Expert Systems With Applications 192 (2022) 116323

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Deep multi-agent reinforcement learning for multi-level preventive maintenance in manufacturing systems

Jianyu Su [a], Jing Huang [b], Stephen Adams [a], Qing Chang [a,b], Peter A. Beling [a,*]

[a] Department of Engineering Systems and Environment, University of Virginia, Charlottesville, VA 22904, USA
[b] Department of Mechanical and Aerospace Engineering, University of Virginia, Charlottesville, VA 22904, USA

ARTICLE INFO

Keywords:
Multi-level preventive maintenance
Deep multi-agent reinforcement learning
Deep reinforcement learning
Serial production line
Production loss

ABSTRACT

Designing preventive maintenance (PM) policies that ensure smooth and efficient production for large-scale manufacturing systems is non-trivial. Recent model-free reinforcement learning (RL) methods shed lights on how to cope with the non-linearity and stochasticity in such complex systems. However, the action space explosion impedes RL-based PM policies to be generalized to real applications. In order to obtain cost efficient PM policies for a serial production line that has multiple levels of PM actions, a novel multi-agent modeling is adopted to support adaptive learning by modeling each machine as cooperative agent. The evaluation of system-level production loss is leveraged to construct the reward function. An adaptive learning framework based on value-decomposition multi-agent actor–critic algorithm is utilized to obtain PM policies. In simulation study, the proposed framework demonstrates its effectiveness by leading other baselines on a comprehensive set of metrics whereas the centralized RL-based methods struggles to converge to stable policies. Our analysis further demonstrates that our multi-agent reinforcement learning based method learns effective PM policies without any knowledge about the environment and maintenance strategies.

1. Introduction

In manufacturing systems, machines suffer from random failures as the result of degradation of parts over time (Wang, 2002). These unexpected failures abruptly interrupt normal production operations, which can not only cause significant production losses but also require considerable resources for prompt maintenance actions. The maintenance procedure in response to random failures is referred to as corrective maintenance (CM). In order to reduce random failures, a common industrial practice is to proactively shut down machines for preventive maintenance (PM) according to predefined policies. However, deriving PM policies that ensure smooth and efficient production has always been a nontrivial task, since PM actions also impact the system in ways that can incur production losses and resource costs. The costs of excessive PM might outweigh the benefits, but conversely, inadequate PM can be ineffective in preventing random failures. It is challenging to balance the delicate decision trade offs that arise in PM for manufacturing systems, which are distinguished by complicated and nonlinear system dynamics due to interactions among machines/buffers and interruptions from production-related activities (Gershwin, Dallery, Papadopoulos, & Smith, 2012; Huang, Chang, Arinez, & Xiao, 2019).

Reinforcement learning (RL) is a machine learning technique that estimates optimal policies through interactions between an agent and the environment. RL has recently emerged as an effective method for obtaining PM policies in manufacturing systems thanks to its capability for dealing with complex and stochastic environments. The complexity and size of the studied manufacturing systems, and hence RL techniques applied, vary across the spectrum in this line of research. Model-based RL methods utilizing known transition probabilities between states are mostly restricted to two-machine-one-buffer manufacturing systems (Fitouhi, Nourelfath, & Gershwin, 2017; Karamatsoukis & Kyriakidis, 2010; Wang, Wang, & Qi, 2016), because it is impractical to obtain the transition probabilities for larger systems. The explosion of state space with increased system size warrants the use of model-free RL methods, which do not require a model of the environment dynamics, and deep RL, which approximates value functions or policy with deep neural networks (Huang, Chang, & Arinez, 2020; Huang, Chang, & Chakraborty, 2019). Whether model-based or model-free, prior RL work shares the approach of modeling the maintenance problem using a centralized RL framework in which all machines across the manufacturing system are precisely coordinated by a central agent (see, e.g., Chen,

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
E-mail addresses: js9wv@virginia.edu (J. Su), jh3ex@virginia.edu (J. Huang), sca2c@virginia.edu (S. Adams), qc9nq@virginia.edu (Q. Chang), pb3a@virginia.edu (P.A. Beling).

https://doi.org/10.1016/j.eswa.2021.116323
Received 24 November 2020; Received in revised form 16 November 2021; Accepted 26 November 2021
Available online 20 December 2021

Liu, & Xiahou, 2021; Fitouhi et al., 2017; Huang et al., 2020; Huang, Chang, & Chakraborty, 2019; Karamatsoukis & Kyriakidis, 2010; Liu, Chen, & Jiang, 2020; Wang et al., 2016; Yousefi, Tsianikas, & Coit, 2020). This commonality in modeling framework arises from the fact that standard RL is designed for a single agent, and manufacturing research aims to improve the performance of the entire manufacturing system rather than individual machines within the system. Traditional group maintenance (GM) and opportunistic maintenance (OM) are inspired by the observation that the maintenance of one unit does not result additional production loss if another unit is already under maintenance (Wang, 2002). Under the centralized framework, the RL-based PM policy not only outperforms traditional GM/OM policies, but also learns to conduct GM/OM when necessary (Huang et al., 2020).

These previously outlined approaches demonstrate the technical feasibility of applying state-of-the-art RL methods in complex maintenance problems. However, there are still quite a few practical constraints in many real production systems that existing methods fail to incorporate. First, most existing approaches (see, e.g., Fitouhi et al., 2017; Huang et al., 2020; Huang, Chang, & Chakraborty, 2019; Karamatsoukis & Kyriakidis, 2010; Wang et al., 2016) assume that maintenance activities restore a machine to its perfect health state. These approaches fail to consider the industrial reality that there are often multiple maintenance options that can lead to different post-maintenance health states and that maintenance activities could be imperfect (Huang, Chang, Zou, & Arinez, 2018; Kijima, 1989). Second, existing methods fail to cover the full spectrum of production systems in the real world, as production system size dramatically varies from a two-stage machining line to automotive assembly plant with dozens of stations. The current approaches, which have issues being scaled to large manufacturing systems, are often restrained to systems of relatively small sizes. For instance, most of target systems for serial production lines range from two-machine-one-buffer to six-machine-five-buffer (Fitouhi et al., 2017; Huang et al., 2020; Karamatsoukis & Kyriakidis, 2010; Wang et al., 2016). This is because single-agent RL methods are prone to action space explosion as the RL agent's action size grows exponentially with number of machines considered in the system. This exponential increase in the size of the action space can lead to poor performance (Foerster, Farquhar, Afouras, Nardelli, & Whiteson, 2018). Third, when a centralized RL framework is adopted, these approaches implicitly assume that all the state information in manufacturing system is observable. For some large-scale manufacturing systems, system-wide data collection and transmission could be inefficient and even infeasible (Chekired, Khoukhi, & Mouftah, 2018; Kim, Kim, Hassan, & Park, 2017). Finally, it appears to be appealing to model each machine in a complex system as a model-free RL-agent and optimize individual performance to alleviate the problem of action space explosion. However, handcrafting individual rewards is often impractical because of the difficulty inherent in quantifying an individual component's contribution to the success or failure of the system. Hence, optimizing individual performance might contradict the collective objective of the system. In addition, the interactions among agents are ignored since they are considered as a part of an agents' environment.

To address the limitations of RL-based PM policies, we extend our previous RL framework (Huang et al., 2020) to a multi-agent setting and apply MARL to obtain optimal policies. Under this setting, the manufacturing system is modeled as a cooperative system where agents make decisions separately while collectively achieving a common goal of the system. MARL then obtains optimal policies through interactions between agents and the environment. Besides addressing the aforementioned limitations, MARL is an appealing and viable technical approach to taking on those challenges in maintenance problem among large-size manufacturing systems for the following reasons. First, similar to autonomous vehicle coordination (Su, Adams, & Beling, 2020) and traffic lights control (Wiering, 2000), it is natural to model complex manufacturing systems as cooperative multi-agent systems where machines interact and cooperate with others. Second, MARL allows us to take advantage of edge computing to achieve real-time policy execution without assuming global observability as it allows each machine to individually make maintenance decision that only condition on their local observation-action trajectories. Third, MARL guarantees that separate maintenance decisions from individual machines could collectively achieve the common goal of the whole system, without the need to handcraft individual rewards for each machine. Our principal contributions are as follows:

- We settle the action space explosion issue in RL-based PM policies by proposing a deep MARL approach as opposed to centralized RL. PM decision-making based on MARL can be scaled up to large manufacturing systems while still outperforming traditional multi-unit maintenance policies;
- Guided by knowledge on manufacturing systems, we formulate the PM decision making problem using a decentralized partially observable Markov decision process (Dec-POMDP) framework, under which we are able to adopt state-of-the-art RL paradigms such as centralized training and decentralized execution (CTDE) that largely increases policy performance;
- Free from action space explosion issue, we further advance the RL applications in PM problems to more realistic industrial scenarios, for example, important practical constraints including imperfect maintenance effect, local observability and production disruptions can be considered;
- We demonstrate the proposed method by successfully implementing a state-of-the-art MARL algorithm called Value-Decomposition Actor-Critic to solve the PM problem in large manufacturing systems efficiently and effectively. Machines make PM decisions independently but still learn to cooperate to perform OM/GM as reported in prior research based on centralized RL (Huang et al., 2020).

The rest of this paper is organized as follows: In Section 2, we summarize prior art. In Section 3, we introduce the preliminary information about MARL. In Section 4, we describe the manufacturing system that we consider in this study. In Section 5, we elaborate on the proposed framework that is based on MARL. In Section 6, we benchmark our MARL-based policies with DQN-based policies, along with traditional policies, on two sets of experiments. The final section concludes the study.

2. Related work

Machine maintenance policies have been studied under numerous application scenarios and are characterized by the target system. Based on the structure of the studied system, the current literature can be categorized into single-unit policies and multi-unit policies.

Single-unit policies provide maintenance policies for one-unit systems. The relation between the maintenance decisions and system maintenance cost is usually known in such systems. Therefore, it is common to model single-unit maintenance as a stochastic process, where the optimal solution can be solved to achieve certain objectives (e.g. minimizing the maintenance cost rate Love & Guo, 1996; Monga, Zuo, & Toogood, 1997; Zheng & Fard, 1991, maximizing the machine availability Chan & Shaw, 1993, maximizing machine reliability Malik, 1979, etc.). Examples are age-dependent policies (Wang & Pham, 1999) and repair limit policies (Koshimae, Dohi, Kaio, & Osaki, 1996). While a multi-unit system can be reduced to multiple single-unit systems, a generalization of single-unit policies to multi-unit systems is questionable due to its rigorous assumption that there is no structural dependencies between subsystems (Wang, 2002). Therefore, single-unit policies are not suitable for serial production lines.

In comparison, multi-unit policies that consider dependencies between subsystems are more suitable to solve maintenance problems

in large-scale production systems because the structural and operational dependencies among machines are often considered in this line of research. However, maintenance policies are often derived by exploiting the specific structure of the system, such as serial systems (Ebrahimipour, Najjarbashi, & Sheikhalishahi, 2015), parallel systems (Barros, Grall, & Bérenguer, 2007), and k-out-of-n systems (de Smidt-Destombes, van der Heijden, & van Harten, 2009) etc, thus are often limited to certain scenarios. Most of the multi-unit policies are based on the fundamental idea of group and opportunistic maintenance, which are inspired by the observation that the maintenance of one subsystem does not incur extra production loss if another subsystem is already under maintenance (Wang, 2002). For instance, (Nicolai & Dekker, 2008; Shafiee & Finkelstein, 2015) propose a group maintenance policy to conduct multiple PM simultaneously, thus avoiding incurring additional production loss. Opportunistic maintenance policies (Ab-Samat & Kamaruddin, 2014; Laggoune, Chateauneuf, & Aissani, 2009; Xia, Jin, Xi, & Ni, 2015) aim to derive a time window where insertions of PM do not result in extra production loss. In addition, those heuristic-based methods tend to consider less general production systems, where buffers among machines are not considered, thus neglecting the delays of machine state propagation caused by buffers (Tan & Gershwin, 2011).

Recent studies that do not assume heuristics of GM and OM policies often adopt the approach of reinforcement learning, by modeling the maintenance problems as MDPs (Amari, McLaughlin, & Pham, 2006; Huang et al., 2020; Robelin & Madanat, 2007), semi-Markov decision processes (Chan & Asgarpoor, 2006; Tomasevicz & Asgarpoor, 2006), and POMDPs (AlDurgam & Duffuaa, 2013; Byon & Ding, 2010; Byon, Ntaimo, & Ding, 2010). Most of these works either assume that a model is given or only derive maintenance policies for simplified multi-unit systems, which impedes those methods from be generalized to realistic scenarios. Karamatsoukis et al. assume the model of the system and derives a PM policy for a two-machine-one-buffer manufacturing system using dynamic programming (DP) (Karamatsoukis & Kyriakidis, 2010). Wang et al. model a two-machine-one-buffer manufacturing system as semi-MDP (Wang et al., 2016). Ramirez et al. propose a simulation-based approximate dynamic programming approach for the optimization of PM scheduling decisions in semiconductor manufacturing systems, while suffering from the assumption that the line is perfectly balanced (Ramírez-Hernández & Fernandez, 2010). Arab et al. derive maintenance policies on systems with simplified dynamics, i.e. the maintenance schedule does not affect machine reliability status (Arab, Ismail, & Lee, 2013). Choo et al. derives a hierarchical maintenance policy for parallel production lines without buffers (Choo, Adams, & Beling, 2017). Huang et al. which derives model-free Q-learning maintenance policies for a six-machine-five-buffer serial production (Huang et al., 2020), represents the work most closely related to this paper. Nonetheless, their PM policies do not consider imperfect maintenance effects and have issues scaling to larger manufacturing systems.

Our study benefits from the recent advances in deep MARL, which has been at the intersection of machine learning and game theory. Early works that model multi-agent systems as general-sum games were demonstrated only on toy examples (Bu, Babu, De Schutter, et al., 2008). By contrast, a number of deep MARL methods, especially ones that utilize a CTDE paradigm, formulate multi-agent systems as a decentralized-POMDP (Dec-POMDP) and report good performance on complex multi-agent tasks, e.g. StarCraft II micromanagement tasks (Samvelyan et al., 2019). The CTDE paradigm takes advantage of extra state information to guide the training of decentralized agents has been proved to be vital to achieve cooperation (Rashid et al., 2018; Su, Adams, & Beling, 2021). Among all CTDE methods, value-decomposition actor-critic (VDAC) stands out for two reasons:

- VDAC achieves state-of-the-art performance on StarCraft II micromanagement tasks among CTDE actor-critic methods (Su et al., 2021).
- Compared with Q-learning methods (Rashid et al., 2018; Son, Kim, Kang, Hostallero, & Yi, 2019; Sunehag et al., 2017), VDAC does not require a replay buffer, which typically has major hardware requirements. Moreover, VDAC is compatible with the A2C framework (Mnih et al., 2016), which enables game experience to be sampled efficiently during the training.

3. Preliminaries and notations

3.1. Preliminaries

Decentralized Partially Observable Markov Decision Processes (Dec-POMDPs): We model the manufacturing system as a fully cooperative multi-agent system with n agents. Each agent represents a machine unit in the system and is identified by $a \in A \equiv \{1, \dots, n\}$. At every timestep, each agent a takes an action $u^a \in U$ simultaneously, forming a joint action $\boldsymbol{u} \in \boldsymbol{U} \equiv U^a, \forall a \in \{1, \dots, n\}$. The environment, namely the manufacturing system, has a true state $s \in S$, a transition probability function $P(s'|s, \boldsymbol{u}) : S \times \boldsymbol{U} \times S \to S$, and a global reward function $r(s, \boldsymbol{u}) : S \times \boldsymbol{U} \to \mathbb{R}$. In this setting, an agent a can only receive its corresponding partial observation $o^a \in O^a$ instead of the true state s. Each agent has its observation-action history $\tau^a \in T \equiv (O^a \times U)$, on which the agent conditions a stochastic multinomial policy $\pi(u^a|\tau^a) : T \times U \to [0, 1]$. Throughout this paper, quantities in bold denote joint quantities over agents, and bold quantities with the superscript $-a$ denote joint quantities over agents other than the agent a. MARL agents aim to maximize the discounted return $R_t = \sum_{l=1}^{\infty} \gamma^l r_{t+l}$. The global value function $V^{\boldsymbol{\pi}}(s_t) = E[R_t|s_t = s]$ is the expected return for following the joint policy $\boldsymbol{\pi}$ from state s. The joint action-value function $Q^{\pi}(s, \boldsymbol{u}) = E[R_t|s_t = s, \boldsymbol{u}]$ defines the expected return for selecting joint action $\boldsymbol{u}$ in state s and following the joint policy $\boldsymbol{\pi}$.

For consistency in notation, we use bold quantities under single-agent RL settings. This is also because quantities output by a single-agent RL agent are equivalent to the joint quantities over agents output by multi-agent RL agents.

Deep Q-Network (DQN): Utilizing neural networks to approximate action-value $Q(s, \boldsymbol{u})$, DQN reports human-level performance in many Atari games (Mnih et al., 2015). Learned parameters in DQN networks are updated by minimizing a sequence of loss functions at each iteration i:

$$L(\theta)_i = E_{(s,\boldsymbol{u},r,s')}[y'_i - Q^{\pi}(s, \boldsymbol{u}; \theta_i)], \tag{1}$$

where $y'_i = r + \max_{\boldsymbol{u}'} Q^{\pi}(\boldsymbol{u}', s'; \theta_{i-1})$. The DQN agent takes observations as input and outputs action-values $Q(s, \boldsymbol{u})$ for $\boldsymbol{u} \in \boldsymbol{U}$, among which the action with the largest action-value is selected. To resolve the strong correlation between consecutive samples, DQN utilizes a replay buffer to store its past experience in tuples $(s, \boldsymbol{u}, r, s')$.

DQN based PM Policy: In our prior work (Huang et al., 2020), a PM policy for serial production line is trained using the DQN algorithm (Mnih et al., 2015). The state is defined to be $s = [g_1, \dots, g_M, b_2, \dots, b_M, d_1^r, \dots, d_M^r]$, where $g_a, a = 1, 2, \dots, M$, denotes the age of machine M_a, $b_a, a = 2, 3, \dots, M$, denotes the level of buffer B_a, and $d_a, a = 1, 2, \dots, M$, denotes the remaining maintenance duration on machine M_a. The action is $u = [u_1, u_2, \dots, u_M]$, where $u_a, a = 1, 2, \dots, M$, is a binary decision variable to determine whether turning off machine M_a for PM or not. The reward function is developed based on the real-time production loss evaluation proposed in Zou, Chang, Lei, and Arinez (2016). The framework was applied to PM control in a six-machine-five-buffer serial production line only considering perfect maintenance effect.

Multi-Agent Policy Gradient Algorithms: Multi-agent policy gradient methods are extensions of policy gradient algorithms with a policy $\pi_\theta(u^a|o^a), a \in 1, \dots, n$ for each agent. Instead of estimating the joint action-value $Q(s, \boldsymbol{u})$, policy gradient methods parameterize

Table 1
Notations used in this paper.

Notation	Description
M_a	ath machine, $a \in \{1, 2, \dots, n\}$
T_a	Cycle time of machine M_a
T_{n^*}	Cycle time of the slowest machine M_{n^*}
g_a	Age of machine M_a
r_a	Maintenance recovery factor for machine M_a
d_a	Maintenance duration on machine M_a
d_a^r	Remaining maintenance duration on machine M_a
C_a	Maintenance resource cost on machine M_a
B_a	ath buffer, $a \in \{2, \dots, n\}$
c_a	Capacity of buffer B_a
b_a	Buffer level for buffer B_a
T	Simulation time horizon
s	Global system state
o^a	Local observation for machine M_a
u^a	Action for machine M_a
$\boldsymbol{u}$	Joint action
$r(t)$	Step reward at time t
$PL(t)$	Production loss at time step t
π^a	PM policy for machine M_a
π	Joint PM policy
$V_{tot}(s)$	Global state value
$V(o_t^a)$	Local state value
θ	Neural network trainable parameters
$Q(s, \boldsymbol{u})$	State–action value

Table 2
Acronyms used in this paper.

Notation	Description
PM	Preventive maintenance
CM	Corrective maintenance
OM	Opportunistic maintenance
GM	Group maintenance
R2F	Run-to-failure scenario
OGM	Opportunistic group maintenance
DQN	Deep Q-network
MARL	Multi-agent reinforcement learning
VDAC	Value decomposed actor-critic
MLP	Multi-layer perceptron
RNN	Recurrent neural network
GRU	Gated recurrent units

policies with learnable parameters and with the objective of learning a joint policy such that J is maximized:

$$J(\theta) = E_{s\sim p^{\boldsymbol{\pi}}, \boldsymbol{u}\sim \boldsymbol{\pi}}[R(s, \boldsymbol{u})]. \tag{2}$$

In the literature of RL, actor-critic methods utilize a critic that can estimate state-values or action-values to mitigate high variances of the policy gradient estimate. Similarly in the domain of MARL, critics that evaluate global state-values or joint action-values are introduced to reduce the gradient variances. Hence, the actors are optimized by following gradients that depend on the estimation of the critic. The gradient is written as follows:

$$\nabla_\theta J = E_{\boldsymbol{\pi}}\left[\sum_a \nabla_\theta \log \pi(u^a|\tau^a)\big(Q(s, \boldsymbol{u}) - V(s)\big)\right], \tag{3}$$

where $V(s)$ is estimated by a central critic. Among all actor-critic methods, value decomposition actor-critic (VDAC) stands out as it not only addresses the credit assignment problem but also achieves good performance on the standard multi-agent benchmark, StarCraft II micromanagement games.

3.2. Notations

The notations used in this paper are listed in Table 1. Table 2 presents all the acronyms adopted in this paper.

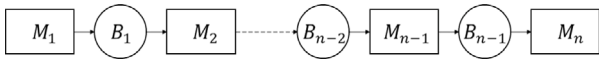


Fig. 1. Serial product line structure.

4. Problem statement

4.1. System description

We consider a serial production line that consists of n machines and $n-1$ buffers with limited capabilities as shown in Fig. 1. The arrows depict the direction of material flow in the system. The material is referred to as a final product once it has been processed by all machines sequentially. Otherwise, it is said to be an intermediate part. The serial production line is described as follows:

- The ath machine in the production line is denoted as M_a, $a \in \{1, 2, \dots, n\}$. The cycle time of machine M_a is T_a.
- The ath buffer in the production line is denoted as B_a, $a \in \{2, 3, \dots, n\}$. Each buffer has a limited capacity c_a. The real-time buffer level of buffer B_a is denoted as b_a.
- An intermediate part that comes out of machine M_a enters its downstream buffer B_a if and only if $b_a < c_a$, otherwise M_a is said to be blocked.
- Machine M_a starts a new cycle by receiving an intermediate part from its upstream B_{a-1} if $b_{a-1} > 0$, otherwise M_a is said to be starved.
- The first machine M_1 is never starved and the last machine M_n is never blocked
- The lifespan of machine M_a is denoted as l_a, which follows a known probability distribution p_a.
- Machine M_a fails when its age g_a reaches its lifespan. A maintenance action reacts to such random failure is referred to CM. Before machine M_a fails, a PM can be conducted to restore machine M_a's health status.

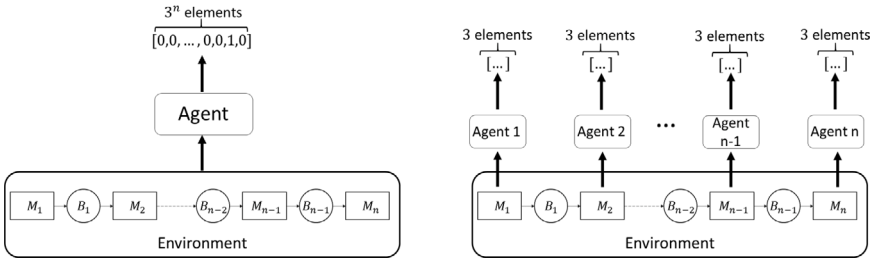
For the serial production line described above, maintenance actions, either PM or CM, would require the machine to temporarily cease production. If a machine M_a is under maintenance, intermediate parts gradually drains in its downstream buffers and increasingly pile up in its upstream buffers, which might lead adjacent machines to idle states due to blockage or starvation. The gradual propagation of machines' idleness could finally cause production loss of the whole system. The analytical model and system property analysis for the described production line can be found in our prior work (Zou, Chang, Arinez, Xiao, & Lei, 2017).

4.2. Maintenance effect and maintenance actions

The extent to which the maintenance action can restore a machine's health state is referred to as maintenance effect. In this work, we use Kijima Model II (Kijima, 1989) to model the effects of different levels of maintenance actions. Let g_a denote the machine M_a's age prior to maintenance, then the machine's age immediately after the maintenance, denoted by g_a', is given by

$$g_a' = g_a \times r_a \tag{4}$$

where $0 \le r_a < 1$ is the recovery factor of the maintenance on machine M_a. $r_a = 0$ indicates a perfect maintenance, since the machine age g_a' is set to be zero after maintenance. By contrast, $0 < r_a < 1$ relates to an imperfect maintenance, as the post-maintenance age g_a', also referred to as 'virtual age' in Kijima (1989), starts somewhere between zero and the original age g_a. In other words, imperfect maintenance does not fully restore machine's health condition and hence earlier



(a) Serial production line RL formulation (b) Serial production line MARL formulation

Fig. 2. Action space comparison between RL and MARL formulations.

arrival of subsequent random failure would be expected after imperfect maintenance than a perfect one.

In this work, we consider two levels of PM for each machine, namely an imperfect (level 1) PM option with and a perfect (level 2) PM option, denoted as PM_1 and PM_2 respectively, with recovery factors $0 < r_a^{PM_1} < 1$ and $r_a^{PM_2} = 0$. Generally, imperfect PM costs less resource and has shorter duration than perfect PM. In addition, CM is conducted in response to random failures and its effect is assumed to be perfect, i.e. $r_a^{CM} = 0$. The cost and duration of CM are much larger than that of any PM options (Chitra, 2003). In summary, maintenance resource costs rank as $C_a^{PM_1} < C_a^{PM_2} \ll C_a^{CM}$ and maintenance duration satisfies $d_a^{PM_1} < d_a^{PM_2} \ll d_a^{CM}$.

4.3. Cost analysis and system objective

Given a time horizon T, the total system cost related to maintenance activities can be broken down into: resource costs of all CM actions $C_{CM}(T)$, resources costs of all PM actions $C_{CM}(T)$, and loss of revenue $C_{PL}(T)$ on account of system production losses. It is noted that both PM and CM would contribute to $C_{PL}(T)$ by introducing machine stoppages that in turn cause system production losses.

Let π^a denote PM policy for machine M_a, and $\boldsymbol{\pi}$ represent system-level joint PM policy. The PM policy $\boldsymbol{\pi}$ instructs when and where to conduct PM, thus largely shaping the cost break-downs and hence the total system cost. We use $C_{\boldsymbol{\pi}}(T)$ to represent total system cost in time horizon T under a joint PM policy $\boldsymbol{\pi}$, and it can be written as:

$$C_{\boldsymbol{\pi}}(T) = C_{PM}(T) + C_{CM}(T) + C_{PL}(T) \tag{5}$$

The objective of this study is to find an optimal joint policy $\boldsymbol{\pi}^*$ to minimize the total cost C in a time horizon T through deep MARL techniques:

$$\boldsymbol{\pi}^* = \arg\min_{\boldsymbol{\pi}} \{C_{\boldsymbol{\pi}}(T)\} \tag{6}$$

It should be noted that there might be other costs that are related to maintenance depending specific production systems, e.g., inventory cost, energy cost and emissions etc. Since the MARL is model-free, one can include extra cost components in Eq. (5) and still obtain optimal PM policies through the proposed framework in this work.

5. Multi-agent adaptive decision framework

Our previous study proposed a centralized adaptive decision framework that is based on DQN (Huang et al., 2020) and successfully applied it to a six-machine-five-buffer production line. However, this RL-based framework has issues being generalized to more realistic applications due to action space explosion. Consequently, the framework only consider one level of PM to limit the action space size of the studied system.

In comparison, MARL is less prone to action space explosion as its action space is independent of number of agents in the system. This is because, under the MARL's decentralized setting, the agents make decisions independently. Fig. 2 depicts the action spaces of a n-machine production line under RL and MARL respectively.

The following sections are organized to describe: (1) how to formulate the PM of serial production line problem to a MARL problem; and (2) how to derive PM policies effectively and efficiently with the aforementioned VDAC algorithm.

5.1. Dec-POMDP formulation

5.1.1. State definition

In the CTDE framework, agents condition their actions on the local partial observations and are optimized by the gradients that are dependent on global states. Therefore, we need to design local observations o that provide the basis for agents' actions and global states s that encapsulate information that is useful for training.

Given a machine M_a, three factors that are essential to its PM decision making are: (1) the age of machine M_a, g_a; (2) the upstream and downstream buffer status related to b_{a-1} and b_a; (3) M_a's remaining maintenance duration, d_a^r, if applicable. In addition, the status of M_a's immediate adjacent machines is also useful because it might relate to the blockage or starvation of machine M_a. Consequently, the local observation for machine M_a is defined as:

$$o^a = [g_a, g_{a-1}, g_{a+1}, b_{a-1}, c_a - b_a, c_{a+1} - b_{a+1}, d_a^r, d_{a-1}^r, d_{a+1}^r]. \tag{7}$$

Note that we use buffer vacancy $c_a - b_a$ instead of buffer level b_a to represent the real-time status of the downstream buffer. This is because the buffer vacancy is a more sufficient criteria than buffer level in determining if the machine is blocked or not. Furthermore, the s that represents the global state of the production line is obtained by concatenating local observations from all machines and it is written as:

$$s = [o^1, o^2, \ldots, o^n]. \tag{8}$$

5.1.2. Action definition

CM is triggered by a machine's random failure and is out of the machine's control. Hence, the action that can be taken for a machine is to conduct a PM or leave it as it is. In this study, we consider two levels of maintenance actions: a perfect (level 2) PM and a imperfect (level 1) PM. This leaves us an action vector of size 3 for each agent. For instance, machine M_a's action can be written as:

$$u^a = \begin{cases} 0 & \text{leave machine } a \text{ as it is} \\ 1 & \text{conduct imperfect (level 1) PM on machine } a \\ 2 & \text{conduct perfect (level 2) PM on machine } a. \end{cases} \tag{9}$$

This action representation can be extended to the cases where more levels of PMs are considered. Note that the action space will be 3^n for the same manufacturing system under the centralized RL framework as shown in Fig. 2. Given a machine in the real industry, the available levels of imperfect maintenance actions are depending on the specific machine structure and maintenance practice (Pham & Wang, 1996). For

a clear demonstration, we only mark three PM options in Eq. (9). However, the action definition can be adapted to include more maintenance options as required by specific applications, and the action space of an agent only increases linearly with available maintenance levels.

5.1.3. Reward definition

On one hand, machine starvation and blockage caused by random failure might lead to production loss. On the other hand, excessive PM would also result in increasing machine down time and cost. To balance the trade-off between PM and random failure, we define the reward as the negative overall cost, which consists of production loss cost due to machine stoppages, and maintenance resource costs incurred by PM and CM. At time step t, the reward $r(t)$ is given by:

$$r(t) = -PL(t)\cdot c_p - \sum_{a=1}^{n} C_a^{PM_1}\mathbb{1}_a^{PM_1}(t) - \sum_{a=1}^{n} C_a^{PM_2}\mathbb{1}_a^{PM_2}(t) - \sum_{a=1}^{n} C_a^{CM}\mathbb{1}_a^{CM}(t) \tag{10}$$

where each component of the reward function is explained as follows:

- c_p is the unit price of a final product
- $PL(t)$ is the system production loss at time step t
- Therefore, $PL(t)\cdot c_p$ represents the production loss cost caused by maintenance activities in a single time step
- $C_a^{PM_1}$ is the resource cost of an imperfect PM on machine M_a
- $\mathbb{1}_a^{PM_1}(t)$ is a binary variable indicates whether an imperfect PM is initiated on machine M_a at time step t. $\mathbb{1}_a^{PM_1}(t) = 1$ if M_a initiates a level 1 PM at time step t, otherwise $\mathbb{1}_a^{PM}(t) = 0$.
- Therefore, $\sum_{a=1}^{n} C_a^{PM_1}\mathbb{1}_a^{PM_1}(t)$ is the total imperfect PM cost on all machines in a single time step
- Similarly, $\sum_{a=1}^{n} C_a^{PM_2}\mathbb{1}_a^{PM_2}(t)$ and $\sum_{a=1}^{n} C_a^{CM}\mathbb{1}_a^{CM}(t)$ are the total cost of total perfect PM and CM costs on all machines at time step t respectively.

Zou et al. (2017) has proved that a downtime event would lead to system-level production loss only when it impedes the slowest machine, and therefore the production loss at time step t can be evaluated by:

$$PL(t) = \frac{D(t)}{T_{n^*}}. \tag{11}$$

Here, n^* indexes the slowest machine in the serial line, $D(t)$ denotes the stoppage time of the slowest machine at step t, and T_{n^*} denotes the cycle time of the slowest machine.

5.2. Applying VDAC to obtain PM policy

It is often elusive to handcraft rewards for individual agents in complex systems. *Difference rewards* $D^a = r(s,\boldsymbol{u}) - r(s,(\boldsymbol{u}^{-a},c^a))$, which adopts the reward changes incurred by replacing agent a's action with a default action c^a as local rewards (Wolpert & Tumer, 2002). The intent of the difference rewards substitution is to shed light on quantifying individual rewards in cooperative multi-agent systems. Unfortunately, difference rewards requires $n-1$ counterfactual simulations to be rolled out at every time step, which can present an extreme computational burden. While it is impossible to implement difference rewards in practice, it is easy to enforce the monotonic relation between shaped local reward D^a and $r(s,\boldsymbol{u})$ as indicated by difference rewards.

Inspired by difference rewards, VDAC is an on-policy multi-agent actor-critic that enforces the monotonic relation between local state-values and global state-values. Specifically, a mixing neural network represents the global state-value as a non-linear function of local state-values. To enforce the monotonic relation between local and global state-values, weights of mixing neural network are restricted to be non-negative. This constraint ensures that given agents other than agent a stay at the same local states, the global state-value should increase if agent a transmit to local states with higher local state-values.

As shown in Fig. 2(b), distributed agents are responsible for making decisions for the corresponding machine under the MARL setting.

5.2.1. VDAC architecture

VDAC consists of distributed actors that make decisions for designated machines and a central critic that estimate the global state-value V_{tot}. As shown in Fig. 3, the actor network takes inputs as observations o_t^a and actions u_{t-1}^a of the previous step $t-1$, outputs a multinomial policy $\pi(o_t^a)$ for timestep t. It also estimates the state-value of the local observation $V(o_t^a)$. To capture the temporal dependencies within agents' trajectories, Gated Recurrent Unit (GRU) is Incorporated in actor networks. Note that to speed up training as well as save memory, actor networks share the same weights. The recent advance of MARL literature (e.g. Foerster et al., 2018; Rashid et al., 2018; Son et al., 2019; Su et al., 2021; Sunehag et al., 2017) is coupled by this parameter sharing technique. The value mixing network, which takes input as local state-values $V(o_t^a)$ and outputs the global state-value $V_{tot}(s)$, serves as a central critic. To incorporate the global state information that is unavailable to actors, the parameters of the value mixing network is generated from a hypernetwork which takes input as the global state s_t. Please refer to Su et al. (2021) for details about VDAC method.

Actors are optimized by following gradients that depend on the central critic. Let θ_π to denote actor network parameters and θ_V to denote hypernet parameters for generality. The actor network is optimized by following the policy gradient given by:

$$\nabla_{\theta_\pi} J = E_{\boldsymbol{\pi}}\left[\sum_a \nabla_{\theta_\pi}\log\pi(u^a|\tau^a)\big(Q(s,\boldsymbol{u}) - V(s)\big)\right], \tag{12}$$

where $V(s)$ is estimated by a central critic and $Q(s,\boldsymbol{u}) = r + \gamma V(s\prime)$. The critic is optimized by minibatch gradient descent to minimize the following loss:

$$L_t(\theta_V) = \Big(y_t - V_{tot}(s_t;\theta_V)\Big)^2, \tag{13}$$

where $y_t = \sum_{i=0}^{k-1}\gamma^i r_t + \gamma^k V(s_{t+k})$ is the target value, k can vary from state to state and is upper-bounded by T_{max}.

5.2.2. Action mask

Note that when machine M_a is under maintenance (namely, $d_a(t) > 0$), the machine cannot conduct actions that interrupt its current maintenance. Hence the only action available for machine M_a is to leave it as it is. To enforce the rules of the studied process, an action mask m_t is often applied to filter out invalid actions (Samvelyan et al., 2019; Silver et al., 2017). More specifically, our multinomial policy network outputs a 3-element action probability vector vec_t, representing the probability of taking each action. Element-wise multiplication $vec_t \otimes m_t$ makes sure that the probability for invalid actions to be 0. The resulted vector is then normalized such that the sum of the vector equals to 1 (see Fig. 4).

5.2.3. Implementation

Fig. 5 demonstrate the procedure to implement VDAC to the maintenance problem in serial production lines. The procedure can be generally divided into two parts: (1) Obtaining experience, and, (2) optimizing parameters. In the phase of obtaining experience, multiple episodes are rolled out independently to increase experience sampling efficiency. For an agent a, its trajectory T_i at episode i is recorded $\{(o_0^a, s_0, u_0^a, r_0, a), \ldots, (o_{T-1}^a, s_{T-1}, u_{T-1}^a, r_{T-1}, a)\}_i$. In addition, its corresponding action masks $\{m_0^a, m_1^a, \ldots, m_{T-1}^a\}_i$ at episode i are also recorded. Note that the central critic is absent during this phase. During the parameter optimization phase, the data acquired in the sampling phase will be discarded once it is used to optimize the network parameters. Therefore, no replay buffer is required.

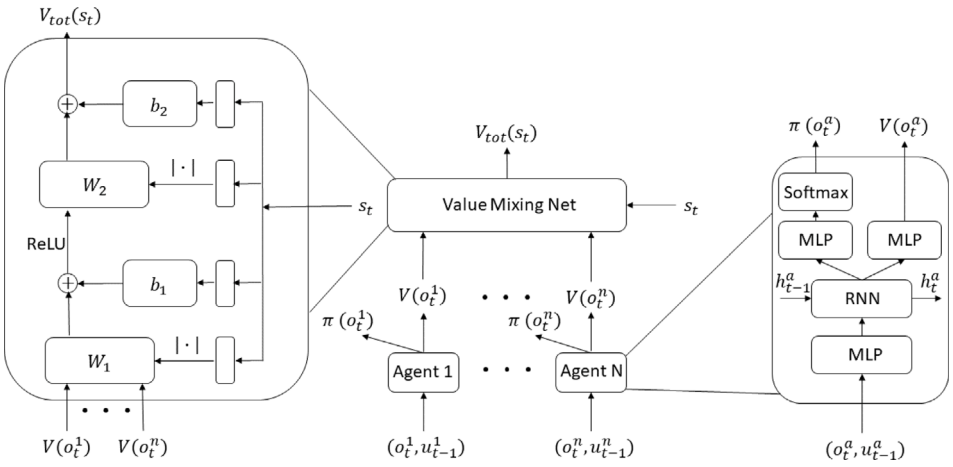


Fig. 3. VDAC architecture.

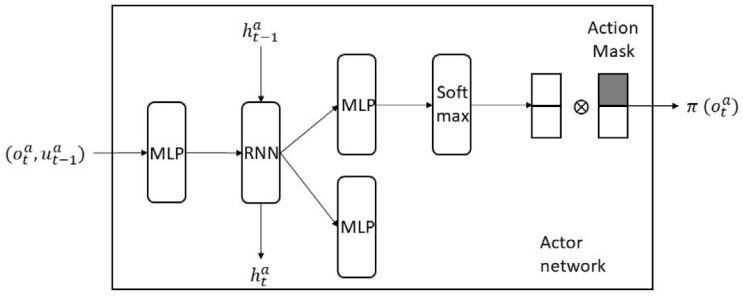


Fig. 4. Action mask.

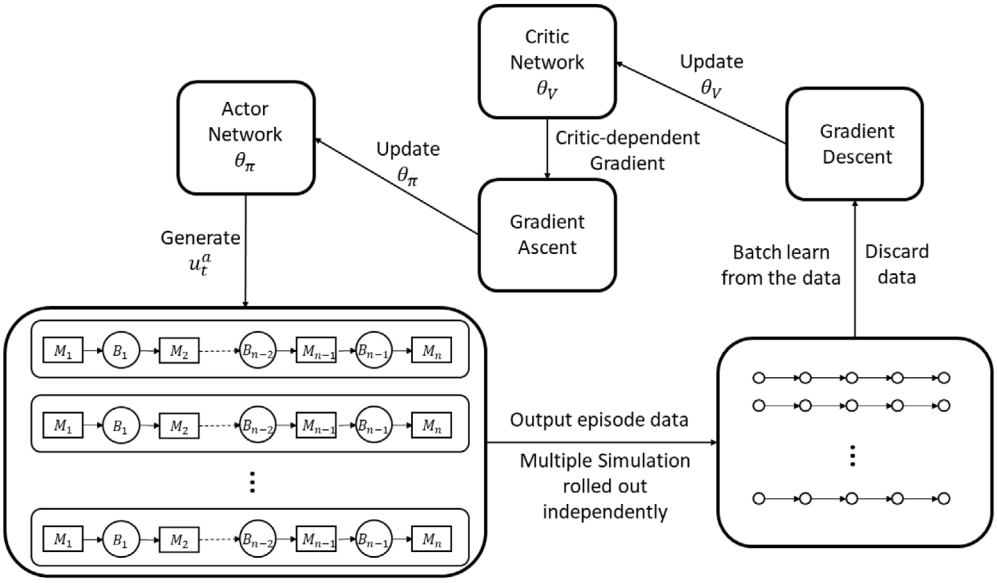


Fig. 5. Implementation details.

6. Numerical study

Two numerical studies are conducted in our experiments to demonstrate the effectiveness of the proposed MARL-base policy. The first numerical study compares the proposed methods with others, utilizing a six-machine-five-buffer serial production line adopted from our previous study on DQN-based policy (Huang et al., 2020). The second numerical study uses a ten-machine-nine-buffer serial production line to further examine the scalability of those methods.

6.1. Environment description

We first test our method on a serial production line simulation that is extended from Huang et al. (2020). Compared with the original simulation introduced in Huang et al. (2020), our simulation is obtained by simply adding an imperfect PM action for each machine. The proposed MARL policy is compared with the DQN-based policy as well as other traditional PM policies. Note that by adding one action for each machine, the action space of the DQN agent increases from

Table 3
Machine parameters.

Parameters	M_1	M_2	M_3	M_4	M_5	M_6
Cycle Time T_a (min)	1.00	0.90	1.20	1.05	1.10	1.05
Level 1 PM cost $c_a^{PM_1}$ ($)	50.0	45.0	55.0	60.0	45.0	50.0
Level 2 PM cost $c_a^{PM_2}$ ($)	105.0	98.0	110.0	120.0	90.0	103.0
Level 1 PM duration $d_a^{PM_1}$ (min)	5	4	4	5	6	4
Level 2 PM duration $d_a^{PM_2}$ (min)	10	9	8	11	12	9
CM cost c_a^{CM} ($)	110.0	100.0	120.0	130.0	110.0	125.0
CM duration d_a^{CM} (min)	28	31	25	32	24	25
level 1 PM degree $r_a^{PM_1}$	0.8	0.8	0.8	0.8	0.8	0.8
level 1 PM degree $r_a^{PM_2}$	0	0	0	0	0	0
Scale parameter α_a	400	430	500	580	550	575
Shape parameter β_a	15	15	15	15	15	15

Table 4
Buffer parameters.

Parameters	B_1	B_2	B_3	B_4	B_5
Buffer capability b_i	8	10	12	6	8

$2^6 = 64$ to $3^6 = 729$. The simulation parameters are provided by industry collaborators and are listed in Tables 3 and 4. Machine M_a's life span follows a Weibull distribution whose scale parameter and shape parameter are α_a and β_a respectively.

For each experiment run, the simulation duration is 1500 timesteps. Each run starts with randomly generated initial system states, including random machine ages and random buffer levels, to make sure that the agents' policies are independent of initial conditions. During the simulation, every time a machine finished its PM or CM, its lifespan is sampled from the Weibull distribution conditioning on post-maintenance age.

6.2. Baselines

6.2.1. Deep Q-learning (DQN)

Following Huang et al. (2020), we implement a DQN policy for multi-level PM in the serial production line. To make the DQN agent consistent with MARL agents, our DQN policy network follows DRQN (Hausknecht & Stone, 2015) where a GRU (Chung, Gulcehre, Cho, & Bengio, 2014) is utilized to memorize the past state of the system. Note that the DQN agent can access global state s whereas our MARL agents can only access partial observations. DQN also need an action mask to filter out invalid actions. At time t, invalid actions $u_t^{invalid}$ are masked out by setting $Q(s_t, u_t^{invalid}) = -\infty$.

6.2.2. Run-to-Failure (R2F)

The Run-to-Failure scenario is used to evaluate the system performance if no PM is conducted throughout the time horizon. Each machine in the production line keeps running until it encounters a random failure. Hence, CM is the only type of maintenance that is conducted in this scenario. Any other PM policy is deemed as effective only when it improves system performance from the Run-to-Failure scenario.

6.2.3. Group Maintenance (GM)

Given GM policy, all the machines would receive PM simultaneously. GM policy tends to reduce the impacts of PM on the whole system by enforcing concurrent machine stoppages. There is one pivotal decision variable in GM policy, which is the time interval τ for carrying out group PM. Since there is no analytical method to derive τ due to the ultra complexity of the production system, the optimal τ can be found through Monte Carlo simulation.

6.2.4. Opportunistic Maintenance (OM)

OM policy is inspired from the observation that once there is a machine undergoing CM, other machines can receive PM without incurring extra system production losses. Under OM policy, whenever one or multiple machines fails in the serial production line, we will conduct CM on those failed machines. In the meantime, we will turn off all other operational machines for PM.

6.2.5. Opportunistic Group Maintenance (OGM)

OGM policy is a combination of OM policy and GM policy. Similar to GM, there is also a predefined time interval τ. If there is a random failure occurs before the time interval τ arrives, OM policy would be triggered, i.e. all other operational machines are turned off for PM. If the time interval τ is reached without machine random failures, GM policy would be carried out so that all the machines receive PM simultaneously. We also leverage Monte Carlo simulation to derive the optimal τ.

6.3. Training description

MARL training: The agent networks contains a GRU (Chung et al., 2014) with a 64-dimensional hidden state, with a fully-connected layer before and after. Algorithms are trained with RMSprop with learning rate 5×10^{-4}. We set $\gamma = 0.99$. The mixing network consists of a single hidden layer of 32 units, whose parameters are outputted by hypernetworks. An ELU activation function follows the hidden layer in the mixing network. The hypernetworks consist of a feedforward network with a single hidden layer of 64 units with a ReLU activation function. Throughout the paper, $Q(s_t, \boldsymbol{u}_t)$ is given by:

$$Q(s_t, \boldsymbol{u}_t) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}), \tag{14}$$

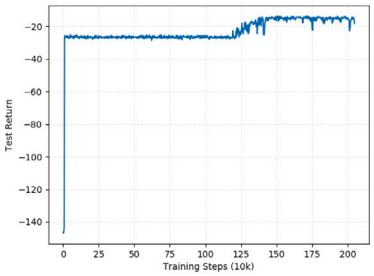
where k can vary from state to state and is upper-bounded by T.

As described in the previous section, the algorithm alternates between sampling experience and optimizing parameters. MARL model is set to train 2050000 timesteps. Eight episodes are run independently during sampling phase. Models are saved every 200000 training timesteps. The best saved model is later benchmarked with other baselines. Note that actions are drawn according to the multinomial probability outputted by the actor network during the training whereas actions with the largest probability are picked during the test.

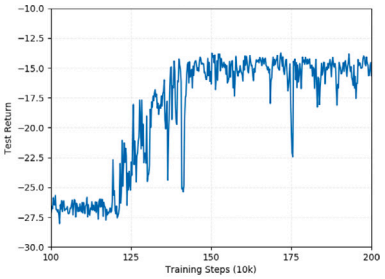
Training experiments are conducted on a PC with a Nvidia RTX 2080 Ti graphics card, a AMD 3700X CPU and a 32 GB RAM, with one run taking approximately 1 h.

RL Training: We also train a RL agent for comparison. The Q-learning network is identical to the actor network described in the previous section except that the Q-learning network does not have a softmax layer and an additional state-value layer. The buffer size is set to be 450000. During the training, actions are selected via ϵ-greedy. ϵ is set to 1 initially and then linearly decreases to 0.05 in 50000 training steps. During the testing, the action with the largest action-value is picked. Other settings are identical to MARL training settings. Note that DQN is an off-policy algorithm, which is not compatible with the A2C framework. Therefore, one episode, instead of 8 independent episodes, is rolled out during training. DQN's training process takes approximately 4 h with the same hardware that described above.

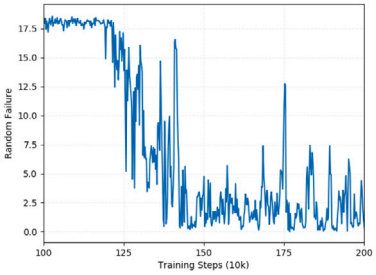
Baselines: OM, OGM, and GM methods cannot condition their choices of action on states of the system. Namely, they can either always conduct level-1 PM or level-2 PM. In addition, OGM and GM have a variable τ that decides when to conduct PM. Since there is no analytical method to derive τ and choice of actions, we determine the best combinations via Monte Carlo simulation: OM, OGM, and GM always take level-1 (imperfect) PM; the optimal τ is 360 for GM and 350 for OGM.



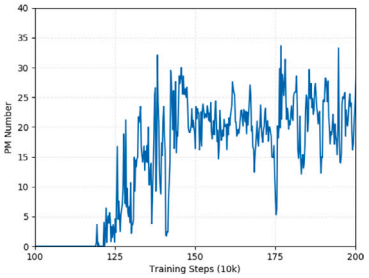
(a) Test return as a function of training step



(b) A closer look at test return



(c) Random failures counts as a function of training steps



(d) PM counts as a function of training steps

Fig. 6. MARL training monitoring.

6.4. Evaluation

6.4.1. Training monitoring

First, we monitor the training of MARL to verify if the algorithm converges. During the course of the training, we periodically halt the training and test the trained model at the current training step for 16 independent episodes. Hence, we can evaluate the model's performance as a function of training steps.

Since the VDAC's objective is to maximize the return $R_t = \sum_t^{T-1} \gamma^{t-1} r_t$, we examine the average of return as a function of training steps. As shown in Fig. 6(a), the average return plateaus until around 1.2 million training steps and then converges to higher values.

Figs. 6(c) and 6(d) depict the average number of random failures and PMs per episode, respectively. They demonstrate that the VDAC algorithm learns to reduce random failure occurrences by applying timely PM at around 1.2 million training steps. Through the course of training after 1.2 million training steps, policies alternate from aggressive PM and passive PM without affecting return values drastically.

With the evidence provided by Fig. 6, we conclude that MARL policy can converge during the 2050000-step training.

We conduct the same analysis on RL training. Fig. 7(a) depicts that the test return oscillates drastically during the course of the training. Fig. 7(c) shows that the RL agent learns to reduce average random failure counts at the end of the training, by excessively conducting PMs. Our further investigation on loss shows that the loss gradually increases during the course of training. Therefore, we conclude RL policy fails to converge for this multi-level PM task. We think that this is due to the fact that the RL agent deals with an action space of size $3^6 = 729$.

6.4.2. Test benchmark

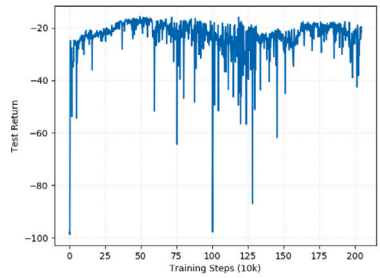
We also benchmark the saved MARL policy with other baseline policies (Note that we also saved the model that achieved the best evaluation profits for Q-learning). During the test, the initial states of machines are randomly generated by the simulation. To account for the randomness resulted from the states initialization, each method is tested for 100 episodes independently and the average performance statistics are reported. We utilize the same random seed to generate the initial health state for every method. Fig. 8 encompasses a set of comprehensive metrics that are used for test evaluation, including profit per episode, throughput numbers per episode, PM counts per episode, random failure per episode, PM cost per episode, and CM cost per episode.

Table 5
Maintenance cost summary.

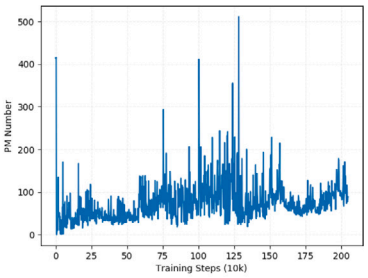
	MARL	DQN	OGM	OM	GM	R2F
PM Cost ($)	2422.21	2920.71	2674.05	2083.19	1771.45	0.00
CM Cost ($)	100.54	468.79	233.00	472.00	690.00	2059.45
Overall Cost ($)	2522.75	3389.5	2907.05	2555.19	2461.45	2059.45

In general, all methods other than R2F are effective policies. This is because they achieved higher average profit over R2F, as shown in Fig. 8(a). Among the effective policies, MARL policy reports the best average profit and throughput per episode. OM policy perform slightly better than OGM in profit although the OM policy reports less throughput than OGM. And GM has the least profit compared with other 3 effective policies (see Table 5).

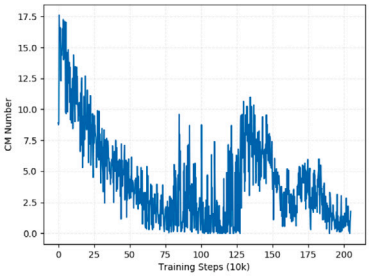
Metrics such as average PM counts and random failures are scrutinized to understand policies. Recall that the goal of this study is to find policies that ensure the smooth operation of the serial production line. Namely, the learned policy should mitigate the stoppage caused by random failures while not committing too many maintenance actions. The R2F policy, which passively performs CM, has the least average maintenance cost per episode of $2095.45. However, it reports the least profit and throughput due to the production loss and CM cost. The OM policy is more aggressive compared with R2F in terms of performing PM. It reduces random failure from R2F's 17.9 to 6.08 per episode. As a result, its profit and throughput improve by 8.68% and 9.11%, respectively, compared with R2F. The variable τ does make OGM and GM more adaptive than R2F as is reflected by the decreasing random failure counts. OGM's PM policy is more aggressive than GM as a result of the opportunistic maintenance performed by OGM. In comparison with OM, OGM conducts 5.7 more PMs per episode and encountered 2.22 less random failures on average. And OGM spends 351.86 more



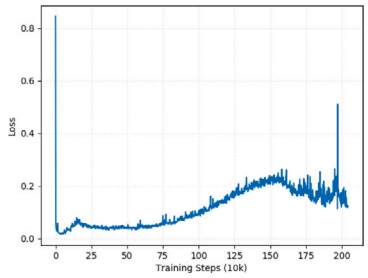
(a) Average test return per episode as a function of training step



(b) Average PM counts per episode as a function of training steps

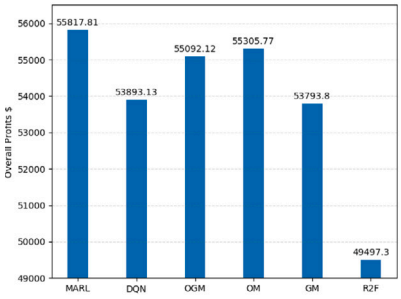


(c) Average Random failures per episode counts as a function of training steps

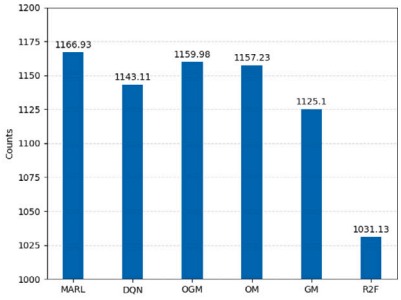


(d) DQN Loss as a function of training steps

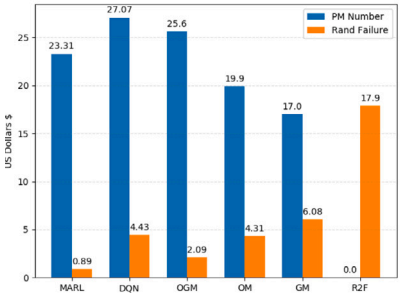
Fig. 7. RL training monitoring.



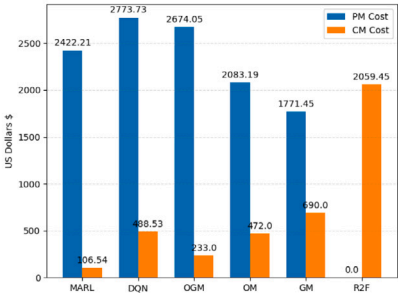
(a) Average profit per episode by methods



(b) Average number of throughputs per episode by methods



(c) Average number of PM and random failure per episode by methods

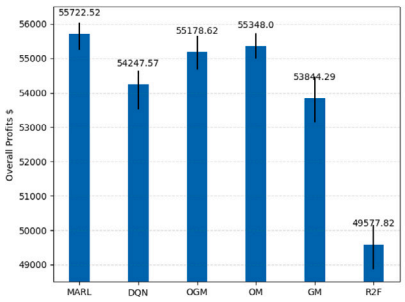


(d) Average cost of PM and CM per episode by methods

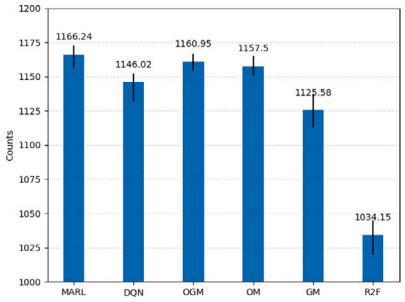
Fig. 8. Evaluation results.

US dollars on maintenance. As a consequence, although OGM produces 2.75 more products per episode than OM, it generates less profit. Compared with OGM, MARL reduces both the number of PMs and random failures. In the comparison with OM, MARL is able to perform 3.41 more PMs while incurring $32.44 less overall cost.

Besides the average stats reported in the above section, the median stats and their 25 to 75 percentile are also recorded. The sensitivity analysis reveals the robustness and generality of the learned policy by methods. We consider profits per episode and throughput per episode as our metrics for sensitivity analysis. Consistent with the previous



(a) Sensitivity analysis on profits per episode by methods



(b) Sensitivity analysis on throughput per episode by methods

Fig. 9. Sensitive analysis for 6-machine-5-buffer study.

Table 6
Throughput sensitivity by methods in 6-machine-5-buffer study.

	MARL	DQN	OGM	OM	GM	R2F
25 Percentile (unit)	1155.35	1131.91	1154.30	1150.72	1112.88	1020.01
Median (unit)	1165.24	1146.02	1160.95	1157.50	1125.58	1034.15
75 Percentile (unit)	1171.67	1152.13	1166.38	1165.28	1137.65	1044.79

Table 7
Profits sensitivity by methods in 6-machine-5-buffer study.

	MARL	DQN	OGM	OM	GM	R2F
25 Percentile ($)	55239.64	53518.54	54671.67	54996.45	53148.29	48871.67
Median ($)	55722.52	54247.57	55178.62	55348.00	53844.29	49577.82
75 Percentile ($)	56041.42	54643.47	55652.40	55729.52	54472.22	50160.57

Table 8
Partial maintenance record.

Time (min)	Failed machines	Machine ages $g_a(t)$	Buffer levels
649	–	[89, 57, 267, 341, 197, 497]	[7, 10, 0, 0, 0]
759	–	[199, 167, 377, 451, 307, 101]	[8, 10, 0, 0, 0]
814	–	[254, 222, 432, 44, 362, 156]	[7, 10, 3, 0, 0]
847	–	[287, 255, 25, 77, 395, 189]	[8, 10, 0, 0, 0]
858	–	[1, 266, 36, 88, 406, 200]	[0, 10, 0, 0, 0]
891	–	[34, 299, 69, 121, 21, 233]	[6, 10, 3, 6, 0]
1144	–	[287, 244, 322, 374, 274, 486]	[8, 10, 0, 0, 0]
1155	–	[1, 255, 333, 385, 285, 497]	[0, 9, 0, 0, 0]
1188	–	[34, 288, 366, 418, 318, 24]	[5, 10, 0, 0, 5]
1221	–	[67, 24, 399, 451, 351, 57]	[7, 9, 0, 0, 0]

Table 9
Partial maintenance record (continue).

Time (min)	Action $\mathbf{u}_t$	PM	GM
649	[0, 0, 0, 0, 0, 2]	Y	
759	[0, 0, 0, 2, 0, 0]	Y	
814	[0, 0, 2, 0, 0, 0]	Y	
847	[2, 0, 0, 0, 0, 0]	Y	
858	[0, 0, 0, 0, 2, 0]	Y	
891	[0, 2, 0, 0, 0, 0]	Y	
1144	[2, 0, 0, 0, 0, 0]	Y	
1155	[0, 0, 0, 0, 0, 2]	Y	
1188	[0, 2, 0, 0, 0, 0]	Y	
1221	[0, 0, 0, 2, 2, 0]	Y	Y

analysis, Tables 6 and 7 depicts that our proposed system achieves the highest median profits and the highest median throughput. Figs. 9(a) and 9(b) visualize Tables 6 and 7, respectively. In Fig. 9, the bar represents the median and the error bar corresponds to the 25 and 75 percentile.

To closely examine the learned MARL policy, we roll out an episode with random state initialization, using the learned MARL policy. Partial record that contains consecutive maintenance is shown in Tables 8 and 9.

At $t = 649$, the MARL policy conducts PM on machine 6 at an age of 497. At $t = 750$, machine 4 conducts PM at an age of 451. At $t = 849$, machine 1 goes into PM at an age of 287. The MARL policy is likely to memorize the likely break-down age for different machines and conduct timely PMs for them. This observation explains why MARL has low random failure and low maintenance cost on average as shown in Fig. 8.

At $t = 1221$, machines 4 and 5 conduct PM simultaneously. Note that machine 5 conducts PM at an age of 406 at $t = 858$ whereas it conducts PM at an age of 351 when $t = 1221$. The PM at $t = 1221$ can be seen as an "group maintenance" as machine 5 accommodates its maintenance schedule to that of machine 4.

Interestingly, the MARL agents favor perfect PM over imperfect PM. This is due to the fact that the cost efficiency (the amount of money spent to reset 1 unit of age) of imperfect PMs is much lower than perfect PMs. In contrast, baselines such OM, GM, and OGM, have trouble capturing the dynamics of the environment and favor imperfect PMs. This is because those methods cannot perform timely PM and because perfect PMs incur higher cost and longer machine down time. In summary, examination of the MARL policy reveals that it is able to schedule PM strategically; that is, the MARK policy conducts group maintenance and captures environment dynamics.

Table 10
Machine parameters for additional machines.

Parameters	M_7	M_8	M_9	M_{10}
Cycle Time T_a (min)	1.20	1.30	1.10	1.05
Level 1 PM cost $c_a^{PM_1}$ ($)	55.0	60.0	45.0	50.0
Level 2 PM cost $c_a^{PM_2}$ ($)	110.0	120.0	90.0	103.0
Level 1 PM duration $d_a^{PM_1}$ (min)	4	5	6	4
Level 2 PM duration $d_a^{PM_2}$ (min)	8	11	12	9
CM cost c_a^{CM} ($)	120.0	130.0	110.0	125.0
CM duration d_a^{CM} (min)	25	32	24	25
Level 1 PM degree $r_a^{PM_1}$	0.8	0.8	0.8	0.8
Level 2 PM degree $r_a^{PM_2}$	0.8	0.8	0.8	0.8
Scale parameter α_a	500	400	390	470
Shape parameter β_a	15	15	15	15

6.5. Additional experiments

To demonstrate the scalability of our proposed framework, we extend the previous serial production line by adding 4 more machines. Parameters for the additional machines and buffers can be found in Tables 10 and 11.

The training process described in the previous section is used here as well. Note that the DQN agent now faces $3^{10} = 59049$ actions whereas the action space of MARL agents is invariant to the number

Table 11
Buffer parameters for additional buffers.

Parameters	B_6	B_7	B_8	B_9
Buffer capability b_i	10	12	9	8

Table 12
Throughput sensitivity by methods in 10-machine-9-buffer study.

	MARL	DQN	OGM	OM	GM	R2F
25 Percentile (unit)	1075.00	–	1086.64	1071.17	1054.46	949.51
Median (unit)	1083.34	–	1093.12	1085.57	1064.26	962.46
75 Percentile (unit)	1092.52	–	1098.00	1096.42	1074.21	975.55

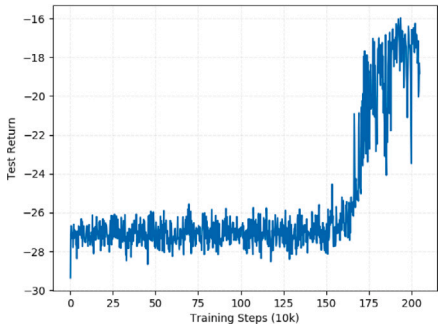


Fig. 10. Average test return per episode as a function of training step.

of agents. Since the memory of DQN is set to 750000 units, each unit stores the action u_t taken by the RL agent and the action mask m_t that keeps track of valid actions, with u_t and m_t being 59049-element vectors. The RAM in our hardware fails to allocate enough memory for DQN's replay buffer. Note that the memory needed for the replay buffer grows drastically with the number of agents or actions, making the DQN impractical to deal with large and complex manufacturing systems. As for other baselines, Monte Carlo simulations are rolled out to find optimal τ and choice of PM action. OM, OGM, and GM still always take level-1 (imperfect) PM; The optimal τ is 280 for GM and 290 for OGM.

Fig. 10 depicts the training of MARL agents in the extended 10-machine-9-buffer serial production line. The test return remains consistent until approximately 1.6 million training steps. Compared to the training process in the previous section, it takes more training steps for the agent to converge to good PM policies. This is due to the fact that the dynamics of the environment is further complicated by the additional 4 machines. Thus, the difficulty of deriving good PM policies also increases.

Consistent with the findings in the previous experiment, all policies except R2F are effective policies and the MARL policy continues to achieve the best average profit. As shown in Figs. 11(c) and 11(a), excessive PM does not necessarily lead to good profit. For instance, the MARL policy reports more random failures than OGM. However, OGM achieves low random failure by excessively conducting PMs. Consequently, the MARL policy spends $1000.68 less in overall maintenance than OGM on average. It is obvious that all baselines except R2F have a tendency to commit over-PM leading to an increase in a machine's downtime and an increase in production loss. This demonstrates that the proposed MARL policies can find the trade-off between production loss and PMs.

The sensitivity analysis shows that the learned MARL-based policy is robust and can be generalized to systems with different initial health states. This is because the MARL-based policy achieves the highest profits as well as demonstrated in Table 13 and Fig. 12.

We also examine the learned policy in the new environment by rolling out an episode with random initialization (see Table 12). Partial

Table 13
Profits sensitivity by methods in 10-machine-9-buffer study.

	MARL	DQN	OGM	OM	GM	R2F
25 Percentile ($)	49385.36	–	49006.38	48836.73	47774.48	43725.61
Median ($)	49815	–	49392.60	49671.07	48270.80	44409.22
75 Percentile ($)	50352.83	–	49633.50	50369.33	48902.99	45035.01

Table 14
Partial maintenance record.

Time (min)	Failed machines	Maint type	Machine ages $g_a(t)$
139	M_1	CM	[340, 130, 468, 128, 254, 130, 174, 84, 116, 75]
143	–	PM	[0, 134, 472, 132, 258, 134, 178, 88, 120, 79]
160	M_2	CM	[0, 151, 489, 149, 6, 151, 195, 105, 137, 96]
297	–	PM	[129, 288, 111, 286, 142, 288, 332, 242, 274, 233]
308	–	PM	[140, 299, 122, 297, 153, 299, 3, 253, 285, 244]
330	–	PM	[162, 321, 144, 319, 175, 321, 25, 275, 10, 266]
385	–	PM	[217, 46, 199, 374, 230, 376, 80, 330, 65, 321]
396	–	PM	[228, 57, 210, 0, 241, 387, 91, 0, 76, 332]
506	–	PM	[338, 167, 320, 110, 351, 101, 201, 110, 186, 442]
600	M_1	CM	[432, 261, 414, 204, 82, 195, 295, 204, 280, 85]
605	–	PM	[0, 266, 419, 209, 87, 200, 300, 209, 285, 90]

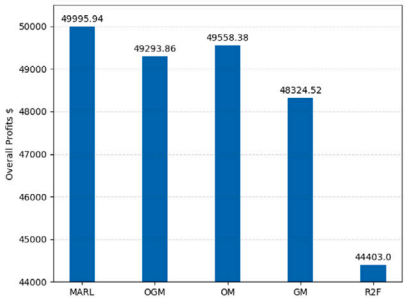
Table 15
Partial maintenance record (continue).

Time (min)	Action $\mathbf{u}_t$	OG	GM
139	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]		
143	[0, 0, 0, 0, 2, 0, 0, 0, 0, 0]	Y	
160	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]		
297	[0, 0, 0, 0, 0, 0, 2, 0, 0, 0]		
308	[0, 0, 0, 0, 0, 0, 0, 0, 2, 0]		
330	[0, 2, 0, 0, 0, 0, 0, 0, 0, 0]		
385	[0, 0, 0, 2, 0, 0, 0, 2, 0, 0]		Y
396	[0, 0, 0, 0, 0, 2, 0, 0, 0, 0]		
506	[0, 0, 0, 0, 2, 0, 0, 0, 0, 2]		Y
600	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]		
605	[0, 0, 0, 0, 0, 0, 0, 0, 2, 0]	Y	

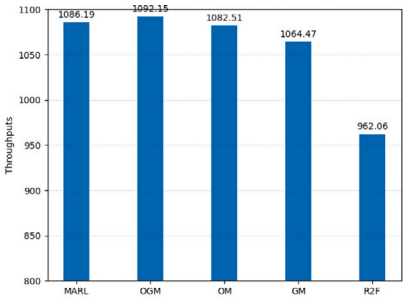
records that contain consecutive maintenance are shown in Tables 14 and 15. Consistent with the policy examination in the previous experiment, the learned policy in the new environment can also perform timely PM and favors perfect PM over imperfect PM. Additionally, we observe increasing occurrences of opportunistic maintenance and group maintenance, i.e. the MARL agent is learning these maintenance strategies through interaction with the system. For instance, machine 1 encounters random failure and goes through CM at time 139. At $t = 143$, machine 5 conducts PM while machine 1 is still under maintenance (CM is longer than PM in general). Machine 5's PM that takes the advantage of unscheduled failure on other machines is said to be opportunistic maintenance. The same incident can be observed at time 605. At $t = 385$ and $t = 506$, group maintenance is conducted, respectively.

7. Conclusion

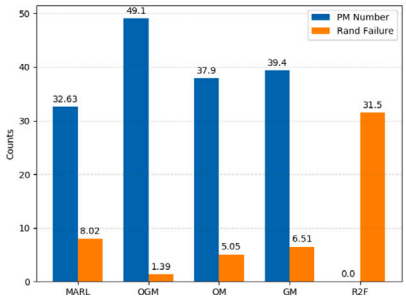
Multi-level PM scheduling in a serial production line is challenging due to the explosion of action space and the non-linear, stochastic nature of the system. We propose to model the PM decision making process in serial production lines as Dec-POMDP and demonstrate how to implement MARL to obtain PM policies. We utilize two numerical experiments to further establish the necessity of modeling the PM decision-making as multi-agent problems instead of as a single-agent problem. In the 6-machine-5-buffer experiment, the DQN policy exhibits convergence issues while the MARL policy achieves the best profit among all baselines. In the 10-machine-9-buffer experiment, the DQN method cannot be implemented efficiently due to the increasing size of its replay buffer and action space. By contrast, MARL does not suffer from this issue and continues to report the best average profit among all policies.



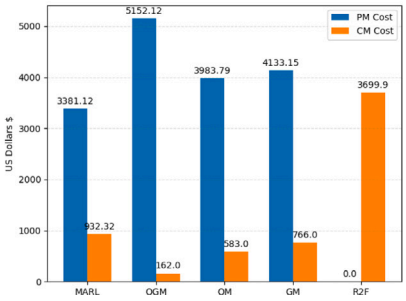
(a) Average profit per episode by methods



(b) Average number of throughputs per episode by methods

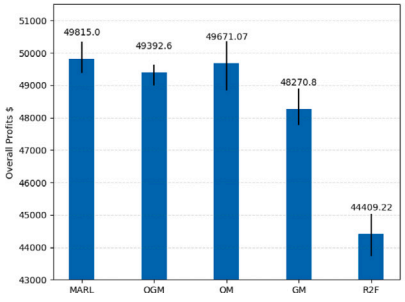


(c) Average number of PM and random failure per episode by methods

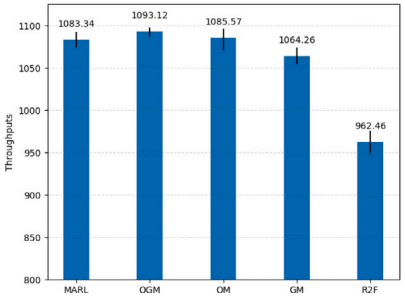


(d) Average cost of PM and CM per episode by methods

Fig. 11. Evaluation results for 10-machine-9-buffer serial production line.



(a) Sensitivity analysis on profits per episode by methods



(b) Sensitivity analysis on throughputs per episode by methods

Fig. 12. Sensitive analysis for 10-machine-9-buffer study.

CRediT authorship contribution statement

Jianyu Su: Conceptualization, Methodology, Software, Formal analysis, Writing – original draft. **Jing Huang:** Validation, Writing – review & editing. **Stephen Adams:** Writing – review & editing. **Qing Chang:** Writing – review & editing. **Peter A. Beling:** Writing – review & editing, Supervision, Funding acquisition.

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.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. CNS: 1650512. This work was conducted in the Center for Visual and Decision Informatics, a National Science Foundation Industry/University Cooperative Research Center.

References

Ab-Samat, H., & Kamaruddin, S. (2014). Opportunistic maintenance (OM) as a new advancement in maintenance approaches: A review. *Journal of Quality in Maintenance Engineering*, *20*(2), 98–121.

AlDurgam, M. M., & Duffuaa, S. O. (2013). Optimal joint maintenance and operation policies to maximise overall systems effectiveness. *International Journal of Productions Research*, *51*(5), 1319–1330.

Amari, S. V., McLaughlin, L., & Pham, H. (2006). Cost-effective condition-based maintenance using Markov decision processes. In *RAMS'06. annual reliability and maintainability symposium, 2006* (pp. 464–469). IEEE.

Arab, A., Ismail, N., & Lee, L. S. (2013). Maintenance scheduling incorporating dynamics of production system and real-time information from workstations. *Journal of Intelligent Manufacturing*, *24*(4), 695–705.

Barros, A., Grall, A., & Bérenguer, C. (2007). Joint modelling and optimization of monitoring and maintenance performance for a two-unit parallel system. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, *221*(1), 1–11.

Bu, L., Babu, R., De Schutter, B., et al. (2008). A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *38*(2), 156–172.

Byon, E., & Ding, Y. (2010). Season-dependent condition-based maintenance for a wind turbine using a partially observed Markov decision process. *IEEE Transactions on Power Systems*, *25*(4), 1823–1834.

Byon, E., Ntaimo, L., & Ding, Y. (2010). Optimal maintenance strategies for wind turbine systems under stochastic weather conditions. *IEEE Transactions on Reliability*, *59*(2), 393–404.

Chan, G., & Asgarpoor, S. (2006). Optimum maintenance policy with Markov processes. *Electric Power Systems Research*, *76*(6–7), 452–456.

Chan, J.-K., & Shaw, L. (1993). Modeling repairable systems with failure rates that depend on age and maintenance. *IEEE Transactions on Reliability*, *42*(4), 566–571.

Chekired, D. A., Khoukhi, L., & Mouftah, H. T. (2018). Industrial IoT data scheduling based on hierarchical fog computing: A key for enabling smart factory. *IEEE Transactions on Industrial Informatics*, *14*(10), 4590–4602.

Chen, Y., Liu, Y., & Xiahou, T. (2021). A deep reinforcement learning approach to dynamic loading strategy of repairable multistate systems. *IEEE Transactions on Reliability*.

Chitra, T. (2003). Life based maintenance policy for minimum cost. In *Annual reliability and maintainability symposium, 2003* (pp. 470–474). IEEE.

Choo, B. Y., Adams, S., & Beling, P. (2017). Health-aware hierarchical control for smart manufacturing using reinforcement learning. In *2017 IEEE international conference on prognostics and health management* (pp. 40–47). IEEE.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

de Smidt-Destombes, K. S., van der Heijden, M. C., & van Harten, A. (2009). Joint optimisation of spare part inventory, maintenance frequency and repair capacity for k-out-of-N systems. *International Journal of Production Economics*, *118*(1), 260–268.

Ebrahimipour, V., Najjarbashi, A., & Sheikhalishahi, M. (2015). Multi-objective modeling for preventive maintenance scheduling in a multiple production line. *Journal of Intelligent Manufacturing*, *26*(1), 111–122.

Fitouhi, M.-C., Nourelfath, M., & Gershwin, S. B. (2017). Performance evaluation of a two-machine line with a finite buffer and condition-based maintenance. *Reliability Engineering & System Safety*, *166*, 61–72.

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., & Whiteson, S. (2018). Counterfactual multi-agent policy gradients. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1), URL: https://ojs.aaai.org/index.php/AAAI/article/view/11794.

Gershwin, S. B., Dallery, Y., Papadopoulos, C. T., & Smith, J. M. (2012). *Analysis and modeling of manufacturing systems, vol. 60*. Springer Science & Business Media.

Hausknecht, M., & Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. In *2015 AAAI fall symposium series*.

Huang, J., Chang, Q., & Arinez, J. (2020). Deep reinforcement learning based preventive maintenance policy for serial production lines. *Expert Systems with Applications*, *160*, Article 113701.

Huang, J., Chang, Q., Arinez, J., & Xiao, G. (2019). A maintenance and energy saving joint control scheme for sustainable manufacturing systems. *Procedia CIRP*, *80*, 263–268.

Huang, J., Chang, Q., & Chakraborty, N. (2019). Machine preventive replacement policy for serial production lines based on reinforcement learning. In *2019 IEEE 15th international conference on automation science and engineering* (pp. 523–528). IEEE.

Huang, J., Chang, Q., Zou, J., & Arinez, J. (2018). A real-time maintenance policy for multi-stage manufacturing systems considering imperfect maintenance effects. *IEEE Access*, *6*, 62174–62183.

Karamatsoukis, C., & Kyriakidis, E. (2010). Optimal maintenance of two stochastically deteriorating machines with an intermediate buffer. *European Journal of Operational Research*, *207*(1), 297–308.

Kijima, M. (1989). Some results for repairable systems with general repair. *Journal of Applied Probability*, 89–102.

Kim, D.-Y., Kim, S., Hassan, H., & Park, J. H. (2017). Adaptive data rate control in low power wide area networks for long range IoT services. *Journal of Computer Science*, *22*, 171–178.

Koshimae, H., Dohi, T., Kaio, N., & Osaki, S. (1996). Graphical/statistical approach to repair limit replacement problem. *Journal of the Operations Research Society of Japan*, *39*(2), 230–246.

Laggoune, R., Chateauneuf, A., & Aissani, D. (2009). Opportunistic policy for optimal preventive maintenance of a multi-component system in continuous operating units. *Computers & Chemical Engineering*, *33*(9), 1499–1510.

Liu, Y., Chen, Y., & Jiang, T. (2020). Dynamic selective maintenance optimization for multi-state systems over a finite horizon: A deep reinforcement learning approach. *European Journal of Operational Research*, *283*(1), 166–181.

Love, C., & Guo, R. (1996). Utilizing Weibull failure rates in repair limit analysis for equipment replacement/preventive maintenance decisions. *Journal of the Operational Research Society*, *47*(11), 1366–1376.

Malik, M. A. K. (1979). Reliable preventive maintenance scheduling. *AIIE Transactions*, *11*(3), 221–228.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., et al. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (pp. 1928–1937).

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529.

Monga, A., Zuo, M. J., & Toogood, R. W. (1997). Reliability-based design of systems considering preventive maintenance and minimal repair. *International Journal of Reliability, Quality and Safety Engineering*, *4*(01), 55–71.

Nicolai, R. P., & Dekker, R. (2008). Optimal maintenance of multi-component systems: a review. In *Complex system maintenance handbook* (pp. 263–286). Springer.

Pham, H., & Wang, H. (1996). Imperfect maintenance. *European Journal of Operational Research*, *94*(3), 425–438.

Ramírez-Hernández, J. A., & Fernandez, E. (2010). Optimization of preventive maintenance scheduling in semiconductor manufacturing models using a simulation-based approximate dynamic programming approach. In *49th IEEE conference on decision and control* (pp. 3944–3949). IEEE.

Rashid, T., Samvelyan, M., de Witt, C. S., Farquhar, G., Foerster, J., & Whiteson, S. (2018). QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *ICML 2018: Proceedings of the thirty-fifth international conference on machine learning*. URL: http://www.cs.ox.ac.uk/people/shimon.whiteson/pubs/rashidicml18.pdf.

Robelin, C.-A., & Madanat, S. M. (2007). History-dependent bridge deck maintenance and replacement optimization with Markov decision processes. *Journal of Infrastructure Systems*, *13*(3), 195–201.

Samvelyan, M., Rashid, T., Schroeder de Witt, C., Farquhar, G., Nardelli, N., Rudner, T. G., et al. (2019). The starcraft multi-agent challenge. In *Proceedings of the 18th international conference on autonomous agents and multiagent systems* (pp. 2186–2188). International Foundation for Autonomous Agents and Multiagent Systems.

Shafiee, M., & Finkelstein, M. (2015). An optimal age-based group maintenance policy for multi-unit degrading systems. *Reliability Engineering & System Safety*, *134*, 230–238.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, *550*(7676), 354–359.

Son, K., Kim, D., Kang, W. J., Hostallero, D. E., & Yi, Y. (2019). Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International conference on machine learning* (pp. 5887–5896). PMLR.

Su, J., Adams, S., & Beling, P. A. (2020). Counterfactual multi-agent reinforcement learning with graph convolution communication. arXiv preprint arXiv:2004.00470.

Su, J., Adams, S., & Beling, P. (2021). Value-decomposition multi-agent actor-critics. *Proceedings of the AAAI Conference on Artificial Intelligence*, *35*(13), 11352–11360, URL: https://ojs.aaai.org/index.php/AAAI/article/view/17353.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., et al. (2017). Value-decomposition networks for cooperative multi-agent learning. arXiv preprint arXiv:1706.05296.

Tan, B., & Gershwin, S. B. (2011). Modelling and analysis of Markovian continuous flow systems with a finite buffer. *Annals of Operations Research*, *182*(1), 5–30.

Tomasevicz, C. L., & Asgarpoor, S. (2006). Optimum maintenance policy using semi-Markov decision processes. In *2006 38th North American power symposium* (pp. 23–28). IEEE.

Wang, H. (2002). A survey of maintenance policies of deteriorating systems. *European Journal of Operational Research*, *139*(3), 469–489.

Wang, H., & Pham, H. (1999). Some maintenance models and availability withimperfect maintenance in production systems. *Annals of Operations Research*, *91*, 305–318.

Wang, X., Wang, H., & Qi, C. (2016). Multi-agent reinforcement learning based maintenance policy for a resource constrained flow line system. *Journal of Intelligent Manufacturing*, *27*(2), 325–333.

Wiering, M. A. (2000). Multi-agent reinforcement learning for traffic light control. In *Machine learning: Proceedings of the seventeenth international conference* (pp. 1151–1158).

Wolpert, D. H., & Tumer, K. (2002). Optimal payoff functions for members of collectives. In *Modeling complexity in economic and social systems* (pp. 355–369). World Scientific.

Xia, T., Jin, X., Xi, L., & Ni, J. (2015). Production-driven opportunistic maintenance for batch production based on MAM–APB scheduling. *European Journal of Operational Research*, *240*(3), 781–790.

Yousefi, N., Tsianikas, S., & Coit, D. W. (2020). Reinforcement learning for dynamic condition-based maintenance of a system with individually repairable components. *Quality Engineering*, *32*(3), 388–408.

Zheng, X., & Fard, N. (1991). A maintenance policy for repairable systems based on opportunistic failure-rate tolerance. *IEEE Transactions on Reliability*, *40*(2), 237–244.

Zou, J., Chang, Q., Arinez, J., Xiao, G., & Lei, Y. (2017). Dynamic production system diagnosis and prognosis using model-based data-driven method. *Expert Systems with Applications*, *80*, 200–209.

Zou, J., Chang, Q., Lei, Y., & Arinez, J. (2016). Production system performance identification using sensor data. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *48*(2), 255–264.