0.042	0.062	0.157
7	0.000	0.000	0.000	0.000	49	0.189	0.277	0.039	0.149
8	0.001	0.021	0.036	0.040	50	0.037	0.145	0.289	0.029
9	0.000	0.000	0.000	0.000	51	0.014	0.341	0.168	0.168
10	0.000	0.027	0.009	0.024	52	0.109	0.098	0.085	0.210
11	0.000	0.063	0.500	0.813	53	0.073	0.100	0.014	0.037
12	0.013	0.087	0.038	0.041	54	0.000	0.573	0.401	1.190
13	0.000	0.255	0.407	0.276	55	0.416	0.309	0.000	0.262
14	0.000	0.176	0.077	0.103	56	0.000	0.627	0.044	0.131
15	0.016	0.247	0.022	0.091	57	0.004	0.057	0.113	0.539
16	0.059	0.213	0.072	0.145	58	0.000	0.063	0.397	0.253
17	0.003	0.026	0.042	0.028	59	0.012	0.020	0.168	0.049
18	0.100	0.103	0.035	0.139	60	0.000	0.324	0.431	0.403
19	0.203	0.700	0.000	1.000	61	0.042	0.029	0.032	0.000
20	0.060	0.076	0.058	0.033	62	0.546	0.583	0.189	0.223
21	0.030	0.055	0.058	0.060	63	0.034	0.176	0.076	0.023
22	0.036	0.021	0.080	0.076	64	0.403	0.337	0.049	0.000
23	0.056	0.000	0.142	0.072	65	0.039	0.194	0.155	0.185
24	0.000	0.072	0.032	0.113	66	0.534	1.111	0.000	0.588
25	0.020	0.135	0.055	0.046	67	0.137	0.000	0.176	0.798
26	0.000	0.083	0.141	0.163	68	0.173	0.398	0.372	0.514
27	0.022	0.119	0.234	0.120	69	0.353	0.192	0.156	0.000
28	0.023	0.346	0.040	0.000	70	0.030	0.062	0.013	0.324
29	0.000	1.160	0.058	0.400	71	0.002	0.719	0.245	0.007
30	0.020	0.134	0.108	0.084	72	0.005	0.095	0.087	0.077
31	0.064	0.063	0.191	0.158	73	0.005	0.052	0.079	0.014
32	0.000	0.038	0.192	0.628	74	0.053	0.053	0.051	0.022
33	0.057	0.073	0.107	0.141	75	0.014	0.327	0.298	0.050
34	0.000	0.025	0.017	0.084	76	0.095	0.083	0.078	0.061
35	0.000	0.099	0.046	0.201	77	0.008	0.086	0.136	0.082
36	0.088	0.378	0.000	0.124	78	1.398	0.204	0.394	0.054
37	0.000	0.040	0.038	0.161	79	0.000	0.166	0.134	0.122
38	0.012	0.000	0.012	0.013	80	0.042	0.142	0.036	0.352
39	0.000	0.428	1.121	0.376	81	0.297	0.000	0.070	0.102
40	0.000	0.134	0.043	0.113	82	0.000	0.138	0.130	0.901
41	0.016	0.168	0.501	0.289	83	0.189	0.274	0.427	0.426
42	0.151	0.017	0.127	0.141	84	0.000	0.176	0.566	0.795

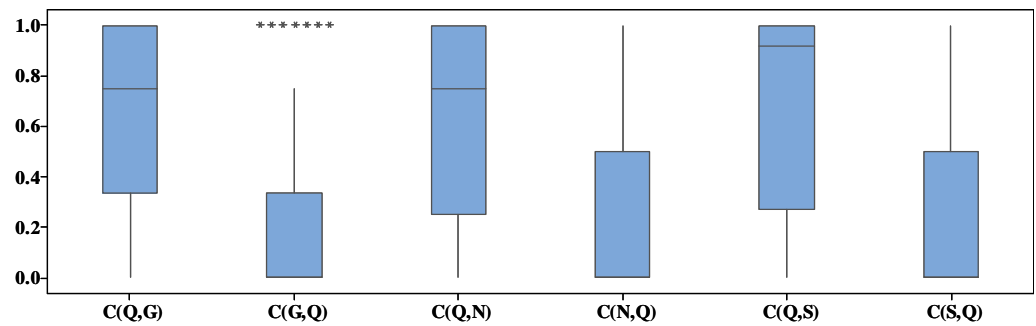


Figure 7. The box plots of metric C for four algorithms.

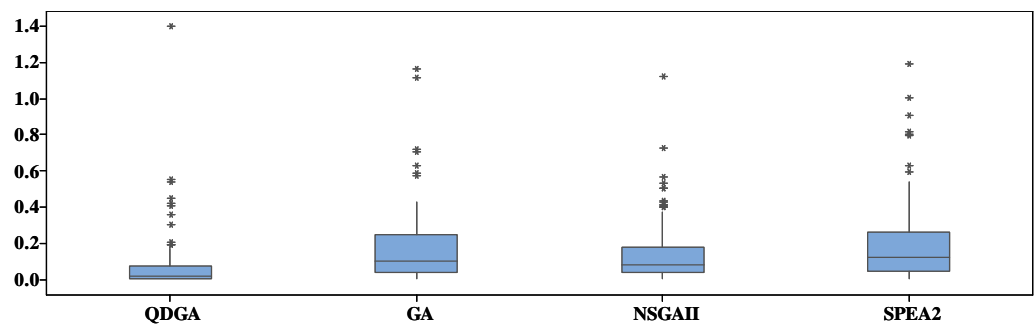


Figure 8. The box plots of metric IGD for four algorithms.

7. Conclusions

A dual-population genetic algorithm with Q-learning is proposed to address the distributed hybrid flow shop scheduling problem. This problem is common in the real-world production process, where multiple objectives need to be considered. QDGA can also solve other problems, such as multi-legged robot control, multiple agents moving cooperatively on another, voltage control of power systems, distribution networks, urban water resource management system, etc. The proposed algorithm employs multiple crossover and mutation operators, and a population assessment method to evaluate the evolutionary state of the population. The algorithm also utilizes Q-learning to guide the search strategy of the second population. The experimental results demonstrate that the proposed algorithm outperforms the basic genetic algorithm. Therefore, the dual-population genetic algorithm with Q-learning can be considered an effective solution for distributed hybrid flow shop scheduling problems.

In the near future, we will continue our research on distributed scheduling problems. We aim to apply meta-heuristics such as the imperialist competitive algorithm to solve these problems. Additionally, we plan to address the problem with energy-related objectives and focus on developing solutions for energy-efficient HFSP. By incorporating energy-efficient objectives, we can create solutions that align with the current trend of sustainability in industrial manufacturing. Therefore, we anticipate that this area of research will continue to gain importance in the future, and our work will contribute to the development of sustainable and efficient manufacturing practices.

Author Contributions: Conceptualization, J.C.; methodology, J.C.; software, J.C.; validation, J.C.; formal analysis, J.C.; investigation, J.C.; resources, J.C.; data curation, J.Z.; writing—original draft preparation, J.C.; writing—review and editing, J.C.; visualization, J.Z.; supervision, J.Z.; project administration, J.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Research Initiation Foundation of Anhui Polytechnic University (2022YQQ002), Anhui Polytechnic University Research Project (Xjky2022002), and the Open Research Fund of AnHui Key Laboratory of Detection Technology and Energy Saving Devices (JCKJ2022B01).

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declared that they have no conflicts of interest in this work.

References

1. Qin, H.X.; Han, Y.Y.; Zhang, B.; Meng, L.L.; Liu, Y.P.; Pan, Q.K.; Gong, D.W. An improved iterated greedy algorithm for the energy-efficient blocking hybrid flow shop scheduling problem. *Swarm Evol. Comput.* **2022**, *69*, 100992. [[CrossRef](#)]
2. Qin, M.; Wang, R.; Shi, Z.; Liu, L.; Shi, L. A Genetic Programming-Based Scheduling Approach for Hybrid Flow Shop With a Batch Processor and Waiting Time Constraint. *IEEE Trans. Autom. Sci. Eng.* **2021**, *18*, 94–105. [[CrossRef](#)]
3. Meng, L.L.; Zhang, C.Y.; Shao, X.Y.; Zhang, B.; Ren, Y.P.; Lin, W.W. More MILP models for hybrid flow shop scheduling problem and its extended problems. *Int. J. Prod. Res.* **2020**, *58*, 3905–3930. [[CrossRef](#)]
4. Wang, G.; Li, X.; Gao, L.; Li, P. An effective multi-objective whale swarm algorithm for energy-efficient scheduling of distributed welding flow shop. *Ann. Oper. Res.* **2021**, *310*, 223–255. [[CrossRef](#)]
5. Zhao, F.Q.; Zhang, L.X.; Cao, J.; Tang, J.X. A cooperative water wave optimization algorithm with reinforcement learning for the distributed assembly no-idle flowshop scheduling problem. *Comput. Ind. Eng.* **2021**, *153*, 107082. [[CrossRef](#)]
6. Zhang, Z.Q.; Qian, B.; Hu, R.; Jin, H.P.; Wang, L. A matrix-cube-based estimation of distribution algorithm for the distributed assembly permutation flow-shop scheduling problem. *Swarm Evol. Comput.* **2021**, *60*, 116484. [[CrossRef](#)]
7. Yang, J.; Xu, H. Hybrid Memetic Algorithm to Solve Multiobjective Distributed Fuzzy Flexible Job Shop Scheduling Problem with Transfer. *Processes* **2022**, *10*, 1517. [[CrossRef](#)]
8. Shao, W.; Shao, Z.; Pi, D. A multi-neighborhood-based multi-objective memetic algorithm for the energy-efficient distributed flexible flow shop scheduling problem. *Neural Comput. Appl.* **2022**, *34*, 22303–22330. [[CrossRef](#)]
9. Meng, L.; Ren, Y.; Zhang, B.; Li, J.Q.; Sang, H.; Zhang, C. MILP Modeling and Optimization of Energy-Efficient Distributed Flexible Job Shop Scheduling Problem. *IEEE Access* **2020**, *8*, 191191–191203. [[CrossRef](#)]
10. Li, Y.; Li, F.; Pan, Q.K.; Gao, L.; Tasgetiren, M.F. An artificial bee colony algorithm for the distributed hybrid flowshop scheduling problem. *Procedia Manuf.* **2019**, *39*, 1158–1166. [[CrossRef](#)]
11. Wang, J.J.; Wang, L. A bi-population cooperative memetic algorithm for distributed hybrid flow-shop scheduling. *IEEE Trans. Emerg. Top. Comput. Intell.* **2020**, *5*, 947–961. [[CrossRef](#)]
12. Cai, J.C.; Lei, D.M. A cooperated shuffled frog-leaping algorithm for distributed energy-efficient hybrid flow shop scheduling with fuzzy processing time. *Complex Intell. Syst.* **2021**, *7*, 2235–2253. [[CrossRef](#)]
13. Zheng, J.; Wang, L.; Wang, J.J. A cooperative coevolution algorithm for multi-objective fuzzy distributed hybrid flow shop. *Knowl.-Based Syst.* **2020**, *194*, 105536. [[CrossRef](#)]
14. Wang, J.J.; Wang, L. A cooperative memetic algorithm with learning-based agent for energy-aware distributed hybrid flow-shop scheduling. *IEEE Trans. Evol. Comput.* **2021**, *26*, 461–475. [[CrossRef](#)]
15. Jiang, E.D.; Wang, L.; Wang, J.J. Decomposition-based multi-objective optimization for energy-aware distributed hybrid flow shop scheduling with multiprocessor tasks. *Tsinghua Sci. Technol.* **2021**, *26*, 646–663. [[CrossRef](#)]
16. Li, Y.; Li, X.; Gao, L.; Zhang, B.; Pan, Q.K.; Tasgetiren, M.F.; Meng, L. A discrete artificial bee colony algorithm for distributed hybrid flowshop scheduling problem with sequence-dependent setup times. *Int. J. Prod. Res.* **2021**, *59*, 3880–3899. [[CrossRef](#)]
17. Lei, D.M.; Xi, B.J. Diversified teaching-learning-based optimization for fuzzy two-stage hybrid flow shop scheduling with setup time. *J. Intell. Fuzzy Syst.* **2021**, *41*, 4159–4173. [[CrossRef](#)]
18. Cai, J.C.; Zhou, R.; Lei, D.M. Dynamic shuffled frog-leaping algorithm for distributed hybrid flow shop scheduling with multiprocessor tasks. *Eng. Appl. Artif. Intell.* **2020**, *90*, 103540. [[CrossRef](#)]
19. Wang, L.; Li, D.D. Fuzzy distributed hybrid flow shop scheduling problem with heterogeneous factory and unrelated parallel machine: A shuffled frog leaping algorithm with collaboration of multiple search strategies. *IEEE Access* **2020**, *8*, 214209–214223. [[CrossRef](#)]
20. Cai, J.C.; Zhou, R.; Lei, D.M. Fuzzy distributed two-stage hybrid flow shop scheduling problem with setup time: Collaborative variable search. *J. Intell. Fuzzy Syst.* **2020**, *38*, 3189–3199. [[CrossRef](#)]
21. Dong, J.; Ye, C. Green scheduling of distributed two-stage reentrant hybrid flow shop considering distributed energy resources and energy storage system. *Comput. Ind. Eng.* **2022**, *169*, 108146. [[CrossRef](#)]
22. Li, Y.L.; Li, X.Y.; Gao, L.; Meng, L.L. An improved artificial bee colony algorithm for distributed heterogeneous hybrid flowshop scheduling problem with sequence-dependent setup times. *Comput. Ind. Eng.* **2020**, *147*, 106638. [[CrossRef](#)]
23. Li, J.Q.; Yu, H.; Chen, X.; Li, W.; Du, Y.; Han, Y.Y. An improved brain storm optimization algorithm for fuzzy distributed hybrid flowshop scheduling with setup time. In Proceedings of the Genetic and Evolutionary Computation Conference, GECCO, New York, NY, USA, 8–12 July 2020; pp. 275–276; Association for Computing Machinery, Inc.: New York, NY, USA, 2020. . [[CrossRef](#)]
24. Qin, H.; Li, T.; Teng, Y.; Wang, K. Integrated production and distribution scheduling in distributed hybrid flow shops. *Memetic Comput.* **2021**, *13*, 185–202. [[CrossRef](#)]
25. Li, J.; Chen, X.L.; Duan, P.; Mou, J.H. KMOEA: A knowledge-based multi-objective algorithm for distributed hybrid flow shop in a prefabricated system. *IEEE Trans. Ind. Inform.* **2021**, *18*, 5318–5329. [[CrossRef](#)]
26. Ying, K.C.; Lin, S.W. Minimizing makespan for the distributed hybrid flowshop scheduling problem with multiprocessor tasks. *Expert Syst. Appl.* **2018**, *92*, 132–141. [[CrossRef](#)]

27. Shao, W.S.; Shao, Z.S.; Pi, D.C. Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem. *Knowl.-Based Syst.* **2020**, *194*, 105527. [[CrossRef](#)]
28. Shao, W.S.; Shao, Z.S.; Pi, D.C. Multi-objective evolutionary algorithm based on multiple neighborhoods local search for multi-objective distributed hybrid flow shop scheduling problem. *Expert Syst. Appl.* **2021**, *183*, 115453. [[CrossRef](#)]
29. Meng, L.; Gao, K.; Ren, Y.; Zhang, B.; Sang, H.; Chaoyong, Z. Novel MILP and CP models for distributed hybrid flowshop scheduling problem with sequence-dependent setup times. *Swarm Evol. Comput.* **2022**, *71*, 101058. [[CrossRef](#)]
30. Cai, J.; Lei, D.; Wang, J.; Wang, L. A novel shuffled frog-leaping algorithm with reinforcement learning for distributed assembly hybrid flow shop scheduling. *Int. J. Prod. Res.* **2023**, *61*, 1233–1251. [[CrossRef](#)]
31. Cai, J.; Lei, D.; Li, M. A shuffled frog-leaping algorithm with memplex quality for bi-objective distributed scheduling in hybrid flow shop. *Int. J. Prod. Res.* **2020**, *59*, 5404–5421. [[CrossRef](#)]
32. Hao, J.H.; Li, J.Q.; Du, Y.; Song, M.X.; Duan, P.; Zhang, Y.Y. Solving distributed hybrid flowshop scheduling problems by a hybrid brain storm optimization algorithm. *IEEE Access* **2019**, *7*, 66879–66894. [[CrossRef](#)]
33. Lei, D.; Wang, T. Solving distributed two-stage hybrid flowshop scheduling using a shuffled frog-leaping algorithm with memplex grouping. *Eng. Optim.* **2019**, *52*, 1461–1474. [[CrossRef](#)]
34. Li, J.Q.; Li, J.K.; Zhang, L.J.; Sang, H.Y.; Han, Y.Y.; Chen, Q.D. Solving type-2 fuzzy distributed hybrid flowshop scheduling using an improved brain storm optimization algorithm. *Int. J. Fuzzy Syst.* **2021**, *23*, 1194–1212. [[CrossRef](#)]
35. Atallah, M.J.; Blanton, M. *Algorithms and Theory of Computation Handbook, Volume 2: Special Topics and Techniques*; CRC Press: Boca Raton, FL, USA, 2009.
36. Charu, A. *Neural Networks and Deep Learning, A Textbook*; Springer: Berlin/Heidelberg, Germany, 2018.
37. Chen, R.; Yang, B.; Li, S.; Wang, S. A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem. *Comput. Ind. Eng.* **2020**, *149*, 106778. [[CrossRef](#)]
38. Wang, J.; Lei, D.; Cai, J. An adaptive artificial bee colony with reinforcement learning for distributed three-stage assembly scheduling with maintenance. *Appl. Soft Comput.* **2021**, *117*, 108371. [[CrossRef](#)]
39. 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](#)]
40. Zitzler, E.; Laumanns, M.; Thiele, L. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization*; Technical Report Gloriestrasse; 103, TIK-Rep; Swiss Federal Institute of Technology: Lausanne, Switzerland, 2001; pp. 1–20.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.