

Technical paper

From Nash Q-learning to nash-MADDPG: Advancements in multiagent control for multiproduct flexible manufacturing systems

Muhammad Waseem, Qing Chang^{*}

Department of Mechanical and Aerospace Engineering, University of Virginia, Charlottesville, VA 22903, USA

ARTICLE INFO

Keywords:

Multiagent control
Mobile robot scheduling
Multiproduct FMS
Nash Q-learning
MADDPG
Deep reinforcement learning

ABSTRACT

The emergence of flexible manufacturing systems (FMS) capable of processing multiple product types is a result of the growing demand for product customization and personalization. Such multiproduct systems are characterized by a higher level of uncertainty and variability when compared to traditional manufacturing systems. This paper proposes a Nash integrated multiagent deep deterministic policy gradient method (Nash-MADDPG) to control the mobile robots' assignment in a multiproduct FMS to enable intelligent decision-making, interaction, and dynamic learning capabilities. A mathematical model of a multiproduct FMS from a previous study is described, and system dynamic property is characterized by permanent production loss (PPL). Then, by observing PPL of the system and market demand for each product type, the multi-agent control scheme is developed to assign mobile robots to load/unload various product types at various machines. First, a Nash game is developed among the mobile robots and to improve the cooperation, a collaboration cost is defined. This collaboration cost is then used in the reward function of the multiagent deep deterministic policy gradient (MADDPG) algorithm. Second, the actions are jointly defined based on the action values of MADDPG, and the mobile robots' strategies in the Nash game, which update the environment to a new state. The performance of the proposed method is verified by comparing it with conventional Nash Q-learning, vanilla MADDPG, Q-learning based single agent reinforcement learning (SARL) and a first-come-first-serve (FCFS) based control. The results demonstrate that the multi-agent control scheme under the proposed Nash-MADDPG is effective in dealing with cooperative FMS environment that involves complicated dynamics and uncertainties.

1. Introduction

The manufacturing sector is undergoing a transformation shaped by evolving customer preferences. As the desire for personalized products surges, flexible manufacturing systems (FMS) have garnered considerable interest from both industry practitioners and academia. This escalating demand for tailor-made goods presents a substantial challenge for manufacturers in meeting these evolving needs.

In response to this challenge, our prior research [1,2] introduced a multiproduct FMS, crafting a robust mathematical model and overseeing its operations through a Q-learning algorithm rooted in reinforcement learning (RL). Nevertheless, as the environmental complexity escalates, exemplified by the presence of extensive state-action spaces, the pursuit of efficient scheduling becomes increasingly intricate. This intricacy is compounded by a range of inherent attributes, including the diversity of product types with distinctive processing workflows, sporadic interruptions, uncertain shifts in product demand, and the substantial

presence of machines, buffers, and mobile robots. To address the intricacies of complex scheduling, the system necessitates a decentralized control approach [3] whereby the mobile robots must possess the intelligence to determine the optimal product-machine assignments at specific times. This task is streamlined within the framework of a multiagent system (MAS), where each mobile robot is regarded as an autonomous agent within the overarching MAS [4].

In the context of managing a MAS, two predominant strategies are commonly employed: game-theoretic models [5] and multiagent reinforcement learning (MARL) [6]. Game-theoretic models depict agents as players who, guided by predefined rules, engage in either competitive endeavors to maximize their individual rewards or cooperative actions aimed at optimizing collective rewards [7]. One advantage of these models is their ability to capture strategic interactions among agents, allowing for a formal representation of scenarios where agents act independently or in concert. For instance, [8] employed Nash Q-learning to enhance assembly processes, while [9] harnessed a game

^{*} Corresponding author.

E-mail address: qc9nq@virginia.edu (Q. Chang).

<https://doi.org/10.1016/j.jmansys.2024.03.004>

Received 10 September 2023; Received in revised form 19 February 2024; Accepted 7 March 2024

Available online 15 March 2024

0278-6125/© 2024 The Society of Manufacturing Engineers. Published by Elsevier Ltd. All rights reserved.

theory-based decision framework for autonomous driving. However, while effective in certain scenarios, game-theoretic models have notable limitations. They tend to excel in relatively static environments where the rules and constraints remain consistent [10]. These models also primarily cater to two-player scenarios, displaying limitations when extended to systems involving multiple agents [11]. Moreover, in dynamic environments characterized by constant changes and interactions, game-theoretic models face challenges in maintaining system stability and optimizing processes. The complexities arising from multiple agents and dynamic settings often hinder these models from achieving efficient coordination and stability in decision-making processes. These limitations are particularly prominent when attempting to scale the models to handle larger groups of agents or highly dynamic scenarios within Multi-Agent Systems.

On the other hand, MARL has received significant attention in the recent literature [12]. It has the ability to deal with dynamic and complex environments. For example [13] used MARL based approach to optimize the energy efficient scheduling in a serial production line, resulting in notable reductions in energy consumption. However, this methodology isn't directly applicable to our multiproduct FMS, given its primary focus on energy optimization and lack of emphasis on material handling, a critical aspect managed through mobile robots in our system. Similarly, [14] used MARL with Graph Neural Networks (GNN) to devise an optimum strategy for a joint scheduling and predictive maintenance. Their approach showed promising outcomes, potentially suitable for our system, where machines and robots can be incorporated into a graph structure. However, it might increase computational complexity, posing a challenge in implementation. The training efficiency of algorithms used for MARL such as MADDPG [15] and MAPPO [16] etc. depends on the complexity of the environment and may take significant time to reach stability.

This paper combines game theory, represented by the Nash game, with a multi-agent reinforcement learning approach known as MADDPG. It employs the joint output of MADDPG and the Nash game to dictate actions within the environment, specifically the allocation of mobile robots and product types to machines. Within the Nash game, there exists a collaboration cost that incentivizes interactive players (the mobile robots) to cooperate and reduce it. At each time step, the mobile robots make strategic choices based on the rewards they receive. In the MADDPG framework, action values generated are merged with the strategies from the game, and the resulting joint actions are applied to the environment to produce a new state. The rewards received from the environment encompass the collaboration cost from the Nash game. These rewards are used to update action values for MADDPG agents and strategies for Nash players. Consequently, the Nash game fosters collaboration, leading to the attainment of equilibrium, while MADDPG rapidly adapts, benefiting from insights into the collaboration cost. The overarching objective of this system is to minimize machine wait times and enhance customer demand satisfaction. The major contributions of this paper are:

1. A Nash integrated MADDPG (Nash-MADDPG) algorithm is introduced where the action values generated by MADDPG are combined with the strategies of players based on a Nash game. The Nash game is used to handle the interactions among mobile robots, which is further used to design collaboration cost functions for players in the game and a unique Nash equilibrium is proved.
2. The proposed Nash integrated MADDPG is evaluated for its feasibility and effectiveness through comparison with vanilla MADDPG, conventional Nash Q-learning, Q-learning based single agent reinforcement learning (SARL), and FCFS based control methods.

The remainder of the paper is structured as follows. In section II, literature review is addressed. In section III, the system description is provided. The system model is covered in section IV, followed by the decision framework in section V. Section VI addresses the proposed

method. The results of the case study, which compares the proposed method to other control methods, are presented in section VII. Finally, the conclusion is given in section VIII.

2. Literature review

Efficient production scheduling is crucial for improving the throughput of a production line without structural changes. However, it is challenging to derive an efficient scheduling solution within a reasonable time because most production scheduling problems are nondeterministic polynomial-time (NP) hard with complex practical constraints [17]. Therefore, several solution approaches have been used in the literature to address the issue of complex scheduling. These solutions comprise analytical [17] and heuristic methods [18]. However, analytical approaches cannot be used for NP-hard problems [17]. Heuristics further divides into search based and learning based approaches. Although search-oriented heuristics cannot be employed everywhere due to the large state space constraint [18], learning based heuristics i. e., reinforcement learning (RL) are commonly used [19]. The selection of learning-based approach further depends on the type and nature of system.

In the case of a multiproduct FMS, each product type follows a unique sequence of operations. The products are handled by mobile robots and the system performance such as production count of each product type, is not only influenced by the control of the mobile robots, but also the dynamic production system state [2]. Therefore, the production system model and control policy are highly coupled. What makes the problem more challenging is that managing multiple mobile robots requires observing not only the production line, but also their colleagues, and negotiating with them before taking any actions. In such a case, the mobile robots should be intelligent enough to decide which product to be assigned to which machine at a certain time. This capability is made easy under a multiagent system (MAS) [4], where each mobile robot is treated as an individual agent. The multi-agent system involves a network of agents that make independent decisions based on their local information, leading to an efficient operation of the overall system [20]. This concept is well-suited for responding quickly and effectively to changing conditions in manufacturing system [21].

For a manufacturing system to be flexible and responsive, it requires three capabilities. First, it must have the ability for agents to make autonomous decisions. Second, agents must have the ability to interact and collaborate with each other. Third, the system must have the intelligence to learn and make final decisions based on the environment. Similarly, in a MAS, there are always two challenges present: coordination and non-stationarity. Firstly, coordination is necessary between agents as the correctness of any one agent's actions can depend on its teammates' actions [22]. Secondly, the agents are simultaneously learning and updating their behaviors, which from the viewpoint of any individual agent makes the learning problem non-stationary; the optimal solution for each agent is constantly changing [23]. To address these requirements, two major approaches are usually adopted in the literature: Game theoretic models and Multiagent reinforcement learning (MARL) algorithms. Both these methods can address a MAS.

The concept of equilibrium in game theory, coupled with multi-agent training methods, is widely recognized as highly effective in addressing complex optimization challenges involving multiple constraints and agents. For instance, Hu et al. [24] introduced a Q-learning-based approach to attain Nash equilibria in stochastic games with general-sum characteristics. In a related context, Nash Q-learning was applied to devise a collaborative dispatch strategy for interconnected power systems [25]. Fu and Chai [7] proposed an online adaptive algorithm for learning Nash equilibrium solutions in nonlinear zero-sum game scenarios. This synergy between equilibrium concepts and multi-agent methodologies extends to various domains, including supply chain management [26,27], cloud manufacturing [28] [29], and Energy management [30]. Additionally, [31] have designed game theory

models for solving complex optimization problems, such as the Flexible Job Shop Scheduling Problem (FJSP). In a different context, [32] applied non-cooperative game theory with complete information to create a novel scheduling model for the FJSP considering machine breakdowns. This approach optimized the conflicting objectives of robustness and stability simultaneously. Similarly, [33] applied non-cooperative game theory to address multi-objective scheduling problems in automated manufacturing systems. However, it's essential to note that while game theoretic models can offer valuable insights into strategic interactions and decision-making among rational agents, they may not be well-suited for complex environments characterized by a vast number of state-action spaces. Additionally, these models often assume agents possess complete information, which may not always be the case. Furthermore, game theoretic models tend to assume static environments, which may not align with real-world scenarios.

On the other hand, RL based algorithms like MADDPG can easily deal with complex and dynamic environments. For instance, [34,35] used RL techniques to analyze and control a serial production line with constant and variable cycle times, whereas [14] used deep RL to develop a predictive maintenance policy for a serial production line. Similarly, [15, 36] used RL and game-theoretic models to solve multi-agent scheduling problems. In the context of multi-agent environments, [37] introduced an actor-hierarchical attention critic to enhance cooperation among agents and facilitate efficient learning in mixed tasks. To combine the advantages of multi-robot systems and hierarchical systems in deep reinforcement learning (DRL), [38] proposed a multi-agent hierarchical deep deterministic policy gradient (DDPG) method. However, it's important to note that the computational complexity of these algorithms is contingent upon the specific problem at hand and may sometimes necessitate a substantial duration to fully grasp the intricacies of the environment.

Therefore, this paper introduces a novel approach known as Nash integrated MADDPG (Nash-MADDPG), specifically tailored to address the multi-agent scheduling challenge posed by mobile robots within a multiproduct Flexible FMS. This innovative method seamlessly combines the Nash game and the MADDPG algorithm, yielding joint actions that navigate the complex landscape of multi-agent coordination.

3. System description

In this paper, a multi-product FMS is considered, which consists of M machines, $M-1$ intermediate buffers, and n mobile robots for loading and unloading parts. The line is shown in Fig. 1 and is made up of machines (denoted as S_i , $i = 1, 2, \dots, M$, represented by rectangles), buffers (denoted as B_i , $i = 1, 2, \dots, M-1$, represented by circles), a source for raw products, and a sink for finished products. The production line processes l types of products. Once a machine has completed its processing, a mobile robot unloads the partially processed product to the

corresponding downstream buffer, depending on product type, and loads a new part from the upstream buffer. The flowlines between stations and buffers with different styles represent the processing flows of different product types.

A summary of the notations used in the environment is presented in Table 1. The following constraints are considered in this study:

- The production line is capable of processing l different product types, each with its own unique sequence of operations;
- Every part must go through the first and last machine (S_1 and S_M);
- Every machine must process at least one product type;
- Part and product type are used interchangeably in this paper;
- Agent and player are used interchangeably, and both address a mobile robot in this paper;
- The intermediate buffer only contains product types meant to be processed by the next machine;
- Reward and payoff in this paper are denoted and calculated similarly. However, the payoff is used in the game while reward is used in the RL;
- Strategy and policy refer to the same concept, but strategy is used in the game and policy in the RL;
- A machine is considered blocked if it is operational, but its downstream buffer is full, and considered starved if it is operational but its upstream buffer is empty.

4. System Modelling

In our previous work [1], a data-driven model is developed for a multiproduct FMS to effectively assess the current states of the system in real-time and control the mobile robot assignment based on Q-learning algorithm. In order to provide a comprehensive understanding of the current paper, a concise overview of the main findings and the modeling approach are presented in this section, without delving into the detailed derivation. The behavior of the system can be depicted through the following state-space equation:

$$\dot{\mathbf{b}}(t) = \mathbf{F}(\mathbf{b}(t), \mathbf{U}(t), \mathbf{W}(t)) \quad (1)$$

In the context of this system, the parameters are defined as.

$\mathbf{b}(t) = [\vec{b}_1(t), \vec{b}_2(t), \dots, \vec{b}_{M-1}(t)]'$ represents the buffer levels at time t ;

$\mathbf{W}(t) = [w_1(t), w_2(t), \dots, w_M(t)]'$ represents the disturbances at time t , where $w_j(t)$ describes whether S_j suffers from a disruption at time t . If $\exists \vec{e}_k \in E$ s.t. $\vec{e}_k = (i, t_k, d_k)$ and $t \in [t_k, t_k + d_k]$, then, $w_i(t) = 1$, otherwise, $w_i(t) = 0$. Define $\theta_i(t)$ as the status of a machine S_i at time t i.e., $\theta_i(t) = 1 - w_i(t)$. A machine S_i is up at time t when $w_i(t) = 0$, and down when $w_i(t) = 1$;

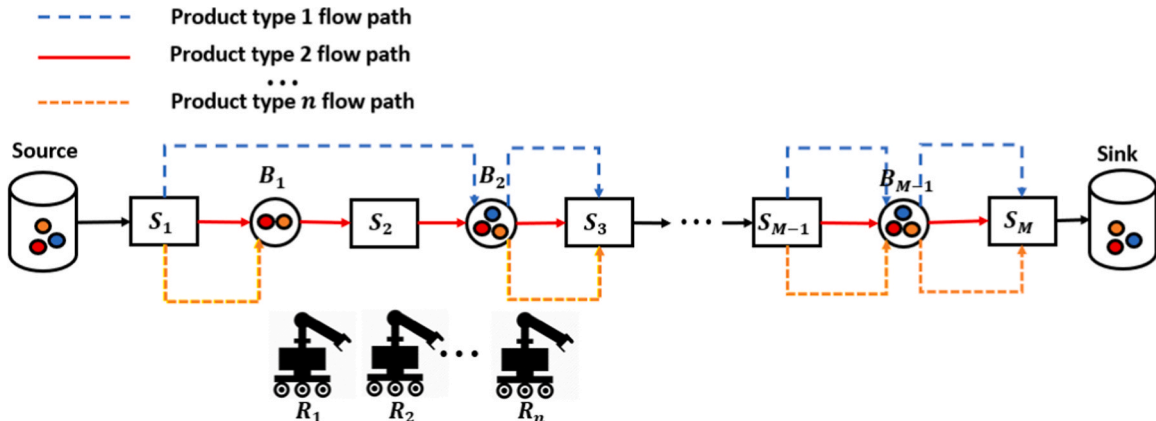


Fig. 1. A general multiproduct FMS with n product types, handled by n mobile robots.

Table 1

Table of notations.

Symbol	Definition
S_i	Machine i , where $i = 1, 2, \dots, M$
S_{sc}	Slowest critical machine
B_j	Buffer j (Also denote buffer capacity), where $j = 1, 2, \dots, M - 1$
l	Product type, $l = 1, 2, \dots, K$
$\vec{b}_j(t)$	Level of buffer j at time t
T_i	Processing time of machine S_i
T_{load}^i	Loading time on machine S_i
T_{unload}^i	Unloading time on machine S_i
T_{travel}	Travel time of mobile robot between machine S_i and S_{i+1}
T_{travel}^{ij}	Travel time between S_i and S_j ; $T_{travel}^{ij} = j - i \times T_{travel}$, ($i, j = 1, 2, \dots, M$) and $i \neq j$
\vec{e}_i	$\vec{e}_i = (j, t_i, d_i)$, Disruption event j happened at time t_i and lasting for a duration of d_i
\vec{e}_{vi}	Virtual disruption event i
$d_i(t)$	Duration of downtime on machine S_i at time t
$D_l(t)$	Market demand of product type l at time t
R_k	Mobile robot k , where $k = 1, \dots, K$
$w_i(t)$	Disturbance i at time t
$u_{ij}^l(t)$	Control input of mobile robot i for machine S_j and product type l at time t
OW_i	Opportunity window of machine S_i
PPL_i	Permanent production loss of machine S_i
a_t^i	Action of agent i at time t
r_t^i	Reward (payoff) of agent (player) i at time t
s_t	Environment state at time t
γ	Discount factor
α	Learning rate
α_a	Actor learning rate in MADDPG
α_c	Critic learning rate in MADDPG
π_t^i	Policy (Strategy) of agent i at time t
π_t^{-}	Policies (Strategies) of agents (players) other than player i at time t
π_t^{*}	Optimal strategy (policy) of agent (player) i at time t
π_t^{-*}	Optimal strategies (policies) of agents (players) other than player i at time t
o_t^i	Observation of agent i at time t
$g_i(t)$	Mobile robot's assignment status to machine i at time t
$\theta_i(t)$	Running status of machine i at time t
\vec{P}_t	Vector of product types loaded on to the machines at time t
CT	Cost of demand dissatisfaction
$CoD(t)$	Cost of delay at time t
ω_d^l	Delay cost rate for product type l
$CoS(t)$	Cost of surfeit at time t
ω_s^l	Surfeit cost rate for product type l
C_i	Collaboration cost of player (Mobile robot) i
$Y_l(t)$	Last machine production (system's throughput) for product type l at time t
θ_i	Constant of state change cost for player (agent) i
φ_i	Constant of disagreement cost for player (agent) i
λ, Ω	Collaboration constants
μ_i	Action probability for agent i in MADDPG
Q_i	State-action value for agent i
Ψ_k^u	Actor-network weights
Ψ_k^Q	Critic-network weights
D	Replay buffer in DQN
\mathcal{D}_k	Replay buffer of agent k in MADDPG

$$U(t) = \begin{bmatrix} u_{11}^1(t) & u_{12}^2(t) & \dots & u_{1M}^l(t) \\ u_{21}^1(t) & u_{22}^2(t) & \dots & u_{2M}^l(t) \\ \vdots & \vdots & \vdots & \vdots \\ u_{K1}^1(t) & u_{K2}^2(t) & \dots & u_{KM}^l(t) \end{bmatrix} \text{ is the control input, where}$$

$u_{ij}^l(t)$ describes whether the mobile robot i is assigned to the machine S_j for loading/unloading of product type l at time t . There must be only one mobile robot assigned to a machine at a time t . Although $U(t)$ is presented as a 2D matrix, it's crucial to note that it actually represents a 3D tensor with i, j , and l axis representing index for robots, machines and product types. Each element of this tensor is a binary variable $u_{ijl}(t)$, which is equal to 1 if a mobile robot i is assigned to machine j to load product type l at time t , otherwise is 0.

The mobile robot's function comprises two major steps: unloading and loading the machine, as well as transporting the material. A mobile robot i can only be assigned to machine j if it is neither starved nor blocked and is not already assigned a mobile robot, i.e., the necessary condition for robot i assignment is $(\sum_{j=2}^{M-2} b_j > 0) \& (B_{j+1} > \sum_{j=2}^{M-2} b_{j+1}) \& (u_{ij}(t-1) = 0)$.

A recursive algorithm has been developed, based on the principle of flow conservation, to evaluate the buffer level $b(t)$ at each time step t in the system. The specific dynamic function $F(*)$ that describes this algorithm can be found in the referenced work [11]. In order to effectively control the assignments of mobile robots in real-time, it is essential to have a metric that assesses the system's dynamic performance. Prior research [2] has shown that not all downtimes result in permanent production loss (PPL) for the system. PPL only occurs when the downtime causes the slowest critical machine, denoted as S_{sc} , to stop. To quantify this characteristic, the concept of an opportunity window (OW) has been introduced. Each machine, denoted as S_i , has an associated OW, denoted as OW_i , which represents the maximum allowable downtime for S_i without causing PPL for the system. It has been established that OW_i corresponds to the duration required for all buffers between S_i and the slowest critical machine S_{sc} to either become empty or full. According to [11], the system experiences PPL only if any disruption event exceeds the OW of the slowest critical machine S_{sc} . Therefore, the occurrence of PPL during a given time interval $[0, T]$ depends on the status of S_{sc} and can be defined as $PPL(T) = \frac{d_{sc}(T)}{T_{sc} + T_{load}^{sc} + T_{unload}^{sc} + T_{travel}^{sc}}$, where $d_{sc}(T)$ represents the duration of downtime for S_{sc} . Detailed derivation and rigorous proof of OW and PPL can be found in [11]. OW_i and PPL_i serve as significant dynamic properties of the system and are utilized in designing the control policy presented in this paper.

5. Mobile robot assignment control problem

Controlling production systems that involve multiple product types and large number of machines and mobile robots poses a greater challenge due to the increased complexity. In this research, the control actions involve assigning a mobile robot, R_i to a machine S_j to load product type l as in Eq. (2). Due to the lack of a closed form representation for the system, the application of traditional control methods becomes challenging. Furthermore, the complexity of the problem, which involves a significant number of states and unknown transition probabilities, poses difficulties in utilizing operations research techniques such as dynamic programming and centralized control scheme. As a result, the problem is formulated as a Decentralized Partially Observable Markov Decision Process (Dec-POMDP) and tackled using Multi-Agent Reinforcement Learning (MARL). The objective of the control problem is to determine an optimal policy in stochastic scenarios, mapping states to actions, in order to maximize the reward.

5.1. Dec-POMDP framework for MARL

In the MARL framework, at state s_t , each agent selects their own action a_t^1, \dots, a_t^n and receives the corresponding reward $r_t^i(s_t, a_t^1, \dots, a_t^n)$. These actions results in a joint action $a_t = (a_t^1, \dots, a_t^n)$, and the state transitions to the next state s_{t+1} with the transition probability $p(s_{t+1}|s_t, a_t)$ such that it satisfies $\sum_{s_{t+1} \in S} p(s_{t+1}|s_t, a_t) = 1$. Each agent strives to

maximize their own discounted accumulated reward by selecting actions a_t^i under the policy π_t^i , which represents the decision-making rule for the distribution of their actions. The policy for each agent i is represented by $\pi^i = (\pi_0^i, \pi_1^i, \dots, \pi_t^i, \dots)$. Given an initial state s , and discount factor $\gamma \in [0, 1)$, the accumulated reward can be expressed as:

$$v^i(s, \pi^1, \dots, \pi^n) = \sum_{t=0}^{\infty} \gamma^t E(r_t^i | \pi^1, \dots, \pi^n, s_0 = s) \quad (2)$$

Since the transition probabilities are unknown in this system, a model-free MARL algorithm is employed. This approach eliminates the need for assumed transition probabilities, and focuses solely on defining observations, actions, and rewards.

5.1.1. Observations

Observations refer to information that each agent has about the current state of the system. These "raw observations" only reflect local information from individual agents and may result in sub-optimal performance in coordinated control problems. The observation of each agent i is represented as,

$$o_t^i = \{ \vec{g}_t, \vec{P}_t \} \quad (3)$$

where $\vec{g}_t = [g_1(t), g_2(t), \dots, g_M(t)]$ represents the status of machines e.g., $\vec{g}_t = [0, 1, 0, 0]$ represent that only second machine is assigned a mobile robot. \vec{P}_t is a vector representing the product types each machine is currently processing e.g., in case of 3-product types and 4-machines, $\vec{P}_t = [[0, 1, 0], [1, 0, 0], [0, 0, 0], [0, 0, 1]]$ represents that the first machine is loaded with product type 1, second with product type 0, third machine is empty, and the fourth one is loaded with product type 2.

In a Dec-POMDP, the state usually comprises the observations of all agents i.e.,

$$s_t = \{ o_t^1, o_t^2, \dots, o_t^n \} \quad (4)$$

However, in this case, besides the observations of all agents, additional information is provided. The state information s_t at time t can be presented as,

$$s_t = \{ \vec{b}_t, \vec{\theta}_t, \vec{g}_t, \vec{P}_t \} \quad (5)$$

where $\vec{b}_t, \vec{\theta}_t, \vec{g}_t$, and \vec{P}_t are vectors representing buffers' levels, machines' status i.e., on/off, machines' assignment status i.e., assigned mobile robot or not, and type of product loaded onto individual machines respectively.

5.1.2. Action

The action an agent can take is the assignment action which defines the product type l , to be assigned to a machine S_j . It is defined as

$$A = \{ a_{ij}^i \} \quad (6)$$

where a_{ij}^i represents the action of agent i to load product type l to machine S_j and $a_{ij}^i \in [0, 1]$. A new action is triggered when a machine is empty or finishes processing a product and a mobile robot is available. However, the action is also subject to Eq. (2).

5.1.3. Reward

A global reward describes how good the combined action of all agents is. Thus, a proper reward setting is required to ensure that the resulting policy does what is expected. A good reward setting requires domain knowledge of the system and is designed in a way that increasing the overall discounted sum of this reward pushes the system to work as per expectations. A variety of possible reward settings were explored for the problem and the following was chosen to be the best.

$$r_t = -PPL(t) - CT - C_i \quad (7)$$

where the first term punishes the system for accruing PPL i.e.,

$$PPL(T) = \frac{d_{sc}(T)}{T_{sc} + T_{load}^{sc} + T_{unload}^{sc} + T_{travel}^{j,sc}} \quad (8)$$

the second term keeps the balance between cost of delay, and the cost of surfeit i.e.,

$$CT = CoD(t) - CoS(t) \quad (9)$$

$$CoD(t) = \sum_{l=1}^K \omega_d^l \max\{0, D_l(t) - Y_l(t)\} \quad (10)$$

$$CoS(t) = \sum_{l=1}^K \omega_s^l \max\{0, Y_l(t) - D_l(t)\} \quad (11)$$

where ω_d^l and ω_s^l are the delay cost rate and the surfeit cost rate for product type l , respectively. The agent is punished when the production is less than the corresponding demand at time t . Similarly, the agent is rewarded when the demand is satisfied.

The last term in Eq. (7), C_i , denotes the collaboration cost calculated in the Nash game (Section 6.2). It helps the agents to improve collaboration.

6. Proposed Nash-MADDPG method

Our proposed Nash-MADDPG method combines game theory and MADDPG algorithm. The framework of the proposed method is given in Fig. 2, which comprises three major parts: MADDPG algorithm (discussed in Section 6.3), Environment (discussed in Section 3), and Nash game (discussed in 6.1 and 6.2). The mobile robots R_i , where $i = 1, 2, \dots, n$ are responsible for the primary control and have to decide product type l and machine S_j , where $j = 1, 2, \dots, M$. Based on the initial state s_t of environment, a Nash game is initialized with random strategies π_i , where $i = 1, 2, \dots, n$. A strategy π_t^i is the probability of actions over the action space for mobile robot i at time t . Utilizing the strategies $\pi = (\pi_0, \pi_1, \dots, \pi_n)$ of all players (mobile robots), a collaboration cost C_i where $i = 1, 2, \dots, n$ is calculated for each player (mobile robot) based on Eq. (13). This cost helps the agents to avoid prisoners' dilemma and focus on the global objective rather than getting stuck in local optima. For a new state s_{t+1} , the strategy π_i is estimated by a deep q-network (DQN). However, the environment is not directly updated with current strategies π_i of agent i .

In a parallel manner, MADDPG algorithm is developed by initializing an actor network $\mu_i(o_i(t) | \Psi_i^{\mu})$ and critic network $Q_i(s_t, a | \Psi_i^Q)$ for each agent i . MADDPG takes the observations $(o_1(t), o_2(t), \dots, o_n(t))$ and actions $(a_1(t), a_2(t), \dots, a_n(t))$ of all agents and generates action probabilities μ_i .

Now, action probability $\mu_i(t)$ and strategy π_t^i is averaged and joint action $a(t) = (a_1(t), a_2(t), \dots, a_n(t))$ is defined based on which the environment is updated to a new state s_{t+1} , and a payoff based on Eq. (7) is assigned to players (agents). The reward function remains the same in the game and MADDPG. However, each agent i receives a different reward r_i based on its corresponding collaboration cost C_i .

6.1. Game modelling of mobile robots

The scheduling of multiproduct FMS is modelled as a Nash game where each mobile robot acts as a player in the game. They make decisions of the processing strategies (i.e., selection of the product type and machine) to achieve their goal individually. The model is expressed as a tuple: $G = (n, \pi, r)$, where n denotes number of players in the game, π denotes the strategy profile of all players, $\pi = (\pi_0, \pi_1, \dots, \pi_n)$, where π_i denotes the strategy of player i , and r denotes payoff, $r = (r_1, \dots, r_n)$, where r_i denotes the payoff of player i .

6.1.1. Players

The scheduling of multiproduct FMS depends on the number of mobile robots handling the product types among machines and buffers.

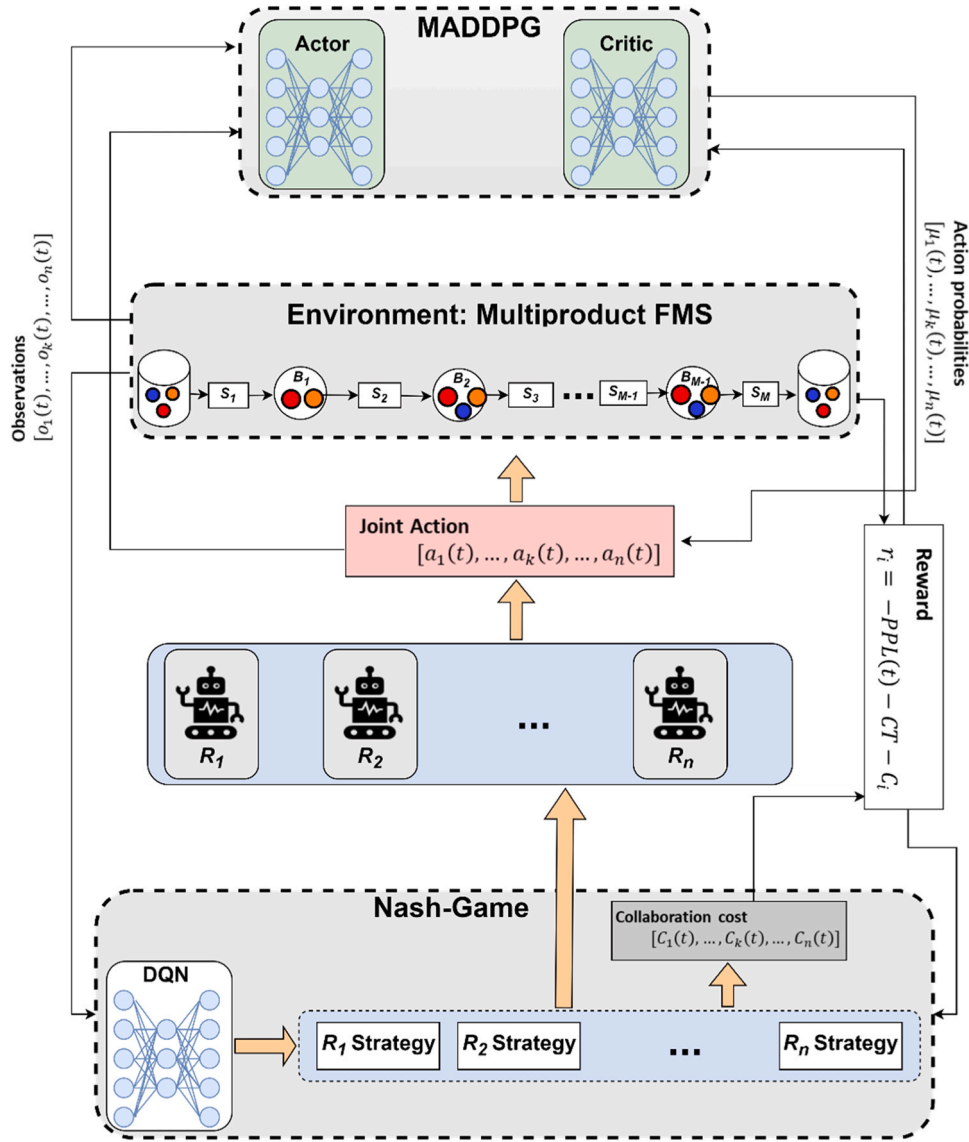


Fig. 2. Framework of the proposed Nash-MADDPG algorithm.

Thus, each mobile robot acts as a player. There are total of n players (mobile robots) in our system.

6.1.2. Strategies

Each product has a unique operations' sequence and different market demand. Thus, each product should be assigned in such a way that the scheduling is optimized. The strategy of player i is represented by π_i . In a multiproduct system with 3 product types and 3 machines, the number of possible strategies for each mobile robot is 9 (only if each product type needs to be processed by every machine). Each player needs to select a strategy from all these possible strategies.

6.1.3. Payoffs

In this scheduling game, each mobile robot tries to assign the products in such a way to maximize its profit. On one hand, the mobile robot doesn't want the slowest critical machine S_{sc} to get starved or blocked. On the other hand, it wants the market demand for each product type to be satisfied. Similarly, each mobile robot also tries to cooperate with the other mobile robots, as its own profit depends on the strategies of other mobile robots. Therefore, the payoff for i^{th} mobile robot r_i can be defined in the same way as in Eq. (7).

6.2. Nash equilibrium

According to [24], in a Markov game, a Nash equilibrium point is defined as a tuple of n policies $(\pi_1^*, \dots, \pi_n^*)$ such that for all $s \in S$ and $i = 1, \dots, n$

$$v^i(s, \pi_1^*, \dots, \pi_n^*) \geq v^i(s, \pi_1^*, \dots, \pi_i^{i-1}, \pi_i, \pi_i^{i+1}, \dots, \pi_n^*) \forall \pi_i \in \Pi^i \quad (12)$$

where Π^i denotes agent i i.e., mobile robot i 's available policies.

As mentioned earlier, in a Nash game, the agents i.e., mobile robots need to collaborate with each other to define an optimum action sequence. To decide if the collaboration is successful or not, a collaboration cost C_i is defined for each mobile robot i . This cost is used as a part of the agent payoff (reward), given in Eq. (7).

To get into details, let $\pi_i(t)$ and $\pi_{-i}(t)$ be the strategies of mobile robot i and other mobile robots at time t respectively, which is defined as the probability of taking an action a_i in the current state s_t from the available actions of mobile robot i at time t . Assuming n mobile robots, the interaction among the mobile robots can be modeled as the following game, which follows the approach of [5,39].

Definition 1. Given n mobile robots $(R_1, \dots, R_k, \dots, R_n)$ with their strategies $(\pi_1^1, \dots, \pi_k^k, \dots, \pi_n^n)$ respectively, the interaction among $R_1, \dots,$

R_k, \dots, R_n is modelled as a game G in which each mobile robot decides its strategy to minimize the collaboration cost C_i . The collaboration cost functions for the mobile robot R_i is defined as.

$$C_i(\pi_t^i, \pi_t^{i-}) = \vartheta_i [\pi_t^i - \pi_t^{i-}]^2 + \prod_{i=1}^n \varphi_i [\pi_t^i - \pi_t^{i-}]^2 \quad (13)$$

where $\varphi_i \in [0, 1]$ and $\vartheta_i = 1 - \varphi_i$, ($i = 1, \dots, k, \dots, n$) are constants. (π_t^i, π_t^{i-}) is called strategy pair of the game.

Definition 2. : For mobile robots ($R_1, \dots, R_k, \dots, R_n$), a strategy pair $(\pi_t^{i*}, \pi_t^{i-*})$ is called the Nash equilibrium of the game if it satisfies the following [5].

$$C_i(\pi_t^{i*}, \pi_t^{i-*}) = \min_{\pi_t^i} C_i(\pi_t^{i*}, \pi_t^{i-*}) \quad (14)$$

This interaction among ($R_1, \dots, R_k, \dots, R_n$) help the mobile robots to decide the best possible action. The overall objective is to reduce the collaborating cost comprising the disagreement cost ($\varphi_i [\pi_t^i - \pi_t^{i-}]^2$) and cost for state change ($\vartheta_i [\pi_t^i - \pi_t^{i-}]^2$) for mobile robot R_i . Thus, each mobile robot not only cares about its own state but also that of its teammate.

Theorem 1.: According to Definitions 1 and 2, a game G with n mobile robots R_1, R_2, \dots, R_n has a unique equilibrium given as.

$$[(1-\lambda)\pi_{t-1}^1 + \lambda\pi_{t-1}^2 + \dots + \lambda\pi_{t-1}^n], [\Omega\pi_{t-1}^1 + (1-\Omega)\pi_{t-1}^2 + \dots + \Omega\pi_{t-1}^n], \dots, [\varnothing\pi_{t-1}^1 + \varnothing\pi_{t-1}^2 + \dots + (1-\varnothing)\pi_{t-1}^n] \quad (15)$$

$$\text{where } \lambda = \frac{\vartheta_2\varphi_1 \dots \varphi_n}{\vartheta_1 + (\vartheta_2\varphi_1 \dots \varphi_n) + \dots + (\vartheta_n\varphi_{n-1} \dots \varphi_1)}, \Omega = \frac{\vartheta_1\varphi_2 \dots \varphi_n}{\vartheta_2 + (\vartheta_1\varphi_2 \dots \varphi_n) + \dots + (\vartheta_{n-1}\varphi_{n-1} \dots \varphi_1)} \text{ and } \varnothing = \frac{\vartheta_n\varphi_{n-1} \dots \varphi_1}{\vartheta_n + (\vartheta_{n-4}\varphi_{n-3} \dots \varphi_1) + \dots + (\vartheta_{n-2}\varphi_{n-1} \dots \varphi_1)}$$

Proof. : For a fixed π_t^{2*} , it follows from (14) that $C_k(\pi_t^{i-}, \pi_t^{i*})$ is a quadratic function of π_t^{i-} . Therefore $C_k(\pi_t^{i-}, \pi_t^{i*})$ has only one global minimum π_t^{i-*} , which satisfies.

$$\begin{cases} \left. \frac{\partial C_k(\pi_t^{i-}, \pi_t^{i*})}{\partial \pi_t^{i-}} \right|_{(\pi_t^{i-*}, \pi_t^{i*})} = 0 \\ \left. \frac{\partial^2 C_k(\pi_t^{i-}, \pi_t^{i*})}{\partial (\pi_t^{i-})^2} \right|_{(\pi_t^{i-*}, \pi_t^{i*})} > 0 \end{cases} \quad (16)$$

According to (14) and (15), (π_t^{1*}, π_t^{2*}) is the Nash equilibrium if and only if (15) holds. By (17), we can conclude that game G has unique Nash equilibrium. Further, (π_t^{1*}, π_t^{2*}) is the Nash equilibrium if and only if

$$\left. \frac{\partial C_k(\pi_t^{i-}, \pi_t^{i*})}{\partial \pi_t^{i-}} \right|_{(\pi_t^{i-*}, \pi_t^{i*})} = 0 \quad (17)$$

By solving (18), we get

$$\begin{cases} \pi_t^{1*} - \varphi_1\pi_t^{2*} - \dots - \varphi_1\pi_t^{n*} = \vartheta_1\pi_{t-1}^1, \\ -\varphi_2\pi_t^{1*} + \pi_t^{2*} - \dots - \varphi_2\pi_t^{n*} = \vartheta_2\pi_{t-1}^2, \\ \vdots \\ \pi_t^{n*} - \varphi_n\pi_t^{1*} - \dots - \varphi_n\pi_t^{n-1*} = \vartheta_n\pi_{t-1}^n \end{cases} \quad (18)$$

Since $(1 - \prod_{i=1}^n \varphi_i) > 0$, (D) can be simplified as

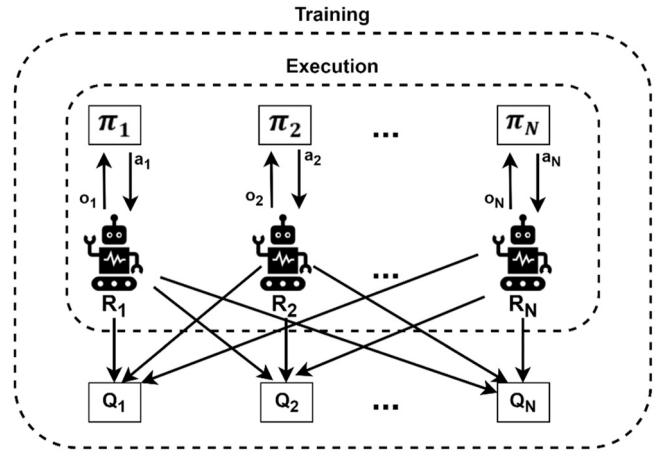


Fig. 3. MADDPG framework [15].

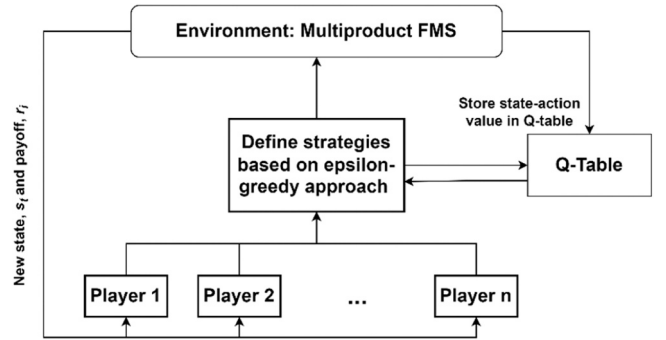


Fig. 4. General framework of Nash Q-learning.

$$\pi_t^{i*} = \frac{\vartheta_i}{1 - \prod_{i=1}^n \varphi_i} \pi_{t-1}^1 + \frac{\vartheta_{i+1}\varphi_1}{1 - \prod_{i=1}^n \varphi_i} \pi_{t-1}^2 + \dots + \frac{\vartheta_n\varphi_{n-1} \dots \varphi_i}{1 - \prod_{i=1}^n \varphi_i} \pi_{t-1}^n \quad (19)$$

Since $\vartheta + \varphi + \dots + \varphi = 1$, we can write

$$1 - \prod_{i=1}^n \varphi_i = \vartheta_1 + \vartheta_2\varphi_1 + \dots + (\vartheta_n\varphi_{n-1} \dots \varphi_i) \quad (20)$$

Hence game G has unique Nash equilibrium (16).

6.3. MADDPG

MADDPG is a multi-agent approach designed to address the challenge of coordinating multiple agents. It builds upon the actor-critic algorithm and extends the DDPG method [40]. In DDPG, the critics receive only the self-observation and action data as inputs. However, in MADDPG, in addition to each agent's own observation and action information, the critics also have access to the observation and action information of other agents. In MADDPG algorithm, each agent uses its own independent Actor-net to output actions A_i by observing the state o_i . Each agent also corresponds to one Critic-net, but this Critic-net is trained by the data produced by all the actors at the same time. That is, what we evaluate in the multi-agent system is the joint strategy $\pi = (\pi_1, \pi_2, \dots, \pi_n)$. Fig. 3 depicts a framework of MADDPG algorithm.

The stepwise implementation of Nash-MADDPG is given in Algorithm 1.

Algorithm 1. Nash-MADDPG Algorithm.

```

Initialize a Nash game of  $k$  players each with a random strategy  $\pi_k$ 
Initialize DQN with a replay buffer  $D$ , action-value function  $Q$ , and target action-value function  $\hat{Q}$ 
for each agent  $k \in \mathcal{K}$  do
    Initialize replay buffer  $\mathcal{D}_k$ 
    Initialize actor network  $\mu_k(o_k|\Psi_k^\mu)$  and critic network  $Q_k(s_k, d, a|\Psi_k^Q)$  with weights  $\Psi_k^\mu$  and  $\Psi_k^Q$ , respectively
    Initialize target networks,  $\mu'_k$  and  $Q'_k$  with weights  $\Psi_k^\mu$  and  $\Psi_k^Q$ , respectively
for each episode  $e = 1, 2, \dots$  do
    for each agent  $k \in \mathcal{K}$  do
        Initialize random process  $\mathcal{N}_k$  for exploration
        Generate initial local observation  $o_k$  from the environment
    for each step  $t = 1, 2, \dots$  do
        Estimate  $Q$  and get strategy  $\pi_k^t$  from the Nash game
        Calculate collaboration cost,  $C_k$ 
        for each agent  $k \in \mathcal{K}$  do
            Observe the action values  $\mu_k(t)$  from MADDPG
            Select action  $a_k(t) = [\mu_k(t) + \pi_k^t]/2$ 
        Execute joint action  $\mathbf{a}(t) = (a_1(t), a_2(t), \dots, a_K(t))$ 
        Collect payoff  $r_k(t)$ , and observe  $s_k(t+1)$ 
        Store the transition  $(s_k(t), \mathbf{a}(t), r_k(t), s_k(t+1))$  into  $D$ 
        Estimate new strategies  $\pi_k$  from the DQN
        for each agent  $k \in \mathcal{K}$  do
            Collect reward  $r_k(t)$ , and observe  $s_k(t+1)$ 
            Store the transition  $(s_k(t), \mathbf{a}(t), r_k(t), s_k(t+1))$  into  $\mathcal{D}_k$ 
            Sample random minibatch of  $B$  transitions  $(s_k^i, a^i, r_k^i, s_k^{i+1})$  from  $\mathcal{D}_k$ 
            Update the critic network by minimizing the loss
            Update the actor policy using the sampled policy gradient
            Update the target networks

```

6.4. Nash Q-learning

A general framework of conventional Nash Q-learning is given in Fig. 4, where n players play around to find the Nash equilibrium. To find the equilibrium policies in a multiproduct FMS, the work of [24] is followed. According to [24], a Nash Q-value represents the expected reward that all agents receive when they follow specific Nash equilibrium policies. The Nash Q-function Q_{t*}^i for agent i at state s_t can be written as:

$$Q_{t*}^i(s_t, \mathbf{a}_t) = r_t^i(s_t, \mathbf{a}_t) + \gamma \sum_{s_{t+1} \in S} p(s_{t+1}|s_t, \mathbf{a}_t) v_t^i(s_t, \pi_*^1, \dots, \pi_*^n) \quad (21)$$

where $(\pi_*^1, \dots, \pi_*^n)$ is the joint Nash equilibrium policy and $r_t^i(s_t, \mathbf{a}_t)$ is the reward agent i receives in state s_t under joint action \mathbf{a}_t .

In contrast to the single-agent Q-learning method, the agents in multiagent Nash-Q-learning update their Q-values based on future Nash equilibrium payoffs. This means that the agent must observe not only its own reward but also the reward of all other agents. At the beginning of the learning process, agent i starts with an arbitrary guess for its Q-value, such as $Q_0^i(s, \mathbf{a}) = 0$ for all $s \in S$. Then, at each time t , the agent i updates its Q-value based on the received reward and the Nash Q-value for the next state:

$$Q_{t+1}^i(s_{t+1}, \mathbf{a}_{t+1}) = (1 - \alpha_t) Q_t^i(s_t, \mathbf{a}_t) + \alpha_t [r_t^i + \gamma \text{Nash} Q_t^i(s_{t+1}, \mathbf{a}_{t+1})] \quad (22)$$

where $\text{Nash} Q_t^i(s_{t+1}, \mathbf{a}_{t+1}) = \pi_*^1(s_{t+1}), \dots, \pi_*^n(s_{t+1}) \cdot Q_t^i(s_{t+1}, \mathbf{a}_{t+1})$ represents

the payoff of agent i in state s_{t+1} for the selected Nash equilibrium and α_t is the learning rate.

Based on Eq. (12), the payoff of each agent i is maximized by applying the equilibrium policy $\pi_*^1(s_{t+1}), \dots, \pi_*^n(s_{t+1})$. Therefore, each agent i must know all other agents' $Q_t^j(s_{t+1}, \mathbf{a}_{t+1})$ to achieve the equilibrium policy. However, at time t , other agents' Q-values are not given. Therefore, agent i has to assume other agents' Q-values. Same as updating its own Q-value, agent i guesses $Q_0^j(s, \mathbf{a}) = 0$ for all other agents at the beginning. As the game moves on, the actions and rewards of other agents will be observed and used to update its belief of the Q-values of all other agents as follows:

$$Q_{t+1}^j(s_{t+1}, \mathbf{a}_{t+1}) = (1 - \alpha_t) Q_t^j(s_t, \mathbf{a}_t) + \alpha_t [r_t^j + \gamma \text{Nash} Q_t^j(s_{t+1}, \mathbf{a}_{t+1})] \quad (23)$$

7. Case study

Numerical studies are performed to verify the effectiveness of the proposed Nash-MADDPG based multiagent control of mobile robots. In this paper, to demonstrate the method, a multiproduct FMS with different complexity levels i.e., different number of machines, buffers, mobile robots and product types are studied. For comparison, a vanilla MADDPG, conventional Nash-Q-learning, Q-learning-SARL, and FCFS approaches are used as alternative ways to solve the same mobile robot scheduling problem. The simple MADDPG and Nash-Q-learning methods are discussed in Sections 6.3 and 6.4 respectively. For Q-learning-SARL method, all the assignments of mobile robots are

controlled by one agent e.g., at each state s_t , the action a_t includes all the combinations of mobile robots' assignments and the Q-values are stored in a single Q-table. The actions are optimized by backing up estimates of the optimally evaluated state-action values, which are always saved by updating the Q-table. Two performance metrics are considered: (1) The training time required for the algorithms to converge to obtain a scheduling policy for the mobile robots (2) The system production throughput using the trained policy for mobile robots' assignments. From the case study, two significant results intend to be concluded: (1) The proposed Nash-MADDPG scheme is effective in optimizing the mobile robots' scheduling; (2) The proposed Nash-MADDPG scheme outperforms the simple MADDPG, Nash-Q-learning, Q-learning-SARL and FCFS method in terms of the performance metrics when the system complexity increases.

7.1. Experiment parameter setting

The mobile robots are trained based on the interactive Nash game played among the mobile robots. The game runs for each episode and after completion of each episode, the game restarts. In the new episode, each mobile robot is randomly assigned. The mobile robot keeps the Q-values learned from previous episodes. The training stops after 20 episodes. Each episode lasts for 3000-time steps (One week time with 5 days a week and 10 h a day) and each time step represents 1 min. The Q-value is updated based on a discount factor $\gamma = 0.95$, learning rate $\alpha = 0.1$ and ϵ set at 0.2. The neural network architecture consists of two fully connected hidden layers, each containing 64 hidden units. The reward function is formulated according to Eq. (7). The discount factor γ remains the same as 0.95. The actor learning rate α_a is assigned a value of 0.0005, while the critic learning rate α_c is set to 0.001.

The information about machines and product types is given in Tables 2 and 3. The loading and unloading time on each machine are constant and is equal to 2 mins. All the buffers have a fixed capacity of $B_i = 15$ units, and the initial buffer levels are selected randomly between 0 and B_i . The data is motivated by real-world data obtained from the automobile industry. However, due to confidentiality issues, synthetic data based on this actual data has been generated to simulate scenarios for analysis, demonstration, and training purposes. After 20 episodes, the results are compared based on market demands given in Table 3.

7.2. Efficacy of the proposed method

The proposed Nash-MADDPG is compared with simple MADDPG, Nash-Q-learning and Q-learning-SARL based on corresponding training time and convergence in a multiproduct FMS with different levels of complexity. During training, machines are randomly selected from the provided list in Table 2. Similarly, the product flow path is determined randomly; specifically, apart from the initial and final machines, a random value between 0 and 1 is generated for intermediate machines to decide if they will process the product type or not. In Fig. 5, the algorithms are trained in a system comprising 4 machines, 3 buffers, 3 product types and 2 mobile robots. As can be seen, the proposed method

Table 2
Parameters of the machines.

Parameter	Processing time, T_i (min)	MTBF (min)	MTTR (min)
S_1	4	100	20
S_2	5	120	30
S_3	4	125	25
S_4	5	150	25
S_5	3	110	30
S_6	2	130	35
S_7	4	120	25
S_8	5	140	20
S_9	2	135	30
S_{10}	3	110	25

Table 3
Weekly demand of product types.

Parameters	Product Type 1	Product Type 2	Product Type 3
Weekly Demand	50	70	150

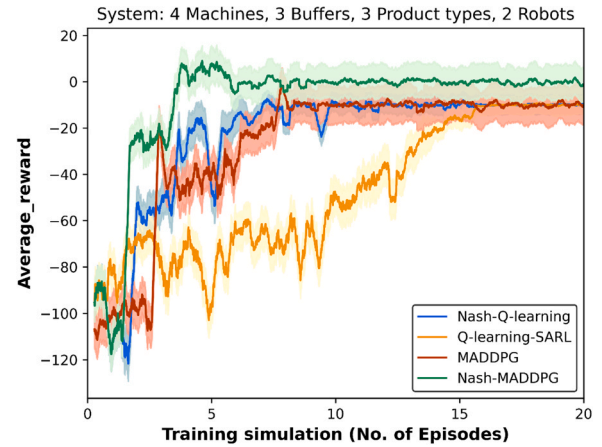


Fig. 5. Training data, Reward versus Time.

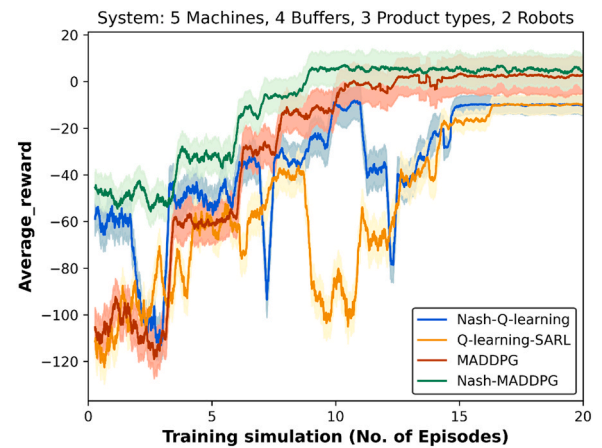


Fig. 6. Training data, Reward versus Time.

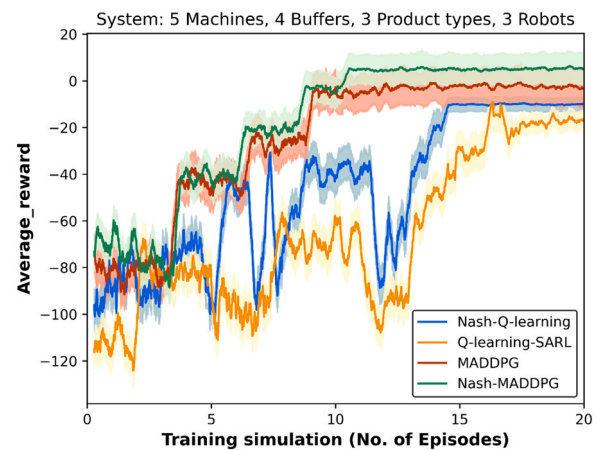


Fig. 7. Training data, Reward versus Time.

converges quickly as compared to other algorithms, which takes relatively more time to get stable. The reward achieved is also higher as compared to the other methods. MADDPG and Nash-Q-learning performed almost similarly. Whereas the Q-learning-SARL took a long time to converge.

In Fig. 6, the system complexity is increased by including an additional machine and a buffer. It can be seen that more time is needed for all the algorithms to converge. Here, in comparison to the Nash-Q-learning and Q-learning-SARL, the proposed method and the MADDPG performed better. They both not only converge faster, but also achieve higher rewards. Overall, the Nash-MADDPG stabilizes first followed by MADDPG, whereas the other two methods take significant time. The Nash-Q-learning converges early as compared to Q-learning-SARL. However, both converge at the end of training time.

In Fig. 7, the complexity is further increased by adding a third mobile robot to the existing system comprising 5 machines, 4 buffers and 3 product types. Again, the proposed Nash-MADDPG and MADDPG performed well and stabilized quickly, but interestingly, the MADDPG catches up quickly and converges faster than the proposed method. However, the reward it achieves is significantly lower than that of the Nash-MADDPG. Nash-Q-learning also converges, but it takes a long time to stabilize, whereas the Q-learning-SARL is unable to stabilize within the given timeframe.

Next, we compare the performance of these trained algorithms during online execution. The average production output of each product type based on market demand is compared among Nash-MADDPG, MADDPG, Nash-Q-learning, Q-learning-SARL, and FCFS based control methods. The first four methods schedule the mobile robots based on the learned policies, while FCFS follow a fixed rule i.e., each machine to be loaded/unloaded is added to a queue and the mobile robots are assigned in the order machines were added. Similarly, product types are also selected in chronological order.

For the given demand in Table 3, Fig. 8 compares the production under the Nash-MADDPG (star pattern), MADDPG (dotted pattern), Nash-Q-learning (lined bars), Q-learning-SARL (crossed bars), and FCFS (bars with circular pattern) for a system comprising 4 machines, 3 buffers, 3 product types, and 2 mobile robots. As can be seen, the Nash-MADDPG, MADDPG, and Nash-Q-learning satisfies the corresponding demand, whereas the Q-learning-SARL performed well, but it is unable to produce the desired quantity of product type-3. Similarly, FCFS could not catch up on the required demand for all product types. However, it performed well for product type 1 and 2 by satisfying the corresponding demand of 50 and 70 units respectively. For product type 3, it produced almost 81 units only. Overall, the proposed Nash-MADDPG performed well and easily achieved the target of required production for individual

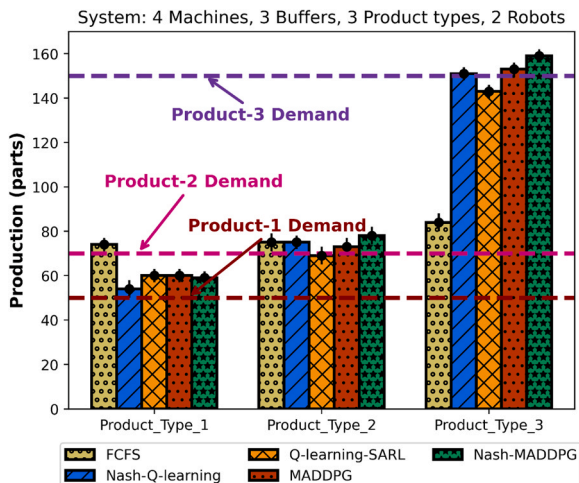


Fig. 8. A comparison of production under different control policies.

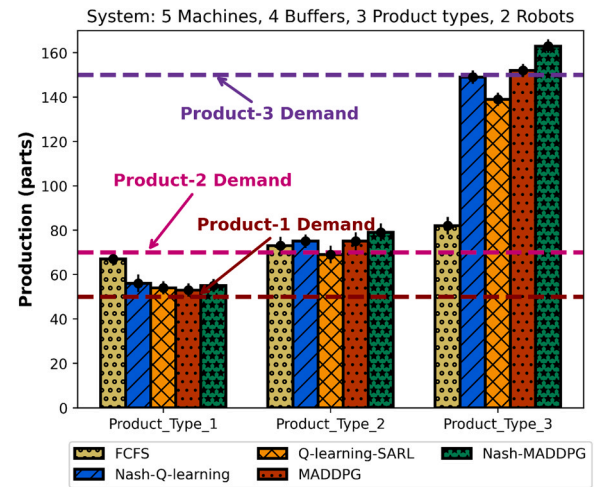


Fig. 9. A comparison of production under different control policies.

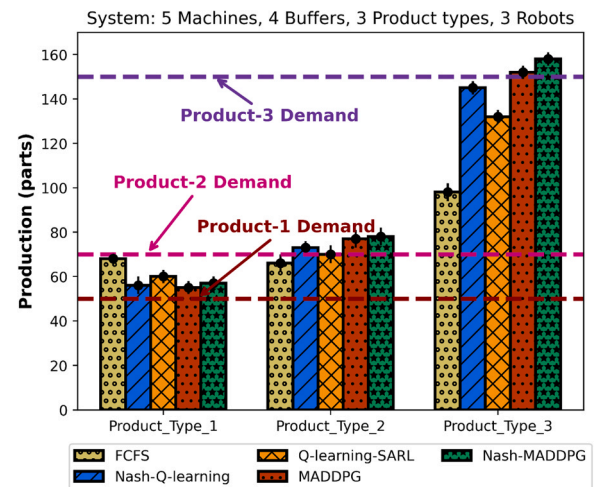


Fig. 10. A comparison of production under different control policies.

product types. It should be noted that our primary focus lies in meeting demand requirements. While our proposed method enables increased production capacity, our attention remains centered on fulfilling the specified demand. As a result, each algorithm ceases operation upon meeting market demand. Given that our proposed method, and occasionally other methodologies like MADDPG and Nash-Q-learning, efficiently fulfill demand ahead of schedule, further execution of these methods is unnecessary.

Fig. 9 compares production under the same control policies based on a different system, comprising 5 machines, 4 buffers, 3 product types and 2 mobile robots. It can be observed that the Nash-MADDPG clearly outperforms the other methods. Both the Nash-MADDPG and simple MADDPG achieve the required target. The Nash-Q-learning barely satisfies the production demand of product type 3. Similarly, Q-learning-SARL and FCFS remain unsuccessful to satisfy the production requirement of product type 3.

Similarly, to go a step further, the complexity of the system is increased by adding a 3rd mobile robot to the existing system comprising 5 machines, 4 buffers, and 3 product types. The production under different policies in such a system is compared in Fig. 10. As can be seen, only Nash-MADDPG and simple MADDPG based control methods are able to satisfy the corresponding market demands for all product types. Although Nash-Q-learning and Q-learning-SARL satisfies the production requirement of product type 1 and 2, they remained

unable to satisfy the demand of product type 3. FCFS on the other hand only succeeded in satisfying the demand for product type 1.

Since FCFS follows a fixed rule, it does not care about demand variations and schedules the mobile robot and assigns the product types in a fixed order, resulting in the corresponding market demand dissatisfaction. Although the Nash-Q-learning and Q-learning-SARL worked well for the first case, it becomes difficult for both methods to manage a large state space and thus could not manage the mobile robots schedule in a complex environment.

Our proposed Nash-MADDPG method stands out distinctively among other methodologies by merging the robust frameworks of game theory and the MADDPG algorithm. Game theory and MADDPG are individually potent in addressing complex problems, and their fusion enhances problem-solving efficacy. Departing from the conventional Nash-Q-learning approach, which encounters limitations due to tabular data storage, we employed DQN to estimate the value function, leveraging deep layers for robust estimation.

In Nash game theory, agents often encounter problems such as the prisoner's dilemma, tragedy of commons, and coordination issues. Addressing these complexities requires mechanisms fostering cooperation, trust, communication, and aligning individual interests with collective goals. Our method addresses this by introducing a collaboration cost based on individual players' strategies for cooperation. Players adapt their strategies to minimize collaboration costs, thereby enhancing global impact. While MADDPG is adept in handling multi-agent and complex environments like a multiproduct FMS, it can lead to computational complexities. However, integrating Nash game dynamics, reliant on the collaboration cost, aids the MADDPG algorithm in swift learning and faster convergence. This integration curtails computational complexity and accelerates learning within intricate environments.

8. Conclusion

In this paper, a multiproduct FMS handled by intelligent mobile robots is studied. The mobile robots' scheduling problem is formulated as a MARL problem and a Nash-MADDPG algorithm is proposed to solve it. The performance of the system is evaluated based on the market demand satisfaction and permanent production loss (PPL). To achieve better performance, besides the MADDPG based control, the interactions of the mobile robots are modelled as a game and a unique Nash equilibrium is verified. The environment is updated based on a joint strategy from the game and MADDPG algorithm which improves the learning process. A case study is used to demonstrate the effectiveness of the proposed Nash-MADDPG based control policy. Compared with simple MADDPG, Nash-Q-learning, Q-learning-SARL and FCFS, the proposed method provides better results. The Nash-Q-learning and Q-learning-SARL are also effective, but they barely satisfy the market demand and do not perform well when the system complexity increases (i.e., increasing number of product types, machine, or mobile robots). On the other hand, FCFS follows a fixed rule and does not get updated based on corresponding demand, thus remains unable to satisfy the different demands of product types.

The performance of the learned policies i.e., Nash-MADDPG, Nash-Q-learning, Q-learning-SARL, and FCFS is tested under three different levels of system complexity. The agents are trained in different system configurations, where the complexity is increased by adding a new machine or buffer or a mobile robot. Overall, the proposed method and the simple MADDPG were able to satisfy the market demand in all system configurations. The market demand in this work is assumed deterministic and remains constant for each period i.e., per week. However, the demand may vary and can affect the production schedule. Therefore, the future work will address a multiproduct FMS considering variable and stochastic market demands.

Funding

This work was funded by the U.S. National Science Foundation (NSF) Grant. CMMI 1853454.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

- [1] Waseem M, Li C, Chang Q. Dynamic modeling and analysis of multi-product flexible production line. *Int J Comput Integr Manuf* 2023;1–15.
- [2] Waseem M, Chang Q. Adaptive mobile robot scheduling in multiproduct flexible manufacturing systems using reinforcement learning. *J Manuf Sci Eng* 2023;1–37. <https://doi.org/10.1115/1.4062941>.
- [3] Liu Y, Ping Y, Zhang L, Wang L, Xu X. Scheduling of decentralized mobile robot services in cloud manufacturing with deep reinforcement learning. *Mob Robot Comput-Integr Manuf* 2023;80:102454.
- [4] Didden JB, Dang Q-V, Adan IJ. Decentralized learning multi-agent system for online machine shop scheduling problem. *J Manuf Syst* 2023;67:338–60.
- [5] Zhou L, Liu J, Zheng Y, Xiao F, Xi J. Game-based consensus of hybrid multiagent systems. *IEEE Trans Cybern* 2022.
- [6] Kim YG, Lee S, Son J, Bae H, Chung BDo. Multi-agent system and reinforcement learning approach for distributed intelligence in a flexible smart manufacturing system. *J Manuf Syst* 2020;57:440–50.
- [7] Fu Y, Chai T. Online solution of two-player zero-sum games for continuous-time nonlinear systems with completely unknown dynamics. *IEEE Trans Neural Netw Learn Syst* 2015;27(12):2577–87.
- [8] Yu T, Huang J, Chang Q. Optimizing task scheduling in human-mobile robot collaboration with deep multi-agent reinforcement learning. *J Manuf Syst* 2021;60:487–99.
- [9] Rahmati Y, Hosseini MK, Talebpour A. Helping automated vehicles with left-turn maneuvers: a game theory-based decision framework for conflicting maneuvers at intersections. *IEEE Trans Intell Transp Syst* 2021;23(8):11877–90.
- [10] Azgomi H, Sohrabi MK. A game theory based framework for materialized view selection in data warehouses. *Eng Appl Artif Intell* 2018;71:125–37.
- [11] Bhatia M. Intelligent system of game-theory-based decision making in smart sports industry. *ACM Trans Intell Syst Technol (TIST)* 2021;12(3):1–23.
- [12] Wang X, Zhang Z, Zhang W. Model-based multi-agent reinforcement learning: recent progress and prospects. *arXiv Prepr arXiv* 2022;2203:10603.
- [13] Zou J, Chang Q, Arinez J, Xiao G. Data-driven modeling and real-time distributed control for energy efficient manufacturing systems. *Energy* 2017;127:247–57. <https://doi.org/10.1016/j.energy.2017.03.123>.
- [14] Huang J, Chang Q, Arinez J. Deep reinforcement learning based preventive maintenance policy for serial production lines. *Expert Syst Appl* 2020;160:113701.
- [15] Lowe R, Wu Yi, Tamar A, Harb J, Pieter Abbeel O, Mordatch I. Multi-agent actor-critic for mixed cooperative-competitive environments. *Adv Neural Inf Process Syst* 2017;30.
- [16] Yu C, Velu A, Vinitsky E, Gao J, Wang Y, Bayen A, et al. The surprising effectiveness of ppo in cooperative multi-agent games. *Adv Neural Inf Process Syst* 2022;35:24611–24.
- [17] Kim M, Park J. Learning collaborative policies to solve NP-hard routing problems. *Adv Neural Inf Process Syst* 2021;34:10418–30.
- [18] Kuhpfahl J, Bierwirth C. A study on local search neighborhoods for the job shop scheduling problem with total weighted tardiness objective. *Comput Oper Res* 2016;66:44–57. <https://doi.org/10.1016/j.cor.2015.07.011>.
- [19] Zhang T, Mo H. Reinforcement learning for mobile robot research: a comprehensive review and open issues. *17298814211007305 Int J Adv Mob Robot Syst* 2021;18(3). <https://doi.org/10.1177/17298814211007305>.
- [20] Karnouskos S, Leita P. Key contributing factors to the acceptance of agents in industrial environments. *IEEE Trans Ind Inform* 2016;13(2):696–703.
- [21] Park JW, Shin M, Kim DY. An extended agent communication framework for rapid reconfiguration of distributed manufacturing systems. *IEEE Trans Ind Inform* 2018;15(7):3845–55.
- [22] Neary C, Xu Z, Wu B, Topcu U. Reward machines for cooperative multi-agent reinforcement learning. *arXiv Prepr arXiv* 2007;01962:2020.
- [23] Hernandez-Leal, P., M. Kaisers, T. Baarslag, and E.M. De Cote, A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*, 2017.
- [24] Hu J, Wellman MP. Nash Q-learning for general-sum stochastic games. *J Mach Learn Res* 2003;4(Nov):1039–69.
- [25] Li R, Han Y, Ma T, Liu H. Nash-Q learning-based collaborative dispatch strategy for interconnected power systems. *Glob Energy Interconnect* 2020;3(3):227–36.
- [26] Adida E, DeMiguel V. Supply chain competition with multiple manufacturers and retailers. *Oper Res* 2011;59(1):156–72.
- [27] Xu X, Choi T-M. Supply chain operations with online platforms under the cap-and-trade regulation: impacts of using blockchain technology. *Transp Res Part E: Logist Transp Rev* 2021;155:102491.

- [28] Ren L, Zhang L, Wang L, Tao F, Chai X. Cloud manufacturing: key characteristics and applications. *Int J Comput Integr Manuf* 2017;30(6):501–15.
- [29] Bai T, Liu S, Zhang L. A manufacturing task scheduling method based on public goods game on cloud manufacturing model. in 2018 4th International Conference on Universal Village (UV). IEEE; 2018.
- [30] Wang H, Zhang C, Li K, Ma X. Game theory-based multi-agent capacity optimization for integrated energy systems with compressed air energy storage. *Energy* 2021;221:119777.
- [31] Zhang Y, Wang J, Liu Y. Game theory based real-time multi-objective flexible job shop scheduling considering environmental impact. *J Clean Prod* 2017;167: 665–79.
- [32] Sun D-h, He W, Zheng L-j, Liao X-y. Scheduling flexible job shop problem subject to machine breakdown with game theory. *Int J Prod Res* 2014;52(13):3858–76.
- [33] Nie, L., X. Wang, and F. Pan. A game-theory approach based on genetic algorithm for flexible job shop scheduling problem. in *Journal of Physics: Conference Series*. 2019. IOP Publishing.
- [34] Bhatta K, Huang J, Chang Q. Dynamic mobile robot assignment for flexible serial production systems. *IEEE Mob Robot Autom Lett* 2022. <https://doi.org/10.1109/LRA.2022.3182822>.
- [35] Li C, Huang J, Chang Q. Data-enabled permanent production loss analysis for serial production systems with variable cycle time machines. *IEEE Mob Robot Autom Lett* 2021;6(4):6418–25.
- [36] Zheng L, Yang J, Cai H, Zhou M, Zhang W, Wang J, et al. Magent: a many-agent reinforcement learning platform for artificial collective intelligence. *Proc AAAI Conf Artif Intell* 2018.
- [37] Wang Y, Shi D, Xue C, Jiang H, Wang G, Gong P. AHAC: actor hierarchical attention critic for multi-agent reinforcement learning. in 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE; 2020.
- [38] Setyawan GE, Hartono P, Sawada H. Cooperative multi-mobile robot hierarchical reinforcement learning. *Int J Adv Comput Sci Appl* 2022;13:35–44.
- [39] Ma J, Ye M, Zheng Y, Zhu Y. Consensus analysis of hybrid multiagent systems: a game-theoretic approach. *Int J Robust Nonlinear Control* 2019;29(6):1840–53.
- [40] Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, et al. Continuous control with deep reinforcement learning. *arXiv Prepr arXiv* 2015;1509:02971.