Received: 14 April 2022 | Revised: 10 July 2022 | Accepted: 26 July 2022

IET Collaborative Intelligent Manufacturing

DOI: 10.1049/cim2.12060

The Institution of Engineering and Technology WILEY

ORIGINAL RESEARCH

A deep reinforcement learning based approach for dynamic distributed blocking flowshop scheduling with job insertions

Xueyan Sun[1,2] | Birgit Vogel-Heuser[1] | Fandi Bi[1] | Weiming Shen[2]

[1]Department of Mechanical Engineering, Chair of Automation and Information Systems, Technical University Munich, Munich, Germany

[2]School of Mechanical Science and Engineering, Huazhong University of Science and Technology, Wuhan, China

Correspondence

Xueyan Sun, Department of Mechanical Engineering, Chair of Automation and Information Systems, Technical University Munich, Boltzmannstr. 15, 85748 Munich, Germany.
Email: sunxueyanxy@163.com

Funding information

China Scholarship Council, Grant/Award Number: NO. 202006160113

Abstract

The distributed blocking flowshop scheduling problem (DBFSP) with new job insertions is studied. Rescheduling all remaining jobs after a dynamic event like a new job insertion is unreasonable to an actual distributed blocking flowshop production process. A deep reinforcement learning (DRL) algorithm is proposed to optimise the job selection model, and local modifications are made on the basis of the original scheduling plan when new jobs arrive. The objective is to minimise the total completion time deviation of all products so that all jobs can be finished on time to reduce the cost of storage. First, according to the definitions of the dynamic DBFSP problem, a DRL framework based on multi-agent deep deterministic policy gradient (MADDPG) is proposed. In this framework, a full schedule is generated by the variable neighbourhood descent algorithm before a dynamic event occurs. Meanwhile, all newly added jobs are reordered before the agents make decisions to select the one that needs to be scheduled most urgently. This study defines the observations, actions and reward calculation methods and applies centralised training and distributed execution in MADDPG. Finally, a comprehensive computational experiment is carried out to compare the proposed method with the closely related and well-performing methods. The results indicate that the proposed method can solve the dynamic DBFSP effectively and efficiently.

KEYWORDS

deep reinforcement learning, distributed blocking flowshop scheduling problem, dynamic scheduling, job insertions, multi-agent deep deterministic policy gradient

1 | INTRODUCTION

Production scheduling is one of the most critical issues in manufacturing systems and has been extensively studied in the literature [1]. As a typical production scheduling problem, flowshop scheduling problem (FSP) plays an important role in modern manufacturing systems. In the classic FSP, there is an infinite buffer capacity between successive machines. However, in many real-world manufacturing environments, such as pharmaceutical workshops, the buffer capacity between machines is limited or even zero due to technical requirements or characteristics of the jobs. Under this circumstance, FSP turns to be the Blocking Flowshop Scheduling Problem (BFSP). At the same time, compared with the traditional single-factory production and processing mode, the distributed manufacturing system makes full use of the resources of multiple factories, realises the overall optimisation through the effective allocation of resources, and quickly achieves the goal of manufacturing at the lowest cost [2]. Therefore, distributed blocking flowshop scheduling problem (DBFSP) has become one of the active research topics in the field of manufacturing scheduling. When the number of machines is greater than two, BFSP has been proved to be a typical NP-hard (non-deterministic Polynomial-hard) combinatorial optimisation problem [3]. Blocking Flowshop Scheduling Problem is a subproblem of DBFSP, and therefore, DBFSP is also an NP-hard optimisation problem.

In real production, the processing environment changes dynamically, and the manufacturing system is often affected by various disturbing events, such as machine failures, new job insertions, worker lateness and absences due to illnesses, which will lead to the low efficiency of the original scheduling scheme. When a dynamic disturbance occurs, the original processing plan needs to be modified to adapt to the new production environment to ensure the continuous execution of the manufacturing system. Therefore, establishing a dynamic DBFSP scheduling system is closer to real production. There are two kinds of dynamic scheduling: one is real-time scheduling, which means that there is no static scheduling scheme in advance, and the processing priority of a workpiece is determined directly by the status of the jobs and equipment in the production system; the other one is that based on the original plan and the dynamic status of the production system; the static scheduling plan will be adjusted in real time to determine the processing priority of the jobs, which is called rescheduling. Both these methods can obtain executable scheduling decisions, but the effectiveness is different. Real-time scheduling often only considers local information in decision-making, so the scheduling results obtained may be feasible, but they are still far from the optimisation scheme. Rescheduling is adjusting the existing static scheduling scheme to ensure its operability. Since the static scheduling scheme is often obtained by considering multiple performance indicators through global optimisation methods, rescheduling can obtain more optimal dynamic scheduling results. In this paper, we combine these two ways to dynamically repair the original static schedule by the status of the jobs and equipment in the production system in order to make a more efficient decision.

Typical scheduling methods can be divided into two categories: precise scheduling methods and approximate scheduling methods. Precise methods are not efficient at solving large-scale scheduling problems because of their high computational complexities. Compared with precise scheduling methods, approximate methods do not search the whole solution space but explore multiple directions based on particular strategies [4]. So, the approximate methods have lower computational complexity and obtain feasible solutions more quickly, while having more advantages in solving large-scale scheduling problems. Approximate approaches for the dynamic scheduling problem include variable neighbourhood search [5], learning-based approaches [6], heuristic algorithms [7], and agent-based methods [8]. Gao et al. [9] proposed a discretisation strategy in the Jaya algorithm to four bi-objective flexible job-shop rescheduling (FJRP) optimisation cases under the condition of new job insertion. An et al. [10] investigated a multi-objective FJRP with both new job insertion and machine preventive maintenance using an improved non-dominated sorting genetic algorithm III with an adaptive reference vector (NSGA-III/ARV). Fu et al. [11] proposed some new artificial molecules to construct and inject into the population adaptively to escape from a local optimum in FSP. They [12] also developed a hybrid multi-objective optimisation algorithm that maintained two populations, executing the global search in the whole solution space and the local search in promising regions to address a dual-objective stochastic hybrid flow shop deteriorating scheduling problem. Zhao et al. [13] applied the memetic algorithm that integrated a population-based non-dominated sorting genetic algorithm II and two single-solution-based improvement methods to solve a bi-objective serial-batch group scheduling problem considering the constraints of sequence-dependent setup time, release time, and due time. Also, Zhao [14] summarised iterated greedy algorithm (IGA) variants and hybrid algorithms with IGA integrated for solving FSPs according to their scheduling scenarios, objective functions, and constraints. But the most approximate methods that are applied in dynamic rescheduling must consider all information and make a totally new schedule instead of adjusting the original plan. For the learning-based methods, they can be adapted to different circumstances flexibly and more suitably to make decisions according to changing environments.

In recent years, reinforcement learning (RL) has emerged as a powerful method to deal with the Markov Decision Process [15]. For actual manufacturing systems, a real-time scheduling algorithm based on RL is worth investigation [16]. Due to the ability of RL to learn the best action at each decision point and react to dynamic events completely in real time, many RL-based methods have been applied to different kinds of dynamic scheduling problems. For single-agent RL methods, Riedmiller [17] proposed an RL approach to learn local scheduling policies in the workshop with the objective of reducing the total latency. A neural network-based agent is associated with each resource and trained via Q-learning. Compared with ordinary heuristic scheduling rules, it shows better performance. Aydin and Öztemel [18] developed an improved Q-learning-based algorithm, called Q-III, training an agent to choose the most appropriate scheduling rules in real-time for dynamic job shops with new job insertions. Wang and Usher [19] used Q-learning to train a single machine agent that selected the best scheduling rule among the three given rules to minimise the average tardiness. Chen et al. [20] proposed a rule-driven approach to develop compound scheduling rules for multi-objective dynamic job shop scheduling. Other researchers applied deep RL approaches to dynamic scheduling problems. Luo [21] studied the dynamic flexible shop floor scheduling problem and proposed seven state indicators, and six dispatch rules, and adopted the double deep Q network. Martin [22] used two heuristic algorithms, Randomised Nawaz-Enscore-Ham and Randomised Clarke Wright Savings (RandCWS), for the permutation FSP and capacitated the vehicle routing problem to increase the autonomy of multiple agents and obtain the current best solution through self-optimisation and communication between multiple agents. Cao et al. [23] applied a Cuckoo Search algorithm with RL and surrogate modelling to solving a semiconductor final testing problem and proposed a parameter control scheme based on RL, which can improve the search efficiency and well balance the diversification and intensification of population. Zhang [24] took corresponding actions according to the status of dynamic occurrence (idle machines with new jobs or machines with surrogate processing jobs) and introduced a feedforward neural network to map state-action pairs to their values based on a simulation-based value iteration algorithm and simulation-based Q-learning algorithm. Zhou et al. [25] gave the mathematical description of the dynamic scheduling in smart manufacturing problem and presented a deep reinforcement learning (DRL)-based framework to minimise the makespan of all tasks over time. Kim et al. [26] presented a smart manufacturing system using a Multi-Agent System (MAS) and RL. Machines with intelligent agents evaluate the priorities of jobs and distribute them through negotiation. Han et al. [27] proposed an end-to-end DRL framework based on 3D disjunctive graph dispatching. They improved the pointer network and trained the policy with 20 static features and 24 dynamic features that described the full picture of scheduling problem. Zinn et al. [28] applied DQN in the control of PLC-based automated production systems to process workpieces depending on their colours and fulfil the schedule automatically. Luo [29] proposed an on-line rescheduling framework named as two-hierarchy DQN for the dynamic multi-objective flexible job shop scheduling problem with new job insertions. The higher-level DQN was a controller determining the temporary optimisation goal for the lower DQN. At each rescheduling point, it took the current state features as input and chose a feasible goal to guide the behaviour of the lower DQN. The lower-level DQN acted as an actuator. It took the current state features together with the higher optimisation goal as input and chose a proper dispatching rule to achieve the given goal. Baer et al. [30] applied the Multi-Agent RL (MARL) version of DQN without explicit exchange of information between agents, with agents who had learnt to guide products efficiently through the plant and achieve near-optimal timing regarding resource allocation. Zinn et al [31] addressed the waypoint-based exploration with Hierarchical RL to the domain of robotic devices. The resulting algorithm utilised a top-level policy, which suggested waypoints to a bottom-level policy that controls the system actuators. Gankin et al. [32] introduced a method for Modular Production Control that combines a DQN and MAS, proving that the approach can foster advantages from both research fields and at the same time showed that the MAS achieves optimal throughput in a static production environment.

To sum up, most researchers tend to apply traditional RL algorithms such as Q learning and DQN to solve the manufacturing scheduling problem. At present, many other RL algorithms with great performance have been designed, such as Deep Deterministic Policy Gradient (DDPG) and Proximal Policy Optimization. But they are less used in manufacturing scheduling problems. Meanwhile, the application of RL is mostly in a single workshop, and there is little research on the scheduling problem of multiple workshops. Furthermore, most actions of RL are dispatch rules, which reduce the selection space. This paper focuses on the DBFSP dynamic scheduling problem with new job insertions by applying the multi-Agent DRL (MADRL). Also, we combine the static optimisation with dynamic DRL-based schedule repair, which is closer to the real production.

The rest of the paper is organised as follows: Section 2 describes the considered problem. Section 3 presents the proposed DRL scheduling framework. Section 4 provides detailed information about the proposed training method. Section 5 shows a series of computational experiments and comparisons. Finally, the conclusion and future research are discussed in Section 6.

2 | PROBLEM DESCRIPTION

The problem to be addressed in this paper is a dynamic scheduling problem with new job insertions. There is a full schedule plan before dynamic events happen. Therefore, this section introduces both the static and dynamic problems.

2.1 | DBFSP problem statement

The DBFSP is an extension of the BFSP, where multiple blocking factories are scheduled simultaneously. The problem is described as follows: There are n workpieces $\{J1, J2, \ldots, Jn\}$, and each job includes S consecutive processing stages (workstations), but different jobs may have different process times in different stages. At the same time, all these jobs can be assigned to f factories that have identical blocking assembly lines. There is no buffer between successive stages; jobs cannot leave the current station until the next station is available for processing. With the goal of total completion time deviation minimisation shown in Formula (1), all the jobs are assigned to different factories, the processing orders of the jobs in different factories are determined, and finally, the start and end times of each process are obtained. The DBFSP is abstracted from real manufacturing scheduling problems. The following assumptions are made: each machine is available at time zero, and the f processing factories are exactly the same, including the machines and their numbers, There is no time constraint between successive operations.

The DBFSP model is formed based on the BFSP model. This paper adopts a sequence-based modelling method and establishes constraint conditions by expressing the connections between processed jobs. Notations in this model are as follows:

Notations:

N	Total number of jobs
S	Total number of stages in each factory
F	Total number of factories
j	Index of job, $j = 1, 2,. . ., N$
k	Index of stage, $k = 1, 2,. . ., S$
f	Index of factory, $f = 1, 2,. . ., F$
$p_{j,k}$	Processing time of job j at stage k
L	A very large positive number
r_k	Release time of stage k
C_j	Completion time of job j
DT_j	Due time of job j
$st_{j,k}$	Start time of job j at stage k
$e_{j,k}$	End time of job j at stage k

The objective is to minimise the total completion time deviation that refers to the sum of the absolute values of all jobs completion times minus the due times:

$$MIN \sim C_d = \sum_{j=1}^{N} abs(C_j - DT_j) \tag{1}$$

Decision variables are:

- Starting process time of job j at stage k: $s_{j,k}$
- Ending process time of job j at stage k: $e_{j,k}$
- If job j is assigned to factory f before job j':

$$y_{f,j,j'} = \begin{cases} 1 \text{ if job } j \text{ precedes job j}' \text{at factory } f \\ 0 \quad \text{otherwise} \end{cases} \tag{2}$$

- If job j is assigned to factory f:

$$z_{j,f} = \begin{cases} 1 \text{ if job } j \text{ is processed on factory } f \\ 0 \quad \text{otherwise} \end{cases} \tag{3}$$

Constraints are:

- For any job j, it can only be assigned to one factory:

$$\sum_{f=1}^{F} z_{j,f} = 1,\ j = 1, 2, \ldots, N \tag{4}$$

- For any job j, the start time of the first stage must be greater than or equal to 0:

$$st_{1,j} \geq 0,\ j = 1, 2, \ldots, N \tag{5}$$

- For any job j, the start time from the second stage is equal to the maximum between stage k release time and end time of the last stage:

$$st_{j,k} = \max\{r_k, e_{j,k-1}\}, j = 1, 2, \ldots, N, k = 2, \ldots, S \tag{6}$$

- For any stage k except last stage, the release time is equal to the maximum between the job j end time of stage k and the job j–1 end time of stage $k + 1$:

$$r_k = \max\{e_{j,k}, e_{j-1,k+1}\}, j = 2, \ldots, N, k = 1, 2, \ldots, S-1 \tag{7}$$

- For any job j, the start time of the next stage must be greater than or equal to the end time of the previous stage:

$$st_{k+1,j} - st_{k,j} \geq p_{k,j},\ j = 1, 2, \ldots, N, k = 1, 2, \ldots, S-1 \tag{8}$$

- The sequences of two jobs at the same factory are determined by:

$$y_{f,j,j'} + y_{f,j',j} \leq 1, j, j' = 1, 2, \ldots, N, j \neq j', f = 1, 2, \ldots, F \tag{9}$$

- Time constraints to ensure that two jobs are in the same factory at the same stage:

$$s_{k,j'} - \left(st_{k,j} + p_{k,j}\right) + L \times \left(3 - z_{j,f} - z_{j',f} - y_{f,j,j'}\right) \geq 0 \tag{10}$$

2.2 | Dynamic DBFSP problem statement

In this paper, we consider new job insertions as the dynamic event. Based on the full schedule, several factories need to choose new jobs to own waiting queues. The problem can be described as follows: when several factories are running with the determined schedule plans, there are m new jobs $\{NJ1, NJ2, \ldots, NJm\}$ arriving randomly that need to be assigned to factories at the decision points. The decision points are the set of the first stage's ending time of all jobs after the first new job insertion. And the optimisation goal is to minimise the total completion time deviation that is shown in Formula (1).

To describe the problem more clearly, this part uses an example to illustrate. We can see from Figure 1 that there are three identical factories with three stages including s1, s2 and s3. Eight jobs are assigned to three factories by the variable neighbourhood descent (VND) algorithm before new job insertions. At time t0, a new job nine arrives, which must be arranged at next decision point. Each factory can calculate the decision point according to the time t0 and decide to choose new job or not if the new job is available. When the decision point comes to the t1 of factory 1, job nine is chosen to the factory and processed after job seven immediately.

3 | DRL SCHEDULING FRAMEWORK

The proposed scheduling framework indicates the whole process of the proposed method. It includes three parts: DBFSP virtual environment (DVE), offline training process, and new instance evaluation, as shown in Figure 2.

Distributed blocking flowshop scheduling problem virtual environment: this part defines the DBFS process. Each episode resets the virtual world randomly such as number of jobs, number of stages, process time and due time. According to such information, the VND algorithm can be used to generate a full schedule for all factories. After the beginning of the process, the virtual environment inserts new jobs randomly, and DVE will update the observations of agents and the global states and transfer this information to agents. According to actions from the agents, DVE chooses the first job in the new job queue for all factories and enters into a new state. The process is repeated until the new job pool is empty and a new schedule is generated. Such a complete process is called an episode.

Offline training process: in this phase, each factory has a DDPG agent. The observations of each agent are input to the actor network and get an action that will provide feedback to DVE and move to the next state. The critic network is used to

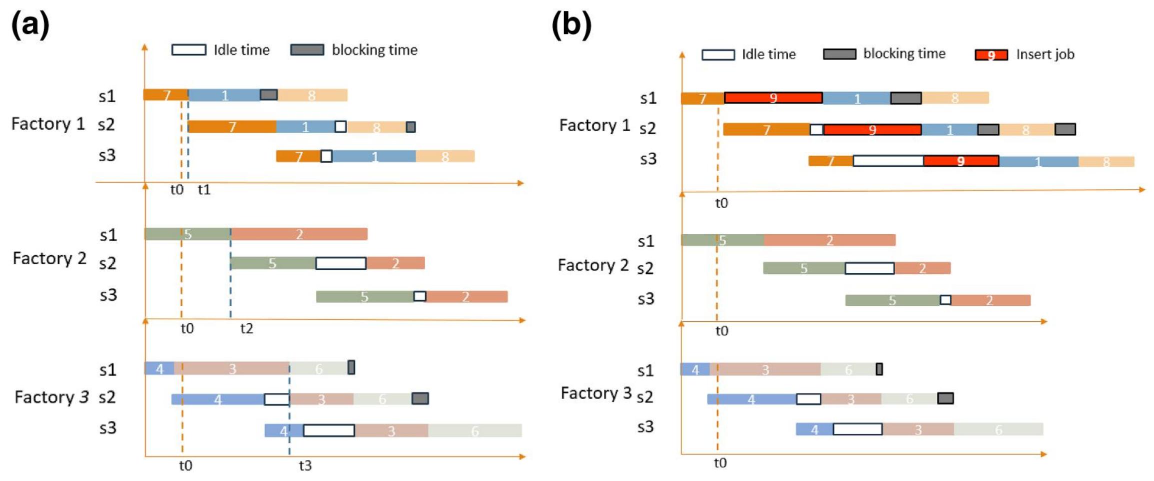


FIGURE 1 (a) Before job insertion (b) After job insertion

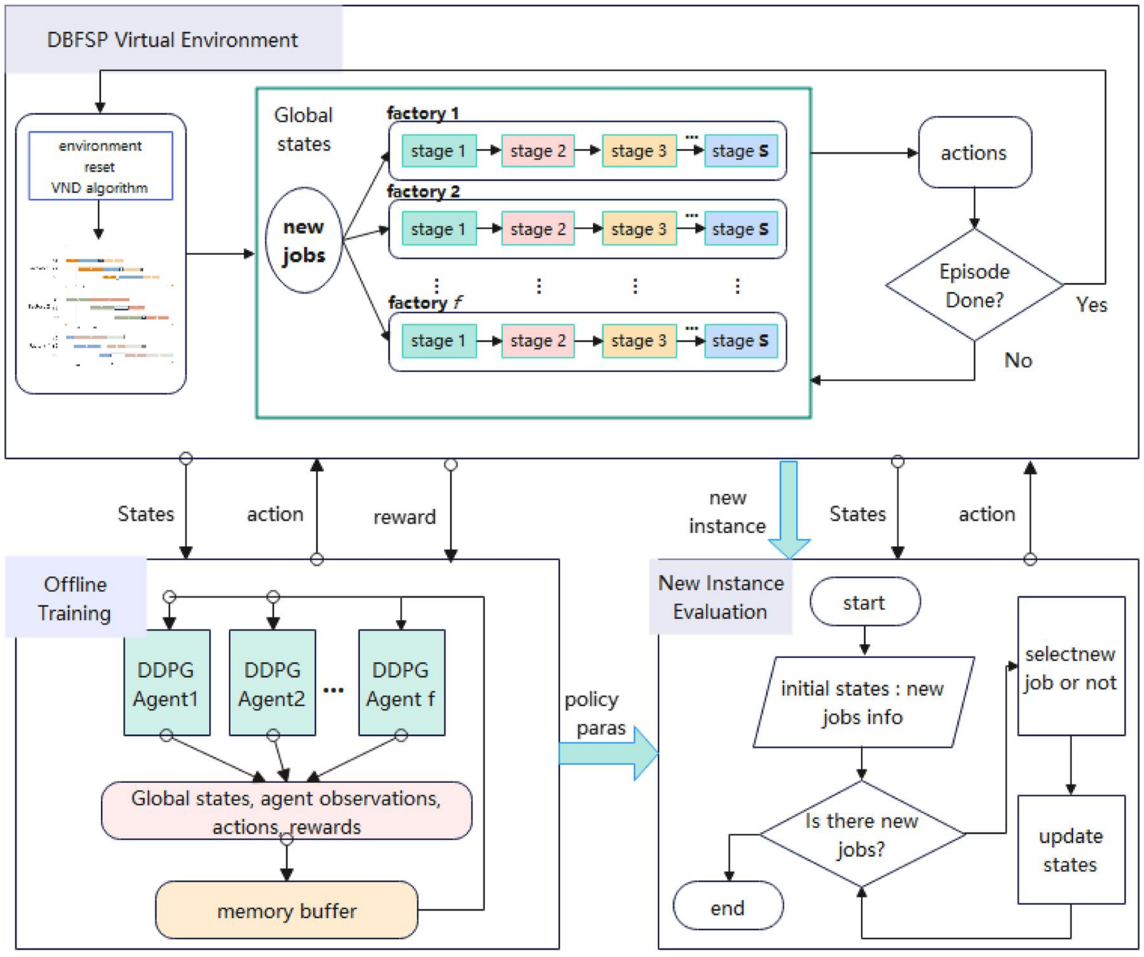


FIGURE 2 Deep reinforcement learning (DRL) scheduling framework of distributed blocking flowshop scheduling problem (DBFSP)

evaluate the pair of state and action from the global view so that all agents can learn cooperatively.

New instance evaluation: after a long period of the training phase, the optimal policies of agents can be obtained. Some new scheduling problem instances can be solved using this model in a short time.

4 | METHOD FOR SOLVING DBFSP

In this section, we introduce the multi-agent deep deterministic policy gradient (MADDPG) algorithm into our solving process. Before executing MADDPG, there should be a full schedule and a sorted new job queue.

4.1 | Full schedule and new jobs sorting

To generate a full schedule before new job insertions, this paper uses an improved VND algorithm from our previous research. There, we use the Insertion, Move and Exchange local search methods. The process of VND is shown in Figure 3.

Insertion: this method is used in all factories. First, there are n_f jobs in the original order of factory f and $n_f - 1$ jobs are selected randomly from the original orders. Then, insert selected jobs one by one to all possible positions of the original order. Finally, get the best inserted orders for individual factories.

Move: this operation is applied between the factory with maximum on-time cost (*fmax*) and the one with minimum on-time cost (*fmin*). The Move operation is to move each job in *fmax* to all insertable positions in *fmin* to find the optimal arrangement.

Exchange: this operation is applied between the factory with maximum on-time cost (*fmax*) and the one with minimum on-time cost (*fmin*). Exchange is to exchange each job in *fmax* with all jobs in *fmin* to find the optimal arrangement.

To build a more adaptive solving policy network, for each decision point, only one new job is left to be chosen so that no matter how many new jobs come in, this model does not have to be learnt again. Therefore, before making a decision, all insert jobs need to be sorted into a new list with the first job being the most urgent to be arranged.

The proposed insert jobs sorting method is shown in Figure 4. At first, according to process times and due times of all jobs DT and decision time U, calculate the left times (H) of all jobs. H of job j can be calculated as follows:

$$H_j = U - DT_j - \sum_{k \in S} p_{j,k} \tag{11}$$

Then, sorting all jobs in an ascending order by H, start the insertion operation in the same way as we mentioned above in the VND algorithm. Stop the insertion until all jobs fall into all positions.

VND algorithm

```
Input ←process time, due time, random schedule
Output← full schedule
Flag←True
  Foreach factory f
        perform Insertion on f
  end
  While flag do
        perform Move between fmax and fmin
        perform Exchange between fmax and fmin
        If there is any improvement on f
              perform Insertion on f
        else
              Flag←false
        end
  end
```

FIGURE 3 Variable neighbourhood descent (VND) algorithm

Insert jobs sorting

```
Input ←process time, due time, decesion time
Output ←final new inserts job order
Calculating all insert jobs left Time (LT)
Sorting jobs in ascending order by LT
Choosing re-insert jobs, left one job as the new job order
While re-insert jobs is not empty do
      perform Insertion on new job order
end
Output final new job order
```

FIGURE 4 Insert jobs sorting method

4.2 | DRL by multi-agent DDPG

After getting a full schedule and sorting the insertion jobs, this paper applies MADDPG to learn the model of making decisions. This section will introduce MADDPG in detail.

4.2.1 | Multi-agent deep deterministic policy gradient

In RL research, DDPG followed a development path of PG → DPG → DDPG. Sutton [33] proposed PG in 2000, which is a classic action-control method for continuous learning. The solving strategy in PG is as follows: using a probability distribution function $\pi_\theta(a|s)$ to represent the optimal strategy for each step and performing action sampling according to the probability distribution at each step to obtain the current optimal action value $Q^\pi(s, a)$. The process of generating an action is essentially a random process and the learnt policy is also a stochastic policy. The gradient of the policy can be written as follows:

$$\nabla_\theta J(\theta) = \mathrm{E}_{s \sim p^\pi, a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a)] \tag{12}$$

The stochastic strategy is obtained, but at each step, we still need to sample the obtained optimal strategy probability distribution in order to obtain the specific action. Also, at most conditions, an action is usually a high-dimensional vector so that frequent sampling of action space will undoubtedly consume computing power. Silver et al. [34] proposed DPG in 2014, which is a deterministic behaviour policy, and the behaviour of each step directly obtains a definite value through the policy $\mu_\theta(a|s)$:

This function, μ, is the optimal behaviour policy, which is no longer a stochastic policy that needs to be sampled. The gradient of the policy can be written as follows:

$$\nabla_\theta J(\theta) = \mathrm{E}_{s\sim D}\left[\nabla_\theta \mu_\theta(a|s)\nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}\right] \tag{13}$$

Based on DPG, Lillicrap proposed Deep DPG in 2016 [35], which is a strategy learning method that integrates deep learning neural networks with DPG. The main improvement over DPG is: using the convolutional neural network as the simulation of policy function μ and Q function, namely policy network and Q network, and then using the deep learning method to train the above neural network.

The implementation and training method of the Q function adopts the DQN method published by Mnih in 2015 [36].

All these methods are applied in single agent problems. If we use them to learn policies in a multi-agent environment without modifications, all agents will continuously learn and improve their policies independently. A decision made by one agent will change the states of the environment immediately so that the environment becomes unstable, changed with every single decision, which does not meet the traditional RL convergence conditions. And to a certain extent, it is impossible to adapt to a dynamically unstable environment by simply changing the agent's own strategy. Due to the instability of the environment, the key skills of DQN, such as experience replay, will not be used directly.

Multi-agent deep deterministic policy gradient overcomes these difficulties because the MADDPG algorithm has the following two characteristics [37]:

(1) Centralised training and distributed execution. Centralised learning is used to train critics and actors during training, and actors can run only by knowing local information. The critic needs the policy information of other agents.
(2) Improved experience replay buffer data. In order to be applicable to dynamic environments, each piece of information consists of $(x, a_1, \dots, a_n, x', r)$, $x = (o_1, o_2, \dots, o_n)$, representing the observations of all agents.

4.2.2 | MADDPG training method

We use $\theta = [\theta_1, \theta_2, \dots, \theta_n]$ to represent the policy parameters of n agents and $\pi = [\pi_1, \dots, \pi_n]$ to represent the policies of n agents. For Agent i, the accumulated reward expectation is as follows:

$$J(\theta_i) = \mathrm{E}_{s\sim p^\pi, a\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma^t r_{i,t}\right] \tag{14}$$

And the gradient of the policy can be written as follows:

$$\nabla_{\theta_i} J(\theta_i) = \mathrm{E}_{s\sim p^\pi, a_i\sim\pi_i}\left[\nabla_{\theta_i}\log\pi_i(a_i|o_i)Q_i^\pi(x, a_1, \dots, a_n)\right] \tag{15}$$

where o_i is the observations of agent i; $x = (o_1, o_2, \dots, o_n)$ is an observation vector, also called global states; $Q_i^\pi(x, a_1, \dots, a_n)$ is the centralised states-action value function of Agent i. Extending the stochastic strategy to deterministic strategy μ_{θ_i}, the gradient of the policy can be written as follows:

$$\nabla_{\theta_i} J(\mu_i) = \mathrm{E}_{x,a\sim D}\left[\nabla_{\theta_i}\mu_i(a_i|o_i)\nabla_{ai} Q_i^\mu(x, a_1, \dots, a_n)|_{a_i=\mu_i(o_i)}\right] \tag{16}$$

Here, the experience replay buffer D contains the tuples $(x, a_1, \dots, a_n, x', r)$ for the batch learning. The update of the centralised states-action value function is the same as the DQN, using temporal difference error, which can be written as follows:

$$L(\theta_i) = \mathrm{E}_{x,a,r,x'}\left[\left(Q_i^\mu(x, a_1, \dots, a_n) - y\right)^2\right], y = r_i + \gamma Q_i^{\mu'}\left(x', a'_1, \dots, a'_n\right)\Big|_{a'_j=\mu'_j(o_j)} \tag{17}$$

where $\mu' = \left[\mu_{\theta_1'}, \dots, \mu_{\theta n'}\right]$ is the set of target policies with delayed parameters θ_i'.

The centralised training and distributed execution process of MADDPG can be shown in Figure 5. All training data are selected randomly from the experience replay buffer. Every agent has four networks to learn, including actor network, critic network, and two target networks. According to observations *obs* of the agent, the actor gives an action A that will be executed by an agent. Combinations of (*obs*, A) of all agents are taken as inputs of the critic, computing a Q value. Meanwhile, target actor could get an action A′ with observations *obs*' that is used to obtain the Q′ from the target critic. Then, parameters of the critic network can be updated using Formula (17), with the gradient descent method. The actor network aims at finding an action with the biggest Q value so that it updates parameters by using the gradient ascent method. Finally, a soft update method is used for two target networks in every training process.

At last, the whole pseudocode of the proposed learning algorithm is given in Figure 6. The learning process runs M episodes. And at start of each episode, the environment resets randomly and gets an original schedule by VND. After that, when the episode does not end, actions will be chosen and executed by agents. New information of $(x, a_1, \dots, a_n, x', r)$ is stored in the replay buffer. At last, the network parameters are updated by a mini-batch of examples until the end of the episode (Figure 6).

4.2.3 | Definitions for states, observations, actions, rewards and networks

According to our problem description, agents have to make a decision to execute an insert job or execute the original schedule. Therefore, we get our action's definition for each agent, which is choosing one job between the first one in the agent waiting line and the one in the insertion pool.

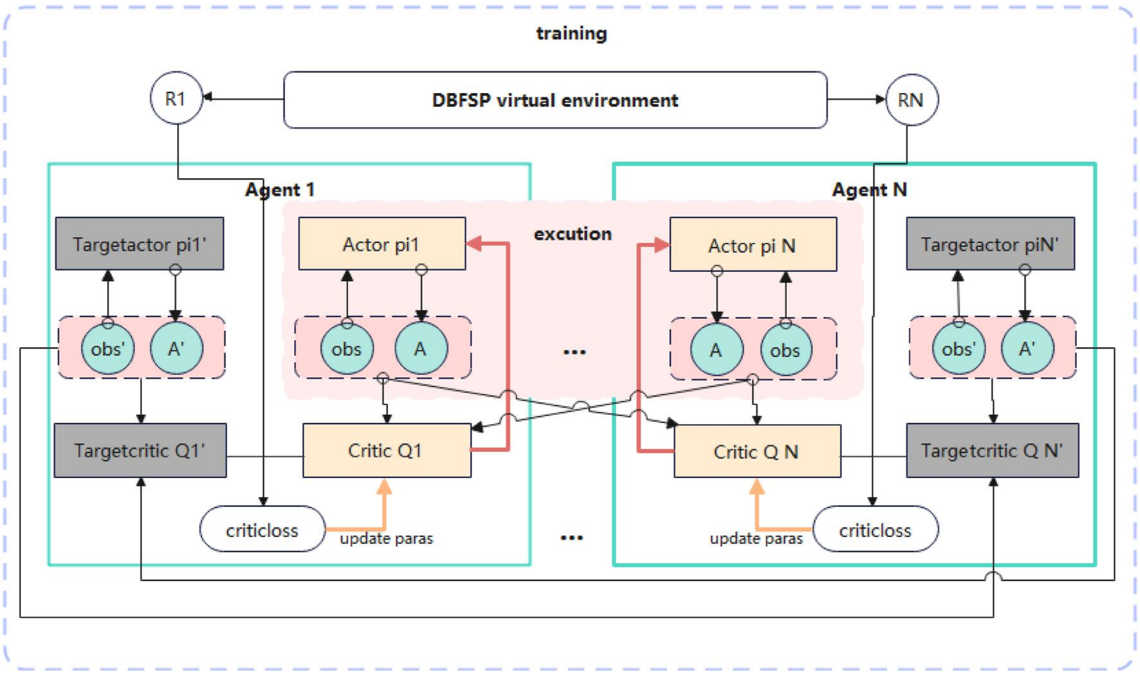
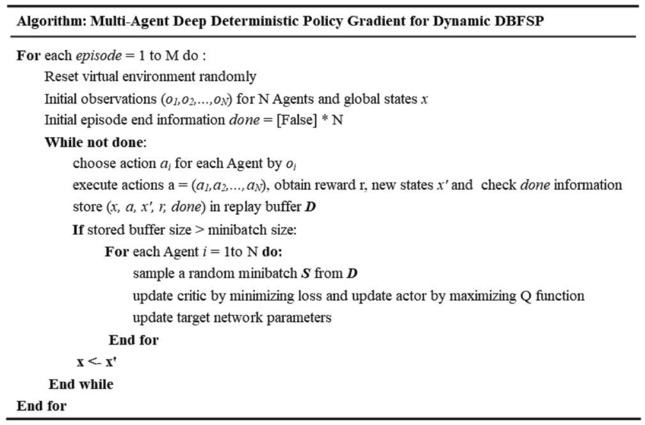
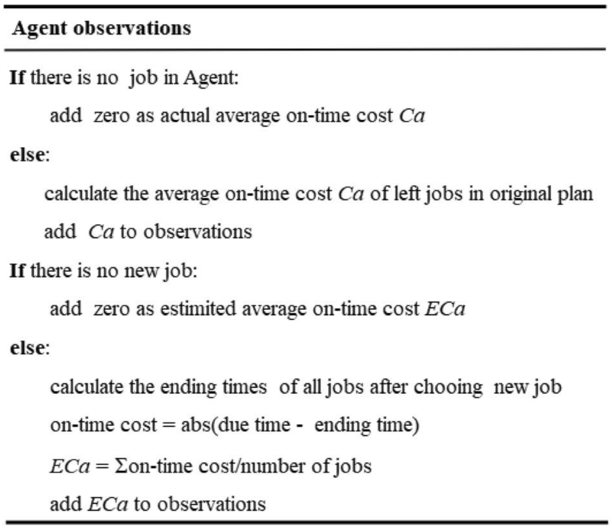


FIGURE 5 Multi-agent deep deterministic policy gradient (MADDPG) training method



```
Algorithm: Multi-Agent Deep Deterministic Policy Gradient for Dynamic DBFSP
For each episode = 1 to M do :
    Reset virtual environment randomly
    Initial observations ($o_1,o_2,\ldots,o_N$) for N Agents and global states $x$
    Initial episode end information done = [False] * N
    While not done:
        choose action $a_i$ for each Agent by $o_i$
        execute actions a = ($a_1,a_2,\ldots,a_N$), obtain reward r, new states $x'$ and check done information
        store ($x$, $a$, $x'$, $r$, done) in replay buffer D
        If stored buffer size > minibatch size:
            For each Agent $i$ = 1to N do:
                sample a random minibatch S from D
                update critic by minimizing loss and update actor by maximizing Q function
                update target network parameters
            End for
        x <- x'
    End while
End for
```

FIGURE 6 whole process of our algorithm



```
Agent observations
If there is no job in Agent:
    add zero as actual average on-time cost Ca
else:
    calculate the average on-time cost Ca of left jobs in original plan
    add Ca to observations
If there is no new job:
    add zero as estimited average on-time cost ECa
else:
    calculate the ending times of all jobs after chooing new job
    on-time cost = abs(due time - ending time)
    ECa = Σon-time cost/number of jobs
    add ECa to observations
```

FIGURE 7 Local observations calculation

Every agent makes decision only based on the local observations including actual total completion time deviation for the original plan and the estimated total completion time deviation (after a new job is inserted). The calculation process is shown in Figure 7. Global states for the centralised state-action value function contains all observations of all agents.

All agents in this problem work cooperatively. They shared all rewards. The reward of each agent is calculated as follows:

$$rew_i = -\left(Ca_i^t - Ca_i^{t-1}\right) \tag{18}$$

where Ca_i^t is the total completion time deviation at time t. The reward recorded in the replay buffer is $\sum rew_i$.

To complete the learning process, three full connection layers are applied to build the Actor network and Critic network. Due to the functions and inputs, the outputs of these two networks are different, and there are subtle differences in their structures that are shown in Figure 8.

5 | EXPERIMENTS AND DISCUSSIONS

This section presents the experimental settings in training, the results of performance comparisons between the proposed MADDPG and two MARL methods as well as three existing well-known dispatching rules.

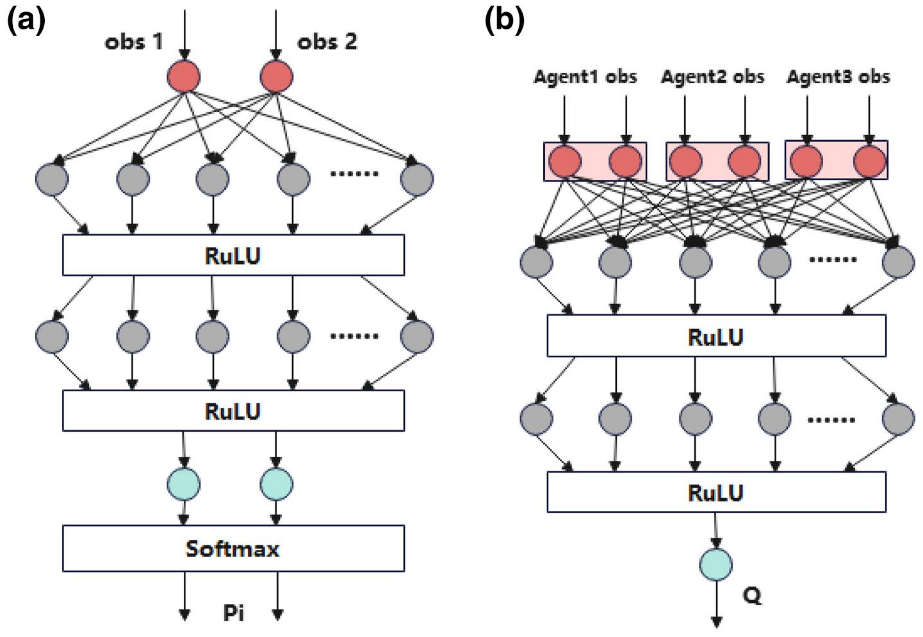


FIGURE 8 (a) Structure of the actor network (b) Structure of the critic network

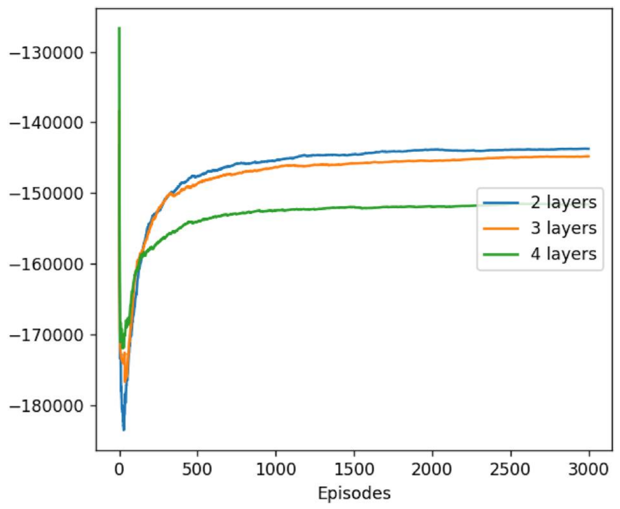


FIGURE 9 Hidden layer comparison

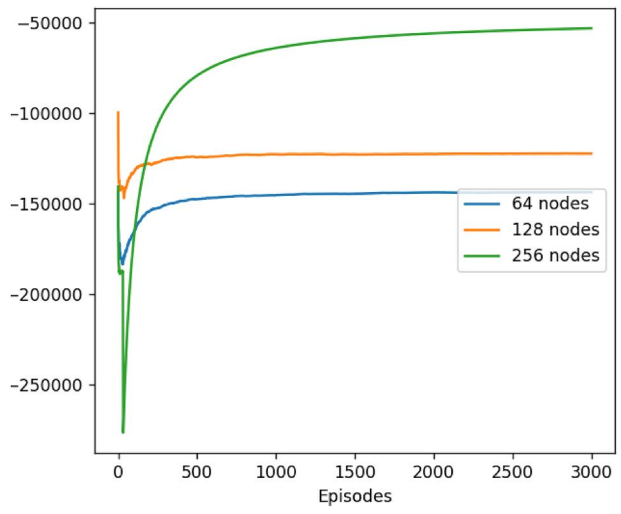


FIGURE 10 Hidden layer nodes comparison

5.1 | Experimental settings

In this study, all the experiments are conducted using Python 3.8.6 and run on the AMD Ryzen 5 2500U CPU. To model the actor and critic networks, the number of hidden layers and hidden layer nodes should be determined at first. An example with three agents, 30 jobs, three stages and 16 new jobs is selected to make a comparison. When the episode is 3000, Relu is applied as the activation function. From Figures 9 and 10, we can see that the number of hidden layers and hidden layer nodes really impact the training results. With the increase in the number of hidden layers, the converged reward value is gradually reduced. And the hidden layer nodes show the opposite effectiveness, which becomes more accurate when the number grows. According to these results, for the comparisons with other MADRL methods and dispatch rules, this paper uses three full connection layers, and each hidden layer contains 256 nodes. In all our experiments, we use the Adam optimiser with a learning rate of 0.01 and $\tau = 0.01$ for updating the target networks. The discounted factor γ is set to be 0.95. The size of the replay buffer is 10^6, and we update the network parameters after every 100 samples added to the replay buffer. We use a batch size of 100 episodes before making an update [37].

5.2 | Experiment results and analyses

To verify the effectiveness of the proposed method, we compare our MADDPG with the other two MADRL methods, which are Multi-Agent Actor Critic (MAAC) and Multi-Agent Deep Q Networks (MADQN). Also, we make a comparison with three dispatch rules, including shortest process time (SPT), longest process time (LPT), and earliest due date (EDD), which are the most used dispatching rules in real production. To make this comparison more effective, we reserve all original plans when applying dispatching rules to optimise and insert jobs according to the SPT, LPT, and EDD of order.

In this paper, we use Due Date Tightness (DDT) to indicate the tightness of a job's slack time from its arrival time to its due date. For a job i with n_i operations arriving at time point A_i, its due date D_i can be calculated as follows:

$$D_i = A_i + \left(\sum_{j=1}^{n_i} p_{i,j} \right) \cdot DDT \tag{19}$$

where $p_{i,j}$ is the process time of job i of the jth operation.

We randomly generate basic 18 examples for the experiments with three factories to generate the original schedule plan. Number of jobs: $J = [40, 50, 60]$; number of stages: $S = [3, 5, 7]$; number of new jobs:I = [20, 30]; processing times range from 1 to 100 and due time range from 300 to 500. And for the new jobs, the due times are calculated using formula (18) with DDT = 0.5, 1.0, 1.5. Instance C3J40S3I20 means inserting 20 jobs into three factories with 40 jobs and three stages.

Three comparisons are shown in Table 1, Table 2 and Table 3 with the best results highlighted in bold font. Table 1 and Table 2 are the comparisons by same instances with three DDT dimentions between MADDPG with dispatching rules and other MADRLs, seperately. Results in Table 3 are calculated by random due dates instead of same DDT in Table 1 and Table 2. All results shown in the tables are the average results of 10 runs. The results in Table 1 are the total completion time deviation of three MADRL methods in different DDTs. It can be seen that the MADDPG outperforms other DRL methods (MADQN and MAAC) for most production environments, indicating that the MADDPG has learnt a better actor policy to choose an action at each decision point. Even the actor policy of MADDPG executes only by observations of the agent, which is the same as the MAAC's, but it uses the target actor to generate the new actions with old observations, which can improve the optimisation ability of the actor policy. Also, DDPG combines advantages of DPG and DQN, and can have a better performance comparing with DQN. Therefore, MADDPG obtains better results than MADQN.

TABLE 1 Comparison among MADRLs

	DDT = 0.5			DDT = 1.0			DDT = 1.5		
Instances	MADDPG	MADQN	MAAC	MADDPG	MADQN	MAAC	MADDPG	MADQN	MAAC
C3J40S3I20	**2.904E + 04**	3.003E + 04	3.004E + 04	**2.797E + 04**	2.841E + 04	2.881E + 04	**2.579E + 04**	2.622E + 04	2.656E + 04
C3J40S3I30	**4.014E + 04**	4.176E + 04	4.196E + 04	**3.949E + 04**	4.039E + 04	4.036E + 04	**3.717E + 04**	3.785E + 04	3.817E + 04
C3J40S5I20	**2.521E + 04**	2.819E + 04	2.805E + 04	**2.286E + 04**	2.386E + 04	2.411E + 04	**1.908E + 04**	1.987E + 04	1.988E + 04
C3J40S5I30	**4.349E + 04**	4.547E + 04	4.510E + 04	**3.733E + 04**	3.952E + 04	3.943E + 04	3.301E + 04	**3.287E + 04**	3.332E + 04
C3J40S7I20	4.215E + 04	4.187E + 04	**4.185E + 04**	4.101E + 04	4.034E + 04	**4.008E + 04**	**3.734E + 04**	3.821E + 04	3.880E + 04
C3J40S7I30	**4.989E + 04**	5.192E + 04	5.227E + 04	**4.397E + 04**	4.486E + 04	4.489E + 04	3.749E + 04	**3.725E + 04**	3.753E + 04
C3J50S3I20	**3.066E + 04**	3.322E + 04	3.445E + 04	**2.993E + 04**	3.321E + 04	3.289E + 04	**2.824E + 04**	3.184E + 04	3.198E + 04
C3J50S3I30	**3.962E + 04**	4.281E + 04	4.282E + 04	**3.527E + 04**	3.838E + 04	3.845E + 04	**3.024E + 04**	3.405E + 04	3.360E + 04
C3J50S5I20	**4.051E + 04**	4.180E + 04	4.201E + 04	**3.521E + 04**	3.552E + 04	3.555E + 04	3.057E + 04	**3.034E + 04**	3.058E + 04
C3J40S5I30	**5.243E + 04**	5.794E + 04	5.840E + 04	**5.008E + 04**	5.186E + 04	5.152E + 04	**4.339E + 04**	4.606E + 04	4.641E + 04
C3J50S7I20	**5.299E + 04**	5.350E + 04	5.334E + 04	**4.935E + 04**	4.981E + 04	4.937E + 04	**4.450E + 04**	4.562E + 04	4.581E + 04
C3J50S7I30	**6.316E + 04**	6.625E + 04	6.579E + 04	**5.755E + 04**	6.051E + 04	6.042E + 04	**5.162E + 04**	5.345E + 04	5.534E + 04
C3J60S3I20	3.822E + 04	**3.781E + 04**	3.818E + 04	3.220E + 04	3.209E + 04	**3.201E + 04**	2.807E + 04	2.790E + 04	**2.771E + 04**
C3J60S3I30	6.878E + 04	6.815E + 04	**6.743E + 04**	6.443E + 04	**6.340E + 04**	6.380E + 04	**5.869E + 04**	5.920E + 04	5.902E + 04
C3J60S5I20	5.809E + 04	5.719E + 04	**5.667E + 04**	5.493E + 04	5.507E + 04	**5.465E + 04**	5.234E + 04	5.214E + 04	**5.090E + 04**
C3J40S5I30	**8.007E + 04**	8.191E + 04	8.287E + 04	**7.772E + 04**	7.949E + 04	7.901E + 04	**7.500E + 04**	7.586E + 04	7.686E + 04
C3J60S7I20	**6.124E + 04**	6.150E + 04	6.192E + 04	**5.482E + 04**	5.522E + 04	5.572E + 04	**4.838E + 04**	4.911E + 04	4.879E + 04
C3J60S7I30	**7.918E + 04**	8.030E + 04	8.027E + 04	**7.321E + 04**	7.435E + 04	7.438E + 04	**6.723E + 04**	6.896E + 04	6.845E + 04

Abbreviations: DDT, Due Date Tightness; MAAC, Multi-Agent Actor Critic; MADDPG, multi-agent deep deterministic policy gradient; MADQN, Multi-Agent Deep Q Networks; MADRLs, multi-Agent DRL.

TABLE 2 Comparison between multi-agent deep deterministic policy gradient (MADDPG) with dispatching rules

	DDT = 0.5				DDT = 1.0				DDT = 1.5			
Instances	**MADDPG**	**LPT**	**SPT**	**EDD**	**MADDPG**	**LPT**	**SPT**	**EDD**	**MADDPG**	**LPT**	**SPT**	**EDD**
C3J40S3I20	2.904E + 04	**2.489E + 04**	2.861E + 04	2.860E + 04	2.797E + 04	**2.406E + 04**	2.710E + 04	2.698E + 04	2.579E + 04	**2.279E + 04**	2.519E + 04	2.550E + 04
C3J40S3I30	**4.014E + 04**	5.739E + 04	6.171E + 04	6.095E + 04	**3.949E + 04**	5.583E + 04	6.022E + 04	5.918E + 04	**3.717E + 04**	5.504E + 04	5.773E + 04	5.546E + 04
C3J40S5I20	**2.521E + 04**	2.639E + 04	2.832E + 04	2.854E + 04	**2.286E + 04**	2.361E + 04	2.344E + 04	2.330E + 04	**1.908E + 04**	2.185E + 04	1.921E + 04	2.045E + 04
C3J40S5I30	4.349E + 04	**4.096E + 04**	4.570E + 04	4.532E + 04	**3.733E + 04**	3.755E + 04	3.816E + 04	3.757E + 04	3.301E + 04	3.438E + 04	3.037E + 04	3.025E + 04
C3J40S7I20	4.215E + 04	**4.014E + 04**	4.188E + 04	4.262E + 04	4.101E + 04	**3.929E + 04**	4.020E + 04	4.057E + 04	**3.734E + 04**	3.797E + 04	3.828E + 04	3.784E + 04
C3J40S7I30	4.989E + 04	**4.521E + 04**	4.683E + 04	4.717E + 04	4.397E + 04	4.120E + 04	3.798E + 04	**3.775E + 04**	3.749E + 04	3.766E + 04	2.898E + 04	3.365E + 04
C3J50S3I20	**3.066E + 04**	3.080E + 04	3.693E + 04	3.606E + 04	**2.993E + 04**	3.027E + 04	3.552E + 04	3.454E + 04	**2.824E + 04**	2.966E + 04	3.468E + 04	3.387E + 04
C3J50S3I30	**3.962E + 04**	4.435E + 04	4.692E + 04	4.881E + 04	**3.527E + 04**	4.135E + 04	4.216E + 04	4.168E + 04	**3.024E + 04**	3.852E + 04	3.657E + 04	3.630E + 04
C3J50S5I20	4.051E + 04	**3.963E + 04**	4.142E + 04	4.155E + 04	3.521E + 04	3.594E + 04	**3.436E + 04**	3.529E + 04	3.057E + 04	3.280E + 04	**2.876E + 04**	3.371E + 04
C3J40S5I30	**5.243E + 04**	5.466E + 04	5.963E + 04	5.889E + 04	**5.008E + 04**	5.075E + 04	5.173E + 04	5.141E + 04	**4.339E + 04**	4.799E + 04	4.393E + 04	4.440E + 04
C3J50S7I20	5.299E + 04	**5.045E + 04**	5.385E + 04	5.358E + 04	4.935E + 04	**4.833E + 04**	4.886E + 04	4.848E + 04	**4.450E + 04**	4.588E + 04	4.504E + 04	4.488E + 04
C3J50S7I30	6.316E + 04	**6.184E + 04**	6.519E + 04	6.569E + 04	**5.755E + 04**	5.807E + 04	5.816E + 04	5.898E + 04	5.162E + 04	5.488E + 04	**5.068E + 04**	5.126E + 04
C3J60S3I20	3.822E + 04	**3.607E + 04**	3.785E + 04	3.761E + 04	3.220E + 04	3.284E + 04	**3.080E + 04**	3.106E + 04	**2.807E + 04**	2.937E + 04	2.599E + 04	2.911E + 04
C3J60S3I30	6.878E + 04	**6.588E + 04**	7.080E + 04	7.089E + 04	6.443E + 04	**6.325E + 04**	6.581E + 04	6.560E + 04	**5.869E + 04**	5.964E + 04	6.098E + 04	6.012E + 04
C3J60S5I20	5.809E + 04	**5.283E + 04**	5.626E + 04	5.661E + 04	5.493E + 04	**5.053E + 04**	5.387E + 04	5.297E + 04	5.234E + 04	**4.893E + 04**	5.062E + 04	4.969E + 04
C3J40S5I30	**8.007E + 04**	8.042E + 04	8.478E + 04	8.465E + 04	**7.772E + 04**	7.814E + 04	8.083E + 04	8.061E + 04	**7.500E + 04**	7.557E + 04	7.761E + 04	7.663E + 04
C3J60S7I20	6.124E + 04	5.874E + 04	5.937E + 04	**5.823E + 04**	5.482E + 04	**5.428E + 04**	5.150E + 04	5.247E + 04	4.838E + 04	5.082E + 04	**4.332E + 04**	5.085E + 04
C3J60S7I30	7.918E + 04	**7.835E + 04**	8.303E + 04	8.320E + 04	**7.321E + 04**	7.482E + 04	7.538E + 04	7.578E + 04	**6.723E + 04**	7.110E + 04	6.829E + 04	6.796E + 04

Abbreviations: DDT, Due Date Tightness; MADDPG, multi-agent deep deterministic policy gradient; SPT, shortest process time.

TABLE 3 Result comparison between multi-agent DRL (MADRL) and dispatch rules

Instances	MADDPG	MADQN	MAAC	SPT	LPT	EDD
C3J40S3I20	2.375E + 04	2.423E + 04	2.473E + 04	2.289E + 04	**2.193E + 04**	2.333E + 04
C3J40S3I30	**3.379E + 04**	3.503E + 04	3.484E + 04	5.389E + 04	5.330E + 04	5.327E + 04
C3J40S5I20	**2.021E + 04**	2.097E + 04	2.133E + 04	2.083E + 04	2.243E + 04	2.081E + 04
C3J40S5I30	**3.185E + 04**	3.509E + 04	3.497E + 04	3.254E + 04	3.490E + 04	3.206E + 04
C3J40S7I20	3.820E + 04	3.754E + 04	3.769E + 04	**3.730E + 04**	3.772E + 04	3.773E + 04
C3J40S7I30	4.186E + 04	4.208E + 04	4.189E + 04	**3.405E + 04**	3.976E + 04	3.461E + 04
C3J50S3I20	**2.720E + 04**	2.928E + 04	2.944E + 04	3.146E + 04	2.746E + 04	3.048E + 04
C3J50S3I30	**3.172E + 04**	3.387E + 04	3.394E + 04	3.614E + 04	3.913E + 04	3.668E + 04
C3J50S5I20	**3.089E + 04**	3.294E + 04	3.285E + 04	3.093E + 04	3.468E + 04	3.464E + 04
C3J40S5I30	4.649E + 04	4.816E + 04	4.805E + 04	**4.594E + 04**	4.918E + 04	4.660E + 04
C3J50S7I20	4.754E + 04	4.822E + 04	4.758E + 04	4.708E + 04	4.725E + 04	**4.672E + 04**
C3J50S7I30	**5.363E + 04**	5.764E + 04	5.734E + 04	5.493E + 04	5.655E + 04	5.480E + 04
C3J60S3I20	2.912E + 04	2.880E + 04	2.873E + 04	**2.708E + 04**	3.041E + 04	2.919E + 04
C3J60S3I30	5.876E + 04	5.826E + 04	**5.806E + 04**	5.942E + 04	5.932E + 04	5.920E + 04
C3J60S5I20	5.240E + 04	5.173E + 04	5.194E + 04	5.042E + 04	**4.876E + 04**	5.054E + 04
C3J40S5I30	**7.476E + 04**	7.567E + 04	7.574E + 04	7.688E + 04	7.522E + 04	7.674E + 04
C3J60S7I20	5.342E + 04	5.371E + 04	5.371E + 04	**4.866E + 04**	5.315E + 04	5.088E + 04
C3J60S7I30	**7.142E + 04**	7.159E + 04	7.198E + 04	7.211E + 04	7.327E + 04	7.199E + 04

Abbreviations: LPT, longest process time; MAAC, Multi-Agent Actor Critic; MADDPG, multi-agent deep deterministic policy gradient.

Next, compared with three dispatching rules as shown in Table 2, the MADDPG can also obtain the best results for most instances. This conclusion reflects that there does not exist a single rule to perform well for all production environments (except the results with DDT = 0.5) and further confirms the effectiveness and generality of the proposed MADDPG.

Furthermore, this paper compares MADDPG with two other MADRL methods and three dispatching rules by random Due date, ranging from 300 to 500. All results shown in Table 3 indicate that the proposed method obtains the best solutions among all the compared methods in most instances. Even for the instances that are not in the best solutions, they perform similar to other MADRL methods. Also, all MADRL methods have better results than single dispatch rules.

With these random examples, we also compare running times among all MADRL methods and dispatch rules in Table 4. Here, the running time means the decision-making time from the initial state to the end of the instance. It can be seen that MADDPG runs faster than MADQN and MAAC with better solutions from the whole. Also, compared with EDD, MADDPG can be quicker in 10 instances. However, this dominant position does not exist in the comparisons between MADDPG with SPT and LPT.

At last, we choose three instances, C3J40S3I30, C3J40S5I30 and C3J60S7I30, to show the generalisation of MADDPG. The following conclusions can be drawn from Figure 11: (1) MADDPG can obtain the best results from the average view; (2) MADDPG has strong generalisation for a small distribution of results; and (3) other methods perform unstably in different examples.

TABLE 4 Time comparison between MADRLs and dispatch rules (seconds)

Instances	MADDPG	MADQN	MAAC	SPT	LPT	EDD
C3J40S3I20	0.4505	0.7389	0.5028	0.0074	0.0114	0.1622
C3J40S3I30	0.6673	1.0738	0.7541	0.0108	0.0161	0.4107
C3J40S5I20	0.4332	0.7122	0.4972	0.0089	0.0133	0.2736
C3J40S5I30	0.7089	1.0897	0.7845	0.0132	0.0219	0.6935
C3J40S7I20	0.4345	0.7402	0.5199	0.0152	0.0208	0.4162
C3J40S7I30	0.6748	1.1160	0.7813	0.0167	0.0292	1.1070
C3J50S3I20	0.4381	0.7094	0.5238	0.0082	0.0128	0.3144
C3J50S3I30	0.6773	1.0350	0.7887	0.0098	0.0183	0.7167
C3J50S5I20	0.4242	0.7153	0.4934	0.0086	0.0155	0.3747
C3J40S5I30	0.6774	1.1276	0.7395	0.0161	0.0305	0.8726
C3J50S7I20	0.4486	0.7330	0.5169	0.0146	0.0225	0.5667
C3J50S7I30	0.6915	1.0949	0.8160	0.0190	0.0358	1.5398
C3J60S3I20	0.4365	0.7134	0.5417	0.0064	0.0092	0.3328
C3J60S3I30	0.6760	1.1443	0.7737	0.0133	0.0215	1.0042
C3J60S5I20	0.4515	0.7559	0.5231	0.0136	0.0200	0.6081
C3J40S5I30	0.7135	1.1326	0.7695	0.0240	0.0371	1.0845
C3J60S7I20	0.4438	0.7068	0.5138	0.0133	0.0208	0.6206
C3J60S7I30	0.7141	1.1380	0.7808	0.0268	0.0439	1.5222

Abbreviations: LPT, longest process time; MAAC, Multi-Agent Actor Critic; MADDPG, multi-agent deep deterministic policy gradient; MADRLs, multi-Agent DRL.

6 | CONCLUSIONS AND FUTURE WORK

In this paper, a MADDPG is developed for the DBFSP with new job insertions aiming at optimising the total completion time deviation. A DRL framework based on MADDPG is proposed based on the characteristics of DBFSP problem first. This framework combines the VND algorithm and new jobs sorting rule with MADDPG. The initial scheduling solution is generated by VND before a dynamic event occurs. Meanwhile, all newly added jobs are reordered before the agents make decisions to select the one that needs to be scheduled most urgently. According to this problem description, the global states, agent observations, actions and rewards are defined before the training process. Numerical experiments are conducted on a set of instances that can be regarded as a high abstraction of actual manufacturing processes to verify the effectiveness and superiority of the proposed MADDPG in practical applications. The results demonstrate that the proposed MADDPG performs better than dispatching rules and other multi-agent DRLs in most experiment instances.

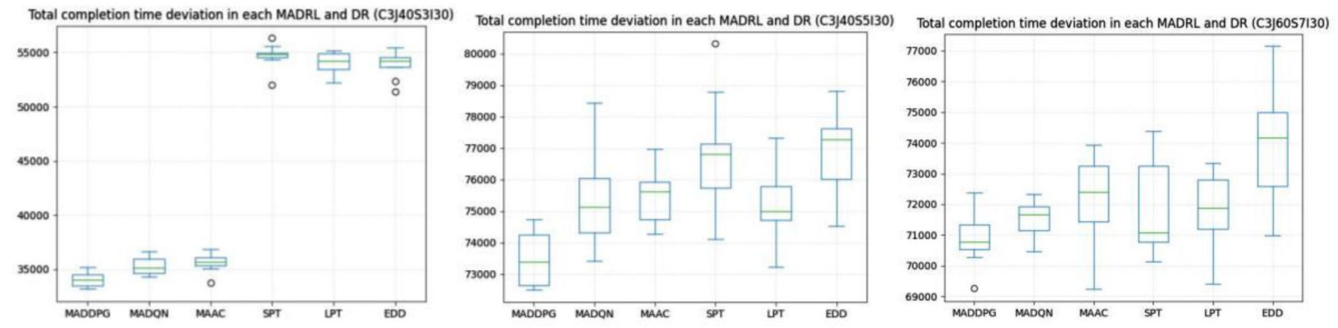


FIGURE 11 Result comparison with all methods

This paper presents some preliminary work of our ongoing work. For future work, other uncertain events such as machine breakdowns will be considered. Also, the observations of agents can be transferred into direct data so that the proposed MADDPG will be an end-to-end DRL. We will also consider combining RL and evolutionary algorithms together for solving the dynamic scheduling problem in our future work in order to achieve better results.

ACKNOWLEDGEMENT

Xueyan Sun was supported by the China Scholarship Council for 1-year study at Technical University Munich.

CONFLICT OF INTEREST

All authors declare that they have no conflicts of interest.

PERMISSION TO REPRODUCE MATERIALS FROM OTHER SOURCES

None.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

ORCID

Xueyan Sun https://orcid.org/0000-0003-1784-5432
Weiming Shen https://orcid.org/0000-0001-5204-7992

REFERENCES

1. Li, X., et al.: Review on flexible job shop scheduling. IET Collab. Intell. Manuf. 1(2), 67–77 (2019). https://doi.org/10.1049/iet-cim.2018.0009
2. Pan, Y., et al.: Solving biobjective distributed flow-shop scheduling problems with lot-streaming using an improved Jaya algorithm. IEEE Trans. Cybern., 1–11 (2022). https://doi.org/10.1109/tcyb.2022.3164165
3. Liu, S., Wang, P., Zhang, J.: An improved biogeography-based optimization algorithm for blocking flow shop scheduling problem. Chin. J. Electron. 27(2), 351–358 (2018). https://doi.org/10.1049/cje.2018.01.007
4. Pezzella, F., Morganti, G., Ciaschetti, G.: A genetic algorithm for the flexible job-shop scheduling problem. Comput. Oper. Res. 35(10), 3202–3212 (2008). https://doi.org/10.1016/j.cor.2007.02.014
5. Adibi, M., Zandieh, M., Amiri, M.: Multi-objective scheduling of dynamic job shop using variable neighborhood search. Expert Syst. Appl. 37(1), 282–287 (2010). https://doi.org/10.1016/j.eswa.2009.05.001
6. Chiu, C., Yih, Y.: A learning-based methodology for dynamic scheduling in distributed manufacturing systems. Int. J. Prod. Res. 33(11), 3217–3232 (1995). https://doi.org/10.1080/00207549508904870
7. Fattahi, P., Fallahi, A.: Dynamic scheduling in flexible job shop systems by considering simultaneously efficiency and stability. CIRP J. Manuf. Sci. Technol. 2(2), 114–123 (2010). https://doi.org/10.1016/j.cirpj.2009.10.001
8. Sahin, C., et al.: A multi-agent based approach to dynamic scheduling with flexible processing capabilities. J. Intell. Manuf. 28(8), 1827–1845 (2017). https://doi.org/10.1007/s10845-015-1069-x
9. Gao, K., et al.: Flexible job-shop rescheduling for new job insertion by using discrete Jaya algorithm. IEEE Trans. Cybern. 49(5), 1944–1955 (2018). https://doi.org/10.1109/tcyb.2018.2817240
10. An, Y., et al.: Multiobjective flexible job-shop rescheduling with new job insertion and machine preventive maintenance. IEEE Trans. Cybern., 1–13 (2022). https://doi.org/10.1109/tcyb.2022.3151855
11. Fu, Y., et al.: Artificial-molecule-based chemical reaction optimization for flow shop scheduling problem with deteriorating and learning effects. IEEE Access 7, 53429–53440 (2019). https://doi.org/10.1109/access.2019.2911028
12. Fu, Y., et al.: Scheduling dual-objective stochastic hybrid flow shop with deteriorating jobs via bi-population evolutionary algorithm. IEEE Trans. Syst. Man Cybern. Systems, 50(12), 5037–5048 (2019). https://doi.org/10.1109/tsmc.2019.2907575
13. Zhao, Z., et al.: Dual-objective mixed integer linear program and memetic algorithm for an industrial group scheduling problem. IEEE/CAA J. Autom. Sin. 8(6), 1199–1209 (2020). https://doi.org/10.1109/jas.2020.1003539
14. Zhao, Z., Zhou, M., Liu, S.: Iterated greedy algorithms for flow-shop scheduling problems: a tutorial. IEEE Trans. Autom. Sci. Eng. 19(3), 1941–1959 (2021). https://doi.org/10.1109/tase.2021.3062994
15. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT press (2018)
16. Li, S., et al.: Machine learning-based scheduling: a bibliometric perspective. IET Collab. Intell. Manuf. 3(2), 131–146 (2021). https://doi.org/10.1049/cim2.12004
17. Riedmiller, S., Riedmiller, M.: A neural reinforcement learning approach to learn local dispatching policies in production scheduling. In: IJCAI, 1999, vol. 2, pp. 764–771. Citeseer
18. Aydin, M.E., Öztemel, E.: Dynamic job-shop scheduling using reinforcement learning agents. Robot. Autonom. Syst. 33(2-3), 169–178 (2000). https://doi.org/10.1016/s0921-8890(00)00087-7
19. Wang, Y.-C., Usher, J.M.: Learning policies for single machine job dispatching. Robot. Comput. Integrated Manuf. 20(6), 553–562 (2004). https://doi.org/10.1016/j.rcim.2004.07.003
20. Chen, X., et al.: Rule driven multi objective dynamic scheduling by data envelopment analysis and reinforcement learning. In: 2010 IEEE International Conference on Automation and Logistics, pp. 396–401. IEEE (2010)
21. Luo, S.: Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. Appl. Soft Comput. 91, 106208 (2020). https://doi.org/10.1016/j.asoc.2020.106208
22. Martin, S., et al.: A multi-agent based cooperative approach to scheduling and routing. Eur. J. Oper. Res. 254(1), 169–178 (2016). https://doi.org/10.1016/j.ejor.2016.02.045
23. Cao, Z., et al.: Scheduling semiconductor testing facility by using cuckoo search algorithm with reinforcement learning and surrogate modeling. IEEE Trans. Autom. Sci. Eng. 16(2), 825–837 (2018). https://doi.org/10.1109/tase.2018.2862380
24. Zhang, T., Xie, S., Rose, O.: Real-time job shop scheduling based on simulation and Markov decision processes. In: 2017 Winter Simulation Conference (WSC), pp. 3899–3907. IEEE (2017)
25. Zhou, L., Zhang, L., Horn, B.K.: Deep reinforcement learning-based dynamic scheduling in smart manufacturing. Procedia Cirp 93, 383–388 (2020). https://doi.org/10.1016/j.procir.2020.05.163
26. Kim, Y.G., et al.: Multi-agent system and reinforcement learning approach for distributed intelligence in a flexible smart manufacturing system. J. Manuf. Syst. 57, 440–450 (2020). https://doi.org/10.1016/j.jmsy.2020.11.004
27. Han, B.A., Yang, J.J.: A deep reinforcement learning based solution for flexible job shop scheduling problem. Int. J. Simulat. Model. 20(2), 375–386 (2021). https://doi.org/10.2507/ijsimm20-2-co7
28. Zinn, J., Vogel-Heuser, B., Ockier, P.: Deep Q-learning for the control of PLC-based automated production systems. In: 2020 IEEE 16th International Conference on Automation Science and Engineering (CASE), pp. 1434–1440. IEEE (2020)
29. Luo, S., Zhang, L., Fan, Y.: Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning. Comput. Ind. Eng. 159, 107489 (2021). https://doi.org/10.1016/j.cie.2021.107489
30. Baer, S., et al.: Multi agent deep q-network approach for online job shop scheduling in flexible manufacturing. Proc. ICMSMM, 78–86 (2020)
31. Zinn, J., et al.: Hierarchical reinforcement learning for waypoint-based exploration in robotic devices. In: 2021 IEEE 19th International Conference on Industrial Informatics (INDIN), pp. 1–7. IEEE (2021)

32. Gankin, D., et al.: Modular production control with multi-agent deep Q-learning. In: 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), pp. 1–8. IEEE (2021)
33. Sutton, R.S., et al.: Policy gradient methods for reinforcement learning with function approximation. Adv. Neural Inf. Process. Syst. 12 (1999)
34. Silver, D., et al.: Deterministic policy gradient algorithms. In: International Conference on Machine Learning, pp. 387–395. PMLR (2014)
35. Lillicrap, T.P., et al.: Continuous control with deep reinforcement learning, arXiv preprint arXiv:1509.02971, (2015)
36. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature. 518(7540), 529–533 (2015). https://doi.org/10.1038/nature14236
37. Lowe, R., et al.: Multi-agent actor-critic for mixed cooperative-competitive environments. Adv. Neural Inf. Process. Syst. 30 (2017)

How to cite this article: Sun, X., et al.: A deep reinforcement learning based approach for dynamic distributed blocking flowshop scheduling with job insertions. IET Collab. Intell. Manuf. 4(3), 166–180 (2022). https://doi.org/10.1049/cim2.12060