Article

Intelligent Decision-Making of Scheduling for Dynamic Permutation Flowshop via Deep Reinforcement Learning

Shengluo Yang [1,2,3], Zhigang Xu [1,2,*] and Junyi Wang [1,2]

1 Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang 110016, China; yangshengluo@sia.cn (S.Y.); jywang@sia.cn (J.W.)
2 Institutes for Robotics and Intelligent Manufacturing, Chinese Academy of Sciences, Shenyang 110169, China
3 University of Chinese Academy of Sciences, Beijing 100049, China
* Correspondence: zgxu@sia.cn

Abstract: Dynamic scheduling problems have been receiving increasing attention in recent years due to their practical implications. To realize real-time and the intelligent decision-making of dynamic scheduling, we studied dynamic permutation flowshop scheduling problem (PFSP) with new job arrival using deep reinforcement learning (DRL). A system architecture for solving dynamic PFSP using DRL is proposed, and the mathematical model to minimize total tardiness cost is established. Additionally, the intelligent scheduling system based on DRL is modeled, with state features, actions, and reward designed. Moreover, the advantage actor-critic (A2C) algorithm is adapted to train the scheduling agent. The learning curve indicates that the scheduling agent learned to generate better solutions efficiently during training. Extensive experiments are carried out to compare the A2C-based scheduling agent with every single action, other DRL algorithms, and meta-heuristics. The results show the well performance of the A2C-based scheduling agent considering solution quality, CPU times, and generalization. Notably, the trained agent generates a scheduling action only in 2.16 ms on average, which is almost instantaneous and can be used for real-time scheduling. Our work can help to build a self-learning, real-time optimizing, and intelligent decision-making scheduling system.

Keywords: permutation flowshop scheduling problem; deep reinforcement learning; actor-critic; dynamic scheduling; real-time scheduling; new job arrival; tardiness cost

check for updates

Citation: Yang, S.; Xu, Z.; Wang, J. Intelligent Decision-Making of Scheduling for Dynamic Permutation Flowshop via Deep Reinforcement Learning. *Sensors* **2021**, *21*, 1019. https://doi.org/10.3390/s21031019

Academic Editor: Kim Phuc Tran
Received: 29 December 2020
Accepted: 27 January 2021
Published: 2 February 2021

1. Introduction

Production scheduling is one of the important issues to consider for a production workshop. Since flowshop is one of the most widely used workshops, the permutation flowshop scheduling problem (PFSP) has received numerous studies during the past several decades. According to a review by Fernandez-Viagas et al. [1], hundreds of heuristic and metaheuristic methods have been proposed to solve the PFSP over the past 60 years. Studies of PFSP have a significant impact on the scheduling field. The permutation criteria in PFSP restricts that the job sequence in the first machine is maintained for all successive machines, i.e., the job sequence is the same for all machines in the flowshop. The PFSP has been proved non-deterministic polynomial-time hard (NP-hard) when the number of machines is more than two [1]. To solve the NP-hard problem, many hybrid meta-heuristics have been proposed [2,3]. For the PFSP, its objective is to find a good job sequence to minimize makespan [4–6], flow time [7–10], tardiness [11–14], multiple objective [15–20], etc.

Given the NP-hard of PFSP, many meta-heuristics, such as genetic algorithm (GA) [21–24], particle swarm optimization (PSO) [25–27], and iterated greedy (IG) algorithm [21–24], have been proposed to solve the PFSP. Generally, the PFSP is regarded as a statistic scheduling problem. However, in a real production environment, the workshop is dynamic with real-time events occurring, such as dynamic order arrival, machine breakdowns, rush order insertion, etc. Thus, dynamic scheduling has more practical implications for a real workshop. Several dynamic characteristics have been considered, such as dynamic job/order arrival [28–31], stochastic processing time [32–34], machine breakdown [35], process interruptions [36,37], etc.

Among these dynamic characteristics, new job arrival has been recently receiving arising attention. In this problem, jobs arrive at the production system randomly rather than at time zero, as generally assumed for statistic PFSP. A rescheduling is needed when a new job arrives. This problem is described as a real-time scheduling problem in some literature [25,38–40].

Several researchers have studied the dynamic PFSP with new job arrival using heuristics and meta-heuristics. Rahman, Sarker and Essam [38] investigated the PFSP with real-time order arrival and used GA to repeatedly re-optimize the solution as each new order arrives. Liu et al. [41] proposed a scheduling strategy by integrating match-up and real-time strategy and provided eleven new heuristics with ten existing and one new priority rule. Li et al. [42] solved the flow shop scheduling problem with new job arrival using the constructive heuristic method. The heuristic method successively puts each job in the current best position and reinserts several jobs based on weight calculations. Liu et al. [43] studied the dynamic PFSP with new job arrival using an improved IG, which is equipped with a novel repair method and new enhanced destruction and reconstruction.

With the advance of artificial intelligence (AI), DRL and reinforcement learning (RL) have also been used to optimize scheduling problems. The DRL method trains a scheduling agent to choose the best job to process next based on current production status. Since a scheduling problem can be regarded as a sequential decision problem, it is suitable to be solved by DRL [44]. Many scheduling problems, such as flowshop scheduling problem (FSP), job shop scheduling problem (JSP), assembly shop scheduling problem (ASP), parallel machine scheduling problem, etc., have been solved by DRL or RL. For FSP, Zhang et al. [45] studied the FSP using an online TD algorithm to minimize makespan. For JSP, several studies have been carried out both for statistic and dynamic JSP. Lin et al. [46] studied the JSP under an edge computing framework and used a multiclass deep Q network (DQN) to generate scheduling decisions for multiple edge devices. Liu et al. [47] solved the JPS using an actor-critic algorithm. Zhang et al. [48] used the DRL to automatically learn priority dispatching rule for JSP. They exploited the disjunctive graph representation of JSP and proposed a Graph-Neural-Network-based scheme to embed sates. Han and Yang [49] studied the adaptive JSP using dueling double DQN. Luo [44] studied the dynamic JSP with new job insertions to minimize total tardiness using DRL. A DQN is used to select appropriate dispatching rules at each rescheduling point, and seven generic state features are extracted to represent production status. For other types of workshops, Wang et al. [50] investigated the adaptive scheduling for assembly job shop with uncertain assembly times using dual Q-learning algorithm, which contains top and bottom level Q-learning methods. Shiue, Lee and Su [39] studied a RL-based real-time scheduling problem using multiple dispatching rules strategy to respond to changes in a manufacturing system. Shiue et al. [51] studied the dynamic scheduling of a flexible manufacturing system and semiconductor wafer fabrication using RL. Zhang et al. [52] studied the scheduling of unreliable parallel machines to minimize mean weighted tardiness using RL.

In addition to the DRL, other AI approaches, such as machine learning, deep learning, and a combination of machine learning and meta-heuristics, are also used to solve scheduling problems in recent years. Jun and Lee [53] addressed the dynamic single-machine scheduling problem to minimize total weighted tardiness by learning dispatching rules from schedules. They proposed a decision-tree-based machine learning method to extract dispatching rules from existing schedules. Wu et al. [54] used deep learning to solve unreliable machines' dynamic dispatching in re-entrant production systems. They combine a deep neural network (DNN) and Markov decision processes (MDP) to assign different priorities to job groups to minimize cycle time or maximize throughput. Li et al. [55] studied the flexible job-shop scheduling problem (FJSP) with sequence-dependent setup times and limited dual resources using machine learning and meta-heuristics. A hybrid meta-heuristic is proposed to solve the FJSP, and the machine learning classification model is trained to identify rescheduling patterns. Chen et al. [56] used RL to control key parameters of GA during evolution in FJSP.

From the above literature review, we can know that the dynamic PFSP with new job arrival has become an active topic in recent years and has been studied using several meta-heuristics. Recently, DRL and other AI techniques serve as a new approach to solve scheduling problems. However, the dynamic PFSP with new job arrival and total tardiness cost criteria has not been solved by DRL. Since the modeling of dynamic flowshop is different from those of other workshops, it is necessary to propose a DRL-based modeling method for dynamic PFSP. In addition, the DRL-based approaches are not compared with traditional meta-heuristics in most literature. It is unclear whether the DRL algorithms outperform traditional meta-heuristics considering solution quality and CPU times.

This paper studies the dynamic PFSP with new job arrival to minimize total tardiness cost using DRL. Our study aims to realize real-time optimization and intelligent decision-making of scheduling for dynamic flowshop using DRL. The procedure of solving dynamic PFSP using DRL is illustrated, and the mathematical model is formulated. The scheduling agent is then modeled by designing state features, actions, and reward for the DRL agent. The A2C algorithm is adapted to train the scheduling agent to choose appropriate actions in different production status. The results show the well performance of the A2C-based scheduling agent compared with other DRL algorithms and meta-heuristics.

In particular, the contributions of this paper are as follows.

(1) To the best of our knowledge, this is the first attempt to solve the dynamic PFSP with new job arrival to minimize total tardiness cost using DRL. Our work can fill the research gap in solving dynamic PFSP by DRL.
(2) An intelligent decision-making scheduling system based on DRL is designed and modeled. The system architecture of intelligent scheduling is proposed. States features, actions and reward for the scheduling agent and system are designed.
(3) Our work shows the DRL-based scheduling method outperforms traditional meta-heuristics (IG and GA) in solution quantity and CPU times by a large margin for dynamic FPSP.

2. Problem Description and Mathematical Model

2.1. Problem Description

In this section, the system architecture of solving dynamic PFSP using DRL is proposed. As shown in Figure 1, a new job arrives at the system randomly. If the job cannot be processed immediately, the job will be stored in a buffer BF. The flowshop needs to determine which job to process next using scheduling agent SCD when a job is finished in M_1. The current state features, including job information and production status, are input to SCD to generate a scheduling action. Since for PFSP, the job sequence is maintained for all machines. Only M_1 needs to determine which job will be processed next. Thus, for step 7 in Figure 1, the system time is only pushed to time $t_{s'}$ (when M_1 finishes job j) to determine which job will be processed next in M_1 using SCD. At each decision point $t_{s'}$, the jobs finished in all machines are moved to the finished jobs area, denoted as FNS.

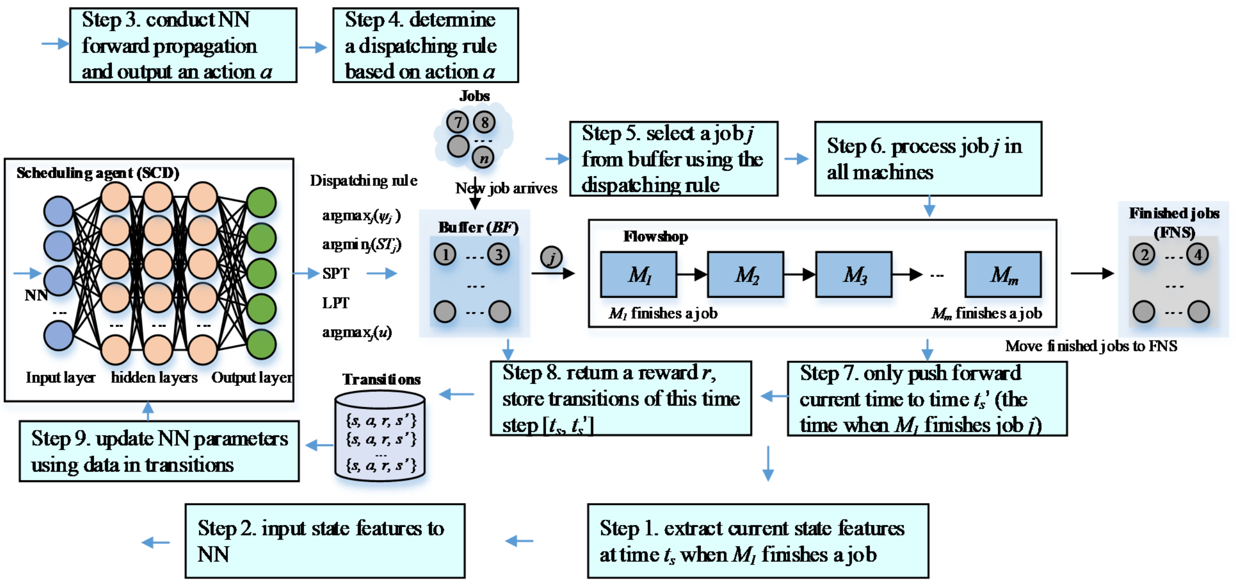
Step 3. conduct NN forward propagation and output an action *a*
Step 4. determine a dispatching rule based on action *a*
Jobs
New job arrives
Step 5. select a job *j* from buffer using the dispatching rule
Step 6. process job *j* in all machines
Scheduling agent (SCD)
NN
Input layer
hidden layers
Output layer
Dispatching rule
argmax_j(ψ_j)
argmin_j(ST_j)
SPT
LPT
argmax_j(u)
Buffer (BF)
Flowshop
M_1
M_2
M_3
M_m
M_1 finishes a job
M_m finishes a job
Finished jobs (FNS)
Move finished jobs to FNS
Transitions
{s, a, r, s'}
Step 8. return a reward *r*, store transitions of this time step [t_s, t_s']
Step 7. only push forward current time to time t_s' (the time when M_1 finishes job *j*)
Step 9. update NN parameters using data in transitions
Step 2. input state features to NN
Step 1. extract current state features at time t_s when M_1 finishes a job

Figure 1. The system architecture of solving dynamic PFSP using DRL.

2.2. Mathematical Model

The mathematical model is established to minimize the total tardiness cost of all jobs arriving at the system. Some notations are listed as follows.

Notations:

j: index of jobs, $j = 1, 2, \ldots, n$
i: index of machines, $i = 1, 2, \ldots, m$
t_{ij}: processing time of job j on machine i, $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$
C_{ij}: completion time of job j on machine i, $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$
C_j: completion time of job j, $j = 1, 2, \ldots, n$
d_j: due date of job j, $j = 1, 2, \ldots, n$
CP: an indicator of completion time of all jobs
TF: tardiness factor
RDD: the relative range of due dates
α_j: unit (per second) tardiness cost of job j
AT_j: arrival time of job j

Based on the notations above and some related formulation works [57–59], the objective function is formulated as Equation (1).

$$\text{Minimize} \sum_{j=1}^{n} \alpha_j \times \max\{0, C_j - d_j\} \tag{1}$$

Subject to:

$$d_j = [CP(1 - \text{TF} - \frac{\text{RDD}}{2}),\ CP(1 - \text{TF} + \frac{\text{RDD}}{2})],\ j = 1, 2, \ldots, n \tag{2}$$

$$CP = \frac{1}{m} \times \sum_{j=1}^{n} \sum_{i=1}^{m} t_{ij} \tag{3}$$

$$C_j = C_{ij},\ i = m, \forall j \tag{4}$$

$$C_{ij} \geq \max\left\{C_{(i-1)j}, C_{i(j-1)}\right\} + t_{ij},\ \forall i, j \tag{5}$$

$$C_{1j} - t_{1j} \geq AT_j,\ \forall j \tag{6}$$

$$C_{0j} = 0,\ \forall j \tag{7}$$

$$C_{i0} = 0,\ \forall i \tag{8}$$

According to [58,60,61], the d_j is set by constraint (2). The d_j follows a uniform distribution and is controlled by CP, TF, and RDD. TF and RDD are constant and are set to 0.5 and 0.5 [58,60]. CP, is calculated by Equation (3). Equation (4) defines the completion time of a job in the system. Equation (5) gives the completion time of a job on a machine. Equation (6) ensures that a job can be processed by the first machine only after the job arrives at the system. Equations (7) and (8) provide some initial values of completion time C_{ij}.

3. Modelling of the Intelligent Scheduling System

This section models the intelligent scheduling system by designing state features, actions, and reward for the scheduling agent and workshop environment. Whenever M_1 finishes a job, the scheduling agent SCD generates a scheduling action based on current state features, and a job is selected based on the scheduling action and processed on M_1. The workshop environment returns a reward for this scheduling action to update SCD parameters. The reward, state features, and actions are designed as follows.

3.1. Reward

Since most literature on using DRL to solve the scheduling problems are aimed to minimize makespan. These reward functions cannot be used to minimize the total tardiness cost. We designed a new reward function for the total tardiness cost criteria in this section.

Recall that the aim of solving PFSP is to find out a job sequence with minimized total tardiness cost. The scheduling agent SCD should learn to choose an appropriate job at every decision point so that the total tardiness cost is minimized after all jobs are finished. Each action the SCD taken should make the total tardiness cost increases as little as possible. Given that DRL is designed to maximize the cumulated reward obtained at each step, the reward in this problem is defined as the inverse of unit newly added total tardiness cost of unfinished jobs in the system during this time step. The newly added tardiness cost comes from work-in-progress (WIP) and jobs in the buffer (BF). Therefore, the reward of time step $[t_s, t_{s'}]$ is calculated by Equation (9).

$$r = -\frac{1}{t_{s'} - t_s}(tp_{BF} + tp_{WIP}) \tag{9}$$

where tp_{WIP}, tp_{BF} denote newly added tardiness penalty cost from WIP and jobs in BF, respectively, during the current time step $[t_s, t_{s'}]$. tp_{BF}, tp_{WIP} are calculated as follows.

$$tp_{BF} = \sum_{j=1}^{n_{BF}} z_{js'}\alpha_j[t_{s'} - \max(t_s, d_j)],\ j \in BF \tag{10}$$

$$tp_{WIP} = \sum_{j=1}^{n_{WIP}} z_{js'}\alpha_j[t_{js'} - \max(t_s, d_j)],\ j \in WIP \tag{11}$$

where n_{BF} denotes the number of jobs in BF, n_{WIP} denotes the number of WIP. $z_{js'}$, defined by Equation (12), indicates whether job j is overdue at $t_{s'}$, the end time of this step. Only the overdue jobs generate tardiness cost. $t_{js'}$, calculated by Equation (13), denotes the actual end time for job j compared with $t_{s'}$. If job j is completed, i.e., finished by all machines, before $t_{s'}$, the time step for calculating tardiness cost is $[t_s, C_j]$ rather than $[t_s, t_{s'}]$.

$$z_{j_{s'}} = \begin{cases} 1, & d_j < t_{s'} \\ 0, & \text{else} \end{cases} \tag{12}$$

$$t_{js'} = \begin{cases} C_j, & \text{if job } j \text{ has been completed at } t_{s'} \\ t_{s'}, & \text{else} \end{cases} \tag{13}$$

3.2. State Features

The scheduling agent SCD determines a scheduling action based on current state features. State features should fully and efficiently reflect current job information and production status. Specifically, state features should provide sufficient information for selecting an action at each rescheduling point. The state features in most existing literature are designed to solve the JSP to minimize the makespan. These features are not applicable for our studied problem. Thus, we designed five state features for the PFSP under the total tardiness cost criteria, considering both jobs in *BF* and current production status. Unlike the existing literature, we calculated four statistic characteristics, maximum, minimum, average, and standard deviation, for several state features rather than only providing the average value to reflect each feature's characteristics better.

The five state features, $ft_1, ft_2, \ldots, ft_5$, are defined as follows.

(1) $ft_1 = \{\psi_j\}, j \in BF$. Current unit tardiness cost of each job in *BF*. where ψ_j is the unit tardiness cost generated by job *j* at present and is calculated as follows.

$$\psi_j = \begin{cases} \alpha_j, & \text{if job } j \text{ overdues at current time} \\ 0, & \text{else} \end{cases} \tag{14}$$

(2) $ft_2 = \{ST_j\}, j \in BF$. Safe time of each job in *BF*. where ST_j is determined by Equation (15). ST_j reflects how much time will be left before the due date d_j when job *j* is finished, if job *j* begins to be processed at present time t_c.

$$ST_j = d_j - \sum_{i=1}^{m} t_{ij} - t_c \tag{15}$$

(3) $ft_3 = \{\sum_{i=1}^{m} t_{ij}\}, j \in BF$. Total processing times in all machines for each job in *BF*.

(4) $ft_4 = \{u_j'\}, j \in BF$. The estimated utilization rate of each job in *BF*. u_j' is the estimated utilization rate of job *j* and is calculated by Equation (16). Each job in *BF* is assumed to be processed under the present production status. The u_j' is calculated based on the waiting times WT_{ij}' of all machines when job *j* is processed.

$$u_j' = 1 - \left(\sum_{i=2}^{m} WT_{ij}' \bigg/ \sum_{i=1}^{m} t_{ij}\right),\ j \in BF \tag{16}$$

$$WT_{ij}' = \max(0,\ C_{i(j-1)} - C_{(i-1)j}), i = 2, 3, \ldots, m, j \in BF \tag{17}$$

where WT_{ij}' is the waiting time of job *j* on machine *i*. WT_{ij}' generates when job *j* is finished on machine $i - 1$, but cannot be processed immediately on machine *i* because machine *i* has not finished its current job $j - 1$. Note that the first machine M_1 does not have a waiting time because M_1 is always idle at a decision point.

(5) $ft_5 = n_{BF}$. The number of jobs in *BF* at present.

For the first four state features, which are array, four statistic characteristics, maximum, minimum, average, and standard deviation, are calculated. Thus, the total dimensions for state features are 4 × 4 + 1 = 17. In addition, all features are normalized to facilitate the learning process.

3.3. Actions

Each action corresponds to a single dispatching rule (SDR) for selecting a candidate job to be processed in M_1 from *BF*. The action space should provide sufficient and effective dispatching strategies under different production status. To lower the learning difficulty, the action space should not be too large. We designed five actions (a_1–a_5) for SCD, considering different production status and well-known dispatching rules.

(1) Select the job j, which has the minimum current unit tardiness cost ψ_j

Where ψ_j is calculated by Equation (14).

$$a_1 = \text{argmax}_j(\psi_j),\ j \in BF$$

a_1 is apparent because the overdue jobs with the maximum unit tardiness cost should be processed first to reduce the increase of tardiness cost in the system. Otherwise, the job will generate tardiness cost every second, and the tardiness cost is the largest compared with the tardiness cost generated by other jobs in BF.

(2) Select the job j, which has the minimum safe time ST_j. Where ST_j is calculated by Equation (15).

$$a_2 = \text{argmin}_j(ST_j),\ j \in BF$$

a_2 may be appropriate when no jobs in BF overdue or other actions do not have too many advantages.

(3) Select a job that has the shortest processing time (SPT).

$$a_3 = \text{argmin}_j(\sum_{i=1}^{m} t_{ij}),\ j \in BF$$

(4) Select a job that has the longest processing time (LPT).

SPT and LPT are well-known dispatching rules and have been used as scheduling actions in [44,46,48].

$$a_4 = \text{argmax}_j(\sum_{i=1}^{m} t_{ij}),\ j \in BF$$

(5) Select a job that obtains the maximum utilization rate $u_j{}'$.

$$a_5 = \text{argmax}_j(u_j{}'),\ j \in BF$$

a_5 selects the most suitable job for current machine status, considering the utilization of machines. a_5 may has larger priority when no jobs overdue or the ψ_j has small standard deviation, etc.

The five scheduling actions above can provide efficient dispatching strategies under different production status. If more than one job in BF fulfills an action, a job is selected randomly among those candidate jobs.

4. A2C

In this section, we adapt A2C to solve the dynamic PFSP with new job arrival. A2C is a DRL algorithm that equips with an actor-network π_θ and critic-network V_ϕ [62]. A2C selects an action with the probability $\pi_\theta(s_t)$ generated by actor-network π_θ, at state s_t. The critic-network is used to estimate the state value of a state s_t in the learning process. Recently, Liu, Chang and Tseng [47] used A2C to solve JSP to minimize makespan and achieved a good balance between makespan and execution time. However, A2C has not been used to solve the PFSP in currently published literature. We adapted the A2C to solve the dynamic PFSP, and the A2C-based training method are shown in Algorithm 1. In Algorithm 1, the new job arrival operation adds the newly arrived jobs during this step to BF. The operation is executed in the beginning and when a job is finished in M_1. When a job is finished in M_1, the WIP is updated by removing jobs that have already been finished in all machines.

The critic and actor networks are updated every T steps using gradients shown in lines 15 and 16 [62] in Algorithm 1. The dr_t denotes the discounted reward of step t and is used as the target state value of step t. The estimated state value of step t is $V_\phi(s_t)$. Thus, the gradient for critic-network is calculated by $\nabla_\phi(dr_t - V_\phi(s_t))^2$. For the actor-network, the difference between target and estimation state values is used to update the probability for selecting actions at state s_t. Also, an entropy $H(\pi_\theta)$, calculated by Equation(18), is considered to expand explorations.

$$H(\pi_\theta) = -\sum_{a_t \in A} \pi_\theta(a_t|s_t) \log \pi_\theta(a_t|s_t) \tag{18}$$

where A is the action space of the agent.

Algorithm 1 shows that an instance with n jobs is used to train the DRL agent for EP epochs, and the agent's parameters are updated every T steps. Thus, the computation complexity of the training process for a single instance is O($EP \times n \times T$). However, after training, the computation complexity for solving an instance is reduced to O(n).

Algorithm 1. The A2C-based training method.

```
1:  Initialize actor and critic network $\pi_\theta$, $V_\phi$
2:  for epoch = 1: $EP$ do
3:  Perform new job arrival at time zero (current system time)
4:  Get current state $s_t$
5:  while step = 1: $n$ do   \\ $n$ is the number of jobs for the selected instance
6:  Determine an action $a_t$ based on probability $\pi_\theta(s_t)$ at state $s_t$
7:  Select a job $j$ from $BF$ using action $a_t$, process job $j$ in all machines, obtain the finished time of job $j$ in each machine
8:  Push forward the system time only to the time when job $j$ is finished in $M_1$
9:  Perform new job arrival and update $WIP$ at current system time
10: Get current state $s_{t+1}$ and reward $r_t$
11: Store transition $\{s_t, a_t, r_t, s_{t+1}\}$ of this step
12: $s_t \leftarrow s_{t+1}$
13: if step % $T$ == 0 then
14: Calculate discounted reward $dr_t$ of the $T$ steps in reverse order using data in transitions,
    $dr_t = \begin{cases} r_t + \gamma V_\phi(s_{t+1}), \text{ for the } T\text{th step} \\ r_t + \gamma dr_{t+1}, \text{ for the first } T-1 \text{ steps} \end{cases}$
15: Update critic-network $V_\phi$ using gradient $\nabla_\phi (dr_t - V_\phi(s_t))^2$
16: Update actor-network $\pi_\theta$ using gradient $\nabla_\theta \log \pi_\theta(a_t|s_t)[dr_t - V_\phi(s_t)] + \beta \nabla_\theta H(\pi_\theta(s_t;\theta))$
17: end if
18: end while
19: end for
```

5. Numerical Experiments

In this section, a large range of instances is generated to train the A2C-based scheduling agent SCD. After training, the trained model of SCD is saved and used to provide intelligent decision-making of scheduling for dynamic PFSP. To evaluate the performance of A2C, we compared it with SDR, two other DRL algorithms, DQN and double DQN (DDQN), and two traditional meta-heuristics, IG and GA. We also tested the trained agent on a new set of extended instances with larger production configurations to verify the generalization of A2C.

A total number of 84 instances are generated for dynamic PFSP with new job arrival using different parameter settings of n and m, following [43,44]. The specific parameters for instances are listed in Table 1. As mentioned above, the due dates d_j are determined by Equation (2). We noticed that when $|r| < 10$, the agent can be trained more efficiently. Based on this consideration, the α_j is set.

The configuration of n and m in Table 1 is repeated three times. Hence, the total number of instances is 7 × 4 × 3 = 84. For each instance, three jobs are set as initial jobs, which arrive at time zero. Other jobs arrive following a poison distribution, which means the interval time of two successive arriving jobs follows an exponential distribution. The mean value of the exponential distribution is set as 30. To better evaluate the performance of the trained model, we divide the 84 instances as 59 (occupies 70% of all instances) training instances and 25 (occupies 30%) test instances randomly. In the training process, 3000 epochs are set. For each training epoch, all the 59 training instances are used to generate 59 episodes to train the scheduling agent. Thus, a total number of 59 × 3000 = 177,000 instances are used to train the agent. All instances, training and test results, and the video of solving dynamic PFSP using the trained agent and SDR are uploaded as Supplementary Materials.

Table 1. Parameters for instances.

Parameter	**Value**
Number of jobs (n)	{20, 50, 80, 100, 120, 150, 200}
Number of machines (m)	{5, 10, 15, 20}
Number of initial jobs for every instance	3
Interval time of job arriving	E(1/30)
Processing time on a machine (t_{ij})	U[1, 100]
Unit tardiness cost of a job (α_j)	U[0, 5]

All algorithms and workshop environments are coded with Python 3.8. The training and test experiments are performed on a PC with Intel(R) Core(TM) i7-6700 CPU @ 3.40 GHz CPU and 12 GB RAM.

5.1. Training Process of A2C

The A2C-based scheduling agent SCD is trained using the aforementioned training instances. Parameter settings for A2C are shown in Table 2. Note that the coefficient β is linearly decreased from 0.005 to 0.0005 during the first 70% epochs and remains at 0.0005 after the 2100th epoch.

Table 2. Parameter settings for the training process and DRL algorithms.

Parameter	**Value**
Number of hidden layers	3
Number of neurons in each hidden layer	100
Number training epochs (EP)	3000
Discount factor (γ)	0.98
Update step iteration (T)	5
Learning rate of actor and critic	$2 \times 10^{-6}, 5 \times 10^{-6}$
Range of entropy coefficient (β)	0.005~0.0005

For each epoch, the SCD is first trained on all of the 59 training instances and then tested on all of the 25 test instances. An instance is used to generate an episode under certain production configurations. Thus, each epoch contains 59 training episodes used to train the agent and 25 test episodes used to evaluate the agent's performance after training at this epoch. The average total tardiness cost and episode reward on all test instances are recorded for each epoch. Note that the total tardiness cost in all experiments is rescaled by dividing 1000.

The average total tardiness cost on all test instances at each training epoch is shown in Figure 2. Figure 2 shows that the average total tardiness cost decreases dramatically with the increase of training epochs, meaning the SCD effectively learned to choose a good scheduling strategy and had a good generality for the unseen test instances. In particular, for the first 1400 epochs, the average total tardiness cost decreased from 163.55 to 91.88, which is a 43.82% improvement. The learning curve remains relatively stable after the 2000th epoch, even with some fluctuations. This indicates the SCD can provide reasonable scheduling solutions at a relatively stable level after training. Note that the solution seems not to keep at the global optimal found in the approximate 1300th epoch. This might be due to the exploration of A2C. The fluctuations and the problem of escaping from global optimal can be reduced by increasing training epochs.

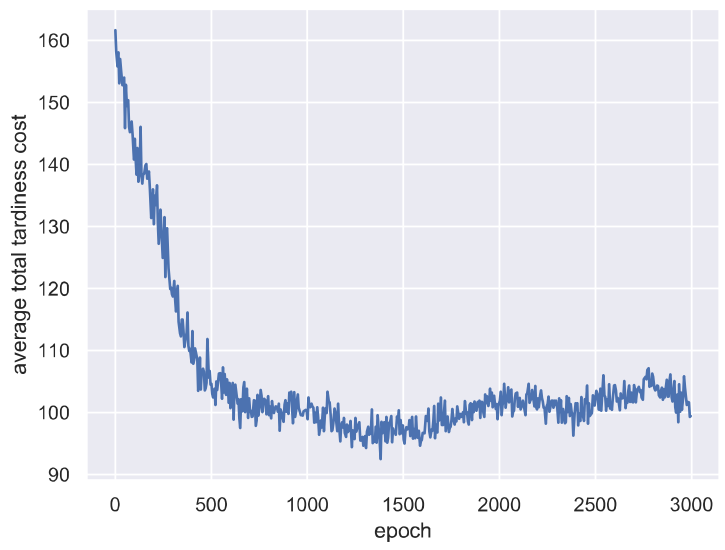

Figure 2. Average total tardiness cost on all test instances at each training epoch.

The average episode reward obtained by SCD during the training epochs is shown in Figure 3, which can reflect the learning effect more directly. Figure 3 shows that the episode reward increases significantly with the increase of training epochs. Recall that a DRL agent learns to maximize the accumulated reward obtained during an episode. Thus, Figure 3 confirms the SCD indeed learned to perform better during the training process. The curve tendency of average episode reward is similar to that of average total tardiness cost, indicating the reward of SCD has a high correlation with the objective function of the studied scheduling problem, verifying the correctness of our reward design for SCD.

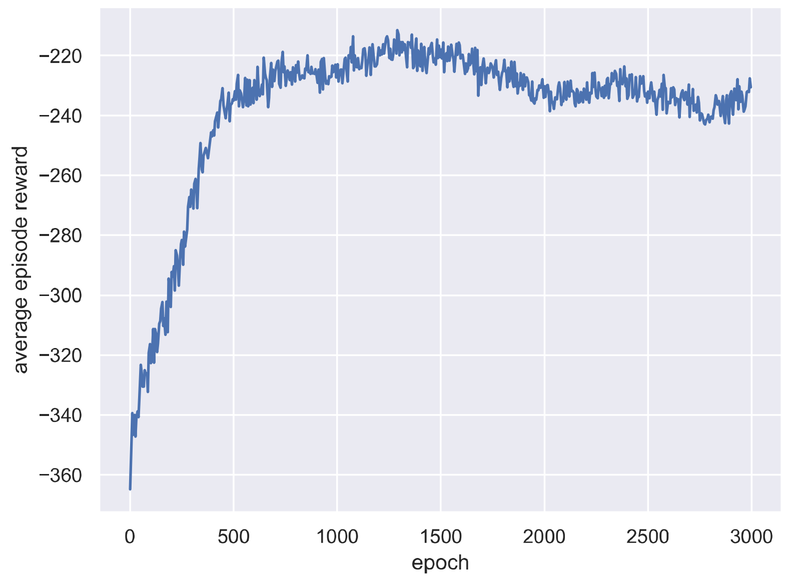

Figure 3. Average episode reward on all test instances at each training epoch.

5.2. Comparison with SDR

Recall that the scheduling agent learns to choose the best scheduling action, i.e., SDR, at every rescheduling point. To evaluate whether the scheduling agent has learned to choose the best scheduling actions, we compared the A2C with SDR and random action strategys, as shown in Figure 4.

Figure 4 shows that the a_1 obtains the best results. This may be because for the tardiness penalty criteria, always selecting the jobs with maximum current unit tardiness cost may be the best strategy, typically when the due date is too tight, and many jobs have waited in the buffer. It indicates that simple dispatching rules exist to generate high-quality solutions for the dynamic PFSP with total tardiness cost criteria. This kind of simple SDR is useful and efficient in practical production scheduling.

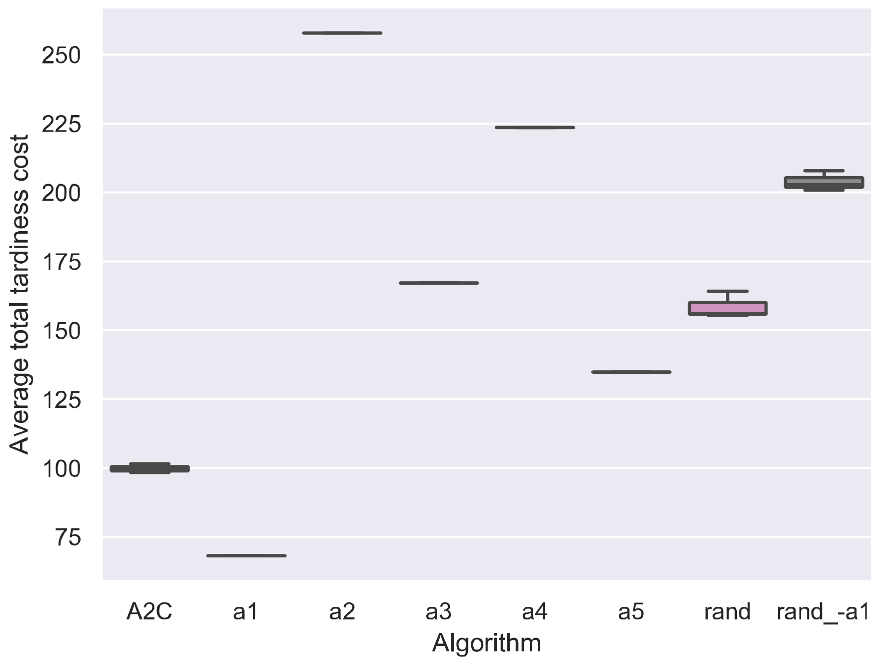

Figure 4. Comparison between A2C and SDR considering average total tardiness cost on all test instances.

The results also indicate that the A2C fails to find the best actions at all rescheduling points. Between all actions, the a_1 is the best one. However, in theory, there are situations when other actions are more appropriate than a_1. Always selecting the a_1 may not be the best strategy. Since the A2C is worse than a_1, the DRL agent did not learn to choose the best actions at every rescheduling points. The trained DRL agent may not outperform its best pure action significantly when the performance of actions differs greatly. For example, the results in Lin, Deng, Chih and Chiu [46] show that the trained DRL agent MDQN only outperforms its best pure action MOPRN by 14.33%.

The random actions are also compared with the SDR strategy. Figure 4 shows that random action results are not the worst, but approximately the average of those of other actions. This may be because the random action is expected to generate a solution that equals the mean value of other actions' solutions. To verify this conjecture, we tested the *rand_-*a_1, the random action exclude a_1. The results of *rand_-*a_1 get worse when the best efficient action a_1 is removed. Also, the *rand_-*a_1 obtains the results close to the average results of other all actions exclude a_1.

Compared between all actions, the a_1 is the best one, followed by a_5, i.e., select jobs with the maximum utilization rate $u_j{}'$, and the worst one is a_2, i.e., select the jobs with the minimum safe time ST_j. Surprisingly, the a_2 is worse than a_3 (SPT) and a_4 (LPT). This may be because the SPT and LPT are more related to the utilization, which is more important than the due date information influencing the safe time. The best action a_1 outperforms the worst one a_2 by 73.55%, which is a large scale. To improve the performance of DRL, more efficient actions should be designed, and less efficient actions are replaced. In addition, more efficient DRL algorithms should be proposed since the A2C fails to choose the best actions in every situation.

5.3. Comparison with DRL and Meta-Heuristics

To evaluate the performance of A2C, we compared it with two other widely used DRL algorithms, DQN and DDQN, and two traditional mate-heuristics, IG and GA.

5.3.1. Training Process of DQN and DDQN

As mentioned above, the DQN has widely used to solve scheduling problems [44,46,49]. We adapt the DQN [63] and its variant DDQN [64] to solve the dynamic PFSP. The training procedure and instance settings are the same as those of A2C. After tuning, the hyper-parameters of DQN and DDQN are set. The learning rate for DQN and DDQN are 1×10^{-5} and 1×10^{-6}, respectively. The memory size and batch size are 1000 and 32, respectively.

The training curves of DQN and DDQN are shown in Figures 5 and 6. Figure 5 shows that the average total tardiness cost decreases during the training epochs, indicating the two DRL agents have learned to choose better actions at rescheduling points. Figure 6 shows that the average episode reward shows an increasing trend, verifying the agents indeed learned to generate better solutions. However, the learning curves of DQN and DDQN show a large fluctuation compared with those of A2C, indicating the A2C is easier to train. For the two DQN algorithms, the training curve of DDQN is smoother than those of DQN.

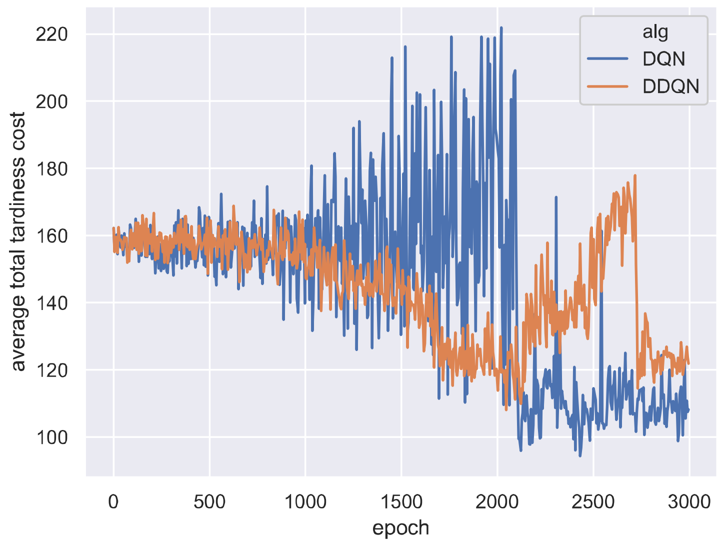

Figure 5. Average episode cost on all test instances at each training epoch for DQN and DDQN.

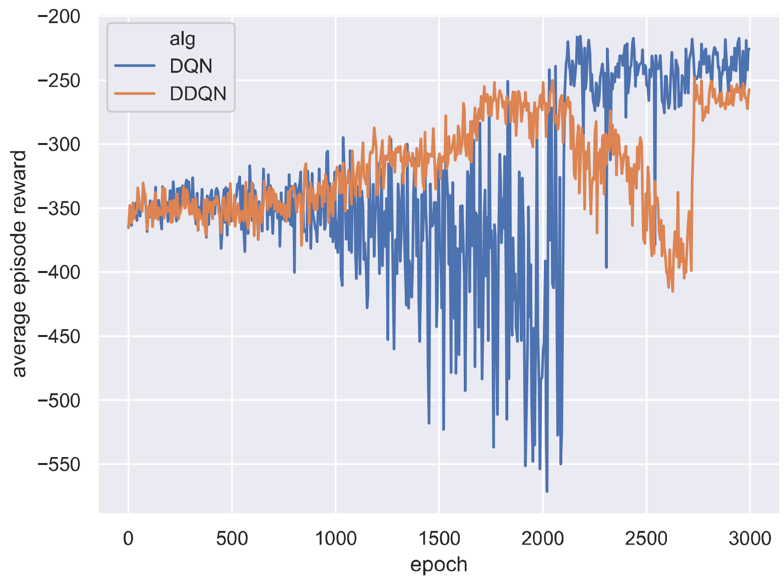

Figure 6. Average episode reward on all test instances at each training epoch for DQN and DDQN.

5.3.2. IG and GA

To test the performance of A2C in solving dynamic PFSP, we also compared it with traditional meta-heuristics, which are extensively used in solving scheduling problems. From the review of [1], the IG-based algorithms show a very efficient performance compared with other meta-heuristics on the PFSP. In addition, GA is one of the most commonly used meta-heuristics in solving PFSP [1]. Thus, IG and GA are selected as traditional meta-heuristics to compare with our A2C-based scheduling approach.

For the dynamic PFSP, jobs arrive randomly. When a new job arrives, a rescheduling procedure is required to schedule the newly arrived jobs with jobs in the system. Following [16], we also set rescheduling times as 5 for an instance. When a rescheduling point is not reached, the newly arrived jobs are appended to the current job sequence to obtain a new job sequence. At each rescheduling point, the new job sequence is searched by meta-heuristics, i.e., IG and GA.

IG starts from an initial solution, executes destruction, reconstruction, local search, and acceptance criteria on the solution sequence for several iterations, and returns the optimized solution [21]. At the beginning of an instance, the initial solution π_T is generated by putting all jobs arrived at time zero into a job sequence randomly. Jobs in π_T are processed successively. When a rescheduling point is reached, the current job sequence π_T is optimized through destruction, reconstruction, local search, and acceptance criteria for several iterations. In the destruction, $\mid \pi_T \mid \times p_{desJ}$ jobs are selected randomly from the current job sequence π_T and removed from π_T to the insertion job sequence π_R. $\mid \pi_T \mid$ denotes the number of jobs in π_T. After the destruction, the insertion job sequence π_R is obtained, and the left job sequence in π_T is denoted as π_D. In the reconstruction, jobs in π_R are selected randomly, inserted to all possible positions in π_D, and finally placed in the position with minimum total tardiness cost to obtain a new job sequence. In the local search, the jobs in the current job sequence are randomly selected without replication. The selected jobs are then tested in all possible positions of the current job sequence and placed in the position with minimum total tardiness cost. When more than one position obtains the minimum total tardiness cost, one of those best positions is selected randomly. For the acceptance criteria, the aggravated solutions are generally accepted with a certain probability to expand searching scopes and escape from local optimal [58,65]. In this paper, worse solutions are accepted at a certain probability τ. The parameters of IG are tuned using the design of experiments (DOE) and analysis of variance (ANOVA) as performed in [58]. After tuning, the parameter settings of IG are p_{desJ} = 0.2, τ = 0.05.

GA starts with an initial population, performs crossover, mutation, and selection between the population or individual for several iterations, and returns the optimized solution. Similar to IG, the initial solution is generated randomly using jobs arrived at time zero. At each rescheduling point, the current job sequence π_T is permuted randomly to generate a population with N individuals. $N = min(N', \mid \pi_T \mid \times (\mid \pi_T \mid - 1))$, where N' is the given population size. The population is evolved through crossover, mutation, and selection operations. In the crossover, the N individuals are paired randomly. Two individuals in a pair perform crossover with a probability p_c. For the pairs that performs crossover, a crossover point is selected randomly. The sequences before the crossover point are switched between the two individuals in the same pair, and two new child job sequences are obtained. The repetition jobs in two child job sequences are replaced with jobs shown in the parent job sequence but not in the child's sequence. In the mutation, every individual l in the population mutates with probability p_m. If an individual mutates, two jobs in this individual's job sequence are selected randomly and switched with each other. In the selection, N new individuals are selected from the current population with replication. Every individual will be selected at a probability p_l, which is calculated by Equation (19).

$$p_l = \frac{f_l}{\sum\limits_{l=1}^{N} f_l}, \; l = 1, 2, 3, \ldots N \tag{19}$$

where f_l is the fitness (total tardiness cost) of individual l. The parameters of GA are tuned using the aforementioned DOE and ANOVA. The parameter settings of GA are $N' = 50$, $p_c = 0.80$, $p_m = 0.10$.

5.3.3. Comparison with DQN, DDQN, IG, and GA

In this section, the A2C is compared with two other DRL algorithms, DQN and DDQN, and two meta-heuristics, IG and GA. After training, the trained models of all DRL agents are saved and used to generate scheduling actions during production.

Figure 7 presents the average total tardiness cost of all compared algorithms on all test instances. As can be seen in Figure 7, A2C obtains the best results compared with all other algorithms. Typically, A2C outperforms two traditional meta-heuristics, IG and GA, by a large margin. We can also see that the three DRL algorithms outperform two meta-heuristics. This may result from the efficient scheduling action a_1. One surprising result is that the DDQN is worse than DQN. This may be because the two algorithms are not fully convergence at the given epochs, and their results may change when more training epochs are given. For the two meta-heuristics, IG and GA are tested at two search iteration levels, 50 iterations and 300 iterations. When the iteration increased from 50 to 300, IG shows a significant statistical improvement, while GA does not. Overall, the two meta-heuristics with two search iteration levels do not differ significantly in the solution quality.

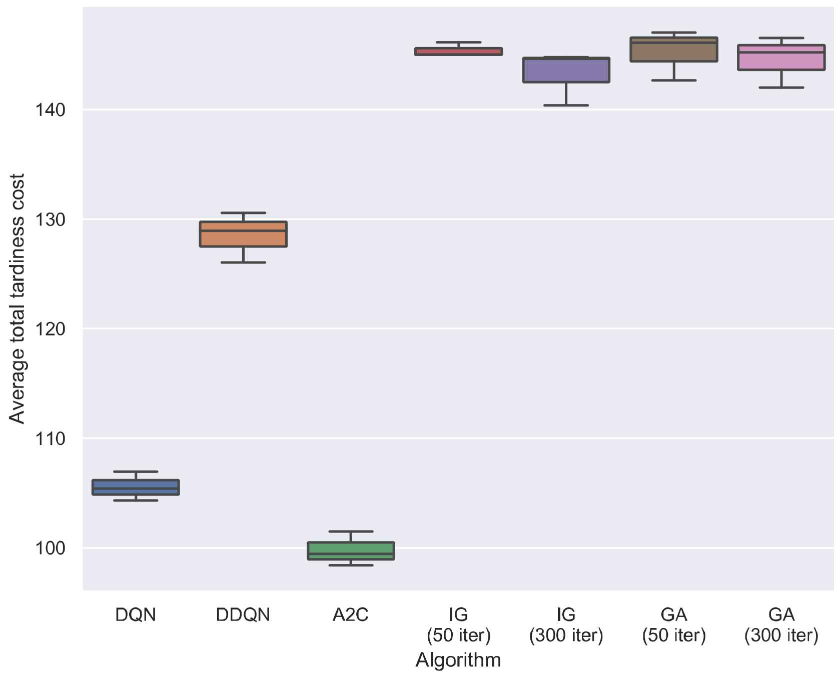

Figure 7. Average total tardiness cost of compared algorithms on all test instances. Three replications are carried out for each test instance. IG and GA are tested under two search iteration levels, 50 iterations and 300 iterations.

The specific average total tardiness cost and CPU times of the compared algorithms are provided in Tables 3 and 4. Table 3 shows that the A2C and DQN generate the best results considering different production configurations. For the A2C, it outperforms IG and GA in solution quality and CPU times under all production configurations considered. Specifically, A2C, on average, outperforms IG and GA by 31.36% and 31.31% in solution quality, using only 0.04% and 1.35% CPU times that IG and GA spent when 50 searching iterations are provided for IG and GA. When IG and GA's iterations increased to 300, A2C outperforms IG and GA by 30.34% and 30.98%, using merely 0.01% and 0.23% CPU times spent by IG and GA. For the two meta-heuristics, when iterations increased from 50 to

300, the CPU times increase by 279.71% and 493.01%, but IG and GA's solution quality only improves slowly. IG obtains slightly better solution quality than GA, using more CPU times.

Table 3. Average total tardiness cost grouped by the number of jobs n and machines m for compared DRL and meta-heuristics. IG and GA are tested under two search iteration levels, i.e., 50 iterations and 300 iterations. The best results are highlighted in bold.

		DRL			**Meta-Heuristics**			
					50 Iterations		**300 Iterations**	
		DQN	**DDQN**	**A2C**	**IG**	**GA**	**IG**	**GA**
n	20	**13.88**	21.52	16.06	19.34	17.68	19.16	19.67
	50	50.09	65.61	**48.98**	75.43	75.83	73.38	77.13
	80	94.76	83.05	**60.79**	114.45	112.9	115.62	112.69
	100	**101.31**	143.59	108.57	161.13	159.64	161.99	165.87
	120	152.35	208.83	**144.45**	205.15	204.78	201.4	207.9
	150	175.06	190.7	**163.28**	222.84	213.67	220.35	211.7
	200	**150.53**	172.42	197.99	255.62	256.19	282.16	246.4
m	5	**43.76**	63.19	46.76	50.94	50.04	50.41	49.21
	10	**73.73**	78.09	74.81	101.02	100.75	106.84	101.21
	15	113.44	191.28	**106.21**	195.85	193.4	192.31	195.48
	20	265.5	249.54	**232.93**	323.72	316.15	317.65	321.87
Ave		105.55	128.52	**99.79**	145.39	145.28	143.25	144.58

Table 4. Average CPU times (s) grouped by the number of jobs n and machines m for DRL and meta-heuristics. IG and GA are tested under two search iteration levels, i.e., 50 iterations and 300 iterations. The CPU time for an instance is the total CPU time required for solving that instance.

		DRL			**Meta-Heuristics**			
					50 Iterations		**300 Iterations**	
		DQN	**DDQN**	**A2C**	**IG**	**GA**	**IG**	**GA**
n	20	0.03	0.03	**0.02**	12.8	56.62	4.35	32.45
	50	0.09	0.09	**0.07**	63.94	230.29	10.31	54.08
	80	0.19	0.18	**0.15**	63.2	278.19	6.39	37.85
	100	0.27	0.26	**0.21**	337.07	1301.06	15.84	94.28
	120	0.33	0.34	**0.27**	751.39	2791.67	25.53	121.65
	150	0.49	0.49	**0.38**	1020.36	3909.1	21.25	165.13
	200	0.68	0.69	**0.49**	3441.1	13023.42	34.01	200.35
m	5	0.17	0.16	**0.14**	246.99	914.52	10.82	70.21
	10	0.24	0.23	**0.18**	594.53	2258.63	13.82	81.65
	15	0.31	0.31	**0.24**	514.63	1665.1	17.72	89.91
	20	0.41	0.42	**0.33**	900.46	3891.24	24.32	156.79
Ave		0.26	0.26	**0.21**	526.99	15.59	2001	92.45

Tables 3 and 4 further show that, as the production configuration n and m increases, the disparity in total tardiness cost between A2C and meta-heuristics increases. For example, A2C outperforms IG and GA with 50 iterations by 8.21% and 7.24%, when m is 5. However, when m is 20, A2C outperforms by 28.05% and 26.67%. This indicates that the A2C-based method has more advantages in solving large instances.

Note that the average CPU times of A2C for calculating a test instance is 0.21 s. Considering the number of jobs in test instances, the average decision time for taking a scheduling action is 2.16 ms. This is almost instantaneous and can be used in real-time scheduling.

The DRL method models the scheduling problems as an MDP where a DRL agent successively determines the job to be processed next [44]. The optimization of scheduling is realized by the successive decisions of DRL agents rather than by searching the job sequence directly as some meta-heuristics do. The generality of NN enables the DRL agents to behave well for unseen instances. In addition, the DRL agent can generate a scheduling decision instantly after trained. Thus, the DRL can be served as an alternative approach for dynamic scheduling problems.

5.4. Generalization to Larger Instances

To further verify the generalization ability of DRL, we tested the trained A2C agent on a novel set of extended instances with larger production configurations. This experiment aims to find out whether the performance deteriorates greatly when production settings different from the original training and test instances.

In the extended instances, the number of jobs n is increased from the original setting $n = \{20, 50, \dots, 200\}$ to $n = \{300, 500, 800\}$. The number of machines m is set to 10. Other production settings are the same as the original instances. Each configuration of n and m are repeated three times. Thus, a total of 9 extended large instances are generated.

The SDR is used as a baseline to evaluate whether the performance of A2C deteriorates when the production configuration becomes larger. The results of A2C and SDR on the extended instances are shown in Figure 8. Figure 8 shows that the A2C is worse than a_1 and a_5. The a_5 becomes the best action, which may be because when too many jobs exist in the system, the utilization becomes more critical.

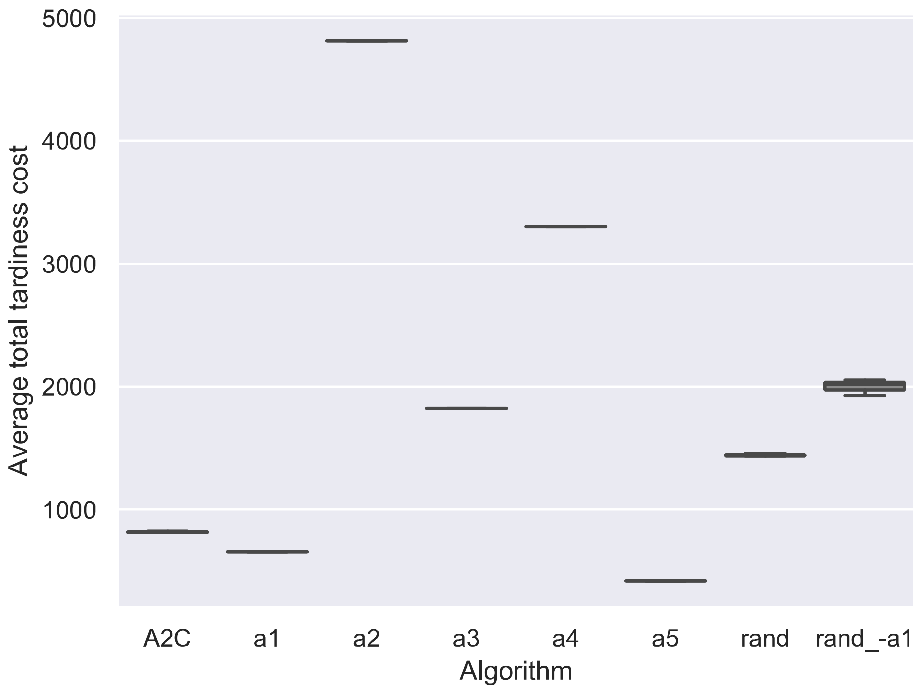

Figure 8. Comparison between A2C and SDR considering average total tardiness cost on extended instances.

To show the change of performance of A2C for larger production configurations, we provide the percentages that A2C outperforms SDR on the original and extended instances in Figure 9. As shown in Figure 9, the percentages on extended instances are larger than those on original instances for most SDR strategies except for the a_5. This indicates that, on the whole, the DRL has a better generalization ability on larger production configurations compared with SDR.

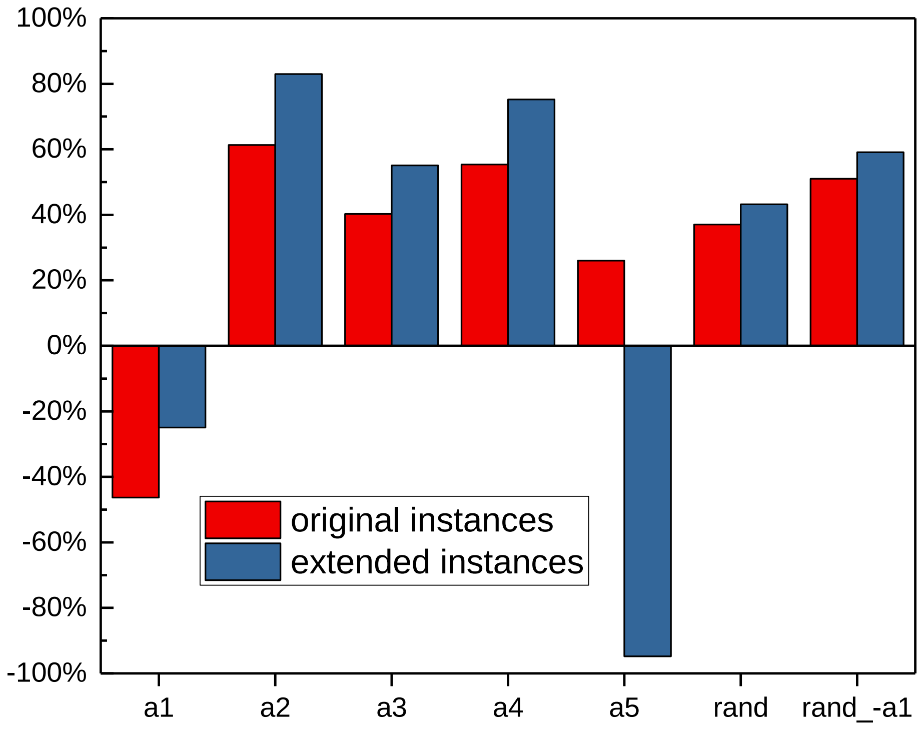

Figure 9. The percentages that A2C outperforms SDR considering average total tardiness cost on the original and extended instances.

6. Conclusions

This paper solved the dynamic PFSP with new job arrival to minimize total tardiness cost using DRL. This study aims to establish an intelligent decision-making scheduling system to provide real-time optimization for dynamic scheduling problems. The whole procedure of solving dynamic PFSP using DRL is illustrated, and the mathematical model is established. The DRL-based scheduling system is proposed with state features, actions, and reward designed for the scheduling agent and workshop environment. Five scheduling actions are designed to perform well at different production states. The state features with 17 dimensions are designed to provide directly related information for choosing scheduling actions. The A2C algorithm is adapted to train the scheduling agent SCD to learn an appropriate scheduling action at different states. A total of 84 instances are generated to train the A2C-based scheduling agent, and extensive comparisons are carried out. The trained scheduling agent is compared with SDR, two other DRL algorithms, DQN and DDQN, two traditional meta-heuristics, IG and GA. The training curve shows that the average total tardiness cost decreases significantly with training epochs increases, indicating the scheduling agent has learned to choose appropriate scheduling actions. The comparison experiments show our A2C-based scheduling agent performs best among all compared DRL and meta-heuristics. Typically, the A2C-based scheduling agent outperforms IG and GA by a large margin, both in solution quality and CPU times. Our A2C-based scheduling agent can generate a scheduling action in 2.16 ms on average. This can be used for real-time scheduling and online decision-making in a real production workshop. In addition, the A2C-based scheduling agent shows well generalization on unseen instances with larger production configurations. The scheduling agent can still be trained during production. Our study contributes to establishing a self-organizing and self-learning scheduling system.

Future research can consider more realistic characteristics, such as machine breakdown, transportation time, etc., and solve real-time scheduling of more complex workshops, such as reconfiguration workshop, hybrid flowshop, and distributed workshop. In addition, more studies can be applied to design more effective state features, action spaces, and reward for the scheduling system. Moreover, more DRL algorithms could be studied to improve the scheduling agent's learning effect and solution performance.

Supplementary Materials: All instances, training and test results, and the video of solving dynamic PFSP using the trained agent and SDR are available at https://osf.io/tynrd/?view_only=bbea6a78dff042e3a5952c3283031676.

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

Funding: This research was funded by the National Natural Science Foundation of China (61803367) and the Natural Science Foundation of Liaoning Province (2019-MS-346).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Fernandez-Viagas, V.; Ruiz, R.; Framinan, J.M. A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation. *Eur. J. Oper. Res.* **2017**, *257*, 707–721. [CrossRef]
2. Nazhad, S.H.H.; Shojafar, M.; Shamshirband, S.; Conti, M. An efficient routing protocol for the QoS support of large-scale MANETs. *Int. J. Commun. Syst.* **2018**, *31*, e3384. [CrossRef]
3. Hosseinabadi, A.A.R.; Kardgar, M.; Shojafar, M.; Shamshirband, S.; Abraham, A. GELS-GA: Hybrid Metaheuristic Algorithm for Solving Multiple Travelling Salesman Problem. In Proceedings of the 14th International Conference on Intelligent Systems Design & Applications, Okinawa, Japan, 28–30 November 2014.
4. Rossi, F.L.; Nagano, M.S.; Neto, R.F.T. Evaluation of high performance constructive heuristics for the flow shop with makespan minimization. *Int. J. Adv. Manuf. Technol.* **2016**, *87*, 125–136. [CrossRef]
5. Lin, J. A hybrid discrete biogeography-based optimization for the permutation flow shop scheduling problem. *Int. J. Prod. Res.* **2015**, *54*, 4805–4814. [CrossRef]
6. Khatami, M.; Salehipour, A.; Hwang, F.J. Makespan minimization for the m-machine ordered flow shop scheduling problem. *Comput. Oper. Res.* **2019**, *111*, 400–414. [CrossRef]
7. Santucci, V.; Baioletti, M.; Milani, A. Algebraic Differential Evolution Algorithm for the Permutation Flowshop Scheduling Problem With Total Flowtime Criterion. *IEEE Trans. Evol. Comput.* **2016**, *20*, 682–694. [CrossRef]
8. Rossi, F.L.; Nagano, M.S.; Sagawa, J.K. An effective constructive heuristic for permutation flow shop scheduling problem with total flow time criterion. *Int. J. Adv. Manuf. Technol.* **2016**, *90*, 93–107. [CrossRef]
9. Abedinnia, H.; Glock, C.H.; Brill, A. New simple constructive heuristic algorithms for minimizing total flow-time in the permutation flowshop scheduling problem. *Comput. Oper. Res.* **2016**, *74*, 165–174. [CrossRef]
10. Deng, G.; Yang, H.; Zhang, S. An Enhanced Discrete Artificial Bee Colony Algorithm to Minimize the Total Flow Time in Permutation Flow Shop Scheduling with Limited Buffers. *Math. Probl. Eng.* **2016**, *2016*, 7373617. [CrossRef]
11. Schaller, J.; Valente, J.M.S. Heuristics for scheduling jobs in a permutation flow shop to minimize total earliness and tardiness with unforced idle time allowed. *Expert Syst. Appl.* **2019**, *119*, 376–386. [CrossRef]
12. Fernandez-Viagas, V.; Molina-Pariente, J.M.; Framinan, J.M. Generalised accelerations for insertion-based heuristics in permutation flowshop scheduling. *Eur. J. Oper. Res.* **2020**, *282*, 858–872. [CrossRef]
13. Ta, Q.C.; Billaut, J.-C.; Bouquard, J.-L. Matheuristic algorithms for minimizing total tardiness in the m-machine flow-shop scheduling problem. *J. Intell. Manuf.* **2015**, *29*, 617–628. [CrossRef]
14. Pagnozzi, F.; Stützle, T. Speeding up local search for the insert neighborhood in the weighted tardiness permutation flowshop problem. *Optim. Lett.* **2016**, *11*, 1283–1292. [CrossRef]
15. Al-Behadili, M.; Ouelhadj, D.; Jones, D. Multi-objective biased randomised iterated greedy for robust permutation flow shop scheduling problem under disturbances. *J. Oper. Res. Soc.* **2019**, *71*, 1847–1859. [CrossRef]
16. Valledor, P.; Gomez, A.; Priore, P.; Puente, J. Solving multi-objective rescheduling problems in dynamic permutation flow shop environments with disruptions. *Int. J. Prod. Res.* **2018**, *56*, 6363–6377. [CrossRef]
17. Xu, J.; Wu, C.-C.; Yin, Y.; Lin, W.-C. An iterated local search for the multi-objective permutation flowshop scheduling problem with sequence-dependent setup times. *Appl. Soft Comput.* **2017**, *52*, 39–47. [CrossRef]
18. Wang, X.; Tang, L. A machine-learning based memetic algorithm for the multi-objective permutation flowshop scheduling problem. *Comput. Oper. Res.* **2017**, *79*, 60–77. [CrossRef]

19. Li, X.; Ma, S. Multi-Objective Memetic Search Algorithm for Multi-Objective Permutation Flow Shop Scheduling Problem. *IEEE Access* **2016**, *4*, 2154–2165. [CrossRef]
20. González-Neira, E.M.; Urrego-Torres, A.M.; Cruz-Riveros, A.M.; Henao-García, C.; Montoya-Torres, J.R.; Molina-Sánchez, L.P.; Jiménez, J.-F. Robust solutions in multi-objective stochastic permutation flow shop problem. *Comput. Ind. Eng.* **2019**, *137*, 106026. [CrossRef]
21. Dubois-Lacoste, J.; Pagnozzi, F.; Stützle, T. An iterated greedy algorithm with optimization of partial solutions for the makespan permutation flowshop problem. *Comput. Oper. Res.* **2017**, *81*, 160–166. [CrossRef]
22. Karabulut, K. A hybrid iterated greedy algorithm for total tardiness minimization in permutation flowshops. *Comput. Ind. Eng.* **2016**, *98*, 300–307. [CrossRef]
23. Fernandez-Viagas, V.; Valente, J.M.S.; Framinan, J.M. Iterated-greedy-based algorithms with beam search initialization for the permutation flowshop to minimise total tardiness. *Expert Syst. Appl.* **2018**, *94*, 58–69. [CrossRef]
24. Fernandez-Viagas, V.; Framinan, J.M. A best-of-breed iterated greedy for the permutation flowshop scheduling problem with makespan objective. *Comput. Oper. Res.* **2019**, *112*, 104767. [CrossRef]
25. Rahman, H.F.; Janardhanan, M.N.; Nielsen, I.E. Real-Time Order Acceptance and Scheduling Problems in a Flow Shop Environment Using Hybrid GA-PSO Algorithm. *IEEE Access* **2019**, *7*, 112742–112755. [CrossRef]
26. Tasgetiren, M.F.; Liang, Y.-C.; Sevkli, M.; Gencyilmaz, G. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *Eur. J. Oper. Res.* **2007**, *177*, 1930–1947. [CrossRef]
27. Marinakis, Y.; Marinaki, M. Particle swarm optimization with expanding neighborhood topology for the permutation flowshop scheduling problem. *Soft Comput.* **2013**, *17*, 1159–1173. [CrossRef]
28. Wang, Z.; Zhang, J.; Yang, S. An improved particle swarm optimization algorithm for dynamic job shop scheduling problems with random job arrivals. *Swarm Evol. Comput.* **2019**, *51*, 100594. [CrossRef]
29. Chen, J.; Wang, M.; Kong, X.T.R.; Huang, G.Q.; Dai, Q.; Shi, G. Manufacturing synchronization in a hybrid flowshop with dynamic order arrivals. *J. Intell. Manuf.* **2017**, *30*, 2659–2668. [CrossRef]
30. Baykasoğlu, A.; Karaslan, F.S. Solving comprehensive dynamic job shop scheduling problem by using a GRASP-based approach. *Int. J. Prod. Res.* **2017**, *55*, 3308–3325. [CrossRef]
31. Moghaddam, S.K.; Saitou, K. On optimal dynamic pegging in rescheduling for new order arrival. *Comput. Ind. Eng.* **2019**, *136*, 46–56. [CrossRef]
32. Framinan, J.M.; Fernandez-Viagas, V.; Perez-Gonzalez, P. Using real-time information to reschedule jobs in a flowshop with variable processing times. *Comput. Ind. Eng.* **2019**, *129*, 113–125. [CrossRef]
33. Baker, K.R.; Altheimer, D. Heuristic solution methods for the stochastic flow shop problem. *Eur. J. Oper. Res.* **2012**, *216*, 172–177. [CrossRef]
34. Villarinho, P.A.; Panadero, J.; Pessoa, L.S.; Juan, A.A.; Oliveira, F.L.C. A simheuristic algorithm for the stochastic permutation flow-shop problem with delivery dates and cumulative payoffs. *Int. Trans. Oper. Res.* **2020**, *28*, 716–737. [CrossRef]
35. Liu, F.; Wang, S.; Hong, Y.; Yue, X. On the Robust and Stable Flowshop Scheduling Under Stochastic and Dynamic Disruptions. *IEEE Trans. Eng. Manag.* **2017**, *64*, 539–553. [CrossRef]
36. Valledor, P.; Gomez, A.; Priore, P.; Puente, J. Modelling and Solving Rescheduling Problems in Dynamic Permutation Flow Shop Environments. *Complexity* **2020**, *2020*, 2862186. [CrossRef]
37. Rahman, H.F.; Sarker, R.; Essam, D. Multiple-order permutation flow shop scheduling under process interruptions. *Int. J. Adv. Manuf. Technol.* **2018**, *97*, 2781–2808. [CrossRef]
38. Rahman, H.F.; Sarker, R.; Essam, D. A real-time order acceptance and scheduling approach for permutation flow shop problems. *Eur. J. Oper. Res.* **2015**, *247*, 488–503. [CrossRef]
39. Shiue, Y.R.; Lee, K.C.; Su, C.T. Real-time scheduling for a smart factory using a reinforcement learning approach. *Comput. Ind. Eng.* **2018**, *125*, 604–614. [CrossRef]
40. Hosseinabadi, A.A.R.; Siar, H.; Shamshirband, S.; Shojafar, M.; Nasir, M.H.N.M. Using the gravitational emulation local search algorithm to solve the multi-objective flexible dynamic job shop scheduling problem in Small and Medium Enterprises. *Ann. Oper. Res.* **2014**, *229*, 451–474. [CrossRef]
41. Liu, W.; Jin, Y.; Price, M. New scheduling algorithms and digital tool for dynamic permutation flowshop with newly arrived order. *Int. J. Prod. Res.* **2017**, *55*, 3234–3248. [CrossRef]
42. Li, G.; Li, N.; Sambandam, N.; Sethi, S.P.; Zhang, F. Flow shop scheduling with jobs arriving at different times. *Int. J. Prod. Econ.* **2018**, *206*, 250–260. [CrossRef]
43. Liu, W.; Jin, Y.; Price, M. New meta-heuristic for dynamic scheduling in permutation flowshop with new order arrival. *Int. J. Adv. Manuf. Technol.* **2018**, *98*, 1817–1830. [CrossRef]
44. Luo, S. Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. *Appl. Soft Comput.* **2020**, *91*, 106208. [CrossRef]
45. Zhang, Z.C.; Wang, W.P.; Zhong, S.Y.; Hu, K.S. Flow Shop Scheduling with Reinforcement Learning. *Asia Pac. J. Oper. Res.* **2013**, *30*, 1350014. [CrossRef]
46. Lin, C.C.; Deng, D.J.; Chih, Y.L.; Chiu, H.T. Smart Manufacturing Scheduling With Edge Computing Using Multiclass Deep Q Network. *IEEE Trans. Ind. Inform.* **2019**, *15*, 4276–4284. [CrossRef]

47. Liu, C.L.; Chang, C.C.; Tseng, C.J. Actor-Critic Deep Reinforcement Learning for Solving Job Shop Scheduling Problems. *IEEE Access* **2020**, *8*, 71752–71762. [CrossRef]
48. Zhang, C.; Song, W.; Cao, Z.; Zhang, J.; Tan, P.S.; Xu, C. Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning. *arXiv* **2020**, arXiv:2010.12367.
49. Han, B.-A.; Yang, J.-J. Research on Adaptive Job Shop Scheduling Problems Based on Dueling Double DQN. *IEEE Access* **2020**, *8*, 186474–186495. [CrossRef]
50. Wang, H.X.; Sarker, B.R.; Li, J.; Li, J. Adaptive scheduling for assembly job shop with uncertain assembly times based on dual Q-learning. *Int. J. Prod. Res.* **2020**, 1–17. [CrossRef]
51. Shiue, Y.-R.; Lee, K.-C.; Su, C.-T. A Reinforcement Learning Approach to Dynamic Scheduling in a Product-Mix Flexibility Environment. *IEEE Access* **2020**, *8*, 106542–106553. [CrossRef]
52. Zhang, Z.C.; Zheng, L.; Li, N.; Wang, W.P.; Zhong, S.Y.; Hu, K.S. Minimizing mean weighted tardiness in unrelated parallel machine scheduling with reinforcement learning. *Comput. Oper. Res.* **2012**, *39*, 1315–1324. [CrossRef]
53. Jun, S.; Lee, S. Learning dispatching rules for single machine scheduling with dynamic arrivals based on decision trees and feature construction. *Int. J. Prod. Res.* **2020**, *20*, 1–19. [CrossRef]
54. Wu, C.H.; Zhou, F.Y.; Tsai, C.K.; Yu, C.J.; Dauzere-Peres, S. A deep learning approach for the dynamic dispatching of unreliable machines in re-entrant production systems. *Int. J. Prod. Res.* **2020**, *58*, 2822–2840. [CrossRef]
55. Li, Y.Y.; Carabelli, S.; Fadda, E.; Manerba, D.; Tadei, R.; Terzo, O. Machine learning and optimization for production rescheduling in Industry 4.0. *Int. J. Adv. Manuf. Technol.* **2020**, *110*, 2445–2463. [CrossRef]
56. Chen, R.; Yang, B.; Li, S.; Wang, S. A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem. *Comput. Ind. Eng.* **2020**, *149*, 106778. [CrossRef]
57. Wang, K.; Luo, H.; Liu, F.; Yue, X. Permutation Flow Shop Scheduling With Batch Delivery to Multiple Customers in Supply Chains. *IEEE Trans. Syst. ManCybern. Syst.* **2018**, *48*, 1826–1837. [CrossRef]
58. Yang, S.; Xu, Z. The distributed assembly permutation flowshop scheduling problem with flexible assembly and batch delivery. *Int. J. Prod. Res.* **2020**, 1–19. [CrossRef]
59. Wang, K.; Ma, W.Q.; Luo, H.; Qin, H. Coordinated scheduling of production and transportation in a two-stage assembly flowshop. *Int. J. Prod. Res.* **2016**, *54*, 6891–6911. [CrossRef]
60. Kazemi, H.; Mazdeh, M.M.; Rostami, M. The two stage assembly flow-shop scheduling problem with batching and delivery. *Eng. Appl. Artif. Intell.* **2017**, *63*, 98–107. [CrossRef]
61. Basir, S.A.; Mazdeh, M.M.; Namakshenas, M. Bi-level genetic algorithms for a two-stage assembly flow-shop scheduling problem with batch delivery system. *Comput. Ind. Eng.* **2018**, *126*, 217–231. [CrossRef]
62. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous Methods for Deep Reinforcement Learning. In Proceedings of the 33rd International conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 1928–1937.
63. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv* **2013**, arXiv:1312.5602.
64. Van Hasselt, H.; Guez, A.; Silver, D. Deep reinforcement learning with double q-learning. *arXiv* **2015**, arXiv:1509.06461.
65. Pan, Q.-K.; Gao, L.; Xin-Yu, L.; Jose, F.M. Effective constructive heuristics and meta-heuristics for the distributed assembly permutation flowshop scheduling problem. *Appl. Soft Comput.* **2019**, *81*, 105492. [CrossRef]