APEM journal
Advances in Production Engineering & Management
Volume 16 | Number 3 | September 2021 | pp 269–284
https://doi.org/10.14743/apem2021.3.399
ISSN 1854-6250
Journal home: apem-journal.org
Original scientific paper

A new solution to distributed permutation flow shop scheduling problem based on NASH Q-Learning

Ren, J.F.[a,b], **Ye, C.M.**[a,*], **Li, Y.**[a]

[a]School of Business, University of Shanghai for Science and Technology, Shanghai, P.R. China
[b]School of Computer and Information Engineering, Henan University of Economics and Law, Zhengzhou, P.R. China

ABSTRACT

Aiming at Distributed Permutation Flow-shop Scheduling Problems (DPFSPs), this study took the minimization of the maximum completion time of the workpieces to be processed in all production tasks as the goal, and took the multi-agent Reinforcement Learning (RL) method as the main frame of the solution model, then, combining with the NASH equilibrium theory and the RL method, it proposed a NASH Q-Learning algorithm for Distributed Flow-shop Scheduling Problem (DFSP) based on Mean Field (MF). In the RL part, this study designed a two-layer online learning mode in which the sample collection and the training improvement proceed alternately, the outer layer collects samples, when the collected samples meet the requirement of batch size, it enters to the inner layer loop, which uses the Q-learning model-free batch processing mode to proceed and adopts neural network to approximate the value function to adapt to large-scale problems. By comparing the Average Relative Percentage Deviation (ARPD) index of the benchmark test questions, the calculation results of the proposed algorithm outperformed other similar algorithms, which proved the feasibility and efficiency of the proposed algorithm.

ARTICLE INFO

Keywords:
Flow shop scheduling;
Distributed scheduling;
Permutation flow shop;
Reinforcement learning;
NASH Q-learning;
Mean field (MF)

**Corresponding author:*
171910083@st.usst.edu.cn
(Ye, C.M.)

Article history:
Received 29 July 2021
Revised 8 September 2021
Accepted 26 September 2021

1. Introduction

Under intelligent manufacturing mode, cross-product value chain integration has become a normal situation, for example, many parts of air conditioners and refrigerators are similar or even the same, so to reduce costs, they can be produced and processed via distributed production. On this basis, this paper abstracted the problem of distributed production in different geographical locations to a DFSP, and divided it into two sub-problems, the reasonable allocation of workpieces among factories, and the processing sequence arrangement within a factory.

Field scholars have achieved various research results in terms of DFSP [1-3], for example, Wang *et al.* [4] analyzed the research status and development direction of DFSPs. Fernandez-Viagas and Framinan [5] proposed a new heuristic algorithm and obtained better upper boundaries. In terms of DPFSP, Gao *et al.* [6, 7] used heuristic algorithm to solve the problem, Naderi and Ruiz [8] proposed a scatter search algorithm, Lin *et al.* [9] proposed an improved iterative greedy algorithm, Fernandez-Viagas and Framinan [5] proposed a bounded search iterative greedy algorithm, Yang *et al.* [10] comprehensively optimized the assembly transport strategy, production process, and production configuration of a reconfigurable flow shop, Wang *et al.* [11] proposed a

distributed estimation algorithm, and they used their respective algorithm to solve the problem. Rifai *et al.* [12] proposed a multi-objective adaptive large neighborhood search algorithm based on the Pareto front to study the distributed reentrant arrangement flow shop. Wang *et al.* [13] proposed a hybrid distributed estimation algorithm based on fuzzy logic to solve the production scheduling problem with the maximum completion time criterion under machine failure conditions. Komaki and Malakooti [14] proposed a generalized variable neighborhood search metaheuristic algorithm to solve the problem of minimizing the maximum completion time of the distributed no-waiting flow shop scheduling problem. Chen *et al.* [15] proposed the non-dominated sorting genetic algorithm (NSGA) to the design of non-compact flow shop scheduling plan, and successfully solved the multi-objective optimization problem considering process connection, Lebbar *et al.* [16] proposed a computational evaluation of a new mixed integer linear programming (MILP) model developed for the multi-machine flowshop scheduling, Rathinam *et al.* [17] proposed heuristic based methodology to solve permutation flow shop scheduling.

As the multi-agent Deep Reinforcement Learning (DRL) technology [18] is developing rapidly, the multi-agent application of the Q-learning algorithm has become more prominent [19-22], and scholars have successively applied this new technology to combinatorial optimization problems such as workshop scheduling, and obtained a series of research results. For example, a few studies [23-26] explored deep into the communication and cooperation between the multiple agents of DRL. Omidshafiei *et al.* [27] studied multi-task and multi-agent RL problem, and proposed a method to upgrade single-task strategy to multi-task strategy. Sunehag *et al.* [28] used a new value decomposition network structure to train individual agents to solve the joint reward problem of multiple agents. Li *et al.* [29] proposed a workflow job scheduling algorithm based on load balancing, and the research results showed that the proposed algorithm can effectively utilize the cloud resources. Neary *et al.* [30] studied the interaction of collaborative multi-agent RL in a shared environment. Some scholars applied Nash equilibrium and multi-agent RL algorithm to optimal control problems [31, 32]. Shah *et al.* [33] studied the Nash equilibrium RL problem of the two-person zero-sum game. Feng *et al.* [34] proposed a polynomial time algorithm for dual-agent scheduling problem to find all Pareto optimal solutions.

In summary, this paper took DFSP as the research object and proposed new NASH-Q DRL algorithm to solve DFSP based on existing multi-agent RL theory and algorithm results, and the problem was solved by a data-driven method, which avoided the design of complex heuristic rules.

2. Problem description

Distributed workshop scheduling can be divided into several types: distributed parallel machine scheduling, distributed flow shop scheduling, distributed job shop scheduling, distributed assembly scheduling, and distributed flexible workshop scheduling. This paper aims at the DPFSP, namely the distributed permutation flow shop scheduling problem, before solving the problem, it's assumed that each factory has one flow production line, the production capacity of each factory is the same, each factory has the same workshop layout, the same number of machines, and the same processing ability for all workpieces.

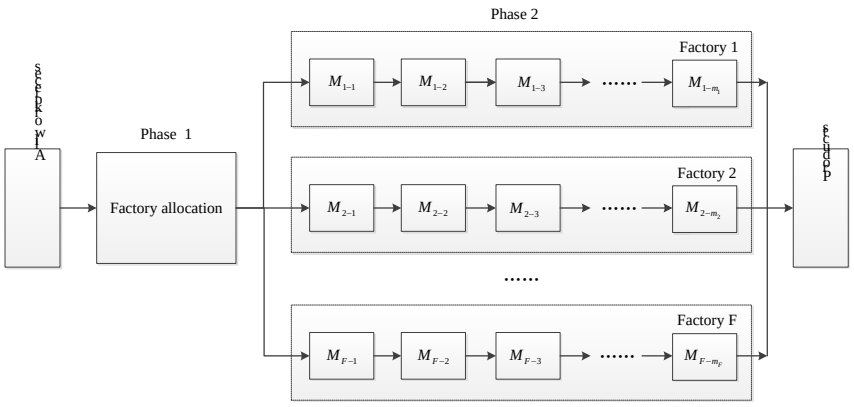

Fig. 1 Scheduling of distributed flow shop

In DPFSP, the first thing to do is to solve the workpiece allocation problem in each workshop, that is, to allocate workpieces to be processed to each workshop according to certain initial conditions; the second is to solve the workpiece processing sequence on the production line in each workshop, that is, to arrange the processing sequence of the workpieces that have been allocated to a same workshop, and use machines to process the sorted workpieces one by one. Due to mutual coupling, the problem shows high complexity, Fig. 1 gives a diagram of the scheduling of distributed flow shop.

In problem solving, the goal is to minimize the maximum completion time of all workpieces. The maximum completion time of DPFSP means the total completion time of all the workpieces, which is determined by the workshop with the longest completion time, and this workshop is called the key workshop in this study. Assume: a processing task has a total of *n* workpieces that need to be processed in *F* workshops, each workshop has *m* machines, and each workpiece needs to go through *m* processing procedures, the processing procedures of all workpieces are the same, once the workshop is decided for a workpiece, all procedures need to be completed within this workshop, the numbers of workpieces allocated to each workpiece are $(n_1, n_2, \dots, n_f)$, the processing time of workpiece *i* on machine *j* in workshop *f* is denoted as $p_{i,j}^f$, the completion time of the *k*-th workpiece on machine *j* is denoted as $C_{k,j}^f$, the processing sequence of workpieces is denoted as π^f, the overall scheduling scheme is denoted as $\pi = [\pi^1, \pi^2, \dots, \pi^F]$, then the objective function is:

$$C_{max}(\pi) = max\{ C_{max}(\pi^f)\} \qquad f = 1,2, \dots, F \tag{1}$$

3. Used methods

3.1 Multi-agent DRL

Multi-agent RL needs to be described by Markov game, because at a same moment, multiple agents respectively take independent actions, after the actions are performed, the reward received by each agent is not only related to itself, but also related to other agents, there're mutual game relationships among agents, and the multi-agent RL system can be described in the following form:

$$(N, S, A, T, R, \gamma) \tag{2}$$

where, N represents the number of multi-agents; S represents the state of the system; A represents the set of actions of the agents, which is consisted of $a_1, a_2, \dots, a_N$; T represents the state transition function, namely $T: S \times A \times S \to [0,1]$, under state S, after action A is performed, the probability distribution of the next state is obtained; R represents the reward obtained by the agents, that is, $R_i(S, A, S')$ represents the reward obtained by agent *i* for performing action A and transforming from state S to state S'; γ represents the learning rate. Each agent performs different actions, and all actions constitute systematic joint actions, which make the environment undergo changes and reach a new state, and each agent gets its own reward value, as shown in Fig. 2.

Each agent in the multi-agent system needs to interact with the environment to obtain reward values, thereby improving the strategy and obtaining the optimal strategy in the current environment. The process of the strategy learning is the multi-agent RL.

The matrix game can be expressed as $(N, A_1, A_2, \dots, A_N, R_1, R_2, \dots, R_N)$, where N is the number of agents in the system, A_n is the action set of the *n*-th agent, and R_n is the reward function of the *n*-th agent, apparently, the reward obtained by each agent is related to the joint actions of the system, $A_1 \times A_2 \times, \dots, \times A_N$ represents the space of joint actions. The strategy of each agent is the probability distribution of the action space, and the goal of each agent is to get the maximum reward value. Assume π_n represents the strategy of agent n, $(\pi_1, \pi_2, \dots, \pi_N)$ represents the joint strategy of the system, then the value function of agent n can be expressed as $V_n(\pi_1, \pi_2, \dots, \pi_n, \dots, \pi_N)$.

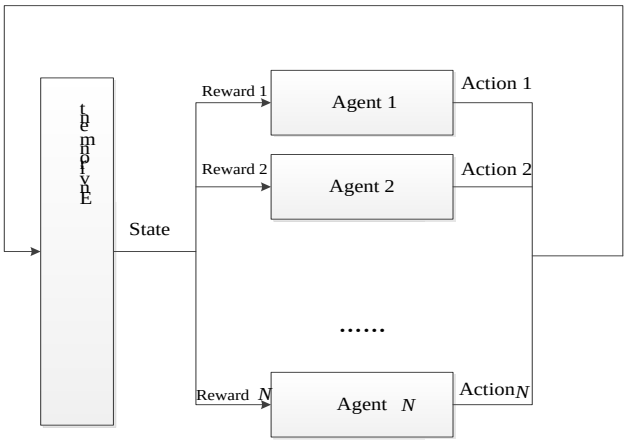

Fig. 2 Multi-agent RL system

If there is $\pi_n \in \prod_n, n = 1,2,\dots,N$ in the joint strategy $(\pi_1^*, \pi_2^*, \dots, \pi_N^*)$ of the matrix game, and it satisfies $(\pi_1^*, \pi_2^*, \dots, \pi_n, \dots, \pi_N^*) \leq (\pi_1^*, \pi_2^*, \dots, \pi_N^*)$, then it is a NASH equilibrium.

In multi-agent RL algorithm, the NASH equilibrium can be described as:

When the system performs joint actions $[a_1, a_2, \dots, a_N]$, the expected reward obtained by agent n is $R_n[a_1, a_2, \dots, a_N]$, $\pi_n(a_n)$ represents the probability of agent n choosing action a_n, then the NASH equilibrium can be defined as:

$$\sum_{a_1,\dots,a_N \in A_1,\dots,A_N} R_n(a_1, a_2, \dots, a_N)\pi_1^*(a_1) \dots \pi_n(a_n) \dots \pi_N^*(a_N)$$
$$\leq \sum_{a_1,\dots,a_N \in A_1,\dots,A_N} R_n(a_1, a_2, \dots, a_N)\pi_1^*(a_1) \dots \pi_n^*(a_n) \dots \pi_N^*(a_N)$$

wherein, $\pi_n \in \prod_n, n = 1,2,\dots,N$.

The Minmax-Q algorithm is an earlier multi-agent method for random decision-making [35]. The algorithm solves the zero-sum game of two agents, and theoretically proves the convergence of the algorithm. However, due to the limitation in actual applications, the algorithm can hardly solve general sum problems. Aiming at this shortcoming of the Minmax-Q algorithm, the NASH-Q algorithm [36] extends to the multi-agent complete information general sum random game problems, and it uses NASH equilibrium to define the value function.

The random game is the random game process of N agents, it can be expressed as Eq. 3:

$$(N, S, A_1, A_2, \dots, A_N, T, R_1, R_2, \dots, R_N, \gamma) \tag{3}$$

where, N represents the number of multi-agents, S represents the state of the system, A_n represents the action space of agent n, T represents the state transition function, namely $T: S \times A \times S \to [0,1]$, under state S, after performing action A, the probability distribution of the next state could be obtained, R_n represents the reward obtained by agent n, γ represents the learning rate, the strategy of agent n is defined as $\pi_n: S \to \Omega(A_n)$, which represents the probability distribution from the state to the agent action space, wherein $\Omega(A_n)$ represents all possible sets of the action space probability distribution of agent n.

The cumulative discount value function of agent n under the current strategy π can be described by Eq. 4 as:

$$V_\pi^n(s) = \sum_{t=0}^{\infty} \gamma^t E_\pi [R_t^n | s = s_0, \pi] \tag{4}$$

The action value function can be described as Eq. 5:

$$Q_\pi^n(s, A) = R_n(s, A) + \gamma E_{s' \sim T}[V_\pi^n(s')] \tag{5}$$

The value function described by Eq. 5 is based on the condition of single agent, the action A in the formula is the joint action adopted by each agent, obviously, the action value function of the agents is based on the system state and the joint action space.

Integrating Eq. 5, Eq. 4 can be rewritten as Eq. 6:

$$V_{\pi}^{n}(s) = E_{A\sim\pi(s)}[Q_{\pi}^{n}(s, A)] \tag{6}$$

It represents that the state value function is obtained through the expectation of the action value function.

Under the state that each agent performs discrete actions, the multi-agent RL is modeled through random games. Under the condition that each agent in the system doesn't know the rewards of other agents, still each agent can observe the previous behaviors of other agents, and respond to the instant rewards generated.

The MiniMax-Q learning algorithm is usually used to solve zero-sum game problems, its core idea is that each agent maximizes the expected reward value in the worst case in the game with the opponent, it solves the Nash equilibrium strategy of the game under specific state s by constructing MiniMax linear programming method, and uses the time-series difference method to iteratively learn the state value function or the action value function. In the zero-sum game of two agents, for a given state s_0, the state value function of the n-th agent can be defined as Eq. 7:

$$V_{n}^{*}(s_0) = \max_{\pi_n(s_0,\cdot)} \min_{a_{-n}\in A_{-n}} \sum_{a_n\in A_n} Q_{n}^{*}(s_0, a_n, a_{-n})\pi_n(s_0, a_n) \tag{7}$$

where, the value of n is 1 or 2, and a_{-n} represents actions other than a_n.

This paper aims to use multi-agent method to solve the distributed production scheduling problem, therefore, based on the MiniMax-Q learning algorithm, it extends to the general sum game problem of multi-agents, namely the NASH Q-learning algorithm, which uses quadratic programming method to solve the NASH equilibrium point, in the game of each state, it could find out the global optimal point or the saddle point, so that the system can converge to the NASH equilibrium point in the cooperative equilibrium or the adversarial equilibrium.

3.2 Multi-agent MF-DRL algorithm

Mean Field Theory is a method for studying complex multi-agent problems using machine learning and physical field theories [37], it can simplify random models with huge scale or complex structure.

The overall goal of using multiple agents to solve distributed production scheduling problems is to minimize the completion time. Each agent must learn the optimal strategy to cooperate with the overall goal of the system, and the solution algorithm was improved based on the work of Ruiz *et al.* [38]. It's assumed that, π represents the joint strategy of the system, π_i represents the strategy of agent i, v_i represents the value function of agent i, N represents the number of agents; π^* represents the joint strategy of the system under NASH equilibrium, and strategy π^* is consisted of N agent strategies $(\pi_1^*, \pi_2^*, \dots, \pi_N^*)$; the system state is s, then the strategy of agent i is π_i^*, and the strategy of other agents except for agent i in the joint strategy π^* is represented as π_{-i}^*, namely $\pi_{-i}^* \doteq (\pi_1^*, \pi_2^*, \dots, \pi_{i-1}^*, \pi_{i+1}^*, \dots, \pi_N^*)$, and the value function of agent i can be described by Eq. 8:

$$v_i(s; \pi^*) = v_i(s; \pi_i^*, \pi_{-i}^*) \tag{8}$$

According to the feature of NASH equilibrium, Eq. 9 could be obtained as:

$$v_i(s; \pi_i^*, \pi_{-i}^*) \geq v_i(s; \pi_i, \pi_{-i}^*) \tag{9}$$

The multi-agent system is in the optimal state at this moment, in the system, agent i executes tasks according to strategy π_i^*, and other agents execute tasks according to strategy π_{-i}^*, obviously, $i \in \{1,2, \dots, N\}$, that is, any agent in the system has the opportunity to act as an independent agent, and each agent has the opportunity to act as a virtual agent and work with other agents to play the game with independent agents in the system.

The initial state of the system is s, then the value function of the system can be described by Eq. 10:

$$v^*(s) \doteq [v_1(s), v_2(s), \ldots, v_N(s)]_{\pi^*} \tag{10}$$

The Q-value of the current state is initialized according to the definition of the NASH Q-learning algorithm, and is updated continuously through iterations, also, a NASH factor is added during Q-value update to ensure that the Q-value update can satisfy the compression mapping requirements. At the same time, the relationship between the Q-value function and the state value function can be known from Eq. 6, that is, the state value function can be updated synchronously and reach the NASH equilibrium.

In a multi-agent system, it can be known from Eq. 3 that the dimension of the contact action is determined by the number of agents, for any agent i in the system, its action value function $Q_i(s, A)$ can be defined as:

$$Q_i(s, A) = \frac{1}{N-1} \sum_{j \in N-1} Q_i(s, A_i, A_j) \tag{11}$$

This definition makes agent i have a connection with action A_j of other agents, and it simplifies the relationships among multiple agents and gives mathematical descriptions. The complex relationships among multiple agents are simplified as pairwise correspondence between agent i and other agents, then by summing and averaging, the internal interactive relationships between agent i and other agents, and between other agents are retained; the pairwise approximation not only reduces the complexity of the interaction between agents, but also implicitly preserves the global interaction between any pair of agents.

In the workshop production scheduling problems, the sorting of workpieces within production line and the exchange of workpieces between production lines are both discrete actions. Assume: agent i has a total of χ actions, the action space of A_i has a total of χ components, and each component represents a selectable action. A_i is encoded by the One-Hot encoding method, the position code of the selected action is 1, and the position of other components is 0.

In the algorithm, the distance or similarity between features is calculated by One-Hot encoding, the values of discrete actions are extended to the Euclidean space, and then a certain value of the discrete action can correspond to a certain point in the Euclidean space. At the same time, One-Hot encoding is also adopted for discrete features, which makes the calculation of distances between actions more reasonable. After discrete actions are subject to One-Hot encoding, the features of each dimension can be regarded as continuous features, and can be normalized using the normalization method of continuous features, for instance, they can be normalized to [-1,1] or to a mean of 0 and a variance of 1. The characteristics of One-Hot encoding provide convenience for calculating the mean action in the mean field theory, assuming $\overline{A_i}$ represents the mean action, which is equivalent to the multinomial distribution of the actions of agents in the agent neighborhood, then it can be expressed by Eq. 12:

$$\overline{A_i} = \frac{1}{N-1} \sum_j A_j \tag{12}$$

where, A_j represents the neighborhood agent action of agent i, and it can be expressed by Eq. 13:

$$A_j = \overline{A_i} + \Delta A_{i,j} \tag{13}$$

where, $\Delta A_{i,j}$represents the distance between the neighborhood agent action of agent i and the mean action code.

Then on this basis, the paired Q functions in Eq. 11 are analyzed, the first-order approximation of the action value Q_i of the agent is used to convert the interaction of multiple agents into the interaction between two agents (as the number of neighborhood agents increases, the accuracy increases, it's because the average value of its high-order terms approaches 0), namely the virtual agent that is composed of an agent and its neighborhood agents.

Thus, Eq. 11 can be rewritten as Eq. 14:

$$Q_i(s,A) = Q_i(s,A_i,\overline{A_i}) \tag{14}$$

Therefore, the multi-agent interaction is converted into two-agent interaction, that is, the mean field theory is applied to simplify the pairwise interaction between an agent *i* and its neighborhood agents to the interaction between the agent *i* and the mean value of its neighborhood agents.

After the agent system performs action A_i, the system state transits from *s* to s', and an instant reward R_i is obtained, then the mean field action value function update formula of the system can be described by Eq. 15 as:

$$Q_i^{t+1}(s,A_i,\overline{A_i}) = Q_i^t(s,A_i,\overline{A_i}) + \eta[R_i + \gamma v_i^t(s') - Q_i^t(s,A_i,\overline{A_i})] \tag{15}$$

where, η represents the learning rate, γ represents the discount factor.

In Eq. 15, the mean field value function is used to replace the commonly used maximum value function to iteratively solve the action value function. The reason is that if the maximum value function is adopted, the cooperation of neighborhood agent strategies is required, and the central agent cannot directly change the strategies of neighborhood agents. Besides, if each agent is greedy to obtain actions, then eventually the algorithm will fail to converge due to the dynamic instability of the environment.

The definition of the state value function is extended to the mean field state value function, which can be expresses by Eq. 16:

$$v_i^t(s) = \sum_{A_i} \pi_i^t(A_i|s,\overline{A_i}) E_{\overline{A_i}(A_{-i}\sim\pi_{-i}^t)}[Q_i^t(s,A_i,\overline{A_i})] \tag{16}$$

In the staged game at each moment, $\overline{A_i}$ is obtained via strategy π_j^t of neighborhood agent *j* at the previous moment, and strategy π_j^t is also described using parameters of the mean action $\overline{A_j}^{t-1}$ of the previous moment, its update process is shown as Eq. 17:

$$\overline{A_i} = \frac{1}{N-1}\sum_j A_j \quad \text{and} \quad A_j \sim \pi_j^t(\cdot\,|s,\overline{A_j}^{t-1}) \tag{17}$$

The strategy π in Eqs. 16 and 17 can also be regarded as the probability distribution of actions taken by an agent. Obviously, the probability that an agent *i* takes the action A_i at time moment *t* depends on the current state and the mean field action $\overline{A_i}$, and it can be described as Eq. 18. The agent strategy is a random behavior in which the action obeys a certain probability distribution. The agent influences itself by observing the history behaviors of other agents in the neighborhood. Under a new state, an agent can determine its own best response action based on the history behaviors of other agents. The strategies of other agents in the neighborhood also obey certain probability distribution rules, and the probability distribution can be determined based on prior knowledge and observed values, that is, to determine the strategy. Therefore, by observing the history behaviors of other agents in the learning process, an agent can learn the strategies of other agents and its influence on the system could be obtained.

$$\pi_i^t(A_i|s,\bar{A}_i) = \frac{\exp\left[-\lambda Q_i^t\left(s,A_i,\overline{A_i}\right)\right]}{\sum_{A_i\in A}\exp\left[-\lambda Q_i^t\left(s,A_{i'},\overline{A_i}\right)\right]} \tag{18}$$

Eqs. 17 and 18 are continuously and interactively updated, which can continuously improve the performance of the strategy and obtain the largest cumulative reward value. In addition, an exploratory factor λ is added to Eq. 18 to try different behaviors by sacrificing some short-term benefits, that is, for each current state, with a certain probability, behaviors that have not been tried before in the current state are tried to collect more information so that the agent could reach the optimal strategy in the macroscopic scale. After the agent takes corresponding actions in the current state, it will observe the new state of the system after the joint action is executed, and timely correct the credibility of other agents in the current state.

In a multi-agent system, the learning of any agent should continuously interact with the learning of other agents and continuously modify parameters to achieve the maximum cumulative

reward value of the system, and the relationships among agents should be established through certain methods. The actions taken when the system transits from the current state to the next state should be jointly determined by the joint actions of the agents, which indirectly realizes the communication between agents.

To ensure the flow of workpieces between different production lines, the action value function of each agent is fitted by the deep neural network, and the loss function is constructed as Eq. 19:

$$L(\theta_i^q) = [y_i - Q_{\theta_i^q}(s, A_i, \overline{A_i})]^2 \tag{19}$$

where, θ_i^q represents the parameters of the agent, $Q_{\theta_i^q}(s, A_i, \overline{A_i})$ represents the action value function fitted by the deep neural network, y_i represents the target value of the mean field value function and is calculated by Eq. 20:

$$y_i = R_i + \gamma v_{\theta_{-i}^q}(s') \tag{20}$$

where, R_i is the reward of agent i, θ_{-i}^q represents the calculation parameter of the mean field value function.

By taking the derivative of Eq. 19, the gradient direction of the parameter could be obtained as shown in Eq. 21:

$$\nabla_{\theta_i^q} L(\theta_i^q) = [y_i - Q_{\theta_i^q}(s, A_i, \overline{A_i})] \nabla_{\theta_i^q} Q_{\theta_i^q}(s, A_i, \overline{A_i}) \tag{21}$$

The parameters can be updated by solving the gradient descent method.

During application, each agent represents a production line. Within the production lines, for any agent i, the action value function under parameters θ_i^q and θ_{-i}^q are initialized and the corresponding mean action $\overline{A_i}$ of the agent is calculated.

The outer loop for sample collection and the inner loop for training are designed. When the collected samples meet the batch size requirement, it enters the inner loop and realizes more efficient learning through the limited samples collected via the interaction between the agents and the environment, as shown in Fig. 3.

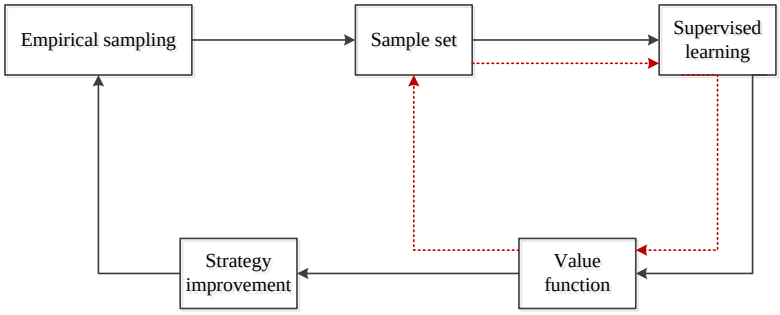

Fig. 3 Algorithm framework

Through the alternation of the two stages: sample collection and storage, and using state action value function to approximate the deep neural network, in the inner loop of the algorithm, the learning algorithm is strengthened through the Q-learning model-free batch processing mode, and it is extended in the multi-agent RL system and iterated by fitting Q. The purpose is to calculate the optimal strategy, and approximate the value function corresponding to the input state and action.

Construct a quintuple as Eq. 22:

$$Tuples = \{(s^t, A^t, \overline{A}^t, R^t, s'^t) | t = 1,2, \ldots, |Tuples|\} \tag{22}$$

where, s represents the current state, A represents the action performed by the agent, $\overline{A}$ represents the mean action, R represents the instantaneous reward value, and s'^t represents the new

state after performing the action. In the outer loop, the algorithm continuously collects and stores samples according to the element information determined by the quintuple, that is:

$$Set = \{(in^t, out^t)|t = 1,2, \dots, |Tuples| \tag{23}$$

where,

$$in^t = (s^t, A^t, \overline{A}^t) \tag{24}$$

$$out^t = R^t + \gamma \max Q^{counter-1}(s'^t, A', \overline{A}') \tag{25}$$

where, *counter* represents the counter constant.

The information of the quintuple is taken as the input, in each loop, after the value function and the counter are initialized as 0, the approximate value of $Q^{counter}$ is calculated through the training set and the regression algorithm. On this basis, a deep neural network is built as the value function approximator to achieve efficient interaction with the environment, each agent calculates and improves via the defined value function to generate high-quality strategies.

When the discrete action space is very large or it is a continuous action space, the deep neural network is taken as the value function approximator to calculate the value function of the state and action $\tilde{Q}(s, A_i, \overline{A_i})$, at the same time, the Bellman equation is used to calculate $R + \gamma \max Q(s', A', \overline{A}')$, and an error function is introduced to calculate the deviation between the two. The back propagation algorithm is used to calculate the connection weight of the deep neural network, so that the error function value is minimized.

$$\left(\tilde{Q}(s, a) - (r + \gamma \, max_{b \in A(s')} \tilde{Q}(s', b))\right)^2 \tag{26}$$

In a multi-agent system, each agent must go through two stages of individual learning and collaborative improvement. For each agent, the specific state, action, mean action are taken as the inputs of the deep neural network, the estimated corresponding state action value function is taken as the output, then the agent will select the subsequent action according to the instruction of the output value. Under the batch processing mode, in each sampling stage, the RL method initializes the state action value function, in the t sampling steps, if the final state s^{final} is not achieved, then the collected sample is stored into set *Tuples*; if it is the final state and the number of samples meets the requirement of |*Tuples*|, then the value function is updated; otherwise, if the number of samples doesn't meet the requirement of |*Tuples*|, then executes the greedy strategy to choose action, and proceed the sampling.

The outer loop in Fig. 3 describes the interaction between the system and the agent whose learning is reinforced under batch processing mode based on the value function. In the algorithm, the orderly dependency between the action set and state is updated, then, it describes the specific realization process of the outer loop of RL framework under the batch processing mode. By constructing the number of tuples that meets the threshold requirement, after sample collection and storage is completed, the algorithm determines whether to enter the inner loop or not, and then executes the value function update of the batch processing mode.

Samples are used to train the deep neural network value function approximator and calculate the out^t of the current mode, the deviation under the batch learning mode can be expressed by Eq. 27:

$$\sum_{t=1}^{|Tuples|} (Q(s^t, A^t, \overline{A}^t) - out^t)^2 \tag{27}$$

Obviously, the optimization goal of the connection parameters of the deep neural network is:

$$Q^* \leftarrow arg \ \min \sum_{t=1}^{|Tuples|} (Q(s^t, A^t, \overline{A}^t) - out^t)^2 \tag{28}$$

In the process of neural network training, the RMSProp algorithm is adopted, this algorithm can well solve the change range of the parameters after update in the optimization, namely the

swing amplitude problem. The RMSProp algorithm uses differential squared weighted averages through the gradients of weights and biases, which is conductive to eliminating the direction with large swing amplitude, and correcting the swing amplitude so that the swing amplitude of each dimension is smaller and the network converges faster. At the same time, the RMSProp algorithm can be built on batch processing training data naturally, and it is easier to be integrated into a batch-processing mode RL algorithm. In this way, the approximator can not only be used for the RL of a single agent, through the RMSProp neural network training algorithm, it also realizes high-quality neural network approximation value function, and it is the key link for the algorithm to be extended to the multi-agent system.

4. Results and discussion of case studies

4.1 Experiment setup

In this study, the experiment constructed 9 states and 10 actions, the features of the states were described by the processing status of the processing machines and the completion status of the workpieces, such as the workload of the processing machines, the estimated earliest possible completion time, and the estimated maximum completion time, etc.

This paper took the minimization of the maximum completion time as the goal to research the workshop scheduling problem, and the global reward was expressed as the sum of local rewards, in this way, the learning system was regarded as a multi-agent learning system with independent rewards, further, in the algorithm, attentions were first paid to the local rewards of each agent obtained according to local scheduling strategies, and then to the global reward. To intuitively understand the characteristics of the distributed scheduling problem, first, the sorting inside each production line should be completed, if the equilibrium condition is not reached, then execute the step of workpiece reallocation between production lines and other steps.

In workshop scheduling, the processing machines were not allowed to perform two operations within a discrete time step. Once a machine starts to perform a certain procedure on the workpiece, it will be in the working state until the end of the procedure. Therefore, for an agent i, its value function was calculated according to the machine state at the decision time point t and the next decision time point $t + \Delta t$.

In the sample collection stage of the agent, the batch size was set to be 100, which means to enter the inner loop of the algorithm when the sample size reaches 100. About the supervised learning part, in the experiment, for the relevant parameters of the RMSProp algorithm, the multi-factor varianceanalysis program was employed to analyze the obtained experimental results and determine the relationships among parameters and generate the best parameter combination.

The parameters were respectively set as

$$dr = (0.75,0.85,0.95,0.99),\ lr = (10^{-4}, 10^{-5}, 10^{-6}),\ sc = (10^{-4}, 10^{-5}, 10^{-6}),$$
$$epoch = (600,800,1000,1200),\ iw = ([-0.06,0.06], [-0.08,0.08], [-0.10,0.10]),$$

and the 5 groups of parameters had a total of $4 \times 3 \times 3 \times 4 \times 3$=432 combinations. After analyzing the experimental results using the variance analysis method, the p-values of the 5 groups of parameters were all lower than the 0.05 confidence interval, indicating that the algorithm was sensitive to these parameters, and finally it's determined that the decay rate of RMSProp algorithm was 0.95, the learning rate lr was 10^{-5}, the small constant sc was 10^{-6}, the epoch of the neural network algorithm was 1000, and the initial weight iw was a random number that is uniformly distributed between [-0.08, 0.08].

The influence of parameters on algorithm performance is shown in Fig. 4.

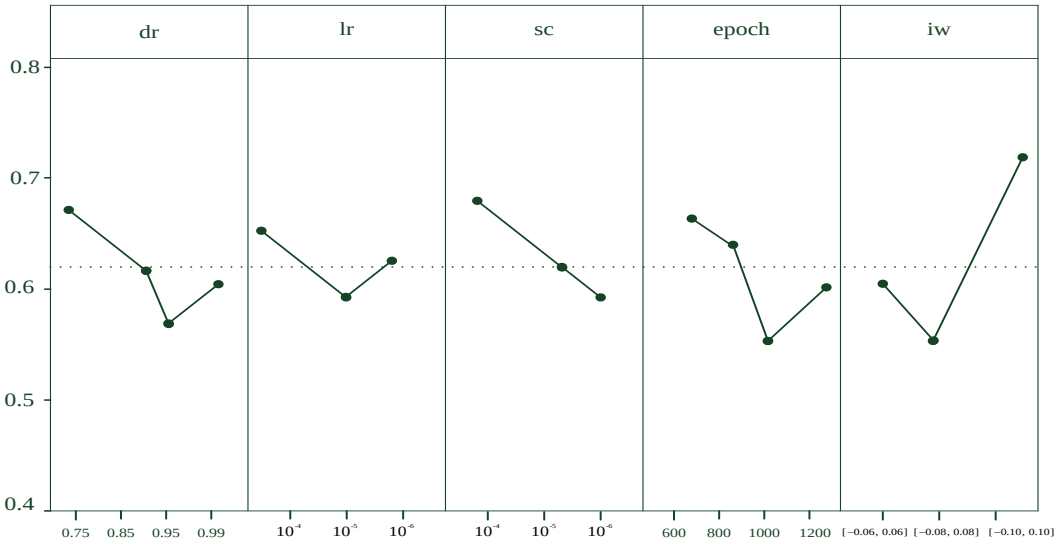

Fig. 4 Influence of parameters on algorithm performance

4.2 Comparison and analysis of results

Based on the benchmark test set Taillard, the performance of the multi-agent RL algorithm was tested, and the results were compared with the results of the iterative greedy algorithm and the benchmark results. In addition, Python was adopted to implement offline training of deep RL and the iterative greedy algorithm was implemented in a Matlab environment. By calculating the average relative percentage deviation (ARPD) index of each test question, the results were compared, as shown in Eq. 29:

$$ARPD = \frac{1}{NR}\sum_{nr=1}^{NR}\frac{C_{max} - C^*_{max}}{C^*_{max}} \times 100 \tag{29}$$

where, NR represents the number of test runs, C_{max} represents the minimum completion time obtained in the nr-th experiment, and C^*_{max} represents the known optimal completion time. The smaller the value of $ARPD$, the better the performance of the algorithm. In the experiment, the number of agents was respectively set as 2, 3, 4, 5, 6, 7, 8, 9, 10 for testing, and the time level was respectively set as 20, 40, and 60. The ARPD comparison results are shown in Tables 1-3.

The calculation results of iterative greedy algorithm (IG), multi-agent RL algorithm (MARL), IG1S, and IG2S [39] were compared. In the small-scale benchmark test, a total of 60 benchmark questions (20 × 5,20 × 10,20 × 20,50 × 5,50 × 10, 50 × 20) in the test set Taillard were selected, and three time levels T = 20, T = 40, and T = 60 were set in the experiment. When T = 60, the average value of ARPD of the MARL algorithm was the smallest among the four algorithms, followed by IG2S, the average ARPD of the IG algorithm designed in this paper was between those of the IG2S and the IG1S, the average ARPD values of the four algorithms were close, and the average ARPD of the IG2S algorithm was 12 % higher than that of the MARL algorithm. When T = 40, the average ARPD values of the four algorithms declined to varying degrees, indicating that within a longer computation time, the algorithms could obtain better experimental results; the average ARPD values of the four algorithms respectively decreased by 5 %, 9 %, 6 %, and 7 %; the IG2S algorithm had the largest average ARPD decrement, followed by the MARL algorithm, whose overall average ARPD still kept the lowest. When T = 60, compared with the results then T = 40, the average ARPD of the algorithms showed no obvious increase, in actual industrial applications, considering the real-time response requirements of the algorithm for the production environment, choosing T = 40 for the algorithm is more appropriate.

Table 1 ARPD comparison (T = 20)

Number of production lines	IG1S	IG2S	IG	MARL
2	0.68	0.52	0.64	0.53
3	0.69	0.60	0.66	0.55
4	0.68	0.63	0.67	0.56
5	0.70	0.64	0.69	0.56
6	0.72	0.66	0.68	0.58
7	0.72	0.68	0.72	0.58
8	0.93	0.79	0.89	0.59
9	0.99	0.89	0.89	0.60
10	1.21	1.05	1.01	0.64
Mean	0.81	0.72	0.76	0.58

Table 2 ARPD comparison (T = 40)

Number of production lines	IG1S	IG2S	IG	MARL
2	0.62	0.48	0.54	0.46
3	0.63	0.49	0.54	0.48
4	0.65	0.52	0.59	0.50
5	0.65	0.52	0.69	0.49
6	0.68	0.66	0.60	0.51
7	0.73	0.68	0.68	0.53
8	0.90	0.73	0.79	0.54
9	0.92	0.83	0.86	0.54
10	1.19	0.92	0.98	0.60
Mean	0.77	0.65	0.70	0.52

Table 3 ARPD comparison (T = 60)

Number of production lines	IG1S	IG2S	IG	MARL
2	0.60	0.46	0.54	0.45
3	0.62	0.46	0.53	0.46
4	0.65	0.50	0.58	0.50
5	0.63	0.50	0.66	0.49
6	0.66	0.65	0.70	0.50
7	0.71	0.66	0.67	0.53
8	0.90	0.73	0.73	0.53
9	0.91	0.81	0.81	0.54
10	1.17	0.91	0.95	0.60
Mean	0.76	0.63	0.69	0.51

The comparison of average ARPD values of the four algorithms at different time levels in the small-scale test is shown in Fig. 5.

In the experiment of large-scale calculation examples, five types of scales (100 × 5,100 × 10,100 × 20,200 × 20,500 × 20) and a total of 50 calculation examples were chosen for the tests. Similarly, three time levels T = 20, T = 40, and T = 60 were set in the experiment, and the results of the ARPD values of the four algorithms are shown in Tables 4-6.

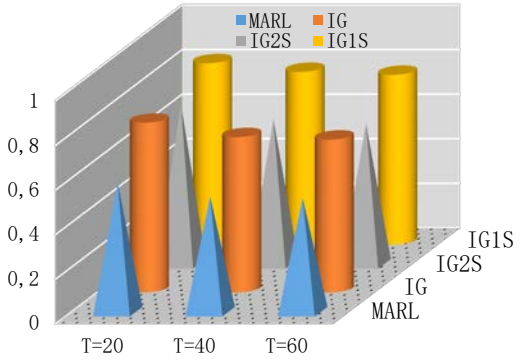

Fig. 5 Comparison of average ARPD of small-scale calculation examples

Table 4 ARPD comparison (T = 20)

Number of production lines	IG1S	IG2S	IG	MARL
2	1.42	1.02	1.37	0.83
3	1.76	1.32	1.66	0.95
4	1.78	1.53	1.79	1.13
5	1.90	1.74	1.94	1.35
6	2.54	2.34	2.44	1.67
7	2.98	2.58	2.68	1.68
8	3.55	2.87	3.34	1.89
9	3.89	3.01	3.90	1.90
10	4.34	3.05	4.39	2.21
Mean	2.68	2.16	2.61	1.51

Table 5 ARPD comparison (T = 40)

Number of production lines	IG1S	IG2S	IG	MARL
2	1.40	1.00	1.35	0.73
3	1.74	1.30	1.64	0.84
4	1.72	1.51	1.77	1.12
5	1.90	1.70	1.94	1.34
6	2.51	2.33	2.43	1.62
7	2.96	2.57	2.67	1.63
8	3.59	2.84	3.33	1.80
9	3.89	3.00	3.90	1.90
10	4.32	3.02	4.38	2.19
Mean	2.67	2.14	2.60	1.46

Table 6 ARPD comparison (T = 60)

Number of production lines	IG1S	IG2S	IG	MARL
2	1.40	1.00	1.34	0.69
3	1.73	1.27	1.64	0.80
4	1.72	1.49	1.77	1.07
5	1.90	1.70	1.92	1.31
6	2.50	2.31	2.43	1.62
7	2.96	2.53	2.67	1.62
8	3.58	2.84	3.31	1.80
9	3.89	3.00	3.90	1.88
10	4.31	3.02	4.36	2.16
Mean	2.67	2.13	2.59	1.44

In the three large-scale experiments, the workpieces and processing machines increased significantly, and the ARPD values of the algorithms had been improved greatly, indicating that when dealing with large-scale problems, the algorithms' problem-solving abilities had reduced. Taking T = 40 as an example, by comparing the ARPD values of the algorithms in small-scale and large-scale scenarios, the MARL algorithm had the smallest increment, followed by the IG2S algorithm, which indicated that when dealing with large-scale problems, the multi-agent RL algorithm designed in this paper also showed obvious superiority.

In the experiment of large-scale calculation examples, the comparison of average ARPD values of the four algorithms at different time levels is shown in Fig. 6, obviously, the ARPD of the MARL algorithm was the smallest.

In order to further test the performance of the algorithm to solve a larger scale and keep other conditions unchanged, when the number of production lines increases to 15 and 20, it can be found that the performance of the proposed algorithm decreases significantly and loses its leading edge compared with other algorithms. This shows that the algorithm has great limitations in solving large-scale problems. How to further improve the algorithm and improve the performance of the algorithm in solving large-scale problems is a problem we are currently exploring.

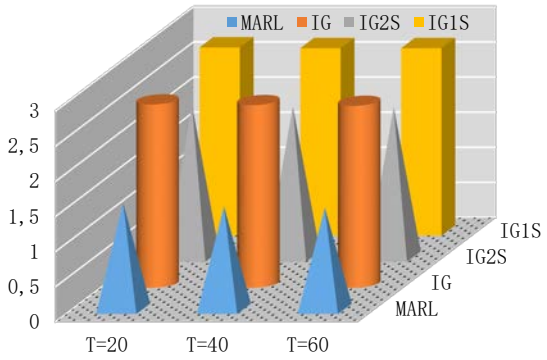

Fig. 6 Comparison of mean ARPD in large-scale computation examples

This paper designed a multi-agent RL method to solve DPFSPs. Based on the NASH equilibrium theory and the NASH Q-learning method, a multi-agent MF-DRL algorithm had been proposed in the study, and global perspective algorithm elements such as joint state and joint actions were constructed. The experimental results showed that, the proposed multi-agent RL method was effective in solving DPFSPs, and it outperformed other algorithms in case of large-scale problems. The RL method proposed in this paper was mainly based on the value function method, it hadn't involved the strategy-based RL method, and now the effect of the strategy-based RL method on the solution of distributed production scheduling hasn't been explored yet, studies of this aspect will be carried out in the subsequent research.

5. Conclusion

This paper designed a multi-agent RL method to solve DPFSPs. Based on the NASH equilibrium theory and the NASH Q-learning method, a multi-agent MF-DRL algorithm had been proposed in the study, and global perspective algorithm elements such as joint state and joint actions were constructed. The experimental results showed that, the proposed multi-agent RL method was effective in solving DPFSPs, and it outperformed other algorithms in case of large-scale problems. The RL method proposed in this paper was mainly based on the value function method, it hadn't involved the strategy-based RL method, and now the effect of the strategy-based RL method on the solution of distributed production scheduling hasn't been explored yet, studies of this aspect will be carried out in the subsequent research.

Acknowledgment

Project supported by the Key Soft Science Project of "Science and Technology Innovation Action Plan of Shanghai Science and Technology Commission, China (No. 20692104300). National Natural Science Foundation, China (No. 71840003), and the Technology Development Project of University of Shanghai for Science and Technology, China (No. 2018KJFZ043).

References

[1] Behnamian, J., Fatemi Ghomi, S.M.T. (2016). A survey of multi-factory scheduling, *Journal of Intelligent Manufacturing*, Vol. 27, No. 1, 231-249, doi: 10.1007/s10845-014-0890-y.

[2] Somashekhara, S.C.H., Setty, A.K.Y., Sridharmurthy, S.M., Adiga, P., Mahabaleshwar, U.S., Lorenzini, G. (2019). Makespan reduction using dynamic job sequencing combined with buffer optimization applying genetic algorithm in a manufacturing system, *Mathematical Modelling of Engineering Problems*, Vol. 6, No. 1, 29-37, doi: 10.18280/mmep.060104.

[3] Zhang, Y.Q., Zhang, H. (2020). Dynamic scheduling of blocking flow-shop based on multi-population ACO algorithm, *International Journal of Simulation Modelling*, Vol. 19, No. 3, 529-539, doi: 10.2507/IJSIMM19-3-CO15.

[4] Wang, L., Deng, J., Wang, S.-Y. (2016). Survey on optimization algorithms for distributed shop scheduling, *Control and Decision*, Vol. 31, No. 1, 1-11.

[5] Fernandez-Viagas, V., Framinan, J.M. (2015). A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem, *International Journal of Production Research*, Vol. 53, No. 4, 1111-1123, doi: 10.1080/00207543.2014.948578.

[6] Gao, J., Chen, R. (2011). An NEH-based heuristic algorithm for distributed permutation flowshop scheduling problems, *Scientific Research and Essays*, Vol. 6, No. 14, 3094-3100.
[7] Gao, J., Chen, R., Deng, W. (2013). An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem, *International Journal of Production Research*, Vol. 51, No. 3, 641-651, doi: 10.1080/00207543.2011.644819.
[8] Naderi, B., Ruiz, R. (2014). A scatter search algorithm for the distributed permutation flowshop scheduling problem, *European Journal of Operational Research*, Vol. 239, No. 2, 323-334, doi: 10.1016/j.ejor.2014.05.024.
[9] Lin, S.-W., Ying, K.-C., Huang, C.-Y. (2013). Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm, *International Journal of Production Research*, Vol. 51, No. 16, 5029-5038, doi: 10.1080/00207543.2013.790571.
[10] Yang, S.L., Xu, Z.G., Li, G.Z., Wang, J.Y. (2020). Assembly transport optimization for a reconfigurable flow shop based on a discrete event simulation, *Advances in Production Engineering & Management,* Vol. 15, No. 1, 69-80, doi: 10.14743/apem2020.1.350.
[11] Wang, S.-Y., Wang, L., Liu, M., Xu, Y. (2013). An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem, *International Journal of Production Economics*, Vol. 145, No. 1, 387-396, doi: 10.1016/j.ijpe.2013.05.004.
[12] Rifai, A.P., Nguyen, H.-T., Dawal, S.Z.M. (2016). Multi-objective adaptive large neighborhood search for distributed reentrant permutation flow shop scheduling, *Applied Soft Computing*, Vol. 40, 42-57, doi: 10.1016/j.asoc.2015.11.034.
[13] Wang, K., Huang, Y., Qin, H. (2016). A fuzzy logic-based hybrid estimation of distribution algorithm for distributed permutation flowshop scheduling problems under machine breakdown, *Journal of the Operational Research Society*, Vol. 67, No. 1, 68-82, doi: 10.1057/jors.2015.50.
[14] Komaki, M., Malakooti, B. (2017). General variable neighborhood search algorithm to minimize makespan of the distributed no-wait flow shop scheduling problem, *Production Engineering,* Vol. 11, No. 3, 315-329, doi: 10.1007/s11740-017-0716-9.
[15] Chen, W., Hao, Y.F. (2018). Genetic algorithm-based design and simulation of manufacturing flow shop scheduling, *International Journal of Simulation Modelling,* Vol. 17, No. 4, 702-711, doi: 10.2507/IJSIMM17(4)CO17.
[16] Lebbar, G., El Abbassi, I., Jabri, A., El Barkany, A., Darcherif, M. (2018). Multi-criteria blocking flow shop scheduling problems: Formulation and performance analysis, *Advances in Production Engineering & Management,* Vol. 13, No. 2, 136-146, doi: 10.14743/apem2018.2.279.
[17] Rathinam, B., Govindan, K., Neelakandan, B.,Raghavan, S.S. (2015). Rule based heuristic approach for minimizing total flow time in permutation flow shop scheduling, *Tehnički Vjesnik – Technical Gazette,* Vol. 22, No. 1, 25-32, doi: 10.17559/TV-20130704132725.
[18] Babu, K.S., Vemuru, S. (2019). Spectrum signals handoff in LTE cognitive radio networks using reinforcement learning, *Traitement du Signal*, Vol. 36, No. 1, 119-125, doi: 10.18280/ts.360115.
[19] Foerster, J., Nardelli, N., Farquhar, G., Afouras, T., Torr, P.H.S., Kohli, P., Whiteson, S. (2017). Stabilising experience replay for deep multi-agent reinforcement learning, In: *Proceedings of 34th International Conference on Machine Learning,* Sydney, Australia, 1146-1155.
[20] Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Vicente, R. (2017). Multiagent cooperation and competition with deep reinforcement learning, *PloS ONE*, Vol. 12, No. 4, Article No. e0172395, doi: 10.1371/journal.pone.0172395.
[21] Xu, X., Jia, Y., Xu, Y., Xu, Z., Chai, S., Lai, C.S. (2020). A multi-agent reinforcement learning-based data-driven method for home energy management, *IEEE Transactions on Smart Grid*, Vol. 11, No. 4, 3201-3211, doi: 10.1109/TSG.2020.2971427.
[22] Zerguine, N., Mostefai, M., Aliouat, Z., Slimani, Y. (2020). Intelligent CW selection mechanism based on Q-learning (MISQ), *Ingénierie des Systèmes d'Information*, Vol. 25, No. 6, 803-811, doi: 10.18280/isi.250610.
[23] Sukhbaatar, S., Szlam, A., Fergus, R. (2016). Learning multiagent communication with backpropagation, In: *Proceedings of 30th Conference on Neural Information Processing Systems (NIPS 2016),* Barcelona, Spain, 2252-2260.
[24] Foerster, J.N., Assael, Y.M., de Freitas, N., Whiteson, S. (2016). Learning to communicate with deep multi-agent reinforcement learning, In: *Proceedings of 30th Conference on Neural Information Processing Systems (NIPS 2016),* Barcelona, Spain, 2137-2145.
[25] He, H., Boyd-Graber, J., Kwok, K., Daumé III, H. (2016). Opponent modeling in deep reinforcement learning, In: *Proceedings of 33rd International Conference on Machine Learning,* New York, USA, 1804-1813.
[26] Palmer, G., Tuyls, K., Bloembergen, D., Savani, R. (2018). Lenient multi-agent deep reinforcement learning, In: *Proceedings of 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018,* Stockholm, Sweden, 443-451.
[27] Omidshafiei, S., Pazis, J., Amato, C., How, J.P., Vian, J. (2017). Deep decentralized multi-task multi-agent reinforcement learning under partial observability, In: *Proceedings of the 34th International Conference on Machine Learning (ICML 2017),* Sydney, Australia, 2681-2690.
[28] Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W.M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J.Z., Tuyls, K, Graepel, T. (2018). Value-decomposition networks for cooperative multi-agent learning based on team reward, In: *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018),* Stockholm, Sweden, 2085-2087.

[29] Li, C., Tang, J., Ma, T., Yang, X., Luo, Y. (2020). Load balance based workflow job scheduling algorithm in distributed cloud, *Journal of Network and Computer Applications*, Vol. 152, Article No. 102518, doi: 10.1016/j.jnca.2019.102518.
[30] Neary, C., Xu, Z., Wu, B., Topcu, U. (2021). Reward machines for cooperative multi-agent reinforcement learning, In: *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '21,* Virtual Event, United Kingdom, 934-942.
[31] Li, R., Han, Y., Ma, T., Liu, H. (2020). Nash-Q learning-based collaborative dispatch strategy for interconnected power systems, *Global Energy Interconnection*, Vol. 3, No. 3, 227-236, doi: 10.1016/j.gloei.2020.07.004.
[32] Li, J., Xiao, Z. (2020). H∞ control for discrete-time multi-player systems via off-policy Q-learning, *IEEE Access*, Vol. 8, 28831-28846, doi: 10.1109/ACCESS.2020.2970760.
[33] Shah, D., Somani, V., Xie, Q., Xu, Z. (2020). On reinforcement learning for turn-based zero-sum Markov games, In: *Proceedings of the 2020 ACM-IMS on Foundations of Data Science Conference, FODS '20,* Virtual Event, USA, 139-148, doi: 10.1145/3412815.3416888.
[34] Feng, Q., Shang, W.-P., Jiao, C.-W., Li, W.-J. (2020). Two-agent scheduling on a bounded parallel-batching machine with makespan and maximum lateness objectives, *Journal of the Operations Research Society of China*, Vol. 8, No. 1, 189-196, doi: 10.1007/s40305-019-00258-9.
[35] Littman, M.L. (1994). Markov games as a framework for multi-agent reinforcement learning, In*: Proceedings of the Eleventh International Conference on International Conference on Machine Learning, ICML'94,* New Brunswick USA, 157-163, doi: 10.1016/B978-1-55860-335-6.50027-1.
[36] Hu, J., Wellman, M.P. (1998). Multiagent reinforcement learning: Theoretical framework and an algorithm, In: *Proceedings of the Fifteenth International Conference on Machine Learning, ICML '98,* Madison, USA, 242-250.
[37] Xiao, L., Bahri, Y., Sohl-Dickstein, J., Schoenholz, S., Pennington, J. (2018). Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks, In: *Proceedings of the 35th International Conference on Machine Learning,* Stockholm, Sweden, 5393-5402.
[38] Yang, Y., Luo, R., Li, M., Zhou, M., Zhang, W., Wang, J. (2018). Mean field multi-agent reinforcement learning, In: *Proceedings of the 35th International Conference on Machine Learning,* Stockholm, Sweden, 5571-5580.
[39] Ruiz, R., Pan, Q.-K., Naderi, B. (2019). Iterated Greedy methods for the distributed permutation flowshop scheduling problem, *Omega*, Vol. 83, 213-222, doi: 10.1016/j.omega.2018.03.004.

Appendix

The abbreviations used in the article:

RL	Reinforcement Learning
MF	Mean Field
DFSP	Distributed Flow-Shop Scheduling Problem
DPFSP	Distributed Permutation Flow-shop Scheduling Problem
DRL	Deep Reinforcement Learning
MF-DRL	Mean Field Deep Reinforcement Learning
ARPD	Average Relative Percentage Deviation
MARL	Multi agent Reinforcement Learning
RMSProp	Root Mean Square Prop
IG	Iterative Greedy
IG1S	The single stage Iterative Greedy
IG2S	The two stage Iterative Greedy