symmetry

MDPI

Article

Integrated Optimization of Blocking Flowshop Scheduling and Preventive Maintenance Using a Q-Learning-Based Aquila Optimizer

Zhenpeng Ge and Hongfeng Wang *

College of Information Science and Engineering, Northeastern University, Shenyang 110819, China; 2100741@stu.neu.edu.cn

* Correspondence: hfwang@mail.neu.edu.cn

Abstract: In recent years, integration of production scheduling and machine maintenance has gained increasing attention in order to improve the stability and efficiency of flowshop manufacturing systems. This paper proposes a Q-learning-based aquila optimizer (QL-AO) for solving the integrated optimization problem of blocking flowshop scheduling and preventive maintenance since blocking in the jobs processing requires to be considered in the practice manufacturing environments. In the proposed algorithmic framework, a Q-learning algorithm is designed to adaptively adjust the selection probabilities of four key population update strategies in the classic aquila optimizer. In addition, five local search methods are employed to refine the quality of the individuals according to their fitness level. A series of numerical experiments are carried out according to two groups of flowshop scheduling benchmark. Experimental results show that QL-AO significantly outperforms six peer algorithms and two state-of-the-art hybrid algorithms based on Q-Learning on the investigated integrated scheduling problem. Additionally, the proposed Q-learning and local search strategies are effective in improving its performance.

Keywords: blocking flowshop; scheduling; preventive maintenance (PM); aquila optimizer (AO); Q-learning (QL)

Citation: Ge, Z.; Wang, H. Integrated Optimization of Blocking Flowshop Scheduling and Preventive Maintenance Using a Q-Learning-Based Aquila Optimizer. *Symmetry* **2023**, *15*, 1600. https://doi.org/10.3390/sym15081600

Academic Editor: José Carlos R. Alcantud

Received: 17 July 2023
Revised: 12 August 2023
Accepted: 16 August 2023
Published: 18 August 2023

1. Introduction

In the field of intelligent manufacturing, manufacturers are generally aiming at more reliable production systems. However, both deterioration and default of machines are becoming critical factors because they inevitably appear in practical production systems [1,2]. In such a scenario, machines are not idealized, and they can break down and deteriorate with cumulative processing time. To restore machine efficiency and reduce faults, executing preventive maintenance (PM) is necessary [3]. Nevertheless, making decisions solely from one side of production or PM is seriously hard to reach a good scheduling solution since they are in conflict with each other. Specifically, PM consumes production time, whereas delaying PM to ensure production on time may increase the probability of machine faults. To this end, integrated optimization of production and PM has become an effective method to handle such problems [4]. In the integrated optimization process, degradation and default of machines are considered, and PM is regarded as a constraint to simultaneously optimize production and PM. In recent years, a large number of studies have demonstrated that this integrated optimization method can achieve a high-quality solution [5]. Consequently, this study aims to optimize an integrated production and PM problem.

Flowshop is a well-known manufacturing system for production scheduling, and complex manufacturing plants often can be abstracted into variants or combinations of this system [6]. Blocking flowshop is one of the most important variants of flowshop. It has limited buffer capacity between machines due to the process characteristics and technological requirements, and it has significant practical applications, e.g., chemical

production [7], steel industry [8], and robotic scheduling [9]. The reason of the blocking may be that jobs are stranded in machines due to the lack of workers, or certain stages are not allowed to be stored [10]. Due to the important practical significance, this study investigates the integrated optimization of blocking flowshop. A general situation, i.e., the blocking flowshop with no intermediate buffers is taken as the research object. In this situation, a job cannot leave a machine until the next machine is free. Following the three-field notation proposed by Graham et al. [11], the problem studied in this paper can be expressed as Fm|PM, blocking|$\omega_1 C_{max} + \omega_2 TMC$, where Fm means the flowshop, PM represents the preventive maintenance, blocking indicates the blocking scheduling, and $\omega_1 C_{max} + \omega_2 TMC$ denotes the object with minimizing the weighted sum of completion time and total maintenance cost.

In terms of algorithm, since the classical blocking flowshop with three or more machines has been proven to be NP-hard [12], it is even more NP-hard for a blocking flow shop with an integrated optimization. It is difficult to obtain a satisfactory solution in a short time for large-scale problems with exact and heuristic algorithms. Therefore, various swarm intelligence algorithms are often as fast and effective tools of solving such problems [13], such as genetic algorithms (GA) [14], particle swarm optimization algorithms (PSO) [15], and bee colony algorithms (ABC) [16]. However, for complex integrated optimization problems, especially large-scale problems, swarm intelligence algorithms often easily fall into local optimal solutions. Thus, scholars are continuously designing high-performance search mechanisms to handle such problems. The aquila optimizer (AO) is a recently developed swarm evolutionary algorithm that simulates the hunting behavior of aquila using multiple update strategies. Recent studies [17] have revealed that AO has advantages of strong search capabilities and fast convergence. Moreover, it has been well applied to problems such as path planning [18], 0–1 backpacking [19] and network node localization [20]. Therefore, in this work, the high-performance AO is used to get a better solution.

The presence of PM and blocking increases the complexity of the scheduling, making it a complex, coupled, and symmetric combinatorial optimization problem. In order to cope with the considered problem with high performance, it is necessary to adopt some strategies to enhance the search capability. In recent years, Q-learning (QL) has become a valid tool to improve intelligent algorithms in the field of scheduling, and the improved algorithms always obtain an excellent performance. For instance, Runfo Li et al. [21] used QL to dynamically adjust the crossover and variance probabilities of GA for the port ship scheduling problem. Mao et al. [22] used QL to select the update strategies of brain storm optimization algorithm for the assembly flow shop scheduling problem. Lixin Cheng et al. [23] designed a QL based on the population size adjustment mechanism for the energy-efficient manufacturing scheduling problem. Zhang et al. [24] designed a QL based ant colony algorithm for an assembly scheduling, where QL is adopted to adjust the search parameters. Learning from the previous studies, this study proposes a QL-based AO (QL-AO), wherein QL is applied to adjust the selection probabilities of the four update strategies of the basic AO. Additionally, a set of local search strategies is designed according to the problem features. To verify the performance of QL-AO, computational experiments are performed. The results show that the proposed algorithm can gain better results when solving the proposed problem.

The contributions of this work are as follows:

(1) This work formulates an integrated optimization model for a blocking flowshop scheduling problem. In this model, deterioration and default of machines, as well as machine maintenance are considered at the same time. To calculate the object values, a recursive formula is established;
(2) This work develops an AO with some special search techniques to enhance its performance and propose the improved algorithm QL-AO. It employs a QL-based mechanism for strategies selection. Other than that, a set of local search strategies is designed to strengthen the search ability via combining the problem's features;

(3) This work conducts a series of experiments to evaluate the performance of the proposed QL-AO by comparing it with eight peer algorithms. They are experiments of parameter settings, components comparison and algorithm comparison. The achieved results suggest that QL-AO is an efficient optimizer compared to its peers.

The remainder of this paper is outlined as follows: Section 2 describes the problem scenario and builds a mathematical model. Section 3 presents the designed algorithm. Section 4 implements numerical experiments and analyzes the results. Section 5 concludes this paper and provides some future research directions.

2. Problem Description

This study investigates an integrated optimization problem of blocking flowshop scheduling and preventive maintenance (PM), in which both deterioration and default of machines are also considered. The problem description is laid out in the following sections.

There are n jobs arriving at the initial time to be processed once on m machines in turn. The sequence of processing jobs on all machines is the same as the first machine. Due to the presence of blocking, the processed job must wait for the next operation on the current machine until the next machine is available for processing. It is noted that two special factors, that is, deterioration and default of machines, are both taken into account on the basis of classic blocking flowshop scheduling. This means that the actual processing time of job requires to be determined by the deterioration and failure parameters of the machine. The purpose of integrated scheduling is to find the optimal job processing sequence $J_{k,}(k = 1, 2, \cdots, n)$, as well as to determine when to execute PM activity.

For machine deterioration, let $p_{i,j}$ denote the normal processing time of i-th job on machine j, $a_{i,j}$ denote the machine age of machine j after processing the i-th job, and γ_i denote the deterioration factor of machine j. The processing time of i-th job on machine j can be calculated as $p'_{i,j} = p_{i,j} + \gamma_i a_{i,j-1}$ when considering linear deterioration [25].

For machine fault, the Weibull distribution is adopted here according to the work of Wang et al. [26]. The expected number of failures when machine j processes the i-th job i can be calculated as $N_{ij} = \int_{a_{i,j-1}}^{a_{i,j}} r(t)dt = \left(\frac{a_{i,j}}{\eta}\right)^{\beta} - \left(\frac{a_{i,j-1}}{\eta}\right)^{\beta}$, where β is the shape parameter and η is the scale parameter. Once fault occurs, corrective maintenance (CM) is executed immediately. CM can only restore the operating state of the machine but not restore the machine age. In this study, the execution time of CM is included in the actual processing time. Therefore, the actual processing time after considering the machine faults and CM can be calculated as $p''_{i,j} = p'_{i,j} + N_{ij}t_{cm}$, where t_{cm} is the time for executing a CM.

This study employs a threshold-based PM strategy to ensure that the machine operates at a consistently high level of reliability [27]. Specifically, the machine undergoes PM every TPM (Time Period of Maintenance) to avoid potential fault or performance degradation. The TPM should always be less than the maximum age limit T_{maxj}. Here, T_{maxj} can be calculated as $T_{maxj} = \eta \cdot exp(\frac{\ln(-\ln(R_j))}{\beta})$, where R_j is the minimal reliability threshold of machine.

Based on this policy, the execution of PM can be determined after obtaining the scheduling sequence. To resolve any potential conflict between PM and job processing, this study employed a conservative PM insertion approach. When scheduling a job, the first step is to calculate whether the machine age exceeds T_{maxj} after processing. If the limit is exceeded, the processing of this job is immediately delayed, and the PM is inserted at the end of the previous job. If the threshold is not exceeded, normal processing the job without executing PM. Therefore, the PM decision matrix can be defined as $pm_{i,j-1} = \begin{cases} 0, if\ a_{i,j} \leq T_{maxj} \\ 1, \ else \end{cases}$, where $pm_{i,j-1} = 1$ denotes executing PM and restoring the machine age, i.e., set $a_{i,j-1} = 0$, and $pm_{i,j-1} = 0$ denotes no execution of PM and calculating the machine age $a_{i,j} = a_{i,j-1} + p''_{i,j}$.

To further explain the impact of PM and blocking on scheduling, let us consider a 3×6 scheduling example, as depicted in Figure 1. Regarding PM, when machine 1 is processing job 5, an immediate execution of job 1 would exceed the machine's maximum age limit. Therefore, PM is inserted after job 5's processing, leading to the postponement of job 1's processing. Regarding blocking, during the processing of job 4 on the first machine, job 6 is concurrently being processed on machine 2. Since there is no intermediate buffer, job 4 remains on the first machine until job 6 completes its processing. Consequently, job 4's transportation to machine 2 for processing is delayed, thereby causing a subsequent delay in starting job 5.

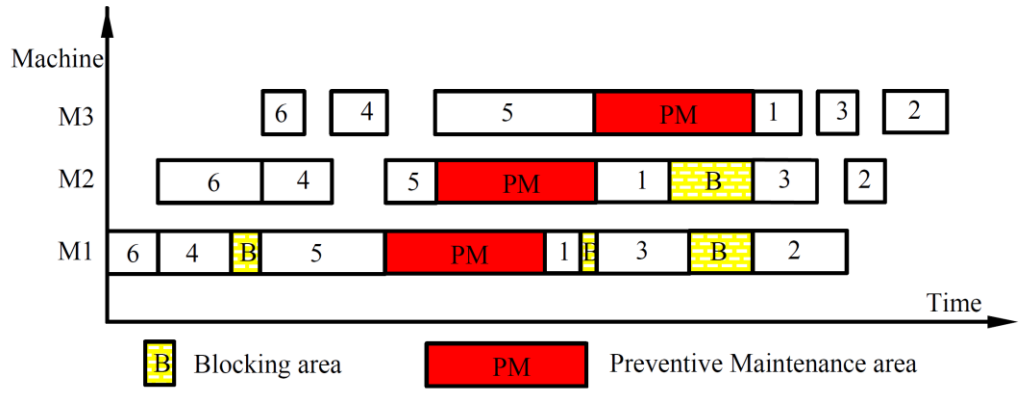

Figure 1. Diagram of blocking flowshop integrated scheduling.

The recursive formula to calculate the makespan (C_{max}) is proposed as shown in Equations (1)–(6). This formula is an extension of the standard blocking flowshop recursive formula [28]. In the recursive process, the calculation starts with determining the completion time of the first job. Then, the departure time is calculated, followed by determining the completion time and departure time of the second job. This process continues iteratively until reaching the last job. And then, C_{max} is obtained.

$$C_{1,j} = \sum_{l=1}^{j} \left(p''_{1,l} \right), j = 1, \cdots, m. \tag{1}$$

$$D_{1,j} = C_{1,j}, j = 1, \cdots, m. \tag{2}$$

$$D_{i,0} = D_{i-1,1}, i = 2, \ldots, n. \tag{3}$$

$$C_{i,j} = \max\left[C_{i-1,j} + Tpm_j \times pm_{i-1,j}, D_{i,j-1} \right] + p''_{i,j}, i = 2, \ldots, n, j = 1, \cdots, m. \tag{4}$$

$$D_{i,j} = max\left\{ C_{i,j}, D_{i-1,j+1} \right\}, i = 2, \ldots, n, j = 1, \cdots, m-1. \tag{5}$$

$$C_{max} = C_{i,m} = D_{i,m}, i = 2, \ldots, n. \tag{6}$$

Notably, $D_{1,j}$ represents the start time of the first job on machine j. Due to the presence of PM, the other job's start time on the machine j depends on the completed time for PM and the departure time of the last job. The computation of $p''_{i,j}$ and $pm_{i,j}$ has been detailed in the preceding paragraphs.

The objective function in this study considers not only the widely used metric C_{max}, but also the substantial costs associated with maintenance activities. The goal of the problem is to minimize the weighted sum of the C_{max} and the total maintenance cost, which encompasses both PM cost and CM cost. The total maintenance cost is calculated

by multiplying the total number of maintenance activities by the unit price of each. The mathematical representation of the objective function is provided in Equation (7).

$$\text{Min } obj = \omega_1 C_{max} + \omega_2 [CPM \times \sum_{i=1}^{n} \sum_{j=1}^{m} N_{i,j} + CMR \times \sum_{i=1}^{n} \sum_{j=1}^{m} pm_{i,j}] \quad (7)$$

where ω_1, ω_2 are weighting factors, and *CPM*, *CMR* are the cost of performing once PM and CM, respectively.

3. The Proposed Algorithm

3.1. Basic Aquila Optimizer

Aquila optimizer (AO) is a novel population-based swarm intelligence algorithm proposed by Laith Abualigah et al. [29] in 2021, which simulates the hunting process of aquila. In the framework of AO algorithm, there are four key population update strategies termed as expanded exploration, narrowed exploration, expanded exploitation and narrowed exploitation. The whole iteration process of AO can be divided into two periods. In the first 2/3 iterations, that is regarded as the exploration period, expanded exploration and narrowed exploration strategies are selected randomly to update the population. In the last 1/3 iterations, that is regarded as the exploitation period, expanded exploitation and narrowed exploitation strategies are employed randomly, as shown in Algorithm 1.

Algorithm 1: Basic AO

```
Initialize the population X and the parameters (i.e., α, δ, etc.) of the AO;
Calculate the fitness values of individuals in the population and find X_best^t;
Set t = 0;
While (the end condition is not met) do
  for each individual in the current population do
      Update X_mean^t, x, y, QF(t), Levy(D);
      if t ≤ (2/3)itermax
          if rand ≤ 0.5
              Update the current individual using expanded exploration;
          else
              Update the current individual using narrowed exploration;
          end if
      else
          if rand ≤ 0.5
              Update the current individual using expanded exploitation;
          else
              Update the current individual using narrowed exploitation;
          end if
      end if
  end for
  Update X_best^t;
  T = t + 1;
end while
return The best solution (X_best).
```

For expanded exploration strategy, the AO widely explores high soar to determine the area of the search space, as shown in the following formula.

$$X^{t+1} = X_{best}^{t} \times \left(1 - \frac{t}{itermax}\right) + \left(X_{mean}^{t} - X_{best}^{t} \times rand\right) \quad (8)$$

where t is the current iteration, $itermax$ is the maximum iteration, X_{best}^{t} is the best individual in the t-th iteration, and $rand$ is the uniformly distributed random numbers between 0

and 1. X_{mean}^{t} denotes the locations mean value of the current population connected at t-th iteration, defined as $X_{mean}^{t} = \frac{1}{N}\sum_{i=1}^{N} X_i^t$.

For narrowed exploration strategy, AO narrowly explores the selected area of the target prey in preparation for the attack, as shown in the following formula.

$$X^{t+1} = X_{best}^{t} \times levy(D) + X_{rand}^{t} + (y - x) \times rand \tag{9}$$

where X_{rand}^{t} is an individual selected randomly from the population in the t-th iteration. y and x are used to present the spiral shape in the search, which are calculated as $x = r \times sin(\phi)$, and $y = r \times \cos(\phi)$. $r = r_1 + 0.00565 \times D_1$, and $\phi = -0.005 \times D_1 + \frac{3\pi}{2}$, where r_1 is a parameter between 1 and 20 for fixed the number of search cycles, and D_1 is integer numbers from 1 to the length of the search space.

$Levy(D)$ is the Levy flight distribution function used to enhance the randomness of the search, as shown in Equation (10).

$$Levy(D) = s \times \frac{u \times \sigma}{|v|^{1/\lambda}}, \sigma = \left(\frac{\Gamma(1+\lambda) \times sine\left(\frac{\pi\lambda}{2}\right)}{\Gamma\left(\frac{1+\lambda}{2}\right) \times \beta \times 2^{\left(\frac{\lambda-1}{2}\right)}} \right) \tag{10}$$

where D is the dimension of the individual and $\Gamma(\cdot)$ is the gamma function. The constant parameters s and λ are set to 0.01 and 1.5, respectively, and the random parameters u and v are taken between 0 and 1.

For expanded exploitation strategy, AO exploits the selected area of the target to get close to the prey and attack, as shown in the following formula.

$$X^{t+1} = \nu \times \left(X_{best}^{t} - X_{mean}^{t}\right) - rand + \delta \times [(UB - LB) \times rand + LB] \tag{11}$$

where UB and LB are the upper and lower bounds, respectively. ν and δ are both the exploitation adjustment parameters.

For narrowed exploitation, AO attacks the prey over the land according to their stochastic movements, as shown in the following formula.

$$X^{t+1} = QF(t) \times X_{best}^{t} - \left(G_1 \times X^t \times rand\right) - G_2 \times Levy(D) + G_1 \times rand \tag{12}$$

where $QF(t)$ is a quality function defined as $QF(t) = \frac{2 \times rand - 1}{t^{(1-itermax)^2}}$, which is used to balance the search strategy. $G_1 = 2 \times rand - 1$ is used to simulate the action of tracking prey, and $G_2 = 2 \times \left(1 - \frac{t}{itermax}\right)$ is used to represent the slope of the flight.

3.2. Individual Representation

Considering that the AO algorithm requires to utilize the individuals with continuous encoding to search for the optimal processing sequence of jobs of the investigated scheduling problem, a random key representation-based ROV rule [30] is used to accomplish the mapping of individual and candidate solution in this study. For example, let us consider an individual $X = [0.82, 0.18, 0.23, 0.68]$, where the minimum value 0.18 is at position 2. This indicates that job 2 will be the first to be processed. Next, the sub-minor value 0.23 is at position 3, indicating that job 3 will be the second to be processed. So, the corresponding candidate solution, that is, the processing sequence of 4 jobs, is $J = \{2, 3, 4, 1\}$.

In order to enhance the quality of the initial population, an NEH heuristic is used to generate a number of initial individuals with higher fitness. In detail, a candidate solution of flowshop scheduling is firstly achieved according to the NEH heuristic, and then 10% individuals in the initial population are generated randomly by executing the abovementioned ROV rule upon the achieved solution reversely.

3.3. Q-Learning-Based Strategies Selection

The balance of exploration and exploitation is very important to intelligent algorithms [31]. In the basic AO, the selection probabilities of four population update strategies are artificially fixed. To cope with the complex integration scheduling problem, a Q-learning (QL)-based selection method is proposed, which is the major contribution of this study. In the proposed selection method, QL is employed to dynamically adjust the selection probabilities of population update strategies according to the feedback information of AO's iteration process. The procedure of the QL update process is given in Algorithm 2. The state, action and reward are presented below.

State: There are 12 states in the state set. It is divide into two metrics, that is *cbad* and *popdiv*, as shown in Table 1. Here, *cbad* is the cumulative unimproved frequency of historical optimal solutions, and c_1, c_2, d_1, d_2, d_3 are the division parameters. The *popdiv* represents the population diversity as shown in Equations (13) and (14) [32].

$$popdiv = \frac{\sum_{k=1}^{D} \sum_{\eta}^{D} f(q)_k (1 - f(q)_k)}{D - 1} \tag{13}$$

where $f(q)_k$ is defined as Equation (14). For notation $\delta_{i,j}(q)$, if the j-th job processed by the machine i is job q, then $\delta_{i,j}(q)$ is set to 1, otherwise it is set to 0.

$$f(q)_k = \frac{\sum_{i=1}^{popnum} \sum_{j=1}^{D} \delta_{i,j}(q)}{popnum}, \forall q, k = 1, \cdots, D \tag{14}$$

Table 1. States definition table.

State	Description	State	Description
1	$cbad < c_1$&& $popdiv < d_1$	7	$c_1 \leq cbad < c_2$&& $d_2 \leq popdiv < d_3$
2	$cbad < c_1$&& $d_1 \leq popdiv < d_2$	8	$c_1 \leq cbad < c_2$&& $popdiv \geq d_3$
3	$cbad < c_1$&& $d_2 \leq popdiv < d_3$	9	$cbad \geq c_2$&& $popdiv < d_1$
4	$cbad < c_1$&& $popdiv \geq d_3$	10	$cbad \geq c_2$&& $d_1 \leq popdiv < d_2$
5	$c_1 \leq cbad < c_2$&& $popdiv < d_1$	11	$cbad \geq c_2$&& $d_2 \leq popdiv < d_3$
6	$c_1 \leq cbad < c_2$&& $d_1 \leq popdiv < d_2$	12	$cbad \geq c_2$&& $popdiv \geq d_3$

Action: Action set $\mathbb{A}$ is composed of four actions. They are defined as increasing the selection probabilities of four update strategies. For example, let $\Delta\xi$ denote the amount of probability increase, and p_1 denote the selection probability of expanded exploration. Action 1 can be described as $p_1 \leftarrow p_1 + \Delta\xi$. To ensure that the probability sum is 1 after the action is executed, the selection probabilities are normalized as $p_i = p_i / \sum_{i=1}^{4} p_i$. The well-known ε-greedy strategy is used to select an action, as described in lines 10–14 in Algorithm 2. In addition, to expedite the convergence of QL, the ε value undergoes a linear decrease from 0.9 to 0.01 within every 100 iterations.

Reward: The reward method is described in lines 2–8 in Algorithm 2. If the optimal solution is improved, the reward is set to 20. If the optimal solution does not improve but there is an increase in diversity, the reward is set to 10. If neither of these conditions is met, the reward is set to -5.

Algorithm 2: Pseudo code of *Q-learning* update

Input: $\mathbb{A}$, Q, s, $cbad$, $popdiv$, c_1, d_1, f^t_{best}, f_{best}, ε.
Output: Q, a′, s′.

```
Get the new state s′ by cbad, popdiv, c1, d1 based on the rules in Table 1.
If f^t_best < f_best
    r = 20
else if popdiv > 0
    r = 10
else
    r = −5
end if
Q(s,a) ← (1 − α)Q(s,a) + α[r + γ·max_{a′∈A} Q(s′,a′)]
if rand > ε
    a′ ← argmax_{a∈A} Q(s′,a)
else
    a′ ← randomChoise(A)
end if
```

3.4. Local Search Strategies

In order to further enhance the search performance, five search strategies are used, which can be described as follows.

1. Machine Age-Based Insert (MI): Insert the job with the highest mean machine age (the average value of machine age for all machines) after the job with the lowest mean machine age but excluding the first job. By applying this strategy, the job with the highest mean machine age is repositioned to a more favorable location. This approach may find a more potential solution.
2. PM-Based Swap (PS): In this strategy, the job with the maximum total times of PM is moved one position backward. By performing this swap operation, the algorithm aims to explore different arrangements of the PM-intensive job, potentially leading to improvements in the scheduling solution.
3. Job Insert (JI): This is a common local search strategy in that two different jobs are randomly selected and the first job is inserted after the second job. This operation introduces a change in the sequence of jobs and may lead to an improved solution.
4. Job Swap (JS): Another commonly used local search strategy is job swap. Two different jobs are randomly selected, and their positions are swapped. This exchange alters the job order, potentially resulting in a better scheduling solution.
5. Random Generation (RG): This strategy randomly generates a scheduling solution. The purpose of this strategy is to introduce diversity into the population. It encourages the discovery of novel and potentially better scheduling solution.

For executing the five local search strategies, the population is first sorted according to objective values. The top 20% individuals use MI and PS with equal probability. The bottom 20% use RG, and the other individuals use JI and JS with equal probability. Among all the local search strategies, the new solution is accepted if it improved, otherwise not.

3.5. The Framework of Proposed Algorithm

The proposed algorithm termed as QL-AO is an effective combination of AO and QL to further improve the performance of AO for the investigated problem. The detailed steps of QL-AO are presented below and the flow chart is shown in Figure 2.

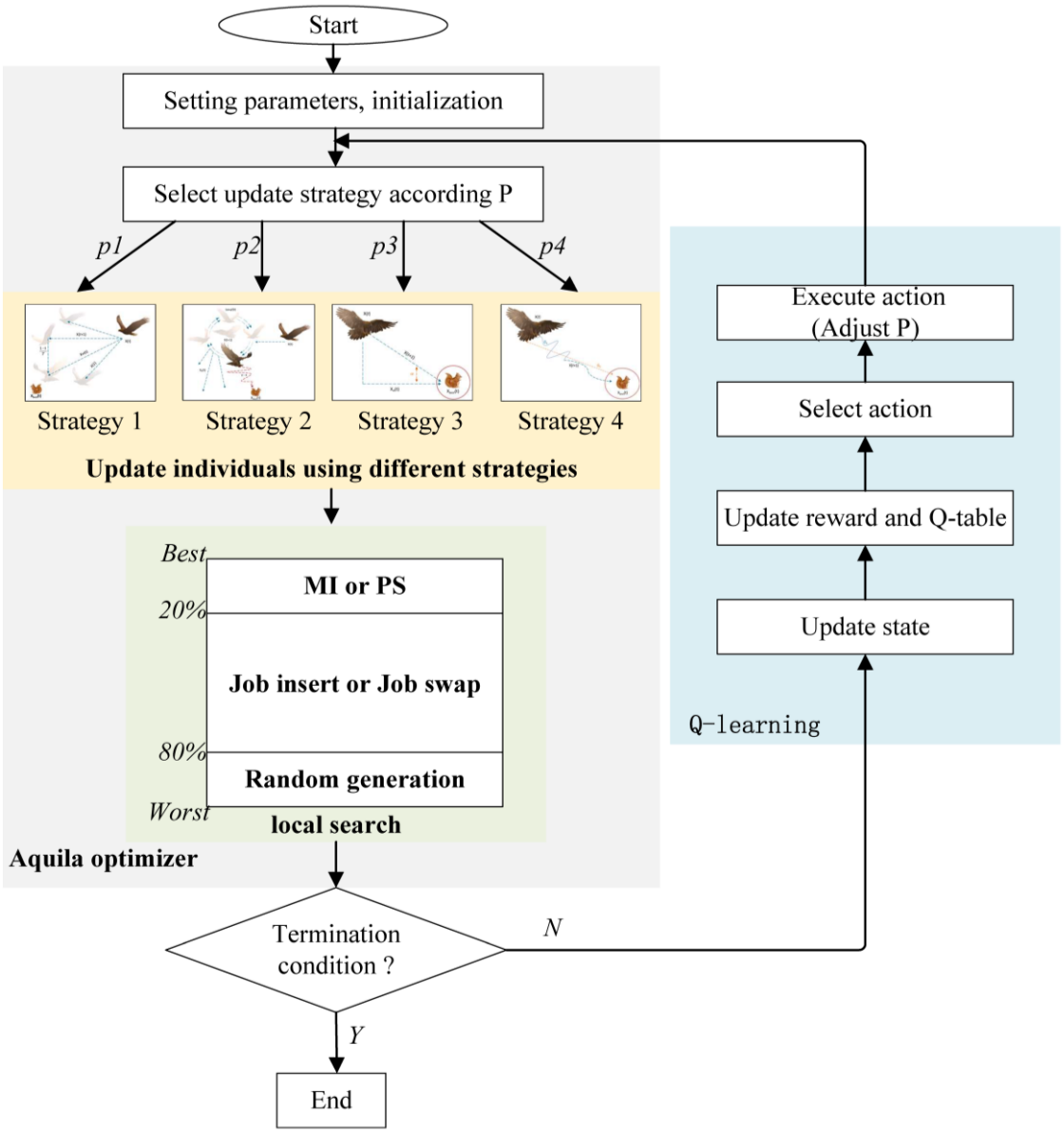

Figure 2. The flow chart of QL-AO.

(1) Initialize the population with 90% random individuals and 10% ones generated via the NEH-based solution.
(2) Select update strategy for each individual according to selection probabilities and then update the population using different strategies.
(3) Execute local search for each individual to further improve its quality.
(4) If the termination condition is not met, execute a new iteration after adjusting the selection probabilities by QL; otherwise, the algorithm is terminated.
(5) Go to QL section. Update the system state, reward, and Q-table. Select the new action, execute the action to adjust probabilities, and then go to step (2).

4. Computational Experiments

In this section, a series of experiments are conducted to evaluate the performance of the proposed QL-AO algorithm for the investigated integrated optimization problem of blocking flowshop scheduling and preventive maintenance. All algorithms run on a computer with an Intel Core i5-10500 @ 3.10 GHz CPU and 16 GB memory and all the algorithms are coded in the software platform MATLAB 2017a.

4.1. Test Instance Settings

In total, the 19 test instances are generated according to two flowshop scheduling benchmark, in which the first 12 are proposed by Taillard [33] and the last 7 come from the work of Ruiz [34], respectively. Additionally, we set the parameters of machine failure and maintenance as follows: two parameters of Weibull distribution $\beta = 2$ and $\eta = 7000$, deterioration factor $\gamma = 0.02$, two parameters of maintenance time $t_{cm} = 20$ and $t_{pm} = 100$, and reliability threshold R = 0.85.

4.2. Key Parameter Settings

In the proposed QL-AO algorithm, there are five key parameters, i.e., the population size n, two exploitation adjustment parameters ν and δ, two QL-related parameters α and γ. To find the promising parameters for the algorithms and the sensitivity analysis of parameters, the Taguchi's orthogonal experiment approach was employed. In this part, three sets of orthogonal experiments were conducted, respectively, on a small-scale instance (with 20 jobs and 20 machines), a medium-scale instance (with 100 jobs and 5 machines), and a large-scale instance (with 400 jobs and 20 machines). In each group of experiment, 5 parameter levels were chosen for each parameter and an orthogonal array with 25 parameter combinations was picked. QL-AO with each parameter combination was run 20 times, and the mean value of the object over 20 independent runs was determined as the response variable (RV), as shown in Table 2. Additionally, Table 3 shows the significant rank of parameter combinations, and the results of the orthogonal experiments are shown in Figure 3.

Table 2. Orthogonal experiment settings of QL-AO.

Trial Number	Factor Levels					RV (20 × 20)	RV (5 × 100)	RV (400 × 20)
	n	ν	δ	α	γ			
1	20	0.1	0.1	0.1	0.5	2919.89	7607.12	54201.77
2	20	0.3	0.3	0.2	0.6	2914.63	7602.14	54207.26
3	20	0.5	0.5	0.3	0.7	2928.07	7617.98	54208.93
4	20	0.7	0.7	0.4	0.8	2952.31	7602.17	54208.01
5	20	0.9	0.9	0.5	0.9	2914.06	7605.10	54205.58
6	40	0.1	0.3	0.3	0.8	2916.63	7577.49	54150.14
7	40	0.3	0.5	0.4	0.9	2933.99	7602.04	54134.41
8	40	0.5	0.7	0.5	0.5	2910.07	7590.76	54137.54
9	40	0.7	0.9	0.1	0.6	2904.77	7597.79	54112.47
10	40	0.9	0.1	0.2	0.7	2907.46	7572.77	54102.59
11	60	0.1	0.5	0.5	0.6	2913.42	7555.60	54114.38
12	60	0.3	0.7	0.1	0.7	2921.55	7570.67	54092.44
13	60	0.5	0.9	0.2	0.8	2900.78	7580.53	54104.55
14	60	0.7	0.1	0.3	0.9	2910.33	7570.79	54076.38
15	60	0.9	0.3	0.4	0.5	2905.61	7583.08	54103.36
16	80	0.1	0.7	0.2	0.9	2918.75	7574.19	54081.89
17	80	0.3	0.9	0.3	0.5	2902.09	7566.89	54084.02
18	80	0.5	0.1	0.4	0.6	2928.77	7532.74	54061.93
19	80	0.7	0.3	0.5	0.7	2893.97	7574.99	54061.09
20	80	0.9	0.5	0.1	0.8	2910.99	7563.49	54098.52
21	100	0.1	0.9	0.4	0.7	2906.50	7565.46	54075.70
22	100	0.3	0.1	0.5	0.8	2904.08	7573.83	54088.92
23	100	0.5	0.3	0.1	0.9	2900.06	7560.35	54070.07
24	100	0.7	0.5	0.2	0.5	2894.66	7576.20	54040.78
25	100	0.9	0.7	0.3	0.6	2901.78	7583.82	54031.82

Table 3. Response and rank of parameters.

(a) Small-scale instances with 20 jobs and 20 machines					
Levels	n	ν	δ	α	γ
1	2925.79	2915.04	2914.11	2911.45	2906.46
2	2914.58	2915.27	2906.18	2907.26	2912.67
3	2910.34	2913.55	2916.23	2911.78	2911.51
4	2910.91	2911.21	2920.89	2925.44	2916.96
5	2901.42	2907.98	2905.64	2907.12	2915.44
Delta	24.38	7.29	15.25	18.32	10.49
Rank	1	5	3	2	4
(b) Medium-scale instances with 100 jobs and 5 machines					
Levels	n	ν	δ	α	γ
1	7606.90	7575.97	7571.45	7579.88	7584.81
2	7588.17	7583.11	7579.61	7581.17	7574.42
3	7572.13	7576.47	7583.06	7583.39	7580.37
4	7562.46	7584.39	7584.32	7577.10	7579.50
5	7571.93	7581.65	7583.15	7580.06	7582.49
Delta	44.44	8.42	12.87	6.30	10.39
Rank	1	4	2	5	3
(c) Large-scale instances with 400 jobs and 20 machines					
Levels	n	ν	δ	α	γ
1	54206.31	54124.78	54106.32	54115.05	54113.49
2	54127.43	54121.41	54118.38	54107.41	54105.57
3	54098.22	54116.60	54119.40	54110.26	54108.15
4	54077.49	54099.75	54110.34	54116.68	54130.03
5	54061.46	54108.37	54116.46	54121.50	54113.67
Delta	144.85	25.03	13.09	14.09	24.46
Rank	1	2	5	4	3

It can be observed from the main effects plots that $n = 100$, $\nu = 0.9$, $\delta = 0.9$, $\alpha = 0.5$, $\gamma = 0.5$ is a promising parameter combination for small-scale instances; $n = 80$, $\nu = 0.1$, $\delta = 0.1$, $\alpha = 0.4$, $\gamma = 0.6$ is a promising parameter combination for medium-scale instances;$n = 100$, $\nu = 0.7$, $\delta = 0.1$, $\alpha = 0.2$, $\gamma = 0.6$ is a promising parameter combination for large-scale instances. From Table 3, it is observed that population size n is the most significant parameter for all situations. An appropriate increase in n is beneficial to the search. The ranking of two exploitation adjustment parameters ν, δ increases and decreases, respectively, with the increase in scale of the instances. This suggests that in large-scale instances, the parameter ν is more important. The learning rate α is ranked low in the medium-scale and large-scale instances but is ranked high in the small-scale instance. This suggests that only the small-scale instances are more sensitive to the setting of the learning rate. The importance of the discount rate γ is not much affected by the scale of the instances, and it keeps around rank 3.

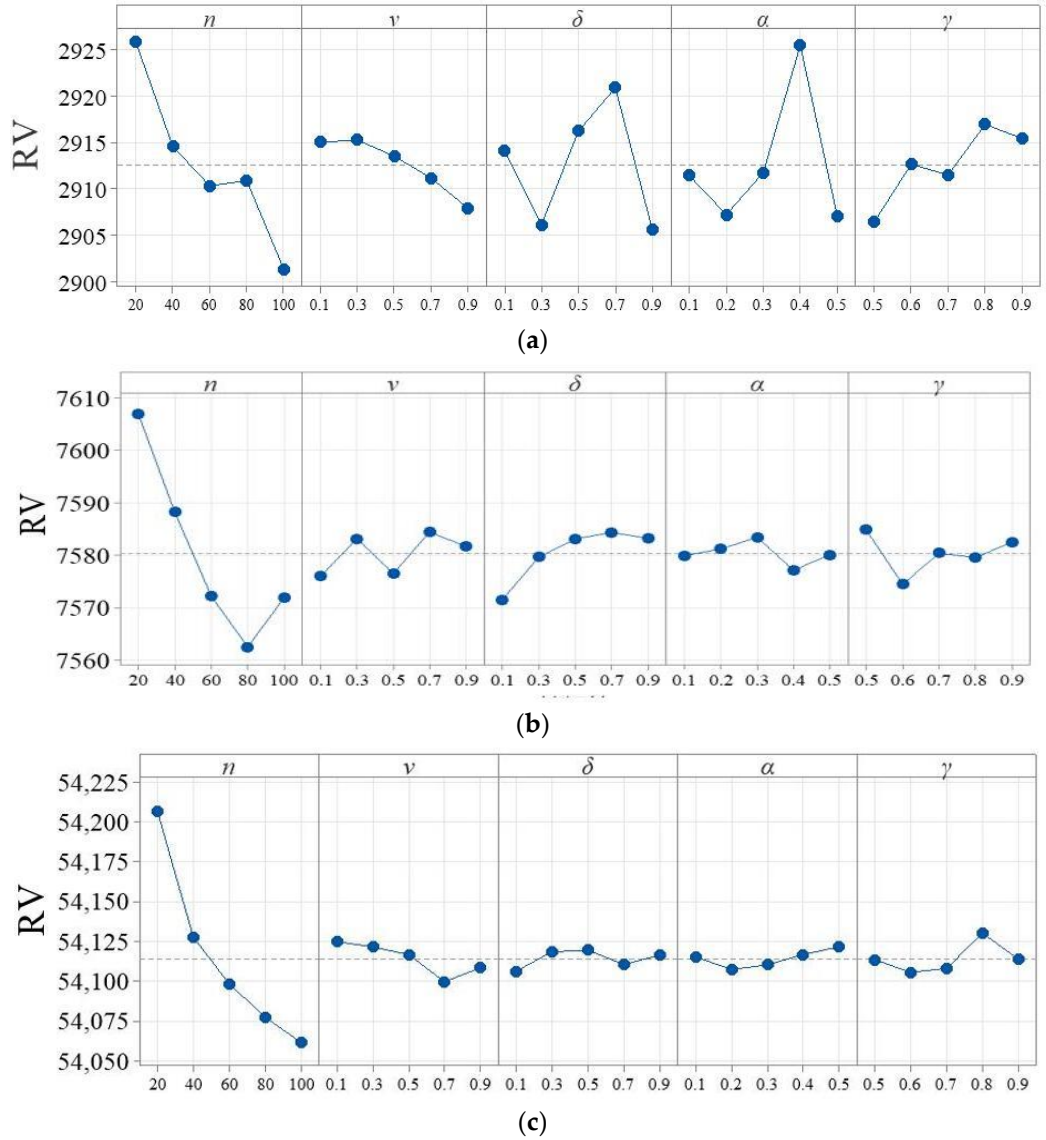

Figure 3. Main effect plots of key parameters. (**a**) Small-scale instance with 20 jobs and 20 machines; (**b**) large-scale instance with 400 jobs and 20 machines; (**c**) large-scale instance with 400 jobs and 20 machines.

4.3. Comparison of the Components on QL-AO

The proposed QL-AO includes the following key components: (1) NEH heuristic method for initialization, (2) Q-learning-based strategies selection, and (3) local search strategies. To illustrate the performance of each component, we compared the QL-AO without them separately, as shown in Table 4. Each algorithm runs 20 times on a large-scale instances with 100 jobs and 3 machines.

Table 4. QL-AO with different components.

Algorithm	Description
1	QL-AO without NEH heuristic method for initialization
2	QL-AO without local search strategies
3	QL-AO without QL based strategies selection
4	QL-AO with all components

The boxplots of result are shown in Figure 4. It can be seen that the algorithm without heuristic initialization and without local search strategies easily fall into a poor solution. The QL-based strategies selection can significantly improve AO. Other than that, in order to illustrate the probabilities adjustment process, the probabilities curves (Figure 5) were printed out. It can be seen that the selection probabilities of different update strategies are continuously adjusted to find the optimal value.

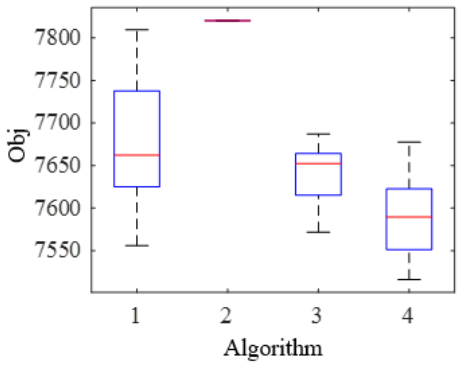

Figure 4. Boxplots of components comparison.

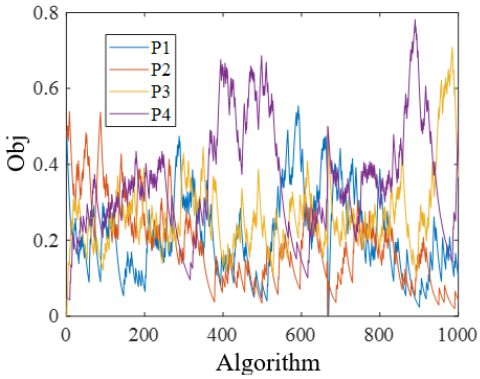

Figure 5. Probabilities adjusting process.

4.4. Algorithm Comparison

To verify the effectiveness of the proposed QL-AO, it is compared with the most classic algorithms, GA and PSO, three novel algorithms, ABC, CS and JAYA, and the basic AO. In addition, Q-learning-based genetic algorithm (QL-GA) and Q-learning-based artificial bee colony algorithm (QL-ABC), proposed by Chen et al. [35] in 2020 and Long et al. [36] in 2022, respectively, are selected as comparison algorithms. In fairness, the competitive algorithms used the same heuristic initialization method as QL-AO. In addition, ABC, QL-ABC, PSO, CS, JAYA and AO used the same encoding method and local search strategies as QL-AO, and GA, QL-GA used the local search strategies of QL-AO as the mutation part.

Table 5 shows the results of QL-AO and its comparative algorithms on Taillard benchmark where Min, Mean, Std., respectively, indicates the best solution, the mean value, the standard deviation of 20 runs. The minimums of the mean and the min for each instance are bolded. In addition, Wilcoxon test was used to show the significance of the difference between QL-AO and the competitive algorithms. The symbol "+" indicates that QL-AO is significantly better than the competitive algorithm, the symbol "−" indicates that QL-AO is significantly worse than the competitive algorithm, and the symbol "≈" indicates that the difference with the competitive algorithm is not significant under the Wilcoxon rank sum test ($\alpha = 0.05$). The symbol "+/≈/−" indicates the number of instances that QL-AO is better, similar or worse than the competitive algorithms under the Wilcoxon test.

Table 5. Experimental results on Taillard benchmark.

Instance	$n \times m$		GA	QL-GA	ABC	QL-ABC	PSO	CS	JAYA	AO	QL-AO
1	20 × 5	Min	1383.40	1380.55	1390.95	1382.09	1381.78	1384.42	**1374.93**	1406.07	1387.86
		Mean	1417.19	1411.31	1405.28	1398.08	1415.45	1402.32	**1401.29**	1422.98	1416.74
		Std.	16.23	16.32	10.43	7.81	18.76	10.80	13.04	15.41	17.57
		Wilcoxon	≈	−	−	−	≈	−	−	≈	
2	20 × 10	Min	**1904.79**	1906.18	1912.58	1910.90	1938.96	1913.57	1910.49	1912.15	1917.31
		Mean	1935.60	1934.50	1943.50	**1933.69**	1971.93	1933.88	1953.81	1944.88	1942.31
		Std.	22.77	21.12	13.26	11.98	22.55	17.78	21.83	22.82	22.41
		Wilcoxon	≈	≈	≈	≈	+	≈	+	≈	
3	20 × 20	Min	2845.92	2844.01	2836.83	2828.02	2866.88	**2824.59**	2851.64	2875.42	2881.18
		Mean	2935.25	2931.72	2879.17	2870.14	2901.16	**2869.03**	2883.96	2926.00	2925.85
		Std.	50.53	39.09	18.86	18.97	22.42	24.15	20.81	36.51	31.86
		Wilcoxon	≈	≈	−	−	−	−	−	≈	
4	50 × 5	Min	3576.51	3528.83	3683.86	3642.64	3544.32	3581.38	3652.11	3494.77	**3488.32**
		Mean	3612.92	3611.97	3733.72	3704.98	3682.18	3657.06	3723.94	3564.16	**3549.49**
		Std.	33.48	33.22	31.21	31.93	50.00	27.97	31.70	34.53	38.09
		Wilcoxon	+	+	+	+	+	+	+	≈	
5	20 × 9	Min	4794.76	4762.25	4845.85	4788.66	4752.50	**4733.64**	4846.58	4793.12	4752.44
		Mean	4855.40	4851.91	4912.50	4871.81	4878.56	4832.86	4923.75	4837.01	**4824.00**
		Std.	33.58	40.51	30.56	35.14	74.37	49.68	35.33	29.09	39.16
		Wilcoxon	+	+	+	+	+	≈	+	≈	
6	50 × 10	Min	7044.37	7033.18	7149.35	7099.14	7026.40	7020.96	7052.65	7003.96	**6963.96**
		Mean	7105.98	7094.31	7195.91	7159.99	7147.39	7114.70	7185.12	7076.00	**7067.55**
		Std.	36.56	34.69	30.66	32.84	53.89	39.90	43.09	33.72	42.41
		Wilcoxon	+	+	+	+	+	+	+	≈	
7	50 × 20	Min	7644.53	7638.80	7811.62	7820.39	7747.91	7693.94	7820.39	7581.37	**7495.38**
		Mean	7710.22	7708.41	7819.69	7820.39	7816.77	7796.97	7820.39	7628.43	**7598.21**
		Std.	32.91	27.53	2.17	0.00	16.21	32.51	0.00	29.79	41.21
		Wilcoxon	+	+	+	+	+	+	+	+	
8	100 × 5	Min	9997.27	9980.79	10,258.11	10,233.13	10,121.03	10,164.82	10,188.47	9920.42	**9885.14**
		Mean	10,079.75	10,076.84	10,259.78	10,256.91	10,252.46	10,223.50	10,255.17	10,003.37	**9957.90**
		Std.	67.14	57.67	0.48	6.44	30.98	29.49	16.18	36.87	48.03
		Wilcoxon	+	+	+	+	+	+	+	+	
9	100 × 10	Min	13,518.95	13,513.64	13,680.83	13,602.60	13,507.90	13,616.91	13,621.38	13,473.78	**13,378.77**
		Mean	13,614.51	13,611.87	13,719.77	13,711.69	13,683.29	13,677.17	13,718.22	13,530.68	**13,497.21**
		Std.	43.95	42.78	27.66	35.88	73.02	33.36	32.92	26.27	44.54
		Wilcoxon	+	+	+	+	+	+	+	+	
10	100 × 20	Min	19,667.65	19,625.42	19,909.16	19,883.45	19,893.75	19,799.03	19,914.51	19,564.39	**19,498.81**
		Mean	19,759.66	19,755.36	19,914.25	19,909.61	19,913.48	19,861.52	19,914.51	19,644.18	**19,588.10**
		Std.	51.08	57.72	1.20	8.54	4.64	28.32	0.00	41.36	60.49
		Wilcoxon	+	+	+	+	+	+	+	+	
11	200 × 20	Min	26,753.76	26,713.04	26,989.87	26,956.33	27,008.19	26,821.13	27,008.19	26,586.15	**26,577.62**
		Mean	26,839.01	26,823.55	27,006.69	26,997.57	27,008.19	26,919.17	27,008.19	26,719.92	**26,652.97**
		Std.	61.16	68.51	4.73	16.39	0.00	43.78	0.00	51.97	45.25
		Wilcoxon	+	+	+	+	+	+	+	+	
12	500 × 20	Min	67,501.67	67,498.71	67,629.04	67,620.43	67,629.04	67,483.90	67,629.04	67,350.62	**67,327.82**
		Mean	67,582.04	67,575.48	67,629.04	67,628.42	67,629.04	67,566.84	67,629.04	67,465.49	**67,444.82**
		Std.	35.89	31.59	0.00	2.06	0.00	31.35	0.00	58.41	55.76
		Wilcoxon	+	+	+	+	+	+	+	≈	
Wilcoxon +/≈/−			9/3/0	9/2/1	9/1/2	9/1/2	10/1/1	8/2/2	10/0/2	5/7/0	

It can be seen from the data in Table 5 that the mean value of QL-AO is always better than that of AO, and it is significant on large-scale cases. This proves the effectiveness of combining QL and AO. GA, QL-GA, ABC, QL-ABC PSO, CS, and JAYA have excellent performance than QL-AO, respectively, on the small-scale instances 1–4. However, they do not do well on other instances. The reason may be that small-scale instances are not hard, and other algorithms such as CS with strong exploration mechanism may be more appropriate. Further, it is observed that QL-AO always obtains best mean value and best min value on the other larger scale instances 5–12, and it is significant under the Wilcoxon

test on instances 7–11. It shows that QL-AO exploitation and exploration abilities have been well balanced by adjusting selection probabilities during the search process compared to other competitors. It can be noted that two advanced algorithms with the involvement of QL, QL-GA and QL-ABC, have certain improvements compared to the basic algorithms GA and ABC, respectively. These are due to the dynamic adjustment of GA's parameters by QL and dynamic control of ABC's search dimensions by QL, respectively. However, under this integration optimization issue they still have certain gaps compared to QL-AO and need further improvement.

Figures 6 and 7 separately show the boxplots of optimal solutions and the convergence curves of algorithms on instances 8–9, respectively. It can be concluded that the solution distribution of QL-AO is significantly better than other algorithms and it converges faster.

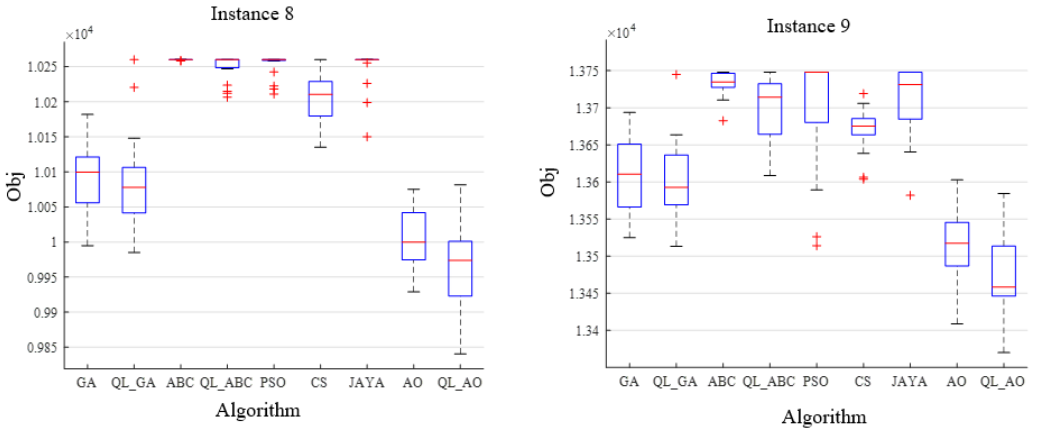

Figure 6. Boxplots of optimal solutions.

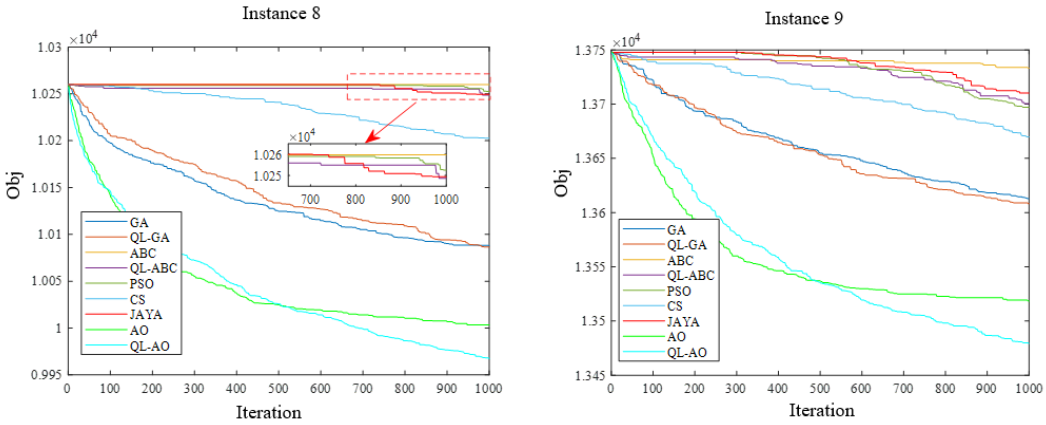

Figure 7. Convergence curves of algorithms.

In order to further verify the performance of QL-AO on more hard instances, we selected seven large-scale instances to test. They are from Ruiz's new hard benchmark. The results of each algorithm are shown in Table 6.

From Table 6, it is obvious that the proposed QL-AO is clearly better and more effective than other comparison algorithms on the hard instances. It can be concluded that the proposed QL-AO is a new efficient algorithm to solve the integrated scheduling of blocking flowshop.

Table 6. Experimental results on Ruiz's hard benchmark.

Instance	$n \times m$		GA	QL-GA	ABC	QL-ABC	PSO	CS	JAYA	AO	QL-AO
13	100 × 20	Min	13,381.89	13,365.69	13,575.12	13,556.27	13,607.59	13,498.32	13,593.09	13,288.05	**13,274.52**
		Mean	13,461.01	13,457.96	13,611.66	13,610.96	13,623.99	13,567.71	13,623.36	13,385.36	**13,363.48**
		Std.	61.51	53.06	18.02	21.90	3.88	23.06	7.12	52.22	58.06
		Wilcoxon	+	+	+	+	+	+	+	+	
14	200 × 20	Min	27,058.88	27,084.77	27,335.85	27,296.28	27,367.67	27,204.34	27,363.98	26,959.71	**26,904.79**
		Mean	27,172.23	27,167.89	27,374.07	27,370.44	27,377.79	27,264.71	27,377.60	27,058.02	**27,013.08**
		Std.	71.05	71.38	12.93	20.34	2.38	36.42	3.21	44.85	49.14
		Wilcoxon	+	+	+	+	+	+	+	+	
15	300 × 20	Min	40,531.17	40,577.87	40,799.60	40,799.01	40,799.60	40,675.93	40,799.60	40,465.17	**40,436.82**
		Mean	40,669.81	40,662.84	40,799.60	40,798.57	40,799.60	40,729.63	40,799.60	40,564.75	**40,515.46**
		Std.	71.38	38.12	0.00	1.13	0.00	27.69	0.00	62.05	41.60
		Wilcoxon	+	+	+	+	+	+	+	+	
16	400 × 20	Min	54,237.72	54,233.34	54,344.92	54,326.63	54,344.92	54,244.88	54,344.92	54,091.23	**54,081.69**
		Mean	54,303.77	54,294.36	54,344.92	54,343.62	54,344.92	54,284.78	54,344.92	54,170.96	**54,164.14**
		Std.	27.75	26.91	0.00	4.09	0.00	20.88	0.00	47.79	52.33
		Wilcoxon	+	+	+	+	+	+	+	≈	
17	500 × 20	Min	68,420.67	68,410.24	68,640.32	68,623.26	68,640.32	68,451.30	68,640.32	68,297.54	**68,282.72**
		Mean	68,562.80	68,531.40	68,640.32	68,638.17	68,640.32	68,544.95	68,640.32	68,418.68	**68,399.91**
		Std.	55.42	38.68	0.00	5.17	0.00	37.85	0.00	55.23	58.95
		Wilcoxon	+	+	+	+	+	+	+	+	
18	600 × 20	Min	81,641.45	81,637.28	81,744.13	81,716.31	81,744.13	81,624.94	81,744.13	81,562.63	**81,504.01**
		Mean	81,713.28	81,711.14	81,744.13	81,740.62	81,744.13	81,663.04	81,744.13	81,627.96	**81,627.42**
		Std.	23.93	17.62	0.00	8.50	0.00	18.70	0.00	42.29	51.35
		Wilcoxon	+	+	+	+	+	+	+	≈	
19	700 × 20	Min	94,930.95	94,925.67	95,010.06	95,003.24	95,010.06	94,830.38	95,010.06	94,853.35	**94,841.62**
		Mean	94,975.91	94,971.35	95,010.06	95,009.48	95,010.06	94,932.49	95,010.06	94,923.47	**94,921.12**
		Std.	20.54	25.09	0.00	1.83	0.00	35.92	0.00	42.72	37.68
		Wilcoxon	+	+	+	+	+	+	+	+	
Wilcoxon +/≈/−			7/0/0	7/0/0	7/0/0	7/0/0	7/0/0	7/0/0	7/0/0	5/2/0	

5. Conclusions

This study investigates the application of a Q-learning-based aquila optimizer (QL-AO) for the integrated optimization problem of blocking flowshop scheduling and preventive maintenance. This QL-AO algorithm can adjust the population update operations during the iteration process adaptively by using Q-learning to update the selection probabilities of four update strategies in the basic aquila optimizer. In addition, an NEH based initialization scheme and five local search methods are employed to further improve the performance of QL-AO. Based on a series of numeric experiments that are carried out on two groups of flowshop scheduling benchmark, this QL-AO algorithm performs significantly better than the other eight comparison algorithms in most test instances. Experimental results also indicate that the proposed Q-learning and local search strategies do well in improving the performance of aquila optimizer for the investigated integrated scheduling problem.

However, there are also some limitations of this study. In terms of modeling, the uncertainty of machine maintenance time, jobs' arrival time and jobs' processing time are not taken into account. And this study is conducted without using actual factory data. In terms of algorithm, this study adopts a weighted approach to deal with multi-objective problems, thus unable to obtain non-dominated optimal solutions. In future research, the following topics will be considered. The first straightforward work is to examine the validity of QL-AO in solving the real-world scenarios considering that the test instances are generated from the general flowshop scheduling benchmark in this paper. The second is the design of a Pareto-based multi-objective AO to obtain a non-dominated set for multi-objective optimization. Moreover, it is also interesting to apply Q-learning to determine how to balance the exploration and exploitation period in the aquila optimizer.

Author Contributions: Writing—original draft preparation, Software, Z.G.; Supervision, H.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research is partially supported by the National Science Foundation of China under Grant Nos. 61973203 and 72271048.

Data Availability Statement: Taillard benchmark: http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html, (accessed on 15 August 2023). Ruiz's hard benchmark: http://soa.iti.es/problem-instances, (accessed on 15 August 2023).

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

References

1. Azimpoor, S.; Taghipour, S.; Farmanesh, B.; Sharifi, M. Joint Planning of Production and Inspection of Parallel Machines with Two-phase of Failure. *Rellab. Eng. Syst. Saf.* **2022**, *217*, 108097. [CrossRef]
2. Ben Ali, M.; Sassi, M.; Gossa, M.; Harrath, Y. Simultaneous scheduling of production and maintenance tasks in the job shop. *Int. J. Prod. Res.* **2011**, *49*, 3891–3918. [CrossRef]
3. Basri, E.I.; Abdul Razak, I.H.; Ab-Samat, H.; Kamaruddin, S. Preventive Maintenance (PM) planning: A review. *J. Qual. Maint. Eng.* **2017**, *23*, 14. [CrossRef]
4. Wang, H.; Yan, Q.; Zhang, S. Integrated scheduling and flexible maintenance in deteriorating multi-state single machine system using a reinforcement learning approach. *Adv. Eng. Inform.* **2021**, *49*, 101339. [CrossRef]
5. Yan, Q.; Wang, H.; Wu, F. Digital twin-enabled dynamic scheduling with preventive maintenance using a double-layer Q-learning algorithm. *Comput. Oper. Res.* **2022**, *144*, 105823. [CrossRef]
6. Liu, L.X.; Shi, L.Y. Automatic Design of Efficient Heuristics for Two-Stage Hybrid Flow Shop Scheduling. *Symmetry* **2022**, *14*, 162. [CrossRef]
7. Merchan, A.F.; Maravelias, C.T. Preprocessing and tightening methods for time-indexed chemical production scheduling models. *Comput. Chem. Eng.* **2016**, *84*, 516–535. [CrossRef]
8. Gong, H.; Tang, L.; Duin, C.W. A two-stage flow shop scheduling problem on a batching machine and a discrete machine with blocking and shared setup times. *Comput. Oper. Res.* **2010**, *37*, 960–969. [CrossRef]
9. Elmi, A.; Topaloglu, S. A scheduling problem in blocking hybrid flow shop robotic cells with multiple robots. *Comput. Oper. Res.* **2013**, *40*, 2543–2555. [CrossRef]
10. Miyata, H.H.; Nagano, M.S. The blocking flow shop scheduling problem: A comprehensive and conceptual review. *Expert. Syst. Appl.* **2019**, *137*, 130–156. [CrossRef]
11. Graham, R.L.; Lawler, E.L.; Lenstra, J.K.; Kan, A.H.G.R. Optimization and approximation in deterministic sequencing and scheduling: A survey. *ADM* **1979**, *5*, 287–326.
12. Hall, N.G.; Sriskandarajah, C. A survey of machine scheduling problems with blocking and no-wait in process. *Oper. Res.* **1996**, *44*, 510–525. [CrossRef]
13. Jiang, J.; An, Y.; Dong, Y.; Hu, J.; Li, Y.; Zhao, Z. Integrated optimization of non-permutation flow shop scheduling and maintenance planning with variable processing speed. *Reliab. Eng. Syst. Saf.* **2023**, *234*, 109143. [CrossRef]
14. Caraffa, V.; Ianes, S.; Bagchi, T.P.; Sriskandarajah, C. Minimizing makespan in a blocking flowshop using genetic algorithms. *Int. J. Prod. Econ.* **2001**, *70*, 101–115. [CrossRef]
15. Liang, J.J.; Pan, Q.K.; Chen, T.J.; Wang, L. Dynamic Multi-swarm Particle Swarm Optimizer for blocking flow shop scheduling. In Proceedings of the IEEE International Conference on Fuzzy Systems, Changsha, China, 23–26 September 2010.
16. Li, M.B.; Xiong, H.; Lei, D.M. An Artificial Bee Colony with Adaptive Competition for the Unrelated Parallel Machine Scheduling Problem with Additional Resources and Maintenance. *Symmetry* **2022**, *14*, 1380. [CrossRef]
17. Liu, H.; Zhang, X.; Zhang, H.; Li, C.; Chen, Z. A reinforcement learning-based hybrid Aquila Optimizer and improved Arithmetic Optimization Algorithm for global optimization. *Expert. Syst. Appl.* **2023**, *224*, 119898. [CrossRef]
18. Ait-Saadi, A.; Meraihi, Y.; Soukane, A.; Ramdane-Cherif, A.; Gabis, A.B. A novel hybrid Chaotic Aquila Optimization algorithm with Simulated Annealing for Unmanned Aerial Vehicles path planning. *Comput. Electr. Eng.* **2022**, *104*, 108461. [CrossRef]
19. Bas, E. Binary Aquila Optimizer for 0–1 knapsack problems. *Eng. Appl. Artif. Intel.* **2023**, *118*, 105592. [CrossRef]
20. Agarwal, N.; Gokilavani, M.; Nagarajan, S.; Saranya, S.; Alsolai, H.; Dhahbi, S.; Abdelaziz, A.S. Intelligent aquila optimization algorithm-based node localization scheme for wireless sensor networks. *CMC-Comput. Mater. Con.* **2023**, *74*, 141–152. [CrossRef]
21. Li, R.; Zhang, X.; Jiang, L.; Yang, Z.; Guo, W. An adaptive heuristic algorithm based on reinforcement learning for ship scheduling optimization problem. *Ocean. Coast. Manage.* **2022**, *230*, 106375. [CrossRef]
22. Mao, J.; Hu, X.L.; Pan, Q.K.; Miao, Z.; Tasgetiren, M.F. An improved discrete artificial bee colony algorithm for the distributed permutation flowshop scheduling problem with preventive maintenance. In Proceedings of the 39th Chinese Control Conference, Shenyang, China, 27–29 July 2020.
23. Cheng, L.; Tang, Q.; Zhang, L.; Meng, K. Mathematical model and enhanced cooperative co-evolutionary algorithm for scheduling energy-efficient manufacturing cell. *J. Clean. Prod.* **2021**, *326*, 129248. [CrossRef]

24. Zhang, Z.; Tang, Q. Integrating preventive maintenance to two-stage assembly flow shop scheduling: MILP model, constructive heuristics and meta-heuristics. *Flex. Serv. Manuf. J.* **2022**, *34*, 156–203. [CrossRef]
25. Sun, L.H.; Ge, C.C.; Zhang, W.; Wang, J.B.; Lu, Y.Y. Permutation flowshop scheduling with simple linear deterioration. *Eng. Optim.* **2019**, *51*, 1281–1300. [CrossRef]
26. Wang, S.; Liu, M. Two-machine flow shop scheduling integrated with preventive maintenance planning. *Int. J. Syst. Sci.* **2016**, *47*, 672–690. [CrossRef]
27. Ruiz, R.; García-Díaz, J.C.; Maroto, C. Considering scheduling and preventive maintenance in the flowshop sequencing problem. *Comput. Oper. Res.* **2007**, *34*, 3314–3330. [CrossRef]
28. Grabowski, J.; Pempera, J. The permutation flow shop problem with blocking. A tabu search approach. *Omega* **2007**, *35*, 302–311. [CrossRef]
29. Abualigah, L.; Yousri, D.; Abd Elaziz, M.; Al-Qaness, M.A.; Gandomi, A.H. Aquila optimizer: A novel meta-heuristic optimization algorithm. *Comput. Ind. Eng.* **2021**, *157*, 107250. [CrossRef]
30. Bean, J.C. Genetic algorithms and random keys for sequencing and optimization. *ORSA J. Comput.* **1994**, *6*, 154–160. [CrossRef]
31. Yang, X.S. Swarm intelligence based algorithms: A critical analysis. *Evol. Intell.* **2014**, *7*, 17–28. [CrossRef]
32. Wineberg, M.; Oppacher, F. The underlying similarity of diversity measures used in evolutionary computation. In Proceedings of the Genetic and Evolutionary Computation Conference, Chicago, IL, USA, 12–16 July 2003.
33. Taillard, E. Benchmarks for basic scheduling problems. *Eur. J. Oper. Res.* **1993**, *64*, 278–285. [CrossRef]
34. Vallada, E.; Ruiz, R.; Framinan, J. New hard benchmark for flowshop scheduling problems minimising makespan. *Eur. J. Oper. Res.* **2017**, *240*, 666–677. [CrossRef]
35. Chen, R.; Yang, B.; Li, S. A Self-Learning Genetic Algorithm based on Reinforcement Learning for Flexible Job-shop Scheduling Problem. *Comput. Ind. Eng.* **2020**, *149*, 106778. [CrossRef]
36. Long, X.; Zhang, J.; Zhou, K. Dynamic Self-Learning Artificial Bee Colony Optimization Algorithm for Flexible Job-Shop Scheduling Problem with Job Insertion. *Processes* **2022**, *10*, 571. [CrossRef]