Article

Deep Reinforcement Learning for Distributed Flow Shop Scheduling with Flexible Maintenance

Qi Yan, Wenbin Wu and Hongfeng Wang *

College of Information Science and Engineering, Northeastern University, Shenyang 110819, China; yanqqz@stumail.neu.edu.cn (Q.Y.); 2070814@stu.neu.edu.cn (W.W.)
* Correspondence: hfwang@mail.neu.edu.cn

Abstract: A common situation arising in flow shops is that the job processing order must be the same on each machine; this is referred to as a permutation flow shop scheduling problem (PFSSP). Although many algorithms have been designed to solve PFSSPs, machine availability is typically ignored. Healthy machine conditions are essential for the production process, which can ensure productivity and quality; thus, machine deteriorating effects and periodic preventive maintenance (PM) activities are considered in this paper. Moreover, distributed production networks, which can manufacture products quickly, are of increasing interest to factories. To this end, this paper investigates an integrated optimization of the distributed PFSSP with flexible PM. With the introduction of machine maintenance constraints in multi-factory production scheduling, the complexity and computation time of solving the problem increases substantially in large-scale arithmetic cases. In order to solve it, a deep Q network-based solution framework is designed with a diminishing greedy rate in this paper. The proposed solution framework is compared to the DQN with fixed greedy rate, in addition to two well-known metaheuristic algorithms, including the genetic algorithm and the iterated greedy algorithm. Numerical studies show that the application of the proposed approach in the studied production-maintenance joint scheduling problem exhibits strong solution performance and generalization abilities. Moreover, a suitable maintenance interval is also obtained, in addition to some managerial insights.

Keywords: permutation flow shop scheduling; distributed manufacturing; machine deterioration; preventive maintenance; deep reinforcement learning

check for updates

Citation: Yan, Q.; Wu, W.; Wang, H. Deep Reinforcement Learning for Distributed Flow Shop Scheduling with Flexible Maintenance. *Machines* **2022**, *10*, 210. https://doi.org/10.3390/machines10030210

Academic Editor: Kai Wang

Received: 16 December 2021
Accepted: 14 March 2022
Published: 16 March 2022

1. Introduction

The permutation flow shop scheduling problem (PFSSP) has been receiving considerable attention by academia and industry as a consequence of its broad applications. Earlier studies have demonstrated it to be an NP-complete problem when the number of machines involved is more than three. Beyond that, various heuristic or meta-heuristic algorithms have been developed [1–8]. However, those studies simply assumed that all scheduling tasks are done in one factory [9]. In the contemporary context of the decentralized and globalized economy, distributed PFSSPs (DPFSSP) in multi-factory production networks are becoming increasingly significant means to increase productivity and achieve lower production costs and higher product quality [10]. In recent years, many models and algorithms have been proposed to solve DPFSSPs [9,11–14], in which two key decision-makings are considered, including allocating jobs to suitable factories, and scheduling operations on machines.

A common assumption in most research of DPFSSPs is that all the machines are continuously available during the whole production process [14]. Nevertheless, machine deterioration is inevitable as their operating times increase, causing an ever-increasing probability of machine failures. Thus, developing a proper maintenance strategy is critical [15]. In contrast to the corrective maintenance (CM) strategy, which has to be carried out immediately after machine failures, two types of preventive maintenance (PM) strategies including time-based maintenance (TBM) and condition-based maintenance (CBM) have become more popular in common environment configurations such as single machine, flow shop and flexible job shop configurations. For TBM, two kinds of assumptions concerning flexible PM strategies from literature are the following: (I) PM must be carried out within a predetermined interval $[u, v]$ whose duration is longer than the PM time [16–18]; (II) the interval between two PMs cannot exceed a maximum allowed continuous processing time [19,20]. For CBM, PM is not limited to a specific duration, but typically depends on the machine's age or multi-state degradation process [21–23]. In this paper, a TBM strategy is integrated into the DPFSSP to minimize the makespan of the whole distributed production system.

A result of the complexity of the integrated optimization of multi-factory production scheduling and PM, the application of exact algorithms has limitations. Limited studies only employed heuristic or meta-heuristic optimization algorithms to solve a large-size instance within a short time. For instance, Chan et al. [24] and Chung et al. [25] studied the distributed flexible job shop scheduling problem with maintenance using improved genetic algorithms, in which the maintenance has to be carried out if the machine's age reaches a given maximum. Lei and Liu [26] proposed an improved artificial bee colony algorithm to solve a distributed unrelated parallel machines problem with flexible PM. Wang et al. [27] investigated a DPFSSP considering event-driven policy and right-shift schedule repair, and developed a fuzzy logic-based hybrid estimation of the distribution algorithm. More recently, Miyata and Nagano [28] proposed a multi-level flexible maintenance strategy in a distributed no-wait flow shop in which sequence-dependent setup times were considered as well. An iterated greedy algorithm with variable search neighborhood was designed for solving small-sized and large-sized instances targeted at achieving a minimal makespan. Similarly, Mao et al. [14] assumed that the same types of machines have the same PM intervals in a DPFSSP and a multi-start iterated greedy algorithm proposed to achieve production-maintenance joint optimization. Jafar-Zanjani et al. [29] developed robust and resilient scheduling approaches in a multi-factory network with periodic maintenance and uncertain disturbances, in which the proposed model in small and medium instances were solved by CPLEX, and large-sized instances were solved by a heuristic method based on the genetic algorithm.

Some reinforcement learning algorithms have been applied to the scheduling field to satisfy requirements of real-time scheduling in actual scenarios. One of them is the Q-learning (QL) algorithm. For instance, Wang and Usher [30] applied QL to address a dispatching rule selection problem on a single machine with different system objectives. After that, Wang et al. [15] applied a scheduling rules-based QL approach to jointly optimize single-machine scheduling and flexible maintenance, in which both deteriorating effects and machine failures were considered. In addition, some researchers combined QL with metaheuristics. For instance, Cheng et al. [31] proposed a multi-objective super-heuristic algorithm based on QL with four heuristic update operators as the action set for mixed scheduling. Lin et al. [32] used multiple heuristic update rules as QL actions to semi-conductor test scheduling problems. Shahmardan et al. [33] used QL to learn the neighborhood deconstruction of SA to solve the problem of truck scheduling. Long et al. [34] focused on the flexible job shop scheduling problem using QL to learn the number of randomly updated dimensions of nectar sources to improve the neighborhood search of the artificial bee colony algorithm. More recently, Wang et al. [35] integrated QL with the well-known artificial bee colony algorithm to efficiently solve distributed three-stage assembly scheduling with maintenance.

However, in the practice of Q-learning, the dimensionality of the state space is usually large for complex large-scale scheduling problems, resulting in Q-tables that are too large for fast convergence. In order to solve it, attempts at neural network-based reinforcement learning have been applied to production scheduling problems [36]; however, maintenance activities were not considered. A deep Q-learning (DQN) approach is employed in this paper, in which the system state is defined by a binary group that consists of the interval between the current time, the next latest start time of PM and the number of available jobs, and the action space is a set of available jobs. In order to evaluate the solving performance of the DQN-based solution framework, several numerical studies are conducted over many instances. It is observed that the DQN-based optimization approach has greater solution potential compared to two metaheuristic approaches.

The remainder of this paper is organized as follows: Section 2 describes the proposed problem in detail; a DQN-based optimization approach is presented in Section 3; numerical studies and discussions are conducted in Section 4; conclusions are provided in Section 5, along with research limitations and future research directions.

2. Problem Description

The application scenario of the studied DPFSSP with PM is described as follows: there are n jobs to be processed on f factories, in which each factory has m machines; each job is available at time zero and its operations number is equal to the machines' numbers in one factory, and all the operations of a job must be processed in the order from machine 1 to machine m according to the precedence constraint; different jobs have the same priority, and there is no priority limit between operations of different jobs; the normal processing time of each operation of each job is known in advance and is slightly different on different available machines; however, it may be extended as a consequence of machines' wear and tear, which is treated as deteriorating processing time in this paper. Inspired by [20], the deteriorated processing time of a job is expressed as a linear function of the machine's age at the start time of processing the job, where the slope is the defined deterioration rate and the intercept is the normal processing time. With increasing machine age, the probability of machine failures increases, thus proactive maintenance strategies become increasingly important; one of them is the time-based PM strategy. Based on the flexible maintenance schedule defined in our previous studies [15,37], i.e., the expected PM schedule $kT(k = 1, 2, \cdots)$ with a fixed interval T can be shifted within flexible time windows, where $kT - \Delta_1$ and $kT + \Delta_2$ are respectively the earliest and latest times at which the machine starts and stops its PM. This paper assumes that each machine is assigned periodic flexible maintenance time windows from the time it is powered on, in which PM times on different machines are identical. However, the occurrence of PM activities consumes an amount of production time; thus, the trade-off between production and maintenance is critical. Hence, the minimal makespan is selected as the optimization objective. Some additional assumptions in this paper are stated as follows: (1) if the assigned machine is being occupied, some jobs have to wait in the buffer, and thus the buffer space is assumed to be sufficient; (2) setup time of machines is ignored and the transfer time between operations from one machine to another machine is negligible; (3) any activities during the process of production and maintenance cannot be interrupted; (4) the proposed PM is perfect, which can restore a deteriorating machine's age to zero.

In a similar manner to the study by Mao et al. [14], the following illustrative case is provided to evaluate the importance of the proposed DPFSSP with PM: there is a set of ten jobs to be assigned to two factories, each of which consists of three machines. The normal processing times of the jobs are shown in Table 1, in which FiMj represents machine j in factory i, and the deteriorating rate is set to 0.1. As for the maintenance activities, T, Δ_1 and Δ_2 are set to 30, 3 and 5, respectively, and the PM time is equal to 4.

Firstly, we refer to the mixed integer linear programming (MILP) model presented by [38] and use the software CPLEX to solve for an optimal solution to the example without considering deteriorating effects and PM activities. As shown in Figure 1, Job 10, Job 1, Job 3, Job 6 and Job 9 are sequentially assigned to Factory 1, and Factory 2 needs to manufacture Job 4, Job 7, Job8, Job5 and Job 2 in sequence. In this allocation model, the maximum completion time for each factory is 83, which also implies an optimal makespan of 83.

Table 1. Normal processing times of the jobs.

	Job 1	Job 2	Job 3	Job 4	Job 5	Job 6	Job 7	Job 8	Job 9	Job 10
F1M1	10	9	12	8	9	17	15	9	13	8
F1M2	11	14	12	11	13	14	7	13	7	14
F1M3	10	13	13	12	6	12	15	18	10	15
F2M1	9	8	11	7	10	15	15	7	13	8
F2M2	10	17	10	11	11	15	7	13	7	15
F2M3	13	12	15	13	7	12	15	18	12	16

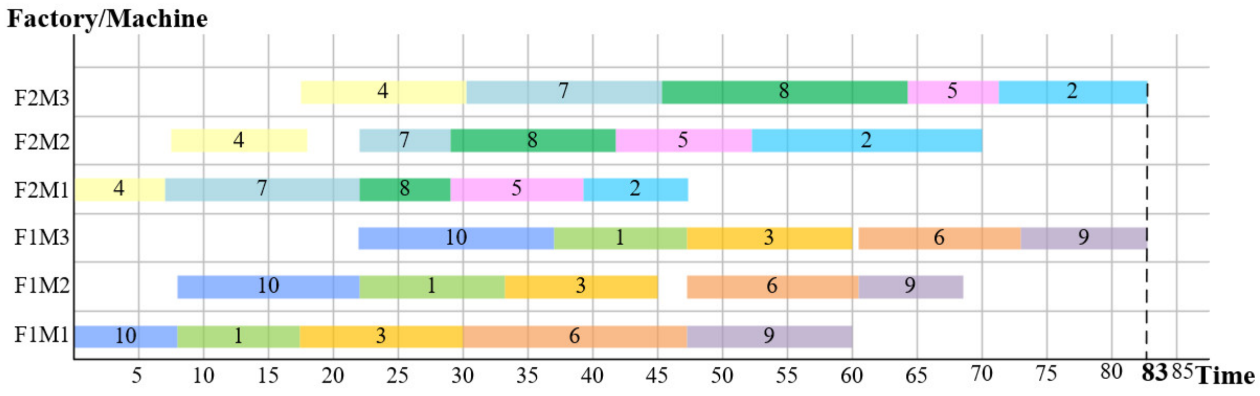

Figure 1. Gantt chart for an optimal solution of the DPFSSP without deteriorating effects and PM.

Next, deteriorating effects and PM are considered in the DPFSSP, however, we do not change the optimal scheduling of the original DPFSSP and only insert the proposed flexible PM constraints into the optimal scheduling considering deteriorating effects. As shown in Figure 2, the processing time of the job is extended as long as a machine's age at the start time of processing the job is not zero. For instance, the deteriorating processing time of the Job 1 of F1M1 can be calculated by $10 + 0.1 \times 8 = 10.8$, in which 10 is the normal processing time, 0.1 denotes the deteriorating rate, and 8 is the machine's age immediately prior to Job 1 of F1M1. It is also observed that the insertion of PM activities disrupts the initially tight schedule. There is a lot of idle time resulting from machine unavailability, which causes a longer makespan. Specifically, take F1M2 as an example to illustrate this phenomenon.

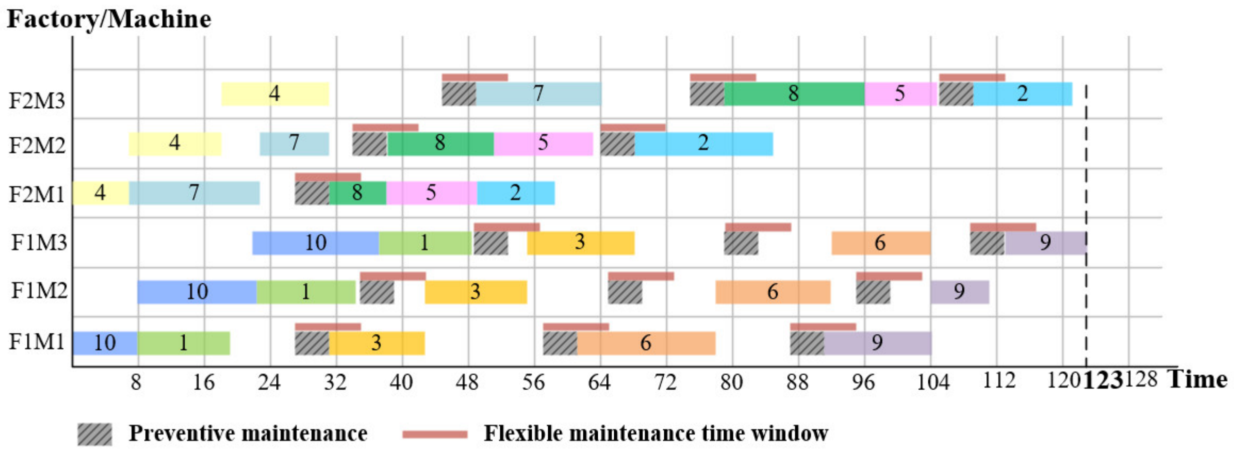

Figure 2. Gantt chart after considering deteriorating effects and PM constraints in the optimal DPFSSP solution.

The first is the initialization of flexible maintenance time windows on Machine 2 of Factory 1. Since the first job, i.e., Job 10 of F1M2, is initially processed immediately after its previous operation on Machine 1 of Factory 1 which is finished at time 8, and T, Δ_1

and Δ_2 are set to 30, 3 and 5, respectively, the set of time windows of F1M2 is noted as $\{[35, 43], [65, 73], [95, 103], \cdots\}$. The next step is to determine if the insertion of each job conflicts with the PM, and update its actual start time and completion time accordingly. Job 10 is finished at time 22 with a normal processing time of 14. Then, the starting time of Job 1 of F1M2 is 22, which depends on the maximum completion time between the last job on Machine 2 of Factory 1 and the last operation on Machine 1 of Factory 1. As a result of the deterioration effect, the deteriorating processing time of Job 1 of F1M2 is equal to 12.4 and its completion time is 34.4. Obviously, processing the first two jobs does not conflict with the first flexible time window, i.e., [35, 43]. However, the next Job 3 cannot be processed immediately after Job 1, since the insertion of Job 3 would make the completion time greater than the difference between the upper boundary of the maintenance time window and the PM time, i.e., it conflicts with the time window, and more importantly, the previous operation to Job 3 has not yet been executed. Therefore, after the processing of Job 1 of F1M2, the machine has 0.6 idle time and PM is performed at the lower bound of the time window, i.e., at time 35, and is finished at time 39. The machine then remains idle until time 43; the first operation of Job 3 is finished on Machine 1 of Factory 1 and the second operation of Job 3 is started on Machine 2 of Factory 1. Similarly, the scheduling of subsequent jobs of F1M2 generates more idle time as a result of the maintenance time window constraints and the completion time constraints of the corresponding previous operations. The same rule is applied to all the machines and a makespan of 123 is obtained.

Considering that the above MILP model applied to the original problem has become difficult to derive an optimal (even a feasible) solution for in a reasonable time using CPLEX as the problem scales up, solving the integrated optimization of the flexible PM and the DPFSSP with deteriorating effects using traditional mathematical programming method is even harder. To this end, a deep reinforcement learning approach is developed in the next section to address this issue. Based on this model-free learning approach, integrated optimization results can be obtained in shorter times, and the better one in limited learning episodes is provided in Figure 3. The integrated optimization result consists of a job sequence {Job 10, Job 2, Job 1, Job 7, Job 9} with a maximum completion time of 95.9 in Factory 1, and a job sequence {Job 4, Job 3, Job 8, Job 5, Job 6} with a maximum completion time of 95 in Factory 2. Due to the flexibility of periodic maintenance times, there is less idle time in addition to fewer executions of PMs. The makespan is equal to 95.9, which implies a decrease in the solution in Figure 2 of 22.03%. This also reflects the necessity of the studied joint optimization problem in addition to the developed flexible maintenance strategy.

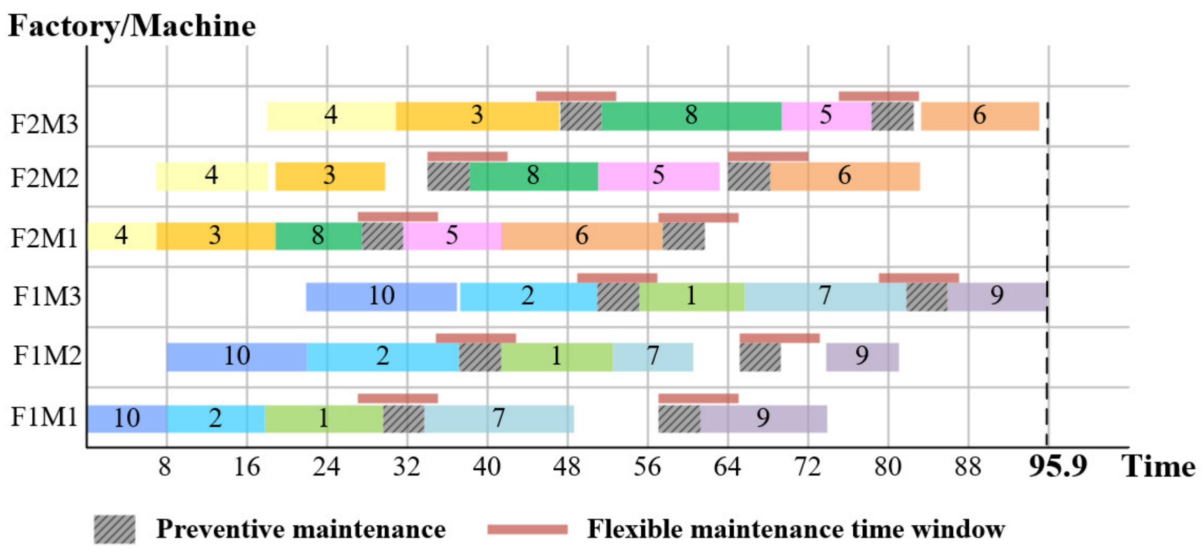

Figure 3. Gantt chart for an optimal solution of the integrated scheduling of the DPFSSP and the PM.

3. Solution Approach Design

3.1. Background of General Q-Learning and Deep Q Networks

Among the various reinforcement learning (RL) based algorithms, Q-learning is a typical model-free RL algorithm which directly interacts with the environment using a trial-and-error approach. In order to find a balance between exploration (of uncharted territory) and exploitation (of current knowledge), ϵ-greedy method is commonly used. Specifically, a random variable $\tau \in (0, 1)$ is generated to compare with the predefined value ϵ; if $\tau < \epsilon$, then the action corresponding to the best Q-value will be selected, otherwise an action a will be performed randomly from an action set at state s. The detailed procedure of the general QL algorithm is shown as Algorithm 1. Each episode is treated as one learning process. In each learning process, an action a is selected by the agent located in state s beforehand and then a new state s' and immediate reward are received, followed by updating the Q-value for the state-action pair (s, a). Through continuous learning, better Q-values will be obtained along with an optimal action-selection policy for the agent [15].

Algorithm 1 General Q-Learning Algorithm

```
Initialize Q-values $Q(s,a)$ arbitrarily for all state-action pairs
Repeat (for each episode) :
Initialize a state $s$
  Repeat (for each episode step) :
    Choose an action $a$ from the state $s$ using $\epsilon$-greedy policy derived from $Q$
    Take action $a$, and observe the new state $s'$ and reward $r(s,a)$
    Update Q-value, $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r(s,a) + \gamma \cdot max_{a'} Q(s',a')]$
    ($\alpha$: learning rate; $\gamma$: discount factor)
    $s \leftarrow s'$;
  until $s$ is terminated
```

As the size of the problem increases, a greater number of states and actions need to be stored, i.e., the Q-table becomes increasingly large, making it difficult to obtain a convergent result in a limited learning time. Fortunately, the introduction of a deep Q network (DQN) can solve this problem by considering states as inputs to the neural network [39,40], and generating Q function values of each state-action pair via analysis of the neural network without taking up space for storage. This approach can deal with complex decision-makings involving large and continuous state spaces. Specifically, two main improvements are included in the DQN, one of which is to establish an experience replay memory D with the capacity of $N\_$ to store the transition $(\phi_t, a_t, r_t, \phi_{t+1})$ at each time-step t. The parameter updating depends on the minibatch of transitions randomly selected from the replay memory, which disrupts the correlation between experiences, and makes the neural network update more efficiently. An old experience is not replaced by a new one until the capacity of the replay memory reaches $N\_$. Another advancement is the use of the target action-value function $\hat{Q}$, which can update parameter θ by calculating the target values and loss function. In addition, $\hat{Q}$ is reset by the action-value function Q every C steps, which can improve the stability of the algorithm. The detailed procedure of the DQN is shown as Algorithm 2 [36].

To the best of our knowledge, little research has been undertaken thus far to apply this DQN approach to the integrated optimization of production scheduling and preventive maintenance. To this end, this paper employs the DQN optimization approach to solve the DPFSSP considering deteriorating effects and flexible maintenance activities. The detailed definitions of states, actions and rewards are provided in Section 3.2.

Algorithm 2 Deep Q-Learning Algorithm with Experience Replay

```
Initialize replay memory $D$ to capacity $N$_
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1: M_ do
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1: T_ do
    With probability $\varepsilon$ select a random action $a_t$
    otherwise select $a_t = argmax_a Q(\phi(s_t), a; \theta)$
    Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from $D$
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network
parameters $\theta$
    Every $C$ steps reset $\hat{Q} = Q$
  End For
End for
```

3.2. Definition of Key Elements

In the studied DPFSSP with flexible PM, each machine is faced with the judgement of whether production and maintenance are in conflict with each other. In order to portray this feature, the array formed by the difference between the current completion time of the job on each machine and the upper bound of the upcoming flexible maintenance time window is treated as the system state. In order to avoid the input of DQN from varying within a wide range, we map the above differences to a number of small intervals. In this difference-based definition approach, it is possible to traverse the defined states several times in a single learning session due to the periodicity of maintenance, which can improve the learning efficiency of DQN.

As a result of the property of the permutation flow shop, assigning a job to a factory is an action in this paper. This means that the capacity of the action space is the product of the number of factories and the number of jobs. Compared to the case where scheduling rules are used as actions [15,36,37], our solution space can be more diverse, and the combination of Q-learning and neural networks can guarantee the solving speed. Moreover, the action space is constantly changing, as it can only be selected from the actions corresponding to jobs that have not yet been assigned. Once a job is assigned to a factory as an action is selected, the actions for that job assigned to other factories are removed from the action space available for this learning round.

Since the objective of the proposed problem is to minimize the makespan of all the factories, it is critical to maintain a balance of maximum completion times across factories in the selection of actions. To this end, variations in the variance of the maximum completion time for each factory before and after the execution of the action are used to develop different immediate rewards. If the variance becomes smaller, i.e., the loading capacity of all the factories is more similar, a positive immediate reward is obtained, which is calculated by a countdown of the current system maximum completion time. This implies that a smaller production cycle corresponds to a larger positive reward. Conversely, a larger variance reflects that the chosen action does not balance the capabilities of all the factories, and a negative reward should be received by calculating the difference between the maximum completion time before and after the action is executed. Additionally, the final reward can be received after a learning process, reflecting the difference between the makespan obj^* in the current learning episode and the historically optimal makespan

obj^{min}. A smaller difference means a bigger final reward. The detailed procedure to achieve a round of interaction between the agent and the environment is provided in Algorithm 3.

Algorithm 3 Interaction Process between the Agent and the Environment at Each Decision Point t

```
Reserve the completion time list $[C_{11}(t-1), \cdots, C_{1m}(t-1), C_{21}(t-1), \cdots, C_{2m}(t-1), \cdots]$ immediately prior to $a_t$ and corresponding maximum completion time list $[maxC_{1m}(t-1), maxC_{2m}(t-1), \cdots]$
Update the completion time list $[C_{11}(t), \cdots, C_{1m}(t), C_{21}(t), \cdots, C_{2m}(t), \cdots]$ mediately after $a_t$ and corresponding maximum completion time list $\left[maxC_{1m}(t), maxC_{2m}(t), \cdots, maxC_{fm}(t)\right]$
Find the latest maintenance start time in upcoming time windows $[T_{11}(t), \cdots, T_{1m}(t), T_{21}(t), \cdots, T_{2m}(t), \cdots]$
Calculate the differences $[T_{11}(t) - C_{11}(t), \cdots, T_{1m}(t) - C_{1m}(t), T_{21}(t) - C_{21}(t), \cdots, T_{2m}(t) - C_{2m}(t), \cdots]$
Update the system state list $[s_{11}(t), \cdots, s_{1m}(t), s_{21}(t), \cdots, s_{2m}(t), \cdots]$ based on the above difference list
Remove all the actions that relate to the assigned job in $a_t$ from the available action space
**If** the system state is terminated, i.e., there are no more jobs that can be assigned to any factory, **then**
  calculate the final reward, i.e., $\left(obj^{min} - obj^*\right)/obj^{min}$
**Else**
  Calculate the variance of $[maxC_{1m}(t-1), \cdots]$ and $[maxC_{1m}(t), \cdots]$ as $\vartheta_1$ and $\vartheta_2$, respectively
  **If** $\vartheta_1 \leq \vartheta_2$ **then**
    receive a negative immediate reward, i.e., $max[maxC_{1m}(t-1), \cdots] - max[maxC_{1m}(t), \cdots]$
  **Else**
    a positive immediate reward is obtained, i.e., $1/max[maxC_{1m}(t-1), \cdots]$
  **End If**
**End If**
```

3.3. Overall Algorithm Framework

The training method is based on the framework of DQN which consists of one input layer and one output layer. The numbers of nodes in input and output layers are equal to the numbers of states and available actions, respectively. In the training process, the decision point t is defined as every time an action a_t is about to be selected, followed by the execution of Algorithm 3. However, this algorithm does not include the scenario where $t = 1$, which means that t is at least greater than or equal to 2. This is a result of the first action being selected without a prior sequence of jobs for comparison and the initial state of the system is still not initialized, for which we design Algorithm 4. The selection of the first action is crucial to ensure the quality of the solution from a certain point of view. Completely random selection, although it guarantees the diversity of solutions, does not consider the propensity to select the first action in the historical optimal solution. In our algorithm, if the makespan learned in a certain round is better than the minimum of historical learning, the first action of this round will be inherited for the next generation of learning; otherwise, the greedy strategy of random exploration is used again. The procedure for initializing the flexible maintenance time window list for each machine is shown in Algorithm 5.

Algorithm 4 Selection of the First Action

```
Input (Optional): The Job List which includes the first job of each factory in the previous solution
Initialize the list of machines in all the factories, i.e., $ML = ['F1M1', \cdots, 'F2M1', \cdots]$
For machine in ML do
  If the machine index is equal to 1 then
    let the starting time of the machine is 0 and initialize the time window list using Algorithm 5
    If the input is empty, i.e., the previous solution is worse than the historical optimal solution, then
      select the first job randomly from the optional job list for the machine
    Else
      select the job of the corresponding machine in the list of entered Job List
    End If
  Else
    Update the current machine and machine index
    Calculate the starting time of the machine, i.e., the completion time of the previous machine
    Initialize the time window list using Algorithm 5
    Identify and reserve the selected operation which depends on the last operation of the previous machine
  End If
  Update the available action space, remaining operation number
  Update the lists of the starting time, processing time, machine's age, completion time
  Initialize the system state referring to the difference calculation in Algorithm 3
  Remove the selected operation from job lists for all the factories
End For
Reserve the maximum completion time for all the factories
Output: System state features, available action space
```

Algorithm 5 Initialization of Flexible Maintenance Time Windows

```
timeWindowList = [ [ ] for machine in ML ]
Input: Machine type (e.g., $'F1M1'$), machine's starting time $st$
For index = 1: N + 2 do
  Append $[st + index * T - \Delta_1,\ st + index * T + \Delta_2]$ in the corresponding list in the timeWindowList
End For
Output: timeWindowList
```

Regarding the selection of subsequent actions based on the observed state features, an improved version of the ϵ-greedy policy presented in Algorithm 1 is implemented. Specifically, an initial period of learning experience is explored using a completely random strategy, followed immediately by a linear reduction in the greedy rate to ensure that the subsequent learning process is more based on prior experience rather than being dominated by randomness. Due to the setting of flexible maintenance time windows and priority constraints between operations, the selected action may not be performed immediately. Specifically, if the direct execution of an action conflicts with the flexible maintenance constraint or the completion time of the last operation, the implementation of the action will be postponed. The detailed judgement and coordination of the production actions and the implementation of PM are given in Algorithm 6. The overall framework is presented in Algorithm 7.

Algorithm 6 Judgement of the Start Time of Actions

```
Input: Selected action
Decode the selected action, i.e., which job is to be assigned to which factory
For machineIndex = 1 : $M$ do
  If the machine index is equal to 1 then
    the earliest start time of the action is initialized as the completion time of the last job on this machine
  Else
    the earliest start time depends on the maximum completion time of the previous job as well as operation
  End If
  Update the available action space, remaining operation number
  Calculate the actual processing time and remove the selected operation from job lists for all the factories
  If the completion time of the previous job does not conflict with the adjacent time window and the insertion of the action still does not conflict with the time window constraint then
    the starting time of the action is equal to the earliest start time of the action
  Else if the PM is performed immediately after the previous job then
    the starting time of the action is equal to the maximum value of the action's earliest start time and the completion time of PM
  Else if the completion time of the previous job does not conflict with the adjacent time window and the insertion of the action still does not conflict with the time window constraint then
    the machine keeps idle until the lower bound of the time window and PM is performed
    Calculate the starting time of the action by evaluating the maximum value of the action's earliest start time and the completion time of PM
  End If
  Update the lists of the starting time, processing time, machine's age, completion time
  Update the system state referring to the difference calculation in Algorithm 3
End For
Reserve the maximum completion time for all the factories
Output: Start time of the selected action
```

Algorithm 7 DQN-Based Solution Framework

```
Input: Learning number, learning rate $\alpha$, greedy rate $\epsilon$ and its rate of change, capacity $N\_$ of replay memory, minibatch size, discount factor $\gamma$, iteration interval $C$ of updating the target network $\hat{Q}$
For episode = 1: Learning number do
  Select the first action using Algorithm 4 to initialize the system state features
  While True:
    Choose an action based on the observed state features using Algorithm 2
    Judge the actual start time of the chosen action using Algorithm 6
    Update new state features and calculate immediate reward using Algorithm 3
    Store the transition process in $D$ using Algorithm 2
    If there are no more jobs that can be assigned to any factory then
      calculate the final reward using Algorithm 3
      Break
    End If
  End While
End For
Output: Algorithm runtime, learning curve, detailed optimal scheduling result
```

4. Numerical Experiments

In this section, parameter settings are provided at first, followed by the description of several algorithm competitors and the analysis of comparative experiments. Last but not least, different maintenance cycle settings are analyzed to derive a better makespan. All algorithms are coded in Python 3.6 and run on an Intel(R) Core(TM) i7-8700 CPU (3.20 GHz/16.00 GB RAM) PC.

4.1. Parameter Settings

To the best of our knowledge, there is no existing instance of the proposed problem, thus some benchmarks are generated based on the following parameters in Table 2, where the normal processing time of a job on each machine follows a uniform distribution from 5 to 20, which is slightly different for different factories. We refer to the research of Mao et al. [14] to set the variation range of maintenance interval T from 50 to 150. Unlike them, this paper assumes that T is the same for each machine and wants to explore a T that makes the makespan optimal using simulation experiments. The time window parameters and the length of maintenance are fixed as 3, 5 and 4, respectively, based on our previous research [37]. In addition, the parameters related to the deep reinforcement learning are presented in Table 3.

Table 2. Problem-related parameters.

Number of jobs	{20, 40, 60, 80, 100}
Number of machines	{3, 5}
Number of factories	{2, 3, 4}
Processing time of a job	[5, 20]
Maintenance cycle	{50, 60, 70, 80, 90, 100 110, 120, 130, 140, 150}

Table 3. Learning-related parameters.

Parameters	**Values**
Learning number	5000
Replay memory size	2000
Batch size of samples to perform gradient descent	128
Greedy rate	Linearly decreases from 0.5 to 0.1
Learning rate	0.1
Discount factor	0.9
Iteration interval of updating the target network	300

4.2. Performance Evaluation of the Developed Algorithm

In this section, the maintenance cycle is fixed as 50, and 30 scenarios in terms of different numbers of factories, machines and jobs are designed to evaluate the performance of the proposed DQN with diminishing greedy rate (DQND) by comparing with three optimization algorithms, including state-of-the-art genetic algorithm (GA) [41], iterated greedy algorithm (IGA) [14] and the DQN with fixed greedy rate of 0.1 (DQNF). The termination condition of all the algorithms is a maximum elapsed CPU time which is estimated by the formula $t = C\_ \times m \times n$ [14], in which $C\_$ is set to 200 for the comparison experiments. The detailed experimental comparison results are shown in Table 4, in which the symbols 'mean' and 'std', respectively, denote the means and standard values of 20 trials for each algorithm. In order to show these results more clearly, a one-tailed t-test with 38 degrees of freedom at a 0.05 level of significance is employed to conduct statistical analysis. The comparative results are shown as '+', '$-$' or '~', respectively, when the proposed DQND is significantly better than, significantly worse than, or statistically equivalent to its algorithm competitors. Obviously, the performance of proposed DQND is better than the others within limited time resources in almost all the scenarios, particularly in complicated scenarios. Besides, the advantage of distributed manufacturing is also confirmed in experiments, i.e., the greater the number of factories, the smaller the makespan for the same number of jobs and machines set up.

Some interesting managerial insights are also obtained by adjusting different T under a specific environment configuration of $F = 2$, $M = 3$ and $J = 40$. As shown in Table 5, the objective function does not exactly show a monotonic trend of variation with increasing T, but there is still a gradual deterioration process, which implies that too large a T results in an exacerbated amplification of the deterioration effect of the jobs within a maintenance cycle. By comparing all the extreme value points, it is not difficult to find the optimal maintenance cycle.

Table 4. Comparison of four approaches to the makespan objective when $T = 50$, $\delta_1 = 3$ and $\delta_2 = 5$.

F	*M*	*J*	DQND		DQNF			GA			IGA		
			Mean	**Std**	**Mean**	**Std**	***t*-Test**	**Mean**	**Std**	***t*-Test**	**Mean**	**Std**	***t*-Test**
		20	178.00	2.19	183.12	1.20	+	177.20	2.25	~	175.11	1.56	–
		40	322.15	1.75	340.52	2.56	+	324.86	3.86	+	321.89	2.49	~
2	3	60	495.63	2.01	533.17	1.79	+	501.89	4.79	+	498.24	3.46	+
		80	642.40	2.98	679.30	3.05	+	658.45	4.35	+	650.75	2.89	+
		100	808.28	3.48	867.95	3.76	+	820.14	5.80	+	812.37	4.01	+
		20	221.29	1.76	235.24	0.99	+	219.54	1.42	–	218.46	1.68	–
		40	380.65	1.45	394.66	1.49	+	383.27	2.43	+	384.12	1.77	+
2	5	60	539.78	1.89	578.20	2.64	+	548.31	2.87	+	545.33	2.20	+
		80	707.51	2.18	746.79	1.86	+	715.24	3.58	+	709.64	3.87	+
		100	891.45	2.45	911.56	2.51	+	901.70	4.76	+	896.34	4.06	+
		20	131.02	0.86	145.28	2.01	+	129.11	1.10	–	130.97	1.92	~
		40	234.64	1.64	252.69	1.94	+	233.98	2.49	~	235.88	2.04	+
3	3	60	344.48	1.82	356.17	2.31	+	350.86	3.47	+	349.18	3.28	+
		80	455.89	2.49	470.92	1.68	+	460.75	2.54	+	459.66	4.08	+
		100	554.10	3.08	581.34	1.99	+	565.34	4.97	+	560.02	3.67	+
		20	168.02	1.54	188.02	1.76	+	167.20	1.16	–	166.16	1.01	–
		40	281.72	1.25	294.32	0.86	+	283.56	2.89	+	282.14	2.04	+
3	5	60	390.15	1.99	410.59	2.14	+	395.68	4.06	+	393.54	3.47	+
		80	495.17	2.51	512.62	3.12	+	506.86	3.85	+	500.02	2.88	+
		100	601.44	1.67	624.25	2.68	+	606.79	5.10	+	604.29	3.79	+
		20	100.12	0.76	120.96	1.58	+	99.59	1.29	–	99.76	0.99	~
		40	194.06	1.05	230.42	2.16	+	196.13	2.16	+	194.32	1.83	+
4	3	60	278.46	1.69	291.68	1.76	+	283.59	3.47	+	280.49	2.56	+
		80	360.17	2.54	379.69	1.53	+	369.96	4.63	+	365.56	3.29	+
		100	437.69	2.34	455.16	2.18	+	442.77	3.57	+	439.76	4.25	+
		20	131.35	1.86	152.30	2.34	+	131.59	2.44	+	130.45	1.85	–
		40	225.16	2.04	248.36	1.58	+	233.75	3.86	+	229.37	2.88	+
4	5	60	315.33	1.66	330.27	1.36	+	318.62	2.76	+	317.60	3.49	+
		80	410.89	2.38	440.86	2.86	+	415.63	4.35	+	415.31	2.95	+
		100	475.51	3.06	501.24	2.76	+	481.21	3.86	+	479.62	4.32	+

Table 5. Comparison of four approaches to the makespan objective when $F = 2$, $M = 3$ and $J = 40$.

T	DQND		DQNF		GA		IGA	
	Mean	**Std**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**
50	322.15	1.75	340.52	2.56	324.86	3.86	321.89	2.49
60	332.71	1.46	348.85	1.95	334.22	2.89	331.89	2.43
70	331.25	2.03	341.55	2.58	332.17	4.21	332.12	1.68
80	335.42	1.45	349.79	1.64	334.69	2.89	336.88	3.49
90	342.56	1.67	356.34	1.35	345.86	2.64	344.64	2.34
100	346.28	1.89	359.60	2.57	346.98	1.97	345.66	2.69
110	361.29	2.56	386.99	2.68	365.47	3.44	363.20	3.64
120	362.75	2.35	385.42	3.04	363.70	1.66	364.18	2.34
130	361.92	1.75	386.48	2.67	362.86	2.76	362.56	3.48
140	366.46	2.41	397.12	1.95	369.79	3.45	365.79	2.99
150	371.11	2.68	402.31	1.86	377.64	2.86	374.55	3.47

5. Conclusions

Distributed manufacturing has attracted a lot of attention from scholars and practitioners in recent years; however, it is rarely studied in conjunction with preventive maintenance. This paper investigated a joint production-maintenance problem in the context of a distributed interchange flow shop scenario. A DQN-based solution framework was applied to minimize the makespan of the proposed problem. The performance of the developed solution framework was validated by comparing it against three other optimization approaches within the same solving time. Some managerial implications regarding the determination of the maintenance interval were also provided for practitioners.

In the near future, we will continue focusing on distributed manufacturing, considering deterioration effects and preventive maintenance. Firstly, other distributed manufacturing scenarios can be addressed, for instance, distributed flexible job shops. Secondly, some dynamic factors, such as new job arrivals and stochastic failures can be considered. Last but not least, single-objective optimization should be extended to multi-objective, and efficient multi-objective optimization algorithms should be studied in depth.

Author Contributions: Software, W.W.; validation, W.W.; formal analysis, Q.Y.; investigation, W.W.; writing—original draft preparation, Q.Y.; writing—review and editing, Q.Y.; supervision, H.W.; funding acquisition, H.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported in part by the National Key Research and Development Program of China under Grant no. 2021YFF0901300, and the National Natural Science Foundation of China under Grant no. 62173076.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The authors confirm that the data supporting the findings of this study are available within the article.

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

References

1. Framinan, J.M.; Gupta, J.N.; Leisten, R. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *Oper. Res. Soc.* **2004**, *55*, 1243–1255. [CrossRef]
2. Zobolas, G.; Tarantilis, C.D.; Ioannou, G.J.C. Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. *Comput. Oper. Res.* **2009**, *36*, 1249–1267. [CrossRef]
3. Yenisey, M.M.; Yagmahan, B.J.O. Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends. *Omega* **2014**, *45*, 119–135. [CrossRef]
4. Wu, X.; Che, A.J.O. Energy-efficient no-wait permutation flow shop scheduling by adaptive multi-objective variable neighborhood search. *Omega* **2020**, *94*, 102117. [CrossRef]
5. Jiang, E.D.; Wang, L. An improved multi-objective evolutionary algorithm based on decomposition for energy-efficient permutation flow shop scheduling problem with sequence-dependent setup time. *Int. J. Prod. Res.* **2019**, *57*, 1756–1771. [CrossRef]
6. Mishra, A.; Shrivastava, D.J.C.; Engineering, I. A TLBO and a Jaya heuristics for permutation flow shop scheduling to minimize the sum of inventory holding and batch delay costs. *Comput. Ind. Eng.* **2018**, *124*, 509–522. [CrossRef]
7. Fu, Y.; Wang, H.; Tian, G.; Li, Z.; Hu, H. Two-agent stochastic flow shop deteriorating scheduling via a hybrid multi-objective evolutionary algorithm. *J. Intell. Manuf.* **2019**, *30*, 2257–2272. [CrossRef]
8. Han, Y.; Li, J.; Sang, H.; Liu, Y.; Gao, K.; Pan, Q. Discrete evolutionary multi-objective optimization for energy-efficient blocking flow shop scheduling with setup time. *Appl. Soft Comput.* **2020**, *93*, 106343. [CrossRef]
9. Wang, S.-Y.; Wang, L.; Liu, M.; Xu, Y. An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem. *Int. J. Prod. Econ.* **2013**, *145*, 387–396. [CrossRef]
10. Fu, Y.; Hou, Y.; Wang, Z.; Wu, X.; Gao, K.; Wang, L. Distributed scheduling problems in intelligent manufacturing systems. *Tsinghua Sci. Technol.* **2021**, *26*, 625–645. [CrossRef]
11. Wang, J.J.; Wang, L. A knowledge-based cooperative algorithm for energy-efficient scheduling of distributed flow-shop. *IEEE Trans. Syst. Man Cybern. Syst.* **2018**, *50*, 1805–1819. [CrossRef]
12. Wang, G.; Gao, L.; Li, X.; Li, P.; Tasgetiren, M.F. Energy-efficient distributed permutation flow shop scheduling problem using a multi-objective whale swarm algorithm. *Swarm Evol. Comput.* **2020**, *57*, 100716. [CrossRef]
13. Deng, J.; Wang, L. A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem. *Swarm Evol. Comput.* **2017**, *32*, 121–131. [CrossRef]

14. Mao, J.-Y.; Pan, Q.-K.; Miao, Z.-H.; Gao, L. An effective multi-start iterated greedy algorithm to minimize makespan for the distributed permutation flowshop scheduling problem with preventive maintenance. *Expert Syst. Appl.* **2021**, *169*, 114495. [CrossRef]
15. 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]
16. Zhao, Z.; Shen, L.; Yang, C.; Wu, W.; Zhang, M.; Huang, G.Q. IoT and digital twin enabled smart tracking for safety management. *Comput. Oper. Res.* **2021**, *128*, 105183. [CrossRef]
17. Chen, J.-S. Scheduling of nonresumable jobs and flexible maintenance activities on a single machine to minimize makespan. *Eur. J. Oper. Res.* **2008**, *190*, 90–102. [CrossRef]
18. Yang, S.-L.; Ma, Y.; Xu, D.-L.; Yang, J.-B. Minimizing total completion time on a single machine with a flexible maintenance activity. *Comput. Oper. Res.* **2011**, *38*, 755–770. [CrossRef]
19. Mosheiov, G.; Sarig, A. Scheduling a maintenance activity to minimize total weighted completion-timeComput. *Math. Appl.* **2009**, *57*, 619–623.
20. Wang, T.; Baldacci, R.; Lim, A.; Hu, Q. A branch-and-price algorithm for scheduling of deteriorating jobs and flexible periodic maintenance on a single machine. *Eur. J. Oper. Res.* **2018**, *271*, 826–838. [CrossRef]
21. Zandieh, M.; Khatami, A.; Rahmati, S.H.A. Flexible job shop scheduling under condition-based maintenance: Improved version of imperialist competitive algorithm. *Appl. Soft Comput.* **2017**, *58*, 449–464. [CrossRef]
22. Rahmati, S.H.A.; Ahmadi, A.; Govindan, K. A novel integrated condition-based maintenance and stochastic flexible job shop scheduling problem: Simulation-based optimization approach. *Ann. Oper. Res.* **2018**, *269*, 583–621. [CrossRef]
23. Ghaleb, M.; Taghipour, S.; Sharifi, M.; Zolfagharinia, H. Integrated production and maintenance scheduling for a single degrading machine with deterioration-based failures. *Comput. Ind. Eng.* **2020**, *143*, 106432. [CrossRef]
24. Chan, F.T.S.; Chung, S.H.; Chan, L.Y.; Finke, G.; Tiwari, M.K. Solving distributed FMS scheduling problems subject to maintenance: Genetic algorithms approach. *Robot. Comput. Manuf.* **2006**, *22*, 493–504. [CrossRef]
25. Chung, S.H.; Chan, F.T.S.; Chan, H.K. A modified genetic algorithm approach for scheduling of perfect maintenance in distributed production scheduling. *Eng. Appl. Artif. Intell.* **2009**, *22*, 1005–1014. [CrossRef]
26. Lei, D.; Liu, M. An artificial bee colony with division for distributed unrelated parallel machine scheduling with preventive maintenance. *Comput. Ind. Eng.* **2020**, *141*, 106320. [CrossRef]
27. Wang, K.; Huang, Y.; Qin, H. A fuzzy logic-based hybrid estimation of distribution algorithm for distributed permutation flowshop scheduling problems under machine breakdown. *J. Oper. Res. Soc.* **2016**, *67*, 68–82. [CrossRef]
28. Miyata, H.H.; Nagano, M.S. Optimizing distributed no-wait flow shop scheduling problem with setup times and maintenance operations via iterated greedy algorithm. *J. Manuf. Syst.* **2021**, *61*, 592–612. [CrossRef]
29. Jafar-Zanjani, H.; Zandieh, M.; Sharifi, M. Robust and resilient joint periodic maintenance planning and scheduling in a multi-factory network under uncertainty: A case study. *Reliab. Eng. Syst. Saf.* **2022**, *217*, 108113. [CrossRef]
30. Wang, Y.; Usher, J.M. Application of reinforcement learning for agent-based production scheduling. *Eng. Appl. Artif. Intell.* **2005**, *18*, 73–82. [CrossRef]
31. Cheng, L.; Tang, Q.; Zhang, L.; Zhang, Z. Multi-objective Q-learning-based hyper-heuristic with Bi-criteria selection for energy-aware mixed shop scheduling. *Swarm Evol. Comput.* **2022**, *69*, 100985. [CrossRef]
32. Lin, J.; Li, Y.-Y.; Song, H.-B. Semiconductor final testing scheduling using Q-learning based hyper-heuristic. *Expert Syst. Appl.* **2022**, *187*, 115978. [CrossRef]
33. Shahmardan, A.; Sajadieh, M.S.J.C.; Engineering, I. Truck scheduling in a multi-door cross-docking center with partial unloading–Reinforcement learning-based simulated annealing approaches. *Comput. Ind. Eng.* **2020**, *139*, 106134. [CrossRef]
34. Long, X.; Zhang, J.; Qi, X.; Xu, W.; Jin, T.; Zhou, K. A self-learning artificial bee colony algorithm based on reinforcement learning for a flexible job-shop scheduling problem. *Concurr. Comput. Pract. Exp.* **2021**, *34*, e6658. [CrossRef]
35. 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]
36. Luo, S. Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. *Appl. Soft Comput.* **2020**, *91*, 106208. [CrossRef]
37. Wang, H.; Yan, Q.; Wang, J. Blockchain-secured multi-factory production with collaborative maintenance using Q learning-based optimisation approach. *Int. J. Prod. Res.* **2021**, *59*, 1–18. [CrossRef]
38. Naderi, B.; Azab, A. An improved model and novel simulated annealing for distributed job shop problems. *Int. J. Adv. Manuf. Technol.* **2015**, *81*, 693–703. [CrossRef]
39. Liu, C.; Zhang, Y.; Sun, J.; Cui, Z.; Wang, K. Stacked bidirectional LSTM RNN to evaluate the remaining useful life of supercapacitor. *Int. J. Energy Res.* **2021**, *46*, 3034–3043. [CrossRef]
40. Hua, Y.; Wang, N.; Zhao, K. Simultaneous unknown input and state estimation for the linear system with a rank-deficient distribution matrix. *Math. Probl. Eng.* **2021**, *2021*, 1–11. [CrossRef]
41. Gao, J.; Chen, R. A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem. *Int. J. Comput. Intell. Syst.* **2011**, *4*, 497–508.