Expert Systems with Applications 160 (2020) 113701

Contents lists available at ScienceDirect

Expert Systems with Applications

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

Deep reinforcement learning based preventive maintenance policy for serial production lines

Jing Huang [a], Qing Chang [a,b,*], Jorge Arinez [c]

[a] Department of Mechanical and Aerospace Engineering, University of Virginia, Charlottesville, VA 22904, USA
[b] Department of Engineering Systems and Environment, University of Virginia, Charlottesville, VA 22904, USA
[c] General Motors Research and Development Center, Warren, MI 48090, USA

ARTICLE INFO

Article history:
Available online 30 June 2020

Keywords:
Preventive maintenance
Production loss
Deep reinforcement learning
Serial production line
Group maintenance
Opportunistic maintenance

ABSTRACT

In the manufacturing industry, the preventive maintenance (PM) is a common practice to reduce random machine failures by replacing/repairing the aged machines or parts. The decision on when and where the preventive maintenance needs to be carried out is nontrivial due to the complex and stochastic nature of a serial production line with intermediate buffers. In order to improve the cost efficiency of the serial production lines, a deep reinforcement learning based approach is proposed to obtain PM policy. A novel modeling method for the serial production line is adopted during the learning process. A reward function is proposed based on the system production loss evaluation. The algorithm based on the Double Deep Q-Network is applied to learn the PM policy. Using the simulation study, the learning algorithm is proved effective in delivering PM policy that leads to an increased throughput and reduced cost. Interestingly, the learned policy is found to frequently conduct "group maintenance" and "opportunistic maintenance", although their concepts and rules are not provided during the learning process. This finding further demonstrates that the problem formulation, the proposed algorithm and the reward function setting in this paper are effective.

1. Introduction

The maintenance activity affects the economics of a manufacturing system in several ways. About 15% to 40% of total production cost is attributed to maintenance activities in a factory (Wang, 2012). Therefore, a good maintenance policy is instrumental to ensure a smooth and efficient production operation but is nontrivial due to the complex nonlinear and stochastic nature of a manufacturing system.

A maintenance action reacting to a machine random failure is referred to as corrective maintenance (CM). The consequences of machine random failures are often unpredictable, and even catastrophic in some situations (Wang, 2002). To reduce such random failures, the preventive maintenance (PM) is much needed, which proactively maintains a machine even it is still operational in order to keep machines in a desired reliability level. However, there is a trade-off in the PM decision making. If PMs were not performed in a timely manner, the system would be interrupted by random failures more frequently, which might lead to significant production losses. On the other hand, if PMs were too frequent, the costs caused by PMs might far outweigh the benefits that PMs could bring due to the fact that some PM actions might be unnecessary.

For years, scheduled and other "preventive" maintenance strategies have been the norm – achieving maintenance objectives through regular equipment inspections and scheduled maintenance at pre-determined intervals based on operational time, cycles, units, etc. However, this "fixed" PM policy may ignore a machine's real degradation and its impact on system level throughput loss. In this paper, an approach based on deep reinforcement learning (DRL) is developed to obtain cost-effective PM policies for serial production lines. A serial production line, which is constructed with multiple machines that are interconnected by intermediate buffers, exhibits very complex system dynamics (Li, E. Blumenfeld, Huang, & M. Alden, 2009). The challenges of deriving the optimal PM policy for the serial production line are three folds.

First, it is important to correctly evaluate the production losses caused by maintenance activities under a PM policy. The immediate outcome of a maintenance action, either CM or PM, is that the production on the specific machine is interrupted until the maintenance is completed. In both cases, the machine stoppages

* Corresponding author at: Department of Mechanical and Aerospace Engineering, University of Virginia, Charlottesville, VA 22904, USA..
E-mail addresses: jh3ex@virginia.edu (J. Huang), qc9nq@virginia.edu (Q. Chang), jorge.arinez@gm.com (J. Arinez).

https://doi.org/10.1016/j.eswa.2020.113701
0957-4174/

propagate in the line and might finally lead to system-level production losses. Therefore, it is crucial to evaluate the production losses caused by maintenance activities, such that the production losses incurred by the PM can be weighed against those incurred by potential random failures when deriving the PM policy. The traditional modeling methods (Li et al., 2009; Zhao & Li, 2015) for production systems based on simplified Markov Chain are only capable of approximately calculating long-term system performance indicators and thus cannot serve this purpose. Therefore, a novel production system modeling and system production loss evaluation method (Liu, Chang, Xiao, & Biller, 2012; Zou, Chang, Lei, & Arinez, 2018) need to be leveraged to establish a novel technique for deriving the optimal PM policy.

Second, the complexity of a production system leads to an extremely large state space of the maintenance problem. In the industrial practice, PM schedules are usually planned solely based on individual machine's ages recommended by machine vendors, which tend to be conservative and largely ignore the intricate interactions among machines in the serial production line. As the information of the production systems have been increasingly transparent and detailed, a PM policy would be preferable if it has fully incorporated all necessary machine-level and system-level information. However, it inevitably leads to a problem with an enormous state space, which is intractable with traditional model-based planning methods. It is promising to embrace new tools and methodologies emerging in Artificial Intelligence (AI) and machine learning (ML) areas to develop intelligent decision-making support systems for production and maintenance management.

Finally, however, desultory deployments of AI and ML techniques in the production system could possibly lead to very unsatisfactory system performance. For example, with abundant real-time data available in the plant floor, it is not advisable to lump all the data into AI programs, where the data interferences are not clear. This could be inefficient or even likely to generate poor PM decisions. Hence, it is critical to systematically formulate the problem with thorough understanding of the properties of the manufacturing system. The domain knowledge of serial production lines should be well utilized to guide the formulation and solving of the PM problem.

This paper is devoted to address the above challenges. The main contributions of this paper are in: 1) formulating the PM decision-making in serial production lines as a DRL problem in the Markov Decision Process framework; 2) proposing a reasonable reward function based on an efficient system modeling and the system production loss evaluation method; 3) implementing a DRL algorithm to solve the formulated problem effectively and efficiently.

The remainder of the paper is organized as the following: literature review is provided in Section 2. Section 3 describes the problem objective and assumptions. Section 4 introduces the production system modeling method and derives the system production loss evaluation based on the model. The maintenance problem is formulated as a DRL problem and solved by the DDQN algorithm in Section 5. Numerical case study and conclusions are provided in Section 6 and Section 7 respectively.

2. Literature review

The machine maintenance is an intricate matter as it relates to many other aspects of modern industrial practices (Huang, Chang, Arinez, & Xiao, 2019). The maintenance policy aims in improving system reliability, preventing the occurrence of system failures, and reducing maintenance costs (Wang, 2002). It is shaped by the specific application scenario and the characteristics of the target system. Regarding the structure of the system of interest, the maintenance policies can be categorized into single-unit policies and multi-unit policies. The single-unit policies are designated for those standalone systems and they have been extensively investigated by Wang (Wang, 2002). Some examples are age-dependent policies (Ross, 2014) and periodic PM policies etc. Since the single-unit system operates independently, the relationship, either deterministic (Ross, 2014) or stochastic (Ye, 1990), between the maintenance decision and the overall maintenance cost is usually known. Therefore, the single-unit maintenance could often be modeled as a stochastic process, in which optimal PM decision variables can be obtained by minimizing the maintenance cost rate, or, equivalently, maximizing the machine availability in some circumstances. For the serial production line, the maximum machine availability does not guarantee an optimal maintenance cost. The system production loss caused by a maintenance action is conditioning on the buffer states (Liu et al., 2012; Zou, Chang, Arinez, Xiao, & Lei, 2017), for which in general we cannot derive the probability distribution (Li et al., 2009).

Unlike in the single-unit system, the components within the multi-unit system have structural or operational dependencies on each other. Maintenance policies have been developed based on the specific structures of the systems, including serial systems (Ebrahimipour, Najjarbashi, & Sheikhalishahi, 2013), the parallel systems (Barros, Grall, & Berenguer, 2007) and k-out-of-n systems (de Smidt-Destombes, van der Heijden, & van Harten, 2009) etc. The "group maintenance" and "opportunistic maintenance" are the building blocks for most of the existing maintenance policies for the close-interconnected serial systems. The group maintenance policy (Nicolai & Dekker, 2008; Shafiee & Finkelstein, 2015) conducts multiple maintenance actions simultaneously to merge and reduce the production losses, while the opportunistic maintenance policy (Ab-Samat & Kamaruddin, 2014; Laggoune, Chateauneuf, & Aissani, 2009; Xia, Jin, Xi, & Ni, 2015) identifies the time window, in which the inserted PM will not incur extra production losses. They are inspired by the observation that when one machine is under maintenance, the others can receive maintenance at the same time without incurring extra production loss. However, it does not hold in a general serial production line because the buffers among machines could delay the propagation of the machine stoppage from the maintained machine to its adjacent machines (Liu et al., 2012; Zou et al., 2018).

It turns out that neither traditional single-unit policies nor multi-unit policies could be directly applicable to the serial production lines. Therefore, considerable research efforts have been devoted to deriving feasible maintenance policies for the serial production lines. There are several works (Fitouhi, Nourelfath, & Gershwin, 2017; Karamatsoukis & Kyriakidis, 2010; X. Wang, Wang, & Qi, 2016) that are aimed to derive maintenance policy for two-machine-one-buffer serial production lines. Fitouhi et al. (Fitouhi et al., 2017) presented a Markov Chain based method to evaluate the system performance under different PM policies, however, the policy considered in that work failed to incorporate the system dynamics since PM actions were determined based on only two variables, i.e. degradation states of the two machines. In the contrast, Karamatsoukis et al. (Karamatsoukis & Kyriakidis, 2010) included the buffer levels in the state definition when they tried to obtain PM policies using Markov Decision Process (MDP). Wang et al. (Wang et al., 2016) derived the PM policy based on semi-MDP for a two-machine-one-buffer production line considering quality inspections. Machine degradation states are assumed to be directly related to product quality performance and non-conforming parts would be scrapped immediately. Although these works found feasible PM policies under different assumptions, the approaches proposed in (Karamatsoukis & Kyriakidis, 2010; Wang et al., 2016) lack scalability and cannot be extended to the more general cases with more machines and buffers.

For longer serial production lines, Arab et al. (Arab, Ismail, & Lee, 2013) searched for the optimal maintenance schedule using genetic algorithm in order to maximize the throughput. Ramirez-Hernandez et al. (Ramírez-Hernández & Fernandez, 2010) used approximate dynamic programming (ADP) to optimize the maintenance schedule in a five-machine production line. However, both works simplifies the maintenance problem as inserting known maintenance tasks, in the form of downtime events, into the production shifts, and assumed that the maintenance schedules would not impact the machine reliability status at all. Kang et al. (Kang & Ju, 2019) proposed an aggregation-based approximation method for obtaining maintenance policies for the synchronous production line, i.e. the cycle time for each machine is identical. But real production lines are usually not perfectly balanced, so that it is important to consider the different machine processing speeds when optimizing PM policies (Fitouhi et al., 2017). In (Huang, Chang, Zou, & Arinez, 2018), a CM policy considering imperfect maintenance effects was proposed for the serial production line, but the PM was not included in the work. Therefore, a systematic approach to deriving PM policies for general serial production lines must be developed to address the above challenges.

Regarding the mathematical techniques used in obtaining maintenance policies, quite a few methods have been applied in literatures, including renewal process (Ross, 2014), Markov Chain (Fitouhi et al., 2017), heuristic methods (Arab et al., 2013), and MDP (Fitouhi et al., 2017; Kang & Ju, 2019; Karamatsoukis & Kyriakidis, 2010; Ramírez-Hernández & Fernandez, 2010; X. Wang et al., 2016) etc. It is noted that MDP is a particularly common modeling method for maintenance problem in complex systems including serial production lines, since maintenance is often a sequential decision-making problem with multi-dimensional states and actions. However, it is important to realize that the performance of MDP-based maintenance policies can vary tremendously depending on the problem formulation and solving techniques.

On the one hand, the problem formulation refers to properly defining the three components, namely state, action, and reward, according to the problem characteristics and objective. It requires thorough understanding of system dynamics in serial production lines. For example, if some key variables are not included in the state definition, the PM decisions would fail to reflect the real system dynamics. In (Fitouhi et al., 2017), the buffer level is not considered when making PM decisions, hence the PM decision might be the identical no matter the buffer is full or empty. But one should also strive not to include redundant variables, especially in today's manufacturing systems, which usually have huge amounts of data from various sources. In this paper, the PM problem is also formulated as an MDP but with the guidance of our previously derived systematic knowledge of the serial production lines.

On the other hand, the solution to an MDP is an optimal policy that gives the best action for each state, such that the expected accumulated reward is maximized. There have been a lot of techniques that can effectively solve the MDP, and some of them have been applied to maintenance problems. Dynamic Programming (DP) is an exact and model-based approach to solving MDP, where model refers to the complete transition probabilities among states. Therefore, in the context of PM problem in serial production lines, DP is only applicable to two-machine-one-buffer line (Karamatsoukis & Kyriakidis, 2010), or needs cruel approximations when applied to longer lines (Kang & Ju, 2019). In the contrast, Reinforcement Learning (RL) is a category of techniques obtaining the optimal policy for MDP through the interactions between agents and the uncertain environment (Sutton & Barto, 2018). Most of the RL algorithms is model-free, i.e. the state transition probabilities are not required. Therefore, the model-free RL algorithms well suit the PM problem in general production lines, for which the system state space explodes exponentially with increased machine numbers. Instead of the transition probabilities, a reliable simulator, or experiment if feasible, that faithfully reflects the uncertain environment needs to be set up for the implementation of RL algorithms. Regarding the serial production line, the general-purpose commercial software, e.g. Simul8 and Simulink etc., have long been used for its simulation. However, the simulation setup is often arduous, and the efficiency and accuracy are not guaranteed. In (Zou et al., 2018), a data-driven model for production lines is established based on dynamic system and conservation of the flow. The model is derived analytically, and therefore it is not only accurate but also has high computation efficiency compared to general-purpose simulation software. In this paper, we will leverage it to simulate the production system dynamics under PM for training process of the RL agent.

The selection of the RL algorithms is also a crucial question. According to how the policy is represented, RL algorithms can be categorized into policy-based methods, value-based methods, and actor-critic (Sutton & Barto, 2018). In RL, policy is a function mapping from state to action. The policy-based methods seek to directly parameterize the policy. Simple parameterizations could be, for example, linear combination of polynomial features or basis functions. The performance of policy-based methods heavily depends on how the features or basis functions are constructed. For problems with high dimension and inherent complexity, simple parameterizations are not sufficient due to their limitations on representation power. In contrast to policy-based methods, value-based methods represent the policy implicitly with state-values or state-action-values, where 'value' is the expected accumulated reward starting from a given state or taking a given action. The naïve Q-learning is one of the most widely used value-based methods in researches on PM problems because of its simplicity and robustness. Wang et al. (Wang et al., 2016) presented the application of naïve Q-learning to PM problem in two-machine-one-buffer line. However, to some extent, naïve Q-learning also suffers from the "curse of dimensionality" mainly because the naïve Q-learning uses a table to record the Q-value for all state-action pairs. The problem with large state spaces is not just the memory needed for large tables, but the time and data needed to fill them accurately (Sutton & Barto, 2018). The actor-critic method adopts both policy parameterization and value function in its algorithm, and therefore suffers from the drawbacks of both. In conclusion, despite the fact that these primitive RL algorithms are robust and accessible, the lack of scalability is preventing them from being applied to solve a range of real-world problems with large state space like the PM problem discussed in this paper. To this end, in recent years, the emergence of deep learning allows the RL to go 'deep' as well and results in a series of DRL algorithms. The DRL scales RL to interesting decision-making problems in practice that were previously intractable (Arulkumaran, Deisenroth, Brundage, & Bharath, 2017). Some of the DRL applications in recent years include, for example, Atari video games (van Hasselt, Guez, & Silver, 2016; Mnih et al., 2015) and the game of Go (Silver et al., 2016) etc. Thanks to its good scalability and efficiency, the DRL shows great potential to solve complex problems in the manufacturing industry that are unsolvable with conventional techniques. The DDQN/DQN algorithm is one of the state-of-the-art DRL algorithms that was originally proposed to play Atari video games (van Hasselt et al., 2016; Mnih et al., 2015). The algorithm has been proved to be effective in solving some practical problems other than the Atari video games (Martinez, Ramasso, Perrin, & Rombaut, 2020; Wei, Bao, & Li, 2020). In this paper, the DDQN algorithm will be applied to solving the PM problem in the serial production lines.

3. Problem description

3.1. System structure and assumptions

A serial production line with M machines and $M-1$ buffers is as shown in Fig. 1. The machines S_i, $i = 1, 2, \cdots, M$, are represented with rectangles. The intermediate buffers B_i, $i = 2, 3, \cdots, M$, are represented with circles. The arrows specify the direction of the material flow in the system.

The following assumptions are made in this paper:

1) Each buffer B_i has a finite capacity. With the abuse of notation, the maximum capacity of buffer B_i is also denoted as B_i;
2) The serial production line is a dynamic system. The buffer levels are changing with the system dynamics. We denote the buffer level of buffer B_i at time t as $b_i(t)$, where $0 \le b_i(t) \le B_i$;
3) The cycle time of machine S_i is denoted as T_i;
4) The lifetime of machine S_i follows a known distribution $p_i(*)$, which can usually be obtained by experiments or from vendors. The age of machine S_i at time t is denoted as $g_i(t)$;
5) When machine S_i fails, it stops operation and the maintenance staff must react to the failure and conduct CM. The duration of a CM on machine S_i is d_i^{CM}. The maintenance effects of CM are *perfect*, i.e. after the CM is completed on machine S_i, the age of machine S_i starts from zero;
6) To reduce machine random failures, an aged machine can be turned off and receive a PM. The duration of a PM on machine S_i is d_i^{PM}. The maintenance effects of PM are also *perfect*, i.e. after the PM is completed on machine S_i, the age of machine S_i starts from zero;
7) The maintenance durations d_i^{PM} and d_i^{CM} include not only the time performing CM or PM but also the response time and preparation time needed before the maintenance starts;
8) Machine S_i resumes production only when the CM or PM is completed. The remaining maintenance duration at time t on machine S_i is denoted as $d_i^r(t)$. For example, if a CM on machine S_2 was started 12 minutes ago, and the it was supposed to take 30 minutes, i.e. $d_2^{CM} = 30$ min, then the remaining maintenance duration is 18 minutes, i.e. $d_2^r(t) = 18$ min;
9) Both CM and PM would incur some fixed resource costs, including costs of new parts and all other consumable expenses. The resource costs of a CM and a PM on machine S_i are c_i^{CM} and c_i^{PM} respectively;
10) The profit per part is c_p.

The notations used in this paper are as the following:

- M is the total number of machines in the line
- M^* is the index of the slowest machine in the line, i.e. $M^* = \arg\max_i \{T_i, i = 1, 2, \cdots, M\}$
- T_i cycle time of machine S_i
- $b_i(t)$ is the buffer level of B_i at time t
- $w_i(t)$ is the indicator of random failures, i.e. $w_i(t) = 1$ if machine S_i fails at time t
- $p_i(x)$ is the lifetime distribution of machine S_i
- $g_i(t)$ is the age of machine S_i at time t
- d_i^{CM} is the duration of a CM on machine S_i
- $d_i^{PM}$is the duration of a PM on machine S_i
- $d_i^r(t)$ is the remaining maintenance duration on S_i at time t
- c_i^{CM} is the resource cost of a CM on machine S_i
- c_i^{PM} is the resource cost of a PM on machine S_i
- $PL(t)$ is the system production loss of the serial production line at time t
- c_p is the profit per part of the production line
- $a_i(t)$ is the PM decision variable at time t, i.e. conduct PM on machine S_i if $a_i(t) = 1$
- $A(s_t)$ is the legal action set for state s_t
- π is the PM policy
- s_t is MDP state at time t
- r_t is MDP reward at time t
- γ denotes discount factor in RL
- ϵ is the parameter for ϵ-greedy exploration in RL
- $C(t;\pi)$ is maintenance cost up to time t given PM policy π
- $N_i^{PM}(t)$ is the total PM counts on machine S_i up to time t
- $N_i^{CM}(t)$ is the total CM counts on machine S_i up to time t
- $\boldsymbol{\theta}$ denotes the parameters for neural network
- $Q(s,a)$ denotes the state-action value, also known as Q-value
- $Q(s,a;\boldsymbol{\theta})$ denotes Q-value approximated by neural network $\boldsymbol{\theta}$

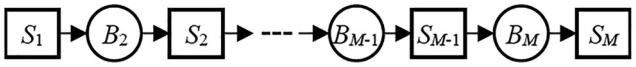


Fig. 1. The structure of a serial production line.

3.2. Problem statement

An intermediate part finished by machine S_i flows to its downstream buffer B_{i+1} if buffer B_{i+1} is not full, and otherwise machine S_i is said to be blocked. Machine S_i starts a new cycle by receiving one part from its upstream buffer B_i if buffer B_i is not empty, and otherwise machine S_i is said to be starved. The blockage or starvation makes an operational machine to stand idle. If one machine is undergoing maintenance, its downstream buffers gradually drain, and upstream buffers fill up due to the machine stoppage and thus causing blockage or starvation in its adjacent operational machines. The stoppage and idleness of these machines might finally lead to the system-level production loss. The production loss due to the maintenance activities accounts for a significant portion of the overall maintenance related costs. We denote the system production loss caused by the maintenance activities as PL.

Let π denotes the PM policy for a serial production line. The PM policy π instructs when and which machine should be turned off and receive a PM. Let $C(t;\pi)$ denotes all the costs caused by the maintenance activities up to time t under the PM policy π, then

$$C(t;\pi) = c_p \cdot \int_0^t PL(t)dt + \sum_{i=1}^{M} c_i^{PM} N_i^{PM}(t) + \sum_{i=1}^{M} c_i^{CM} N_i^{CM}(t) \tag{1}$$

where c_p is the profit per part and $\int_0^t PL(t)dt$ is the accumulative system production loss up to time t. $N_i^{PM}(t)$ and $N_i^{CM}(t)$ are the total numbers of PM and CM conducted on machine S_i up to time t respectively.

$$N_i^{PM}(t) = \int_0^t a_i(\tau)d\tau \tag{2}$$

$$N_i^{CM}(t) = \int_0^t w_i(\tau)d\tau \tag{3}$$

An optimal PM policy π^* should minimize the long-run maintenance cost rate, which is maintenance cost per unit time. Therefore, the objective of the maintenance problem in this paper can be represented as:

$$\pi^* = \arg\min_{\pi} \left\{ \lim_{t\to\infty} \frac{C(t;\pi)}{t} \right\} \tag{4}$$

With this objective function, the problem to be studied in this paper can be stated as: *Under assumptions 1 to 10, develop a method to find the optimal PM policy π^* for the serial production line, such that the long-run maintenance cost rate is minimized.*

Before we proceed to solve the problem, a proper modeling method for the serial production line must first be established and the system production loss, i.e. $PL(t)$, should also be evaluated such that the impacts of the maintenance activities could be identified.

4. System modeling and production loss evaluation

Our previous work (Zou et al., 2018) proposed a data-driven modeling method for serial production lines, which can efficiently evaluate real-time dynamic behavior of the production system. In addition, based on the proposed model, the system production loss can be derived. To keep this paper self-contained, the main conclusions in the modeling methodology are briefly introduced in this section.

4.1. Serial production line modeling

In this model, the serial production line is modeled by the state space equations:

$$\dot{\boldsymbol{b}}(t) = \boldsymbol{F}(\boldsymbol{b}(t), \boldsymbol{U}(t), \boldsymbol{W}(t)) \tag{5}$$

$$\boldsymbol{Y}(t) = \boldsymbol{H}(\boldsymbol{b}(t)) \tag{6}$$

The buffer levels $\boldsymbol{b}(t)$, where $\boldsymbol{b}(t) = [b_2(t), \cdots, b_M(t)]'$, is treated as the system state. The physical meanings of other variables are as the following:

(1) $\boldsymbol{F}(*) = [f_2(*), \cdots, f_M(*)]$ is the dynamic function of the system state $\boldsymbol{b}(t)$;

(2) $\boldsymbol{U}(t) = [u_1(t), \cdots, u_M(t)]'$ is the control input, which describes whether there is an ongoing maintenance activity, either PM or CM, at each machine at time t, where

$$u_i(t) = \begin{cases} 0, & \text{if machine } S_i \text{ is under PM/CM} \\ 1, & \text{if machine } S_i \text{ is operating} \end{cases} \tag{7}$$

(3) $\boldsymbol{W}(t) = [w_1(t), \cdots, w_M(t)]'$ is used to indicate whether there is a random failure on each machine at time t, where

$$w_i(t) = \begin{cases} 1, & \text{if machine } S_i \text{ fails at time } t \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

(4) $\boldsymbol{Y}(t) = [Y_1(t), \cdots, Y_M(t)]'$ is the accumulated production counts of each machine up to time t, and $\boldsymbol{H}(*) = [H_1(*), H_2(*), \cdots, H_M(*)]$ are measurement functions, which map the system states to the system throughput.

The dynamic function $\boldsymbol{F}(*)$ and the measurement function $\boldsymbol{H}(*)$ have been analytically derived based on the conservation of the flow. Given the control input $\boldsymbol{U}(t)$ and random disturbance $\boldsymbol{W}(t)$, the system states $\boldsymbol{b}(t)$ and throughputs $\boldsymbol{Y}(t)$ can be quickly calculated. One may refer to (Zou et al., 2018) for detailed derivations and rigorous proof. It has been demonstrated that the presented model is effective and has great computation efficiency.

It is noteworthy that the DRL algorithm trains its policy with large amounts of stepwise transition samples from the target environment. The data-driven introduced in this section is used to construct such an environment. The DRL agent would make the PM decisions, which are corresponding to the control input $\boldsymbol{U}(t)$ in the model. In order to simulate the target environment, machine random failures would also be generated according to the given machine reliability model. The random failures are corresponding to the disturbances $\boldsymbol{W}(t)$ in our model. Based on the state space equation in Eqs. (5) and (6), the system states and outputs can be efficiently computed. Therefore, by using the presented modeling method, one can computationally efficiently acquire reliable transition samples for the DRL training process.

4.2. System production loss evaluation

For simple serial systems without intermediate buffers, any machine stoppage would immediately count towards the system production loss. However, with the existence of intermediate buffers, the relationship between system production loss and machine stoppage duration is not trivial. It has been proved in (Liu et al., 2012) that *in a serial production line a maintenance action causes system production loss if and only if the slowest machine is impeded,* i.e. *blocked or starved, by the stoppage*. Let S_{M^*} denotes the slowest machine in a serial production line, where

$$M^* = \arg\max_i \{T_i, i = 1, 2, \cdots, M\} \tag{9}$$

If we discretize the time into steps, and let $D(t)$ be the idleness time of the slowest machine S_{M^*} during the time step t then the system production loss $PL(t)$ during the time step t is

$$PL(t) = \frac{D(t)}{T_{M^*}} \tag{10}$$

where T_{M^*} is the cycle time of the slowest machine S_{M^*}. Note that the idleness time $D(t)$ on machine S_{M^*} includes not only the duration of maintenance activities on machine S_{M^*} itself, but also, more importantly, the starvation and blockage time of machine S_{M^*} caused by the maintenances on other machines. One may refer to reference (Liu et al., 2012) for details in derivation and rigorous proof for system production loss.

In the stepwise simulation scenario, with the system outputs given by Eq. (6), the system production loss $PL(t)$ can also be conveniently calculated as

$$PL(t) = 1/T_{M^*} - (Y_{M^*}(t) - Y_{M^*}(t-1)) \tag{11}$$

where $1/T_{M^*}$ is the ideal production increment of the slowest machine S_{M^*} without any disturbance, and the second term $Y_{M^*}(t) - Y_{M^*}(t-1)$ is the actual incremental production counts of the slowest machine S_{M^*}. The difference between them is the system production loss during the time step.

5. Obtaining PM policy through DRL

For serial production lines, the state space of the PM problem stated in Eqs. (1)–(4) explodes with the increasing length of the line. Considering the large state space, it is very difficult to obtain the optimal PM policy by using exact approaches such as DP. Alternatively, RL is a proper method to tackle this problem.

Firstly, most of the RL algorithms are model-free, which means that the system transition probability is not required. Secondly, RL algorithms usually train its policy through sampling transitions in the state space and action space with either experiments or simulations. The data-driven modeling method described in previous section offers us great efficiency in the simulation of serial production lines. Finally, RL is particularly well-suited to the sort of problems that bear a trade-off between long-term and short-term reward. In our case, the PM is increasing the short-term cost, however, it avoids the long-term cost by reducing the probability of potential machine failures.

Despite its great potential as a suitable and effective approach to the PM problem, two key challenges are: 1) how to formally fit PM problem into the RL framework, and 2) how to solve it with an efficient and scalable RL algorithm. In this section, the PM problem will first be formulated as an MDP, which is the general

framework for RL. A state-of-the-art DRL algorithm, i.e. DDQN, will be applied to solve the MDP and obtain PM policy afterwards.

5.1. MDP formulation for PM problem

The most common framework for RL is MDP, which is a stochastic process that models the sequential decision making in uncertain environments. There are three components in an MDP, including state s_t, action a_t and reward function r_t. As an extension of Markov Chain, MDP also describes the stochastic state transition, however, the transition is driven partially by the environment uncertainties and partially by the actions. Fig. 2 illustrates two steps of state transition in a simple MDP. Note that the MDP is a framework mostly based on discretized time. In the following, we will follow this convention to use notation t to represent discrete time step.

Based on the state s_t at time step t, an action a_t would be selected according to some rule and implemented in the environment. In the context of PM problems, this is to determine whether or not we conduct PM actions on machines. The rule governing the action selection is referred to as a policy, denoted as $\pi(a|s)$.

$$\pi(a|s) = \Pr(a_t = a|s_t = s) \tag{12}$$

Subsequently, a scalar reward r_t will be observed, which reflects the goodness of the action a_t in state s_t. The accumulated reward is referred to as return, denoted as G_t.

$$G_t = r_t + \sum_{k=1}^{\infty} \gamma^k r_{t+k} \tag{13}$$

where γ is a discount factor, which is used to make a trade-off between immediate reward and future rewards. If the immediate reward is preferred, a small γ will be used and vice versa. An optimal policy π^* is supposed to maximize the expected return, i.e.

$$\pi^* = \arg\max_{\pi} \{\mathbb{E}_{\pi}[G_t]|s = s_t\} \tag{14}$$

Before we can apply RL algorithms to obtaining the ultimate PM policy π^*, we need to first properly define the three key components, i.e. state s_t, action a_t, and reward function r_t, in the MDP that models the PM problem in serial production lines.

5.1.1. State definition

Given the state s_t, one should be able to fully comprehend the production system status such that an action that suits the status can be determined. Three factors are essential for the PM decision making in the serial production line, namely

1) The machine ages $g_i(t), i = 1, 2, \cdots, M$, specify the probability of random failures on each machine;
2) The buffer levels $b_i(t), i = 2, 3, \cdots, M$, denote the status of the production line, which directly relate to the system production losses caused by PM actions and random failures;
3) The remaining maintenance duration $d_i^r(t)$, $i = 1, 2, \cdots, M$, indicate all the ongoing maintenance activities on each machine.

Consequently, the state s_t is defined as:

$$s_t = [g_1(t), \cdots, g_M(t), b_2(t), \cdots, b_M(t), d_1^r(t), \cdots, d_M^r(t)] \tag{15}$$

Remark 1:. *It is important not to confuse MDP state s_t with the system state $\boldsymbol{b}(t)$ of the serial production line. System state $\boldsymbol{b}(t)$ is raised when modeling the production system dynamics and evaluate the production losses $PL(t)$ given PM decisions and random failures. However, the state s_t is defined in order to formulate the PM problem as an MDP. The state s_t assembles all the machine-level and system-level information, including $\boldsymbol{b}(t)$, that are essential to make the PM decisions.*

5.1.2. Action definition

In contrast with the PM, the CM is passively triggered by a random failure, which is beyond the control of the agent. Therefore, the action a_t is limited to the PM decisions. The action a_t is a vector consisting of M binary variables indicating whether we turn off machines for PM or not at time t.

$$a_t = [a_1(t), a_2(t), \cdots, a_M(t)] \tag{16}$$

where

$$a_i(t) = \begin{cases} 0, & \text{leave machine } S_i \text{ as it is} \\ 1, & \text{turn off machine } S_i \text{ for PM} \end{cases} \tag{17}$$

Note that $d_i^r(t) > 0$ implies that there is an ongoing maintenance on machine S_i, and thus machine S_i cannot receive a PM under such circumstance. The action a_t is said to be illegal if it intends to assign a PM on machine S_i given $d_i^r(t) > 0$. At each time step, it is only allowed to select an action from the legal action sets $A(s_t)$.

$$A(s_t) = \{a_t | \, \forall i, \; a_i(t) = 0, \text{ if } d_i^r(t) > 0\} \tag{18}$$

5.1.3. Reward function definition

The overall maintenance cost includes the resource cost of all PMs and CMs, and the system production loss caused by those maintenances. Therefore, the reward function r_t is defined as:

$$r_t = -c_p \cdot PL(t) - \sum_{i=1}^{M} w_i(t) c_i^{CM} - \sum_{i=1}^{M} a_i(t) c_i^{PM} \tag{19}$$

where $\sum_{i=1}^{M} w_i(t) c_i^{CM}$ and $\sum_{i=1}^{M} a_i(t) c_i^{PM}$ are the resource costs at time step t incurred by CMs and PMs respectively, and $c_p \cdot PL(t)$ is the profit loss caused by the system production loss $PL(t)$ during the time step. The reward is negative because we seek to maximize the accumulated reward, and equivalently the overall maintenance cost can be minimized.

It is noted that the reward function r_t is consistent with the problem objective function presented in Eqs. (1)–(4) in Section III. The reward function is the stepwise version of the objective function, such that the RL could deliver the policy that meets our ultimate objective.

5.2. Applying DDQN to obtain PM policy

In order to obtain the optimal policy π^*, a lot of algorithms have been proposed in the past, among which the Q-learning is one of

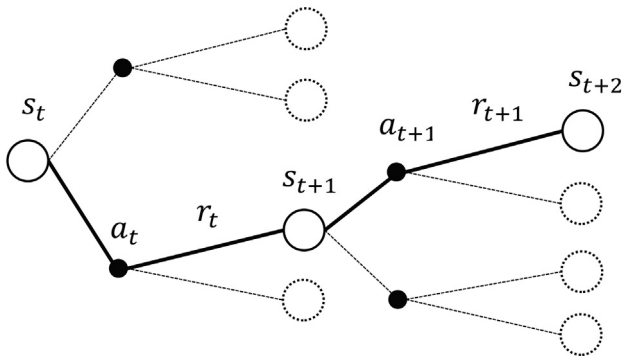


Fig. 2. An illustrative example of state transitions in MDP.

the most widely used algorithms (Sutton & Barto, 2018). It possesses good efficiency and robustness, and therefore it has also been applied to solve maintenance problems in a two-machine-one-buffer production line (X. Wang et al., 2016). One may refer to (Sutton & Barto, 2018) for details of the naïve Q-learning algorithm.

Naïve Q-learning Algorithm
Input: ϵ, γ
Output: $Q(s,a)$
Initialize $Q(s,a)$ table arbitrarily
Initialize s_0 randomly
For $t = 0, 1, \cdots, K$ **do**
Choose a_t from using policy derived from $Q(s,a)$ (e.g. ϵ-greedy)
Take action a_t, observe r_t, s_{t+1}
$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_t + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t)]$

The naïve Q-learning algorithm utilizes a large table to record the state-action values, which is also called Q-values. Q-values are the expected returns if action a is taken in state s and the same policy is followed afterwards.

$$Q(a,s) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a] \tag{20}$$

The naïve Q-learning has a lack of scalability and cannot be applied to problems with large state spaces. The problem with large state spaces is not just the memory needed for large tables, but the time and data needed to fill them accurately (Sutton & Barto, 2018). In recent years, the emergence of DRL algorithms have well addressed the scalability issue in Q-learning. The Double Deep Q-Network (DDQN) (van Hasselt et al., 2016) is one of the state-of-the-art algorithms in DRL. It is a stable and scalable approach to complex and ultra-high-dimensional RL problems, such as Atari video games (van Hasselt, Guez, & Silver, 2015), in which the DDQN largely outperforms both human players and its precedent, Deep Q-Network (DQN) (Mnih et al., 2015). The policy in Q-learning (including naïve Q-learning, DQN and DDQN etc.) is implicitly embedded in the state-action values, which is also called Q-values, which are the expected returns if action a is taken in state s and the same policy is followed afterwards. In DDQN, the Q-values are approximated with a neural network $\boldsymbol{\theta}$, i.e.

$$Q(s,a;\boldsymbol{\theta}) \approx Q(a,s) \tag{21}$$

Instead of filling a large table, the DDQN seeks to iteratively update the neural network parameters $\boldsymbol{\theta}$, which could well approximate the Q-values, until the ultimate policy $\pi^*(a|s)$ is obtained. Similar to the naïve Q-learning, the ultimate policy given by the neural network $\boldsymbol{\theta}$ is a deterministic policy.

$$\pi^*(a|s) = \begin{cases} 1, & \text{if } a = \arg\max_{a' \in A(s)} \{Q(s,a';\boldsymbol{\theta})\} \\ 0, & \text{otherwise} \end{cases} \tag{22}$$

The PM problem for general serial production line also has a huge state space. For instance, for a four-machine-three-buffer line, suppose the machine age ranges are [0, 500], buffer capacities are 10 parts, and maintenance duration ranges are [0, 30], the state space size is approximately 5×10^{19}. In this context, the DRL algorithms such as DDQN are well suited to problems with such large state space. In this section, we will use DDQN to solve the PM problem MDP defined by Eqs. (15)–(19). The full algorithm is as the following.

Algorithm 1
Input: $\boldsymbol{M}, \boldsymbol{T_i}, \boldsymbol{B_i}, \boldsymbol{p_i}(\boldsymbol{t}), \boldsymbol{c_i^{CM}}, \boldsymbol{c_i^{PM}}, \boldsymbol{c_p}, \boldsymbol{d_i^{CM}}, \boldsymbol{d_i^{PM}}$
Output: $\boldsymbol{\theta}$
Initialize replay memory $\mathscr{D}$ to capacity N_{mem}
Randomly initialize the neural network $\boldsymbol{\theta}$
For $t = 0, 1, \cdots, K$ **do**
Every C_1 steps, initialize s_t randomly and normalize it to $\bar{s}_t$
Every C_2 steps, set $\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}$
Find legal action list $A(s_t)$ according to Eq. (22)
Draw a random number $\xi \sim$ Uniform$(0,1)$
If $\xi > \epsilon$ **then**
Select $a_t = \arg\max_{a \in A(s_t)} Q(\bar{s}_t, a; \boldsymbol{\theta})$
Else
Select a random action $a_t \in A(s_t)$
End if
Input action a_t to the production line
Run the production line for one step per Eqs. (5) and (6)
Observe $w_i(t), g_i(t), d_i^r(t)$
Calculate production loss $PL(t)$ using Eq. (11)
Calculate r_t according to Eq. (23) and normalize it to $\bar{r}_t$
Observe next state s_{t+1} and normalize it to $\bar{s}_{t+1}$
Store transition sample $(\bar{s}_t, a_t, \bar{r}_t, \bar{s}_{t+1})$ in replay memory $\mathscr{D}$
Sample a minibatch of size b of transitions $(\bar{s}_j, a_j, \bar{r}_j, \bar{s}_{j+1})$ from $\mathscr{D}$
Set $y_j = \bar{r}_j + \gamma Q\left(\bar{s}_{j+1}, \arg\max_{a \in A(s_{j+1})} Q(\bar{s}_{j+1}, a; \boldsymbol{\theta}); \boldsymbol{\theta}^-\right)$
Perform a gradient descent step on $\left(y_j - Q(\bar{s}_j, a_j; \boldsymbol{\theta})\right)^2$
End for

The flow chart for **Algorithm 1** is as shown in Fig. 3. There are two key techniques that are inherited from the original DDQN (van Hasselt et al., 2016), including experience replay and target network. The experience replay is used to stabilize the learning process by storing past experiences, i.e. one-step transitions (s_t, a_t, r_t, s_{t+1}), in a replay memory $\mathscr{D}$, and sampling mini-batches from $\mathscr{D}$ to train the neural network $\boldsymbol{\theta}$ afterwards (van Hasselt et al., 2016). While the target network is used to generate targets when updating the neural network with gradient descent. The target network $\boldsymbol{\theta}^-$ is a fixed and delayed replica of the current network $\boldsymbol{\theta}$, and the use of target network is intended to overcome the over-optimism issue. One may refer to references (van Hasselt et al., 2016; Mnih et al., 2015) for details regarding the original DDQN algorithm.

Note that this paper is not aimed to develop new DRL algorithms, rather the contribution is to apply a state-of-the-art technique to complex engineering problems. However, since the DDQN was originally designed to play video games, some *knowledge-guided adjustments* are much needed when applying it to obtaining the PM policy. These adjustments are discussed and clarified in the following:

(1) Neural network and its architecture

In Atari video games, the observations are raw images and the true states are hidden. Therefore, the convolutional neural network (CNN) is adopted in the original DDQN algorithm. However, in the context of the PM problem, the state consists of buffer levels, machine ages and maintenance activities, which can all be directly observed or tracked. As shown in Fig. 4, the neural network used in this paper consists of several fully connected layers, whose depth and heights should be determined by the dimension of the specific PM problem. The inputs are the system state s, and the outputs are the Q-values for all actions in state s.

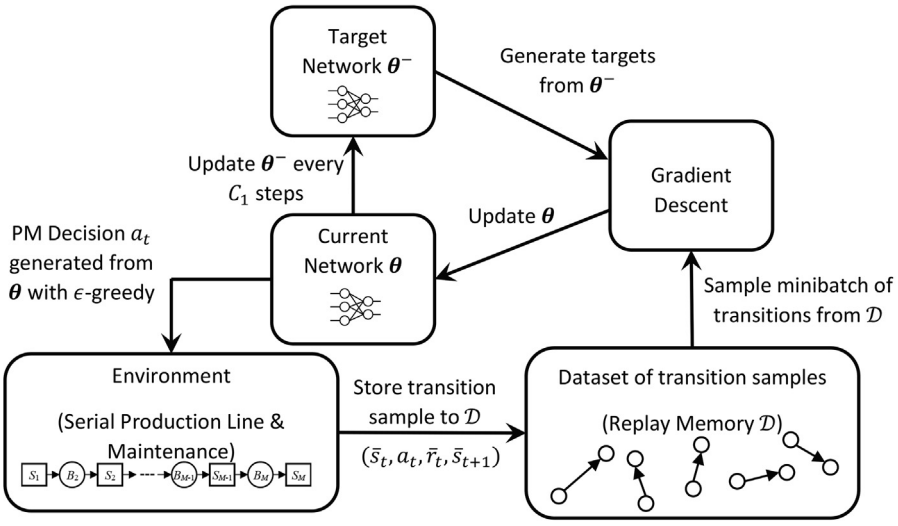


Fig. 3. The flow chart of Algorithm 1.

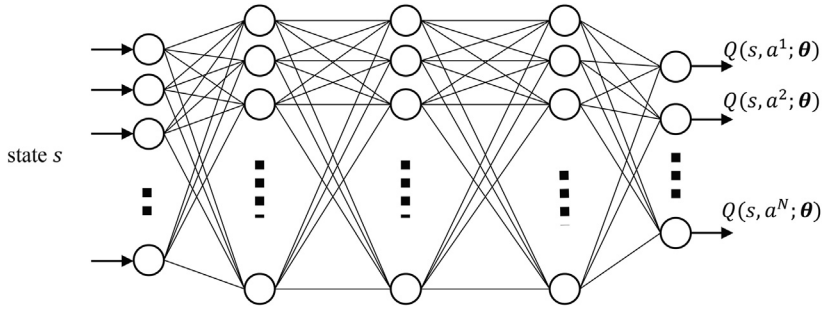


Fig. 4. The achitecture of the Q-network.

(2) Replay memory initialization

In DDQN, before the training iteration begins, we need to initiate the memory buffer $\mathscr{D}$ with experiences obtained from a random policy. Considering that the age is the most important factor in determining the PM, we generate a random policy based on age thresholds. It is very likely that the machine is turned off for PM before its expected age. We construct a distribution $p_i'(x)$, which is similar to but has a slightly smaller expectation than the machine lifetime distribution $p_i(x)$. Every time machine S_i comes back in operation, we generate a random number from $p_i'(x)$ and use it as the next PM age threshold for machine S_i.

(3) Inputs and outputs normalization

The normalization is an operation to ensure that all inputs and outputs have a similar scale to ease the implementation of the training process of the neural network. For the state s_t as inputs, an intuitive normalization method is

$$\bar{s}_t = \left\{ \frac{g_1(t)}{\mathbb{E}[X_1]}, \cdots, \frac{g_M(t)}{\mathbb{E}[X_M]}, \frac{b_2(t)}{B_2}, \cdots, \frac{b_M(t)}{B_M}, \frac{d_1^r(t)}{d_1^{CM}}, \cdots, \frac{d_M^r(t)}{d_M^{CM}} \right\} \tag{23}$$

where $\mathbb{E}[X_i] = \int_0^\infty x p_i(x) dx$ is the expected lifetime of machine S_i, $i = 1, 2, \cdots, M$. Thus, we normalize the elements of state s_t to an approximate range of $[0, 1]$. Regarding the outputs, the Q-values and their ranges are not readily available before the training starts. It is recommended to run a few experiments and obtain the approximate range of the Q-values to decide the proper normalization scale applied to r_t.

5.3. PM policy implementation

In practice, the implementation of DDQN-based PM policy will follow a two-stage scheme, including offline training and online decision making. First of all, the parameters regarding the production line and maintenances should be collected. These parameters are listed as the inputs to the Algorithm 1. In addition, there are a few hyperparameters that we need to define before the learning begins, which are listed in Table 1 for quick reference.

Given all the parameters and hyper-parameters, **Algorithm 1**, along with the production line simulator, will be launched offline and outputs the neural network parameters $\boldsymbol{\theta}$ through iterative training. The neural network $\boldsymbol{\theta}$ conveys the ultimate PM policy

Table 1
Hyperparameters in Algorithm 1.

Notation	Meaning
L	Number of hidden layers in the neural network
h_l	Number of hidden units in layer l, $l = 1, \cdots, L$
K	Training iterations
γ	Future reward discount factor
ϵ	Parameter for ϵ-greedy
C_1	Step interval that the state is randomly generated
C_2	Step interval that the function parameter is held for $\boldsymbol{\theta}^-$
b	Size of the minibatch
N_{mem}	Size of the experience memory

for the specific production line. In the time of implementing, the PM decision making is conducted online following **Procedure 1**.

Procedure 1
Input: real-time states
$s_t = [g_1(t), \cdots, g_M(t), b_2(t), \cdots, b_M(t), d_1^r(t), \cdots, d_M^r(t)]$
Output: PM decision a_t
Normalize s_t to $\bar{s}_t$ according to same scale in Algorithm 1
Find legal action list $A(s_t)$ according to Eq. (22)
Run a forward propagation in neural network θ to get $Q(\bar{s}_t, a; \theta)$
Find the optimal action as $a_t = \arg\max_{a \in A(s_t)} Q(\bar{s}_t, a; \theta)$
Output a_t

The PM decision is always made based on the real-time status of the production line. The buffer states $b_i(t)$ are collected by the distributed sensors in the production system. The machine ages $g_i(t)$ remaining maintenance duration $d_i^r(t)$ can be tracked through the real-time production and maintenance management systems. Therefore, the PM policy proposed in this paper is also a condition-based PM policy. The training process (**Algorithm 1**) is computationally expensive since it requires the training of the neural network, however it is conducted totally offline and could be accomplished with abundant computing resources on edge or cloud. The decision making (**Procedure 1**) contains only one forward propagation, therefore it should be fast in the real-time control of the production system.

6. Numerical case study

6.1. Parameters setting and training process

In order to validate the proposed method, we conduct the following numerical experiment and in-depth analysis based on a 6-machine-5-buffer serial production line. Note that the experiment is not conducted on the real plant floor. Nonetheless, the production system parameters used in this case study are provided by an industrial collaborator. The parameters are listed in Table 2. The machine lifetime follows Weibull distribution with two parameters, i.e. scale parameter α_i and shape parameter β_i (Dodson, 2006). The slowest machine is machine S_3, i.e. $M^* = 3$. The profit per part is assumed to be c_p= $50.

Given the system parameters, we implement Algorithm 1 to train the PM policy. The neural network has two fully connected hidden layers, and each layer has 64 hidden units. Since the machine number is six, the size of the input layer is 17 and that of output layer is 2^6, i.e. 64. The size of the experience memory is N_{mem} = 500,000. The minibatch size is $b = 32$. The step intervals for the state randomization and neural network duplication are $C_1 = 10,000$ and $C_2 = 20,000$, which means that every 10,000 steps the state s_t will be replaced by a random state and every 20,000 steps a target network θ^- will be copied from the current network θ.

The future reward discount factor is set to be γ = 0.95. The parameter for ϵ-greedy is initially set to be 0.8, and linearly reduced to 0.1 when the iteration reaches 300,000 steps and fixed to 0.1 afterwards. Regarding the gradient descent, the optimizer used in this case is RMSprop (Tieleman & Hinton, 2012), in which three parameters are η = 0.00025, $\epsilon' = 0.01$, and ρ = 0.95. We implement the proposed algorithm using TensorFlow with 4 GPUs and 4 CPUs. The total training iterations are set to be 2,000,000. The total computation time is 25.42 h.

To monitor the training progress, every 10,000 steps we run the production system with the PM policy defined by the latest neural network parameters θ for 100,000 min and observe the average rewards per minute, which is as shown in Fig. 5(a). Although the training rewards are noisy before one million steps, but the underlying trend is that the rewards are increasing with training steps. Since the rewards are in different scales before and after one million steps, the rewards after one million steps are replotted in a suitable scale in Fig. 5(b). It can be observed that the agent is still making steady progress throughout the training process despite some noises.

6.2. Evaluation of the learned policy

To evaluate the performance of the learned policy, three other scenarios are considered, including the run-to-failure scenario, the age-dependent PM policy, and opportunistic PM policy.

1) **Run-to-failure scenario**

In this scenario, no PM decision would be made, and the machines keep running until failure. The run-to-failure scenario serves as a baseline. Comparison with the run-to-failure policy would interpret in what scale the PM policy improves the system performance.

2) **Age-dependent PM policy scenario**

In this policy, we follow the traditional norm - one machine will receive a PM whenever its age exceeds a predetermined age threshold. The age-dependent policy is the most widely used PM policy in the real industry. The optimal age threshold is derived to minimize the cost rate of each individual machine according to Ross (Ross, 2014). Details regarding the optimal age thresholds for each machine are included in the appendix part of this paper.

3) **Opportunistic PM policy scenario**

In the opportunistic PM policy, there exists a pre-determined PM interval τ. Two conditions would trigger PM actions. If the no

Table 2
Machine and buffer parameters.

Parameters	Machines					
	S_1	S_2	S_3	S_4	S_5	S_6
Cycle time T_i (min)	1.00	0.90	1.20	1.05	1.10	1.05
PM cost c_i^{PM} ($)	105.0	98.0	110.0	120.0	90.0	103.0
CM cost c_i^{CM} ($)	110.0	100.0	120.0	130.0	110.0	125.0
PM duration d_i^{pm} (min)	10	9	8	11	12	9
CM duration d_i^{cm} (min)	28	31	25	32	24	25
Scale parameter α_i	400	430	500	580	550	575
Shape paramter β_i	2	2	2	2	2	2
	Buffers					
	B_2	B_3	B_4	B_5	B_6	
Buffer capacity B_i	8	10	12	6	8	

machine fails before the interval arrives, then all machine will receive PM at interval τ. In the contrast, if there is any machine fails before the PM interval arrives, then the failed machine has to receive an CM and other machines would receive opportunistic PM. In this study, we adopt the opportunistic PM policy in Laggoune et al. (Laggoune et al., 2009) and its proposed method for searching optimal τ, which is based on Monte Carlo simulation. Given the parameters in this case study, the optimal τ is found to be 363 min.

Ten sets of initial states are randomly generated for the evaluation of each policy, in order to ensure that the performance of PM policies is independent of its initial states. With each initial state, the production line is simulated with the four scenarios independently for one week, i.e. 10,800 min. Each simulation is repeated for 50 times. The overall maintenance cost per minute, which is also called the maintenance cost rate, is calculated from the simulation results. Table 3 presents the average maintenance cost rates and their 95% confidence intervals for three scenarios starting from ten randomly generated initial state sets.

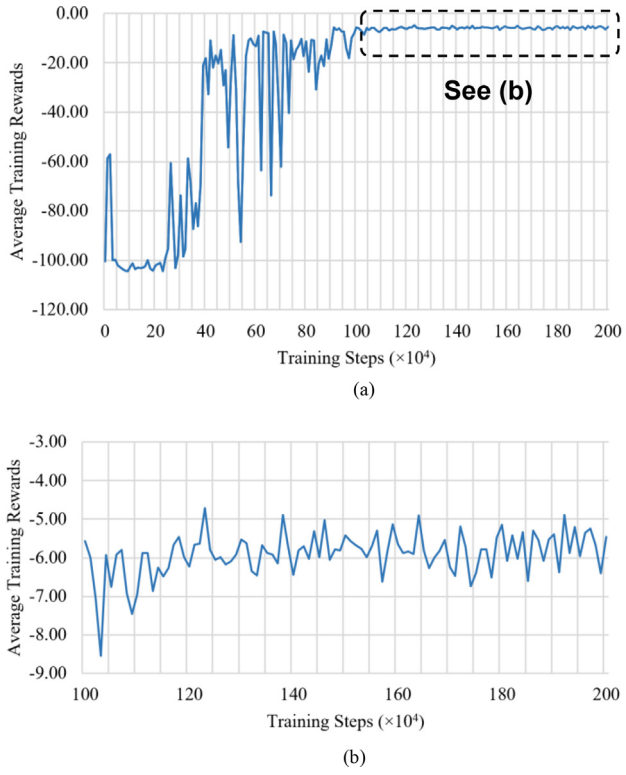


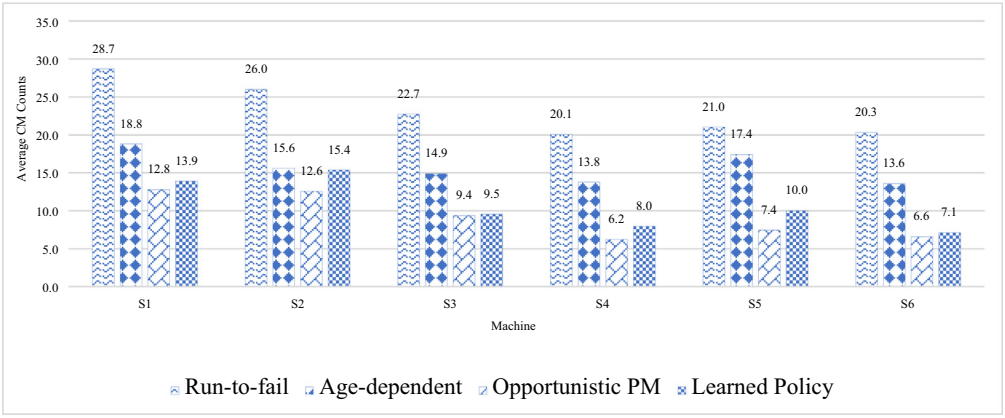
Fig. 5. (a) Average training rewards monitoring; (b) Average training rewards after one million steps.

According to the experiment results, all the three policies are effective PM policies, because they all improve the system performance from the run-to-failure scenario. The opportunistic policy performs slightly better than the age-dependent policy. Despite the optimality of the age-dependent policy, its optimality is limited to the single-machine scenario. While the decision variable τ in the opportunistic policy is searched through large amounts of simulations in serial production line, and therefore it could possibly capture part of the dynamics of the serial production line. However, given that the opportunistic policy has only one decision variable, the PM would be conducted whenever there are random failures or the interval arrives, and therefore it is hard to comprehend the real dynamics in the production system, such as buffer level fluctuations. To sum up, the learned policy outperforms both the age-dependent policy and opportunistic policy in all the ten initial states sets. On average, the learned policy reduces the overall maintenance cost rate by 8.77% and 6.25% comparing to the age-dependent policy and opportunistic policy respectively. Table 3 also lists the 95% confidence interval for the average maintenance cost rates under difference PM policies.

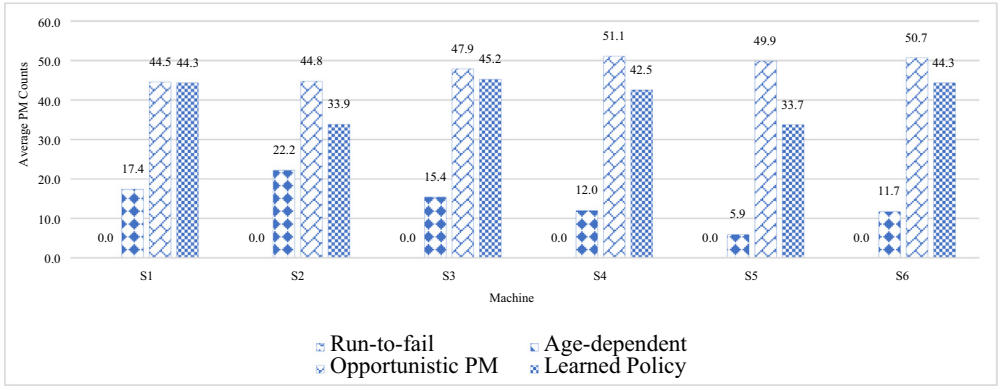
Furthermore, we compare the average numbers of random failures and PM conducted in all four scenarios starting from initial state 1. Fig. 6(a) is the comparison of the numbers of random failures on each machine. The three PM policies both reduce the random failures from the baseline by conducting PMs. From Fig. 6(b), one can tell that both opportunistic policy and learned policy tend to conduct PMs more frequently that age-dependent policy, which accordingly results in much less random failures. It implies that the age-dependent policy is more conservative, since the impact of intermediate buffers is not considered in its derivation. The latter two policies would find more opportunities for conducting PM. The opportunistic policy is even more aggressive than the learned policy regarding the PM decisions, which also results in more maintenance resource costs according to Fig. 6(c). In Fig. 6(d), it shows that opportunistic policy leads to less final system throughputs, which might be attributed to the interruptions introduced by the extra PMs.

Table 3
Average maintenance cost rates and 95% confidence intervals.

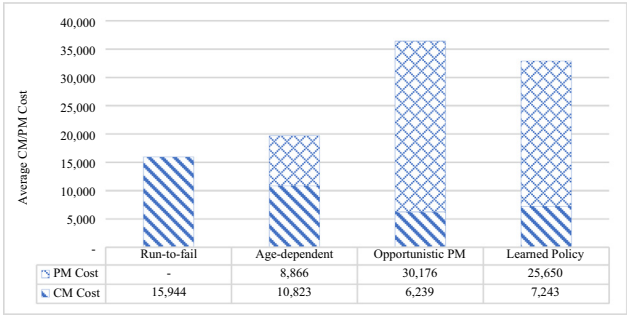
Initial State	Run-to-failure	Age-dependent Policy	Opportunistic Policy	Learned Policy
1	7.92	6.68	6.56	6.18
	[7.81, 8.02]	[6.53, 6.83]	[6.41, 6.7]	[6.06, 6.31]
2	7.98	6.84	6.57	6.15
	[7.9, 8.07]	[6.72, 6.95]	[6.44, 6.69]	[6.03, 6.28]
3	7.89	6.86	6.54	6.12
	[7.77, 8.01]	[6.75, 6.96]	[6.41, 6.68]	[6.0, 6.24]
4	7.88	6.71	6.6	6.1
	[7.76, 7.99]	[6.59, 6.82]	[6.49, 6.71]	[5.98, 6.22]
5	7.92	6.72	6.47	6.16
	[7.82, 8.02]	[6.59, 6.86]	[6.36, 6.58]	[6.03, 6.28]
6	7.8	6.61	6.53	6.18
	[7.69, 7.92]	[6.48, 6.74]	[6.41, 6.66]	[6.04, 6.33]
7	7.93	6.78	6.49	6.09
	[7.81, 8.04]	[6.65, 6.9]	[6.38, 6.6]	[5.97, 6.21]
8	7.78	6.7	6.51	6.03
	[7.66, 7.9]	[6.55, 6.84]	[6.36, 6.65]	[5.92, 6.14]
9	7.9	6.64	6.59	6.09
	[7.77, 8.03]	[6.51, 6.77]	[6.47, 6.72]	[5.99, 6.2]
10	7.8	6.65	6.53	6.2
	[7.69, 7.92]	[6.53, 6.76]	[6.39, 6.67]	[6.09, 6.31]



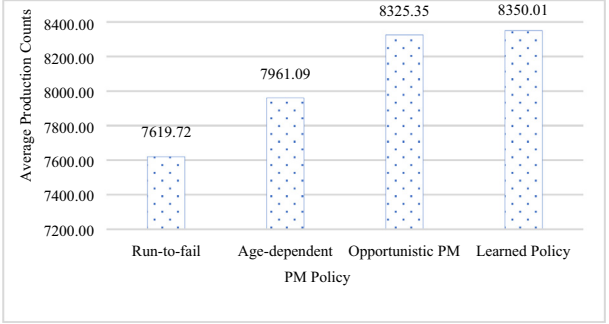
(a) Average number of random failures for four scenarios



(b) Average number of PM conducted for four scenarios



(c) Total resource costs for four scenarios



(d) Final system throughputs for four scenarios

Fig. 6. Cost and throughput analysis for four scenarios in initial state 1.

In conclusion, the age-dependent policy takes the age as the only criteria for the PM decisions. Although the age thresholds are analytically optimal, it seems to be too conservative and doesn't ensure the best system performance. The opportunistic policy takes the PM interval τ as the only decision variable, and it tends to conduct aggressive PMs and ignore the underlying system dynamics. The results demonstrate that PM policy learned by the proposed algorithm is able to balance the tradeoff in the PM decision making. It proves that the problem formulation and the proposed algorithm in this paper is effective in delivering PM policies for serial production lines.

6.3. Further observation – Group maintenance and opportunistic maintenance

To closely examine the learned policy, we take a random initial state and run the production line with the learned policy. 20 consecutive maintenance records along with the states when each maintenance was conducted are drawn from the maintenance history. The records are listed in Table 4.

The 15 consecutive maintenances include 9 PMs and 6 CMs. Interestingly, the "opportunistic maintenance" (OM) and the "group maintenance" (GM) can be observed in the records. To be specific, OM is to conduct PMs on certain machines when there is an unscheduled failure or repair "opportunity" on other machines, and GM is to conduct PMs on multiple machines simultaneously to reduce the total maintenance related cost.

From Table 4, we have observed 7 OMs. For example, at time $t = 5781$ min, machine S_6 failed and a 25-minnute-long CM was conducted, which created an opportunity for PMs on other machines. The learned policy determined to conduct the PM on machine S_1, whose age reached 326 min. In addition, 5 out of the total 9 PMs in the table can be deemed as the GM. For example, at time $t = 6588$ min, machine S_3, S_4, S_5 and S_6 received PMs simultaneously.

The last two columns in the table mark whether the PM record is GM or OM. It turns out that the PM actions at $t = 6019$ min, $t = 6218$ min, $t = 6917$ min and $t = 6818$ are not exclusively GM or OM, but a combination of the two. The OM and GM observed partially explain the reason why the learned policy has less production losses, even though it conducts much more frequent PMs than the age-dependent policy and opportunistic PM policy.

In many existing studies on the maintenance problem in multi-unit systems, the GM and OM are the starting points for deriving maintenance policies (Shafiee & Finkelstein, 2015; Xia et al., 2015). In other words, those studies first restrict the maintenance actions to GM or OM, and further optimize the decision variables for GM and OM to obtain the final policies. For example, the opportunistic PM policy that was used for comparison purpose in this paper has one pivotal decision variable τ, which is the time interval to conduct PM on all machines. However, in this paper we do not take this heuristic approach. Rather, we formulate the policy as a learning problem based on system property (i.e., permanent production loss) and indeed the learned policy is able to make GM and OM decisions when appropriate. From the construction of the reward function, the agent is never explicitly rewarded if it conducts the GM or OM. The decisions of OM and GM are something that the agent learned itself throughout the training process.

When looking into the learning process, it is not difficult to explain the phenomenon. For the GM, the action space includes all the possible combinations of the PM actions on each machine, which means that the GMs are always available for the agent to select. If the GM action in some state yielded better accumulated reward, then the algorithm would increase state-action value to favor GM in the particular state in the future. Similarly, during the training process the agent might also conduct OM when there are random failures on other machines, which would also change the state-action value to encourage or discourage OM in the future. Therefore, the GM and OM observed in the learned policy is a logical outcome as long as the problem formulation is rational, and the solution technique is effective. This interesting finding further validates the DRL based approach proposed in this paper.

6.4. Discussion

Although the proposed framework is proved to be effective, we acknowledge that there are still some limitations that should be addressed. Based on the numerical case and problem assumptions, we discuss some of the potential future work directions.

1) Interpretability of Machine Learning

One might notice that there are some other behaviors that could not be well interpreted well in the learned policy. For example, the learned policy made PM decision $[0, 0, 1, 1, 1, 1]$ at time $t = 6218$ min, and another PM decision $[1, 0, 0, 0, 0, 0]$ in the subsequent time step $t = 6219$ min. Considering the time interval is relatively small compared to the maintenance duration, these two decisions are almost equivalent to a PM decision $[1, 0, 1, 1, 1, 1]$ at time $t = 6218$ min. Although the phenomenon is not truly affecting the system performance, it reveals one of the heated criticisms on machine learning models, i.e. the lack of interpretability. The interpretability of machine learning models is the ability to explain or to present in understandable terms to a human (Doshi-Velez &

Table 4
Maintenance record and GM/OM analysis.

Time t (min)	PM/CM	Failed machines	Machine ages $g_i(t)$	Buffer levels $b_i(t)$	Remaining maintenance duration $d_i^r(t)$	PM Decisions a_t	GM	OM
5781	CM	S_6	[326, 17, 104, 101, 100, 152]	[6, 5, 0, 0, 0]	[0, 0, 0, 0, 0, 25]	–		
5782	PM	–	[327, 18, 105, 102, 101, 0]	[5, 5, 0, 0, 1]	[9, 0, 0, 0, 0, 24]	[1, 0, 0, 0, 0, 0]		Y
6018	CM	S_6	[227, 254, 341, 338, 337, 212]	[8, 10, 0, 0, 0]	[0, 0, 0, 0, 0, 25]	–		
6019	PM	–	[228, 255, 342, 339, 338, 0]	[7, 10, 1, 0, 0]	[0, 0, 0, 0, 0, 24.]	[1, 0, 0, 1, 1, 0]	Y	Y
6020	PM	–	[0, 256, 343, 0, 0, 0]	[7, 10, 1, 0, 0]	[8, 0, 0, 9, 10, 23.]	[0, 0, 1, 0, 0, 0]		Y
6044	CM	S_2	[16, 280, 17, 15, 14, 1]	[8, 10, 1, 5, 8]	[0, 31, 0, 0, 0, 0]	–		
6217	CM	S_2	[189, 142, 190, 188, 187, 174]	[8, 10, 0, 0, 0]	[0, 31, 0, 0, 0, 0]	–		
6218	PM	–	[190, 0, 191, 189, 188, 175]	[8, 10, 0, 0, 0]	[0, 30, 7, 10, 11, 8.]	[0, 0, 1, 1, 1, 1]	Y	Y
6219	PM	–	[191, 0, 0, 0, 0, 0]	[8, 10, 0, 0, 0]	[9, 29, 6, 9, 10, 7]	[1, 0, 0, 0, 0, 0]		Y
6588	PM	–	[360, 340, 361, 358, 357, 360]	[8, 10, 0, 0, 0]	[0, 0, 7, 10, 11, 8.]	[0, 0, 1, 1, 1, 1]	Y	
6589	PM	–	[361, 341, 0, 0, 0, 0]	[8, 10, 0, 0, 0.]	[9, 0, 6, 9, 10, 7.]	[1, 0, 0, 0, 0, 0]		
6680	CM	S_1	[81, 432, 85, 82, 81, 84]	[8, 10, 0, 0, 0.]	[28, 0, 0, 0, 0, 0.]	–		
6916	CM	S_5	[208, 668, 321, 318, 317, 320]	[8, 10, 0, 0, 0]	[0, 0, 0, 0, 24, 0]	–		
6917	PM	–	[209, 669, 322, 319, 0, 321]	[8, 10, 0, 0, 0]	[0, 0, 7, 10, 23, 0.]	[0, 0, 1, 1, 0, 0]	Y	Y
6918	PM	–	[210, 670, 0, 0, 0, 322]	[8, 10, 0, 0, 0]	[9, 8, 6, 9, 22, 8]	[1, 1, 0, 0, 0, 1]	Y	Y

Kim, 2017). The interpretability could be one of the prerequisites to the implementation of any DRL-based PM policies in the real plant floor. Moreover, if we can interpret the DRL agent's decision making, it would also deepen our understanding on production systems and maintenance problems.

2) Combination with Predictive Maintenance (PdM)

In this paper, the machine lifetime is assumed to follow a known distribution. This assumption is reasonable in reality, since machine reliability is one of the important parameters that vendors have to provide. Hence, it has been a common assumption in not only most of the researches on PM (Fitouhi et al., 2017; Kang & Ju, 2019; Karamatsoukis & Kyriakidis, 2010), but also a broad range of studies on production systems (Li et al., 2009). Instead of following a given reliability model, PdM aims at inferring the real-time machine reliability status, e.g. remaining useful life or failure rate, from sensor or inspection data (Wang, 2016). However, the prediction itself does not guarantee a good maintenance decision, and therefore there is promising combination of the PdM techniques and the DRL framework proposed in this paper. To be specific, PdM predicts the machine status based on real-time data, and the DRL agent takes the prediction as one of the inputs to make PM decisions in system level.

3) Coordination between PM and Other Activities

In this paper, we assumed that the maintenance control be carried out independently. However, in the real industry, maintenance is only one of the many activities that support the functioning of a production system. Others include production scheduling, quality assurance, material handling and shift scheduling etc. There are inevitably complex interactions among some of these essential activities. In this case, a good coordination is required between PM decision making and other activities. For example, in some production systems, PM has to be conducted either when the machine is down or it starts outputting inferior products (Pandey, Kulkarni, & Vrat, 2010). The DRL-based framework proposed in this paper should be further extended to multi-agent deep reinforcement learning (MDRL) (Hernandez-Leal, Kartal, & Taylor, 2019) to include these considerations, as the state and action space continue to increase in this case.

7. Conclusion and future work

The PM decision making in a serial production line is a complex problem due to its exploding state space and complicated interactions among machines. The problem is proposed to be solved using a DRL approach in this paper. One of the important prerequisites to successfully solving the problem is that a modeling method for the serial production line is adopted, such that we can correctly capture the dynamics of the system and ensure the good computation efficiency during learning process.

The numerical experiment proves that a good PM policy for the serial production line can be obtained by using the proposed DRL approach. In addition, we observe GM and OM in the learned policy. As two of the most important building blocks for the maintenance policies in the multi-unit systems, the GM and OM were originated from human reasoning. In this paper, they are obtained by DRL without giving the agent any prior concepts and rules. Therefore, if the problems are properly formulated based on thorough understanding of the system properties, we can further exploit the great potentials of the artificial intelligence (AI) and machine learning techniques to facilitate complicated decision-makings in the manufacturing industry.

CRediT authorship contribution statement

Jing Huang: Methodology, Software, Validation, Visualization, Writing - original draft. **Qing Chang:** Conceptualization, Methodology, Supervision, Writing - review & editing, Funding acquisition, Project administration. **Jorge Arinez:** Writing - review & editing, Investigation, Resources.

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.

Acknowledgements

This work was supported by National Science Foundation [grant numbers CMMI 1351160, 1853454].

Appendix

1) Derivation of the Age-dependent PM policy

In the age-dependent policy, machine S_i receives a PM when it reaches an age of X_i and receives a CM when it fails before that age. The process can be modeled as a renewal reward process. Let $C_i(t)$ be the accumulated maintenance costs, including both production loss and resource cost, up to time t, then the long-run maintenance cost rate R_i is

$$R_i = \lim_{t\to\infty} \frac{C_i(t)}{t} \tag{A1}$$

According to Renewal Reward Theorem (Ross, 2014), the long-run cost rate can be evaluated by

$$R_i = \frac{\mathbb{E}[C_i]}{\mathbb{E}[d_i]} \tag{A2}$$

where $\mathbb{E}[C_i]$ is the expected cost in one renewal cycle, and $\mathbb{E}[d_i]$ is the expected duration of one renewal cycle.

The CM on machine S_i lasts for d_i^{CM} units of time and it will lose d_i^{CM}/T_i parts. The CM cost is $c_i^{CM} + c_p \cdot d_i^{CM}/T_i$ and the PM cost can be calculated likewise. The expected cost $\mathbb{E}[C_i]$ is

$$\mathbb{E}[C_i] = \int_0^{X_i} \left(c_i^{CM} + c_p \cdot \frac{d_i^{CM}}{T_i}\right) p_i(x)dx + \int_{X_i}^{\infty} \left(c_i^{PM} + c_p \cdot \frac{d_i^{PM}}{T_i}\right) p_i(x)dx \tag{A3}$$

If one renewal cycle ends up with a CM, the duration of the cycle is the age at failure plus the CM duration d_i^{CM}. If in one renewal cycle ends up with a PM, the duration of the cycle is $d_i^{CM} + X_i$. Therefore, the expected cycle duration $\mathbb{E}[d_i]$ is

$$\mathbb{E}[d_i] = \int_0^{X_i} \left(d_i^{CM} + x\right) p_i(x)dx + \int_{X_i}^{\infty} \left(d_i^{PM} + X_i\right) p_i(x)dx \tag{A4}$$

Table 5
Optimal age thresholds for age-dependent PM policy.

Machine	S_1	S_2	S_3	S_4	S_5	S_4
Optimal age threshold X_i^* (min)	349	316	416	492	646	505

An optimal age threshold X_i^* is

$$X_i^* = \arg\min_{X_i} \frac{\mathbb{E}[C_i]}{\mathbb{E}[d_i]} \tag{A5}$$

Given the parameters of the four machines in Table 3, the optimal age thresholds for the age-dependent PM policy are computed according to Eq. (A5) and shown in Table 5.

References

Ab-Samat, H., & Kamaruddin, S. (2014). Opportunistic maintenance (OM) as a new advancement in maintenance approaches. *Journal of Quality in Maintenance Engineering, 20*(2). https://doi.org/10.1108/JQME-04-2013-0018.

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. https://doi.org/10.1007/s10845-011-0616-3.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine, 34*(6), 26–38. https://doi.org/10.1109/MSP.2017.2743240.

Barros, A., Grall, A., & Berenguer, C. (2007). Joint modelling and optimization of monitoring and maintenance performance for a two-unit parallel system. *Proceedings of the IMechE, 221*(1), 1–11. https://doi.org/10.1243/1748006XJRR31.

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. https://doi.org/10.1016/j.ijpe.2008.08.058.

Dodson, B. (2006). *The Weibull analysis handbook*. ASQ Quality Press.

Doshi-Velez, F., & Kim, B. (2017). Towards A Rigorous Science of Interpretable Machine Learning. ArXiv Preprint ArXiv:1702.08608. Retrieved from http://arxiv.org/abs/1702.08608.

Ebrahimipour, V., Najjarbashi, A., & Sheikhalishahi, M. (2013). Multi-objective modeling for preventive maintenance scheduling in a multiple production line. *Journal of Intelligent Manufacturing, 26*(1), 111–122. https://doi.org/10.1007/s10845-013-0766-6.

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 and System Safety, 166*(March), 61–72. https://doi.org/10.1016/j.ress.2017.03.034.

Hasselt, H. van, Guez, A., & Silver, D. (2016). Double DQN.pdf. Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16), 2094–2100. Retrieved from https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/viewPaper/12389.

Hernandez-Leal, P., Kartal, B., & Taylor, M. E. (2019). A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems, 33*(6), 750–797. https://doi.org/10.1007/s10458-019-09421-1.

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. https://doi.org/10.1016/j.procir.2019.01.073.

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, 1–1*. https://doi.org/10.1109/ACCESS.2018.2876024.

Kang, Y., & Ju, F. (2019). Flexible Preventative Maintenance for Serial Production Lines with Multi-stage Degrading Machines and Finite Buffers. *IISE Transactions, 5854*, 1–28. https://doi.org/10.1080/24725854.2018.1562283.

Karamatsoukis, C. C., & Kyriakidis, E. G. (2010). Optimal maintenance of two stochastically deteriorating machines with an intermediate buffer. *European Journal of Operational Research, 207*(1), 297–308. https://doi.org/10.1016/j.ejor.2010.04.022.

Laggoune, R., Chateauneuf, A., & Aissani, D. (2009). Opportunistic policy for optimal preventive maintenance of a multi-component system in continuous operating units. *Computers and Chemical Engineering, 33*(9), 1499–1510. https://doi.org/10.1016/j.compchemeng.2009.03.003.

Li, J., E. Blumenfeld, D., Huang, N., & M. Alden, J. (2009). Throughput analysis of production systems: recent advances and future topics. International Journal of Production Research, 47(14), 3823–3851. Doi: 10.1080/00207540701829752

Liu, J., Chang, Q., Xiao, G., & Biller, S. (2012). The Costs of Downtime Incidents in Serial Multistage Manufacturing Systems. *Journal of Manufacturing Science and Engineering, 134*(2). https://doi.org/10.1115/1.4005789 021016.

Martinez, C., Ramasso, E., Perrin, G., & Rombaut, M. (2020). Adaptive early classification of temporal sequences using deep reinforcement learning. *Knowledge-Based Systems, 190*. https://doi.org/10.1016/j.knosys.2019.105290 105290.

Mnih, V., Kavukcuoglu, K., Silver, D., Andrei A. Rusu, Veness, J., Bellemare, M. G., ... Shane Legg. (2016). Human-level control through deep reinforcement learning. Nature, 2016-Janua(7540), 2315–2321. Doi: 10.1038/nature14236

Nicolai, R. P., & Dekker, R. (2008). Optimal Maintenance of Multi-component Systems: A Review. *Springer Series in Reliability Engineering, 8*, 263–286. https://doi.org/10.1007/978-1-84800-011-7_11.

Pandey, D., Kulkarni, M. S., & Vrat, P. (2010). Joint consideration of production scheduling, maintenance and quality policies: A review and conceptual framework. *International Journal of Advanced Operations Management, 2*(1/2), 1. https://doi.org/10.1504/ijaom.2010.034583.

Ramírez-Hernández, J. A., & Fernandez, E. (2010). Optimization of preventive maintenance scheduling in semiconductor manufacturing models using a simulation-based approximate dynamic programming approach. Proceedings of the IEEE Conference on Decision and Control, 3944–3949. Doi: 10.1109/CDC.2010.5717523.

Ross, S. M. (2014). Introduction to Probability Models. *Academic Press*. https://doi.org/10.1080/00401706.1998.10485493.

Shafiee, M., & Finkelstein, M. (2015). An optimal age-based group maintenance policy for multi-unit degrading systems. *Reliability Engineering and System Safety, 134*, 230–238. https://doi.org/10.1016/j.ress.2014.09.016.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Driessche, G. Van Den, ... Kavukcuoglu, K. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7585), 484–489. Doi: 10.1038/nature16961.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning*. An Introduction. MIT Press.

Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for. Machine Learning, 4*(2), 26–31.

van Hasselt, H., Guez, A., & Silver, D. (2015). *Deep Reinforcement Learning with Double Q-learning*. In Thirtieth AAAI conference on artificial intelligence: Retrieved from. http://arxiv.org/abs/1509.06461.

Wang, H. (2002). A survey of maintenance policies of deteriorating systems. *European Journal of Operational Research, 139*(3), 469–489. https://doi.org/10.1016/S0377-2217(01)00197-7.

Wang, K. (2016). Intelligent Predictive Maintenance (IPdM) system-Industry 4.0 scenario. *WIT Transactions on Engineering Sciences, 113*(2016), 259–268. https://doi.org/10.2495/IWAMA150301.

Wang, W. (2012). An overview of the recent advances in delay-time-based maintenance modelling. *Reliability Engineering and System Safety, 106*, 165–178. https://doi.org/10.1016/j.ress.2012.04.004.

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. https://doi.org/10.1007/s10845-013-0864-5.

Wei, S., Bao, Y., & Li, H. (2020). Optimal policy for structure maintenance: A deep reinforcement learning framework. *Structural Safety, 83*. https://doi.org/10.1016/j.strusafe.2019.101906 101906.

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. https://doi.org/10.1016/j.ejor.2014.08.004.

Ye, M.-H. (1990). Optimal replacement policy with stochastic maintenance and operation costs. *European Journal of Operational Research, 44*(1), 84–94. https://doi.org/10.1016/0377-2217(90)90317-5.

Zhao, C., & Li, J. (2015). Analysis and Improvement of Multiproduct Bernoulli Serial Lines: Theory and Application. *IEEE Transactions on Systems, Man, and Cybernetics: Systems, 45*(9), 1218–1230. https://doi.org/10.1109/TSMC.2015.2399868.

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. https://doi.org/10.1016/j.eswa.2017.03.025.

Zou, J., Chang, Q., Lei, Y., & Arinez, J. (2018). Production System Performance Identification Using Sensor Data. *IEEE Transactions on Systems, Man, and Cybernetics: Systems, 48*(2), 255–264. https://doi.org/10.1109/TSMC.2016.2597062.