Deep Reinforcement Learning for Optimal Planning of Assembly Line Maintenance

Journal of Manufacturing Systems 69 (2023) 170–188

Contents lists available at ScienceDirect

Journal of Manufacturing Systems

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

Deep reinforcement learning for optimal planning of assembly line maintenance

M. Geurtsen [a,b,*], I. Adan [a], Z. Atan [a]

[a] Department of Industrial Engineering, Eindhoven University of Technology De Zaale, 5600 MB, Eindhoven, The Netherlands
[b] ITEC, Nexperia, Jonkerbosplein 52, 6534 AB, Nijmegen, The Netherlands

ARTICLE INFO

Keywords:
Scheduling
Maintenance
Deep reinforcement learning
Simulation
Case-study
Flexibility

ABSTRACT

Discovering the optimal maintenance planning strategy can have a substantial impact on production efficiency, yet this aspect is often overlooked in favor of production planning. This is a missed opportunity as maintenance and production activities are deeply intertwined. Our study sheds light on the significance of maintenance planning, particularly in the dynamic setting of an assembly line. By maximizing the average production rate and incorporating flexible planning windows, buffer content, and machine production states, a unique problem is addressed in which a policy for planning maintenance on the final machine of a serial assembly line is developed. To achieve this, novel average-reward deep reinforcement learning techniques are employed and pitted against generic dispatching methods. Using a digital twin with real-world data, experiments demonstrate the immense potential of this new deep reinforcement learning technique, producing policies that outperform generic dispatching strategies and practitioner policies.

1. Introduction

The vast majority of research in production scheduling assumes that machines are always available during a planning horizon [1]. However, in practice, machines may become unavailable due to variety of reasons, including planned and/or unplanned maintenance activities. Although maintenance interrupts the production process, it guarantees a better equipment condition which might lead to higher production rates. As production and maintenance both utilize the available time of machines, decision support on the optimal timing to execute maintenance can be highly beneficial [2].

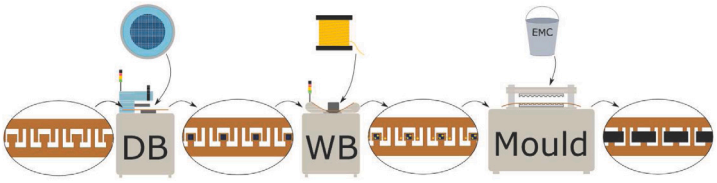
This study is motivated by the production of integrated circuits (ICs) at Nexperia, a global semiconductor manufacturer that produces more than 90 billion ICs annually. In particular, in the back-end assembly process ICs are produced on assembly lines. The assembly process involves three steps in series: (1) die-bonding, (2) wire-bonding and (3) molding. Three different machines are connected by a support structure, that flows through the line. An illustration of this process is provided in Fig. 1(a). An assembly line with multiple machines of each type and buffers in between machines is depicted in Fig. 1(b). This type of assembly line can be considered as a general N machines, $N-1$ buffers serial production line.

Maintenance on these serial production lines is necessary to prevent major failures and keep the equipment (machines) in appropriate operational conditions. Performing Preventive Maintenance (PM) on serial production lines is a complicated endeavor due to the close connections of the equipment in the line. Wrongly timing of PM can have a significant impact on system throughput. Therefore, when executing PM it is crucial to take the active equipment status and buffer levels into account. Due to equipment failures or small hiccups amidst production, the number of produced products between two consecutive PM activities is not constant. Hence, in order to better reflect the deterioration process of equipment, a usage-based maintenance approach would be preferred. Adopting a usage-based maintenance approach rather than performing the maintenance at predetermined moments in time results in less certainty on the future time at which maintenance will be required. Consequently, in case of limited maintenance resources, machines could become idle. To prevent such scenarios, flexibility on the usage-based maintenance activities can be applied. An example of this maintenance planning method can be found in the semiconductor industry, more specifically at the back-end assembly process of Nexperia, in which PM must be scheduled on the last machine of the assembly line in such a manner that congestion on the upstream machines is minimized.

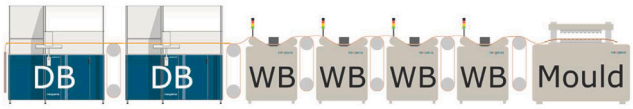
Deriving a policy based on these complex and intricate interactions among machines in the serial production line, requires keeping track of the many environmental conditions. This requirement implies an extremely large state space of the maintenance optimization problem.

* Corresponding author at: Department of Industrial Engineering, Eindhoven University of Technology De Zaale, 5600 MB, Eindhoven, The Netherlands.
E-mail address: michaelgeurtsen@protonmail.com (M. Geurtsen).

https://doi.org/10.1016/j.jmsy.2023.05.011
Received 10 February 2023; Received in revised form 21 April 2023; Accepted 17 May 2023



(a) An assembly line with a die-bond (DB) machine, a wire bond (WB) machine and a molding machine in series. On the DB the sawed wafer is loaded, on the WB a roll of wire is loaded (copper or gold) and the Mold uses EMC-shielding material to form an encapsulation for the IC. The lead frame flows through all machines and is shown in the figure between consecutive machines.



(b) Configuration of an assembly line considered in this study: two die-bonders, followed by four wire-bonders and ending with one mold machine. A buffer of varying size is positioned between each machine.

Fig. 1. Assembly line.

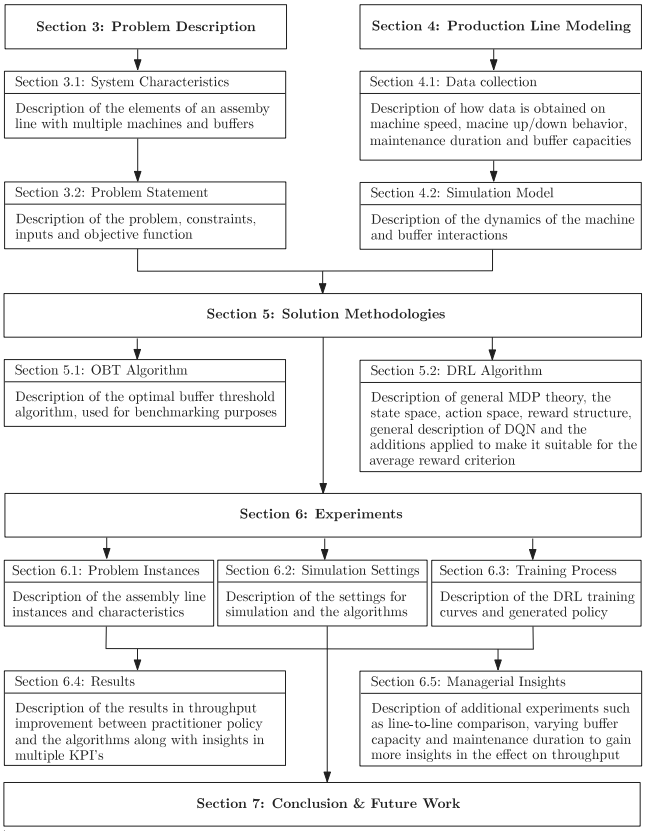


Fig. 2. Flowchart showing the layout of the paper.

The problem rapidly becomes intractable with traditional model-based planning methods. New tools and methodologies emerging in Artificial Intelligence (AI) and machine learning (ML) areas such as Deep Reinforcement Learning (DRL) might provide promising techniques that can be applied for intelligent decision-making in maintenance management.

This paper is devoted to address the complex maintenance planning problem described above. The main contributions of this paper are: (1) studying a new maintenance problem where a policy must be derived based on multiple characteristics such as the machine production states, buffer contents and allowed flexibility of the maintenance activity; (2) formulating the problem as a Markov Decision Process (MDP) and applying DRL as a solution method; (3) being the first study to apply Q-network based DRL techniques for the average-reward setting; (4) implementing the problem in a discrete-event simulation that models the production as a fluid flow which uses real production data as input.

This study shows that adopting usage-based maintenance over time-based maintenance leads to significant throughput improvements. In addition, allowing flexibility for maintenance and taking into account the machine production states when initiating PM extends the improvement even further. The newly proposed DRL technique is the best performing method for this particular problem and can be a useful approach for any manufacturing problem that deals with decision-making based on multiple inputs and long-term goals.

The remainder of the paper is organized as follows. In Section 2, a review of the relevant literature is provided. Section 3 describes the problem in more detail. Section 4 introduces the modeling method of the production line and the simulation model. Section 5 describes two solution methods, a heuristic approach and a DRL approach. Then, in Section 6, experiments are performed to compare the DRL method to other dispatching strategies. Finally, conclusions and recommendations are provided in Section 7. Fig. 2 provides a flowchart, depicting the organizational structure of the paper.

2. Literature review

Maintenance has a strong relationship to production. The purpose of maintenance is to allow production, yet to execute maintenance production often has to be interrupted. This negative effect must therefore be considered in planning and optimizing maintenance along production. Reviews on the topic of maintenance scheduling and planning in combination with production are provided by Budai et al. [2] and Geurtsen et al. [3]. As this study only addresses maintenance decisions, we do not give advice on how to plan production. Therefore, we set the scope of our literature review to the setting where (1) production is not explicitly scheduled, but instead taken into account in the form of conditions or requirements, and (2) the decision on when to do maintenance is determined based on characteristics of the production line, such as machine states and buffer contents. Following this reasoning, we identify two streams of research. The first stream addresses studies that consider machine and buffer interactions in a production line for scheduling maintenance. The second stream deals with studies where a maintenance activity is planned at those moments that certain machines are not needed for production, also referred to as Opportunistic Maintenance (OM). Each stream is dealt with in a separate section.

2.1. Scheduling maintenance based on machine and buffer interactions in a production line

Van der Duyn Schouten and Vanneste [4] are one of the first to consider a downstream buffer for planning maintenance. A two-machine single-buffer problem is considered where inspections occur at discrete time epochs. Both CM and PM are considered where the time to failure is a stochastic variable with known probability distribution function. To prevent frequent failures, PM is allowed, which can be initiated at any time epoch. A class of control-limit policies are developed in which maintenance is triggered based on the buffer content and the machine age. A similar setting is examined by Kyriakidis and Dimitrakos [5]. While Van der Duyn Schouten and Vanneste [4] only consider a cost when maintenance is initiated while the buffer is empty, Kyriakidis and Dimitrakos [5] extend the cost function by including operating, maintenance and storage costs. Later, they extend their study in Karamatsoukis and Kyriakidis [6] by including costs due to lost production. A discrete-time Markov decision model is presented with which they show that for fixed buffer content and fixed deterioration degree of one machine, the average-cost optimal policy initiates PM on the other machine if its degree of deterioration exceeds some critical level. The two-machine single-buffer problem is also studied in Meller and Kim [7] where the goal is to determine the optimal buffer level that triggers PM on the first machine. While these studies proved that the optimal policy is of control-limit type, only a simple setting with a fixed buffer level is analyzed. This is therefore less applicable when buffer levels fluctuate significantly.

Another approach considers both the deterioration of the equipment and the buffer level to determine when to initiate maintenance. For a review on maintenance policies for deteriorating systems, we refer the reader to the study by Wang [8]. Liu et al. [9] study a single machine-single buffer problem where the machine deteriorates and the optimal maintenance interval and buffer level have to be determined to maximize system availability. In Fitouhi et al. [10], a two-machine single-buffer problem is studied where both machines transition from one degraded state to another. The decision when to perform PM depends on the degraded state of the machines and the action can be chosen to restore the machine to any other less degraded state, which influences the costs of PM. A two-machine single-buffer problem is also studied by Zhou and Zhang [11], where each machine has two components that suffer from degradation. The decision to make is on which component of what machine to perform PM, based on buffer status and component degradation state. The goal is to optimize the expected revenue per unit time, which consists of the production revenue and costs of maintenance, operation and inventory. Wang and Qi [12] study a similar problem but without multiple components. Instead, they consider imperfect production and imperfect PM, which is constrained by a scarce resource. Deterioration is represented by multiple decreasing yield levels. They present a multi-agent reinforcement learning approach to decrease the long-run average cost. Li and Zhou [13] extend the similar problem to a more complex production line with multiple machines and buffers. The degradation of the machine follows a discrete-time discrete-state Markov process. Gu et al. [14] also study a serial production line with multiple machines and buffers where each machine deteriorates according to a geometric distribution. Different distributions for maintenance durations are evaluated. PM decisions are based solely on machine degradation states. Arab et al. [15] go a step further and study multiple complex production lines, including both serial and parallel processes with multiple buffers. Decisions on when to perform maintenance is based on work-in-process (WIP) and remaining reliability of equipment. A genetic algorithm is developed to create a schedule that maximizes the long-run throughput. In general, the aim of modeling deterioration is to generate more precise maintenance schedules. However, this is usually paired with assumptions on deterioration processes that differ significantly in literature, as seen in Wang [8]. These approaches therefore lack general applicability in practice.

Another technique is to use the production rate of the machine as a decision variable. Zequeira et al. [16] study a production facility of two machines with an intermediate buffer, where at random times, the first machine is able to receive extra production capacity to build up a buffer before initiating maintenance. They are interested in finding the optimal time to do maintenance based on both the decision when to use the extra production capacity and the ideal buffer threshold. A similar problem is studied by Magnanini and Tolio [17]. In a two-machine single-buffer production line, the upstream machine has a higher production rate than the downstream machine and the upstream machine is characterized by a degradation profile on which PM can be performed. A threshold-based control policy is proposed where switching points define the buffer level and the machine state for which PM should be activated. In addition, hedging points are defined to reduce the production rate of the upstream machine in order to avoid surplus in the buffer.

A different approach is based on using the buffers as a measure for bottleneck detection and utilize this information to schedule maintenance. The study by Langer et al. [18] is the first to adopt such an approach. They consider a serial production line with reactive and PM operations. PM has a fixed schedule where the time between two consecutive maintenance activities is assigned randomly. Priority on which machine to perform maintenance is assigned based on a dynamic bottleneck-approach. The studies by Li et al. [19] and Gopalakrishnan et al. [20] also use bottleneck identification based on buffer utilization as an approach to prioritize maintenance on a serial production line. However, only reactive maintenance is modeled. Gopalakrishnan et al. [20] propose a dynamic shifting-bottleneck approach where the priority of which machine to serve changes in real-time by looking at the momentary bottlenecks. Li et al. [19] on the other hand use a data-driven approach where the solution method is developed without an analytical or simulation model. A combination of policy-based and bottleneck-based decision making is studied by Lu et al. [21]. A serial production line with multiple workcenters and intermediate buffers is considered. The workcenter consists of machines that require either corrective maintenance or PM, performed by a scarce technician. First, it is decided whether to perform maintenance, which is based on both the number of failures a machine has experienced and thresholds for the buffers. Then, maintenance is dispatched based on which machine is a greater bottleneck. Although the studies show an interesting approach to determine on which machine to schedule maintenance in case it must be executed immediately, mostly CM activities are considered. In addition, maintenance activities that have some flexibility are completely neglected.

2.2. Opportunistic maintenance

Opportunistic maintenance originates from the thought that the downtime of a system is often an opportunity to combine preventive and corrective maintenance. Especially for systems where components are connected in series, the effect of a single failure could result in disturbances on the other components. These systems are also known as multi-component systems. A review on multi-component systems is provided by Nicolai and Dekker [22]. Reviews on the application of opportunistic maintenance for these multi-component systems can be found in Ab-Samat and Kamaruddin [23] and Werbińska-Wojciechowska [24]. Serial production lines can also be interpreted as a multi-component system. As disturbances on one machine in the line may affect upstream or downstream machines. Indeed, intermediate buffers between the machines could mitigate the direct connection of the machines and therefor reduce the effect of disturbances, however the connection will not cease to exist. Most literature on opportunistic maintenance assumes a direct connection

between the components, i.e. no buffers exist in-between the components. According to Werbińska-Wojciechowska [24], four distinct groups of policies within opportunity based maintenance can be defined: (1) age-based opportunity maintenance models, (2) failure-based opportunity maintenance models, (3) opportunity and condition-based maintenance models and (4) mixed PM models that consider different types of maintenance policies. We will only consider research where maintenance opportunities arise due to either unexpected component failures or flexible windows for PM, as this best matches the problem studied in this paper.

van der Duyn Schouten and Vanneste [25] consider a serial system of two components. The times to failure are stochastic variables with a known probability distribution function and inspections occur at discrete time epochs. Upon inspection, the choice to make is which component to replace. The choice depends on the lifetime of both components, the breakdown cost and replacement cost. Laggoune et al. [26] study a system of multiple components in series where the failure of any component leads to the failure of the whole system. Each component has a time interval for PM. The decision to make is whether to take the opportunity to replace preventively some of the non-failed components in case the system is down, due to either CM or PM. The decision is based on component degradation and the risk of failure of the components before reaching the next scheduled PM. In Sarker and Faiz [27], similar to Laggoune et al. [26], a problem is studied where the failure of one component causes the whole system to stop. Upon a component failure, corrective maintenance is done, but opportunities arise to perform PM as well. The decision whether to perform maintenance on the other components is based on the age group the component is in. In addition, the degree of preventive action per component can be set as well. Maintenance times are considered negligible and the goal of the optimal policy is to minimize overall costs. A different problem is considered by Gunn and Diallo [28]. They study a system of multiple components where each component requires PM. The time between two consecutive maintenance activities differs per component. Failure of the component is not considered. In addition, before the end due of maintenance is reached, a time window is available in which maintenance may be flexibly scheduled. This enables an opportunity to group maintenance of multiple components together. Zhou et al. [29] examine a similar problem, but additionally consider stochastic failures of the components. A failure of a component now also becomes an opportunity to group PM together, based on the flexible time windows of PM. A policy is obtained by minimizing the cumulative maintenance cost. Above studies do not explicitly model production and therefor neglect the duration of maintenance and the impact it can have on production throughput.

Ferreira Neto et al. [30] take production activities into account. They study a serial production line where the first machine supplies material to the downstream machines downstream and a buffer exists between the first machine and downstream machines. The first machine is composed of n components and the machine fails if k-out-of-n components fail. Inspections are performed when the machine fails, the buffer is full or empty or when the optimal thresholds for the operation time and the buffer level are exceeded, which is viewed as the opportunistic window. The occurrence of empty buffers as opportunistic windows is also studied by Wu et al. [31] and Yang et al. [32]. While Wu et al. [31] first inspection to determine if PM is required, Yang et al. [32] propose a condition-based strategy. Both model the arrival of the empty buffer opportunities as a Poisson process. More recently, Huang et al. [33] study a serial production line of multiple machines with intermediate buffers. Both corrective and PM is considered and the lifetime of a machine follows a known distribution. A Deep Reinforcement Learning approach is presented. In the state space, the machine ages, the buffer levels and the remaining maintenance durations of the ongoing maintenance activities are included. The policy obtained by the agent is compared to a run-to-failure policy, an age-based replacement policy and an opportunistic policy. Interestingly, the agent learns a policy that combines aspects of both opportunistic maintenance and group maintenance, which outperforms the other policies in terms of maintenance cost. Valet et al. [34] also successfully apply DRL to an opportunistic maintenance scheduling problem. They consider opportunities induced by approaching condition-based breakdowns, as well as opportunities triggered by external factors such as empty/full buffers. A job shop environment is considered and the decision to perform PM is based on a time-to-failure distribution for each machine and the content of the buffer prior to the machine. States of neighboring machines are not considered. Kuhnle et al. [35] propose exactly a similar opportunistic maintenance approach as in Valet et al. [34], but apply it to a less complicated environment of single machines. A different study is considered in Zhang et al. [36], where a two-machine single-buffer production line is examined. Machines are always producing but deteriorate over time. Their goal is to define the best levels for when to perform CM, PM and OM, based on buffer level and deterioration state.

Similar as in Section 2.1, bottleneck-based approaches can also be used for opportunistic maintenance. Zhou et al. [37] study a serial production line with intermediate buffers where each machine has a unique failure rate. PM is performed at fixed time intervals. Bottlenecks are continuously monitored and in case PM is performed on a bottleneck machine, the opportunity to perform small repairs on other machines is taken as well. The study by Chang et al. [38] utilizes the buffer contents as well as machine starvation and blockage states to define bottleneck machines and obtain opportunities for performing maintenance during production. They develop a continuous flow model to determine the maintenance opportunity window in a serial production line. However, stochasticity is not modeled and only the direct production losses during maintenance are considered. Gu et al. [39] extend the study by Chang et al. [38] by including stochastic machine failures, but only consider machine starvation and blockage. Later, they extend their work in Gu et al. [40] and propose a method based on active maintenance opportunity windows to seek real-time opportunities for performing PM. The method is based on estimating how long a machine can be shut down for while still satisfying the system throughput requirement by considering both the production losses during and after PM. Bottleneck-based approaches are interesting methods as using starvation and blocking information to plan maintenance can improve production efficiency. However, it does not capture the complete information of a production line, such as down states and machine speeds and therefore misses possible maintenance opportunities.

As shown in this review, literature that studies the planning of maintenance based on production environment characteristics is comprehensive and has many directions. There is a large separate stream of literature on the modeling of machine deterioration in combination with PM. However, these studies usually consider maintenance cost and barely focus on the impact of maintenance on production throughput. In case deterioration is not modeled and throughput is considered, corrective maintenance activities or the presence of buffers are typically introduced instead. Studies that plan maintenance based on buffer levels often only consider a fixed set of levels for which to trigger PM or only use machine starvation and blocking modes to define a maintenance policy. In studies where maintenance is actually triggered on a more detailed buffer level, such as in Valet et al. [34], decisions based on machine production states and the use of corrective maintenance activities as opportunities are neglected. Using corrective maintenance activities to plan PM is often considered in the literature stream on opportunistic maintenance. However, these studies usually do not model production as a continuous flow of products, such as in Huang et al. [33], and thereby need to make use of functions to model a production environment, resulting in a less detailed and inefficient simulation.

To address these shortcomings, this paper attempts to combines both streams together by constructing a maintenance policy based

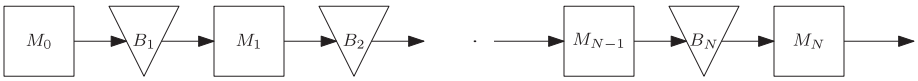


Fig. 3. Schematics of a serial production line.

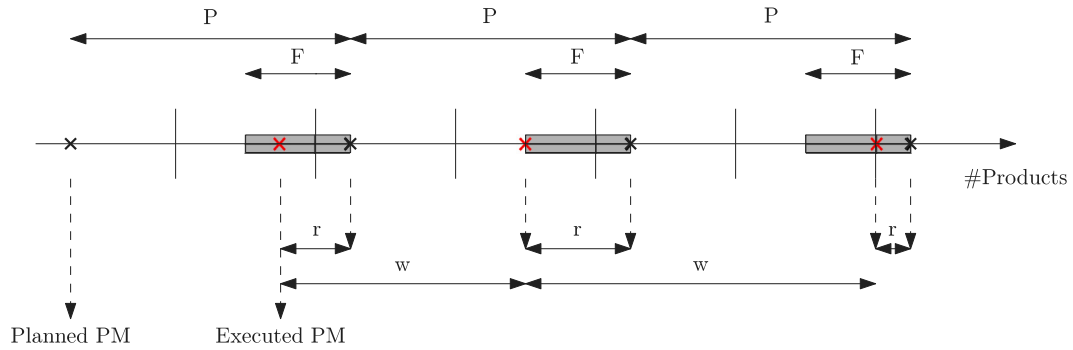


Fig. 4. Schematic overview of the procedure when cleaning is performed based on produced products with flexibility. The black cross marks the planned PM activities based on the fixed number of products, while the red crosses mark examples of actual executions of PM activities. The gray rectangle represents the flexibility window.

on real-time buffer levels, machine production states and the available flexibility of the maintenance activity. Additionally, a discrete event simulation is used to model production as a continuous fluid flow, typically observed in high-speed production lines. To the best of our knowledge, creating a maintenance policy by combining flexible maintenance activities in a real-time environment with buffer levels and machine production states has not been studied before. This study extends the analysis based on an MDP of the proposed problem in Geurtsen et al. [41] to the more complex and stochastic setting of a full production line, based on simulation.

3. Problem description

3.1. System characteristics

A serial production line with N machines and $N-1$ buffers as shown in Fig. 3, is considered. The machines M_i, $i \in \{0, 1, \dots, N\}$, are represented with rectangles and the intermediate buffers B_i, $i \in \{1, 2, \dots, N-1\}$, are represented with triangles. The arrows specify the direction of the material flow in the system.

Each buffer B_i has a finite capacity. The maximum capacity of buffer B_i is B_i^{max}. The state of buffer B_i is described by $b_i \in \{e, f, n\}$. The state is e when the buffer is empty and it is f when the buffer is full. When the buffer is neither empty nor full, it is state n. The buffer levels are changing with the system dynamics. We denote the level of buffer B_i as l_i. The buffer is in state e if $l_i = 0$, in state f if $l_i = B_i^{max}$ and in state n if $0 < l_i < B_i^{max}$.

The state of machine M_i is described by s_i. A machine can be in four different states; up (u), down (d), blocked (b) and starved (s). Machine M_i is blocked if the downstream buffer B_{i+1} is full and the downstream machine M_{i+1} is not in state u, i.e., it is not manufacturing. Machine M_i is starved if upstream buffer B_i is empty and the upstream machine M_{i-1} is not in state u. When the machine is in state d, it means that an activity is being performed on the machine for which the machine must be stopped. Among others, this can be a corrective maintenance activity, calibration activity, tool change or material change. In addition, the last machine can be in the maintenance state (m). The production times, down times and maintenance duration are independent follow known distributions. These are empirically sampled from the real-world data of the Original Equipment Manufacturer (OEM).

Each machine M_i in the line has an operating speed v_i and a maximum production speed v_i^{max}. If buffer B_i is not empty and buffer B_{i+1} is not full, the operating speed v_i of machine M_i equals the maximum production speed v_i^{max}. Otherwise, machine M_i adapts its operating speed to the speed of neighboring machines, v_{i-1} or v_{i+1}, depending on whether the upstream buffer is empty or the downstream buffer is full, and only if the neighboring machines have a lower maximum speed and are in a production state. In case the neighboring machines are not in a production state, v_i will be 0. By modeling the speed in such manner, the speed of machine M_i could be equal to the speed of a machine multiple positions down the line. As an example, $v_i = v_{i+5}$, if all buffers from B_{i+1} until B_{i+5} are full, all machines from M_i until M_{i+5} are in a production state and the maximum production speeds of machines M_i to M_{i+4} are larger than v_{i+5}^{max}.

Buffer B_i is filled with rate $v_{i-1} - v_i$, which can be either positive or negative, depending on whether machine M_{i-1} or M_i has a faster operating speed. With a positive fill rate, the content of the buffer between machines M_{i-1} and M_i increases until it reaches its maximum capacity B_i^{max}. Conversely, with a negative rate, the buffer is emptied until the content reaches zero.

3.2. Problem statement

Based on the characteristics described above, a formal problem statement can be defined. The last machine in the assembly line, M_N, requires periodic PM. The PM activities are scheduled at fixed intervals with length P (in terms of number of products manufactured), as shown in Fig. 4. This guarantees that maintenance is executed once every P number of products, which is convenient for the operators. A PM activity cannot exceed this predefined limit. However, before this limit is reached, a flexibility interval of length F emerges. Maintenance can be performed anywhere in the interval F. Then, the limit for the next PM event will not be P products after the last PM execution, but instead remains as planned. In doing so, the window between two consecutive PM activities w is not fixed, but varies throughout the scheduling horizon as is seen in Fig. 4. The new interval is equal to $P + r$, where r defines the number of products that were still left in the previous flexibility window. In this setting, the length of the window w varies, where the maximum and minimum length is respectively $P + F$ and $P - F$. This method ensures that the long-run average window length will be equal to P.

Let Π denote the set of PM policies for a serial production line. The PM policy $\pi \in \Pi$ instructs when to execute PM on the last machine M_N, once the flexibility interval is reached. This decision is based on the production line characteristics such as machine state and buffer contents, as described in Section 3.1. Let $P(t; \pi)$ denote the total number of produced products up to time t under the PM policy π. An optimal policy π^* should maximize the long-run average throughput of the

Table 1
List of notations.

Notation	Description	Notation	Description	Notation	Description
N	Total number of machines in the line	M_i	ith machine in the line	B_i	ith buffer in the line
b_i	State of buffer B_i	n	Neutral state of a buffer	e	Empty state of a buffer
f	Full state of a buffer	l_i	Buffer content of buffer B_i	B_i^{max}	Maximum buffer capacity of buffer B_i
s_i	State of machine M_i	u	Production state of a machine	d	Down state of a machine
b	Blocked state of a machine	s	Starved state of a machine	v_i	Operating speed of machine M_i
w	Interval between 2 consecutive PM activities	P	Periodic interval of PM	F	Flexibility window for PM
r	Number of products left in the flexibility window	v_i^{max}	Maximum speed of machine M_i	π	PM policy
$P(t;\pi)$	Production count up to time t under PM policy π	μ_i	Fill rate of buffer B_i	S_t	State at time t
X_u	Distribution of the up state	X_d	Distribution of the down state	R_t	Reward at time t
A_t	Action at time t	C_M	Set of machines in the state space	C_B	Set of buffers in the state space
Q	Value for state action pair	θ	Neural network	θ^-	Target network
δ	error; difference between target and neural network	η	Constant controlling the average reward	$\bar{R}$	Average reward

serial production line. Hence, the maintenance optimization problem can be formulated as:

$$\pi^* = \arg\max_{\pi \in \Pi} \left\{ \lim_{t \to \infty} \frac{P(t;\pi)}{t} \right\} \tag{1}$$

A summary of the notations used in this paper is provided in the Table 1.

4. Production line modeling

To learn and evaluate PM policies, a model is required that captures the behavior of the serial production line described in Section 3 as realistically as possible. Simulation is an efficient method that can accurately represent real-world physical systems. In addition, this method is robust, easy to interpret and implement. This section presents a fluid flow discrete event simulation, which accurately models the serial production line.

4.1. Case description and data collection

In 2001, Nexperia's Industrial Technology and Engineering Centre (ITEC) introduced its advanced warning and data collection system, abbreviated AWACS, which is used for the analysis of machine performance. Machine status monitoring forms the core of this system and is responsible for the collection of a wide range of machine data. In particular, the changes in machine states and production count logs are of interest for this work. A machine can be in one of three aggregated states: production, standby or down. Fig. 5 shows the three aggregate states and their respective sub-states. The production state indicates that the machine is up and products are being produced. The down state means that a machine is unable to produce and is divided into multiple sub-states indicating the reason. It is noted that the error sub-state is also an aggregate state that contains hundreds of specific errors that may occur. These sub-states are not modeled in the simulation of the serial production line. Instead, solely the aggregate down state of all these sub-states is used in the model, to describe the down behavior of a machine. The standby state indicates that the machine could produce products, but is not. The cause for this is indicated by one of the four standby sub-states. The sub-state wait input means that the upstream buffer is empty and that the upstream machine is down, i.e., the machine is starved. The wait output state means that the downstream buffer is full and the downstream machine is down, i.e., the machine is blocked.

We use actual machine data as input to the simulation model. The required data for this problem includes (1) up and down state distributions for each machine in the line, (2) machine speeds for each machine in the line, (3) maintenance duration of the last machine and (4) maximum capacities of the buffers in-between the machines.

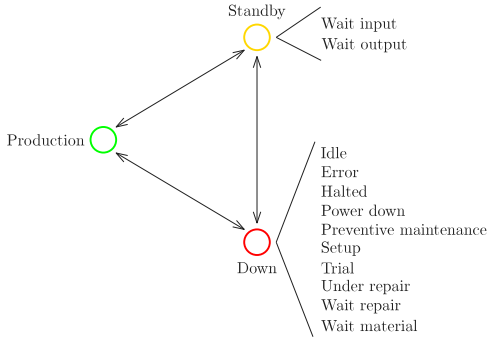


Fig. 5. Overview of machine states and sub-states.

Up and down state distribution: The distribution for the up state, X_u, can be obtained by monitoring the time a machine is in a production state until it changes its state to a down state. It is important to note that starvation and blocking states are not part of the up- and downtimes. Therefore, production times before and after a period of starvation or blocking are added together to obtain one up-time realization. When this procedure is applied for a long enough time period, many samples can be acquired wherewith an accurate distribution can be constructed. We perform a similar procedure to attain the distribution of the down state, X_d. Durations in up and down states are assumed to be independent.

Machine speed: Machine i has a maximum speed of v_i^{max}. This value is derived from the data. In accurately deriving the machine speed from data for serial production lines with buffers, two important factors should be taken into account. The first factor is internal speed losses of the machine due to the machine itself, for instance when the machine is not calibrated correctly or incorrect settings are used. The second factor is external speed losses when the machine should adapt its speed when the upstream buffer is empty and the upstream machine is producing at a lower speed or when the downstream buffer is full and the downstream buffer is producing at a lower speed. The maximum machine speeds that are used as input for the simulation model should include internal speed losses, but they should not include speed adaption because of empty or full buffers, since speed adaptation will be determined by the simulation model itself.

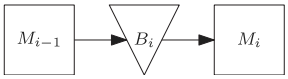


Fig. 6. Schematics of a serial production line of 2 machines and 1 buffer.

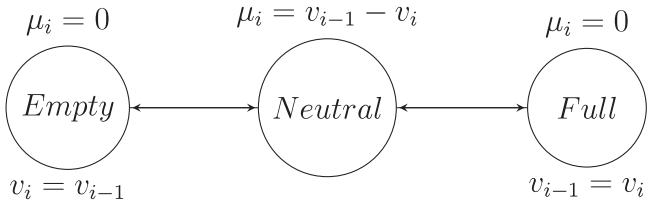


Fig. 7. Schematic overview of the buffer dynamics.

The production count is monitored by the machine. To derive an accurate estimate for the machine speed, speed adaptation should be corrected for. Therefore, only the speed during the up times is used, during which continuous production is observed with neither empty or full buffers upstream and downstream. The maximum speed of the machine in the simulation is the average speed over all up times, i.e. total production count divided by total up time.

Maintenance duration and buffer capacities: A distribution of the maintenance duration d_{PM} is created through monitoring the start and end time of the PM activity on the last machine of the line. The maximum buffer capacities B_i^{max} are provided by the assembly site.

4.2. Fluid simulation model

The speed at which production lines create products can be very high. Therefore, it is natural to describe the product flow as a fluid flow model instead of a discrete model.

For simplicity, we consider a system with two machines and a finite buffer in between, as depicted in Fig. 6. We use the same notation for the buffer states, machine states and production rates as in Section 3. Additionally, we define μ_i as the buffer fill rate.

Buffer dynamics: It is important to accurately model the behavior of the buffer into the simulation model, as buffers can have a large impact on the production line throughput. As mentioned in Section 3, a buffer can be in three different states: empty ($b_i = e$, $l_i = 0$), full ($b_i = f$, $l_i = B_i^{max}$) and neutral ($b_i = n$, $0 < l_i < B_i^{max}$). In case buffer i is empty, $\mu_i = 0$ and machine M_i has to slow down the production and adapt its speed to the speed of machine M_{i-1}, i.e., $v_i = v_{i-1}$, and conversely if the buffer is full. When the buffer is in a neutral state, speeds of machines M_i and M_{i-1} are not affected and the fill rate of the buffer is defined by the difference in speeds of the machines, i.e., $\mu_i = v_{i-1} - v_i$. Fig. 7 graphically explains these dynamics.

Machine dynamics: The behavior of a machine is directly connected with the behavior of a buffer. As mentioned in Section 3, machines can have four different states: up (u), down (d), blocked (b) or starved (s). The first machine is never starved and the last machine is never blocked. A product that leaves one machine can immediately be produced by the next machine in case the buffer is completely empty. In a down state, the machine cannot produce products, thus $v_i = 0$. In an up state, a machine produces products with a rate that ranges between 0 and v_i^{max}. The actual rate v_i depends on the machines and buffer statuses of neighboring machines. If machine M_{i-1} is up and buffer B_i is empty, the production rate of machine M_i will be the same as for machine M_{i-1}, i.e., $v_i = v_{i-1}$. However, if machine M_{i-1} is down and buffer B_i is empty, the machine M_i will become starved and its speed drops to 0. Likewise, in case machine M_{i+1} is up and buffer B_{i+1} is full, the production rate of machine M_i will match machine M_{i+1}, i.e., $v_i = v_{i+1}$. However, if machine M_{i+1} is down and buffer B_{i+1} is full, machine M_i will be blocked and its speed is 0. Failures of the machine are operational-dependent instead of time-dependent, i.e., a machine cannot break down when it is in a starvation or blocking state. Fig. 8 graphically explains these dynamics.

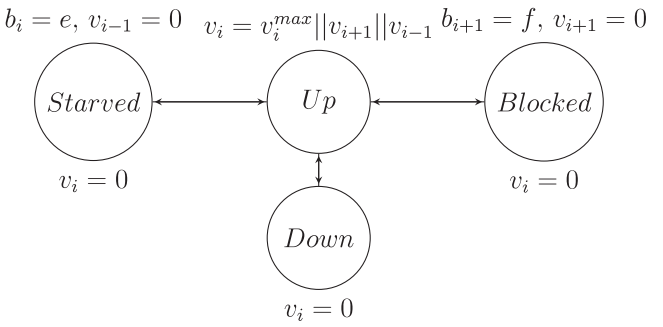


Fig. 8. Schematic overview of the dynamics of a machine.

5. Solution methodologies

In this section, two algorithms are presented that aim to find a PM policy for the problem described in Section 3. The purpose of the first algorithm, presented in Section 5.1, is not to find an optimal policy π^* but instead to serve as a baseline policy for bench-marking purposes. Then, in Section 5.2, a DRL algorithm is presented with the objective of finding an optimal policy π^*.

5.1. Optimal buffer threshold algorithm

Initiating PM at an inopportune moment could be detrimental for the long-run average throughput. For instance, initiating PM when the buffer prior to the last machine in the line is full, will result in more congestion of the upstream machines during the PM activity. As described in Section 3, the decision of when to initiate a PM is based on three elements: (1) the machine states, (2) the buffer levels and (3) the number of products left in the flexibility window.

The baseline policy presented in this section only optimizes for two of the three elements; the buffer level and the machine state, while ignoring the number of products left in the flexibility window. Adding this last element to the optimization increases the search space dramatically, which results in an impractical procedure. Therefore, adding this final element is left to the DRL method proposed in Section 5.2.

While DRL is able to handle large state spaces, the baseline policy presented here cannot. Therefore, in order to reduce the search space for the baseline policy even further, only the buffer prior to the last machine in the line and the states of the maintenance machine and the machine prior to the maintenance machine are considered. These components of the serial production line are the most influential factors for making PM decisions, as they are closest to the machine that receives maintenance. Consequently, the baseline policy ignores some information of the entire system, while the DRL approach presented later in this section will be able to include all information. The method to determine the optimal buffer threshold level and machine state combination is presented in Algorithm 1. In the remainder of this paper, this policy is referred to as the Optimal Buffer Threshold (OBT) algorithm. It involves a straightforward iterative greedy optimization. The objective of this method is to find the best buffer thresholds for each pair of machine state combinations. A list of all pairs is provided in Table 2. The buffer threshold indicates a level below which PM may be initiated for the given machine state combination. The optimal thresholds are found by incrementally increasing the threshold for each machine state combination, starting with 0, and then verifying whether the long-run average throughput improves. Table 2 shows all 9 machine state combinations. However, the total number to iterate over equals

Table 2
List of all possible combinations for two consecutive machines.

1. $u-u$	2. $u-d$	3. $d-u$	4. $d-d$	5. $d-s$	6. $b-d$	7. $s-s$	8. $s-d$	9. $s-u$

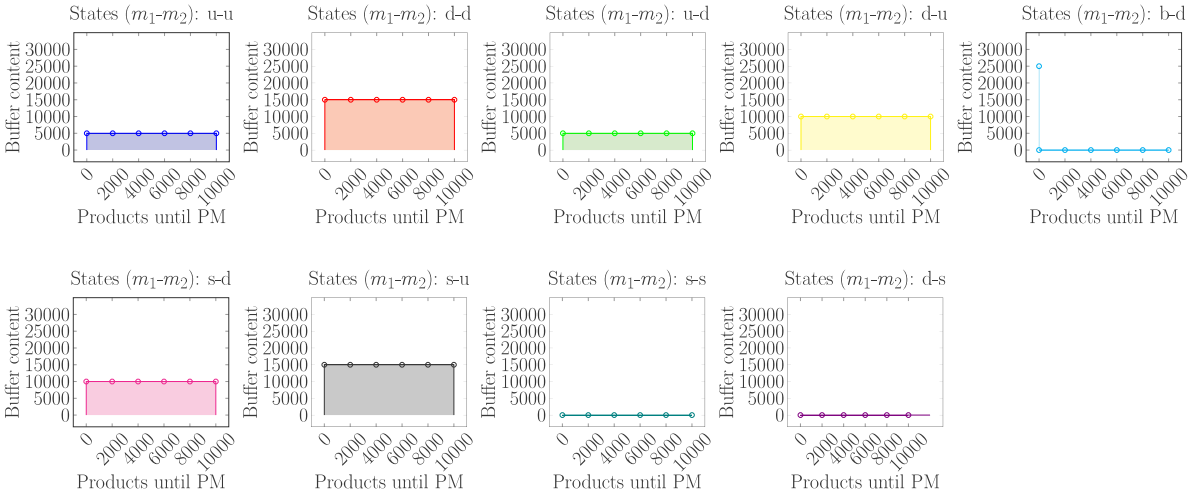


Fig. 9. Policy describing what action to take, given a pair of machines with a buffer in-between. The decision depends on the buffer content, depicted on the y-axis, and how much products are left in the flexibility window until the limit is reached, depicted on the x-axis. The shaded planes shows the region under which it is allowed to perform PM.

6 as there are 3 combinations where the threshold is fixed; when the buffer is empty or the buffer is full. For these exceptions, the policy is to always initiate PM when the buffer is empty and never to do it when the buffer is full.

Algorithm 1: Optimum Buffer Threshold (OBT)

Input: Set of buffer levels L to iterate over
Set of machine state combinations C to iterate over
Result: Optimal buffer level per machine state combination $O(c)$

```
Initialize O(c) ∀c to zero
while no more improvement do
    for combination c ∈ C do
        set buffer threshold to next buffer level l ∈ L
        run simulation
        if improvement then
            O(c) = l
        end
    end
end
```

An example of a policy for two machines and one buffer may look like the policy shown in Fig. 9. In this example, the buffer capacity and the flexibility window are arbitrarily chosen to be 30.000 products and 10.000 products, respectively. For each combination of machine states, different thresholds can be observed. In the example, the flexibility window is divided into 5 bins of 2000 products. Similarly, the buffer thresholds are split into 6 bins of 5000 products. Thus, the search space can be further controlled by defining the number of bins for each element.

5.2. Deep reinforcement learning algorithm

As mentioned in Section 5.1, the state space for the problem explodes when all three state elements (states of machines, buffer levels and number of products in flexibility window) are considered. The state size can be further controlled by defining the number of bins for the buffer level and the flexibility window. Exact approaches such as Dynamic Programming (DP) do not suffice in obtaining an optimal PM policy, given this large state space. Instead, DRL is more suitable to address the problem. In particular, the DRL methods train a policy through sampling transitions in the state and action space from an environment. The data-driven simulation environment described in Section 4 provides an ideal setting for employing DRL-based methods.

Most problems for which DRL approaches have been applied have a trade-off between short-term and long-term rewards. For the particular PM problem considered here, only the long-term average reward is of interest. Conventional DRL methods are therefor most likely not well-suited for our PM problem. Accordingly, a novel state-of-the-art DRL algorithm is presented, specifically designed for problems that aim to optimize the long-run average reward. In the next sections, the basic elements of DRL for the PM problem of this study are described first. Afterwards, the novel DRL algorithm is presented.

5.2.1. MDP formulation

A Markov Decision Process (MDP) is the mathematical foundation of RL and essentially describes a framework for decision making under uncertainty. In an MDP, a decision maker inhabits an environment which changes state randomly in response to actions made by the decision maker. Formally, an MDP is defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, p)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{R}$ is the set of rewards, and $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \longrightarrow [0,1]$ is the dynamics of the environment. At each discrete moment in time $t \in \{0,1,2,\dots\}$, the decision maker observes state $S_t \in \mathcal{S}$, selects an action $A_t \in \mathcal{A}$ and receives a reward $R_{t+1} \in \mathcal{R}$. Then, the system transitions to the next state $S_{t+1} \in \mathcal{S}$. The state transition probability is $p(s', r|s, a) = Pr(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a)$ for all $s, s' \in \mathcal{S}, a \in \mathcal{A}$ and $r \in \mathcal{R}$.

In the context of the PM problem considered in this work, the state represents the status of the serial production line, the action is whether or not to perform PM on the production line and the reward is the number of produced products by the production line. These three components need to be properly defined for the DRL algorithms to be applied and the optimal policy π^* to be obtained. These components are described in more detail below.

State space

The state $S_t \in \mathcal{S}$ should fully describe the status of the production line and the PM-related conditions. There are three essential elements to describe the state, which are necessary for an agent to make proper decisions:

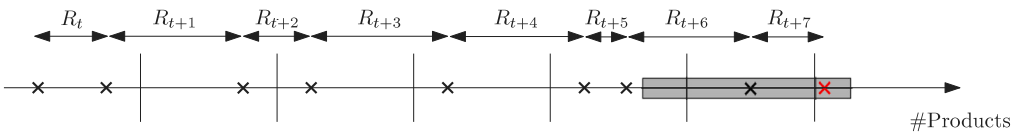


Fig. 10. Example trace of reward over time. Here, the black crosses indicate the moment a time-step has passed.

1. The state of the machines in the line at time t, $s_i(t) \in \{u, d, b, s\}$. Note that not all machines need to be included into the state space, i.e., any combination of machines of the production line can be selected; $C_M \subset \{1, 2, \dots, N\}$, where N is the total number of machines in the line.
2. The buffer levels of the buffers in the line, $0 \leq l_j(t) \leq B_j^{max}$. Again, not all buffers need to be included into the state space and any combination can be selected; $C_B \subset \{1, 2, \dots, N-1\}$. Also note that the categorical buffer states (empty, neutral, full) need not to be included since the empty and full state can be characterized by the minimum and maximum values of the buffer levels.
3. The active number of products left in the flexibility window until the PM limit is reached, $r(t)$.

State S_t is then defined as:

$$S_t = [s_i(t), l_j(t), r(t)] \quad \forall i \in C_M \quad , \quad \forall j \in C_B \tag{2}$$

Action space

The action $A_t \in \mathcal{A}$ describes all PM-related maintenance actions that an agent can choose. For the PM problem in this study, there are only 2 actions an agent can select:

$$A_t = \begin{cases} 0 & \text{do nothing} \\ 1 & \text{perform PM} \end{cases} \tag{3}$$

An agent can only choose between these two actions once it has arrived in the flexibility window. Action 0 is selected otherwise. In addition, when the PM production count limit is reached, only one action can be performed, which is action 1. Thus, depending on state S_t, there is a legal action set $A(S_t)$. When training the agent, action-masking should be applied such the agent can only choose feasible actions from $A(S_t)$. This prevents that the agent trains on infeasible actions, which can slow down the learning process.

Reward function

The reward R_t received by the agent at time t is the production output from the production line between times $t-1$ and t. The length of the discrete time step, T, can be chosen by the user and is measured by the actual time during simulation. The time step T is a fixed value throughout the simulation. This time step T is not coupled in any way to the model described in Section 4. It means that during a time step of length T, the simulation model can have a multitude of transitions between up and down states on the machines. Therefore, the simulation keeps a record of the production output it produced during the time step. Also notice that this production output completely depends on the state of the last machine in the line. If it was in a down state during the complete time step, the reward received by the agent will be 0. Whereas the agent will receive the maximum reward in case the last machine was continuously producing during the entire time step. As the machine might also need to adapt its speed in case of empty buffers and machine downs in the upstream line, the reward per time step ranges between 0 and $T \cdot v_N^{max}$. The reward can be described as:

$$R_t = T \cdot \bar{v}_N \tag{4}$$

where $\bar{v}_N$ is the average speed realized by the last machine during time step T. Fig. 10 shows an example of the reward for a certain period, demonstrating that while the time step may be fixed, the reward can vary significantly.

5.2.2. Algorithm implementation

The goal of reinforcement learning is to learn good policies for sequential decision-making problems [42]. Multiple algorithms have been proposed for this task. The Q-learning algorithm by Watkins [43] is one of the most popular reinforcement learning algorithms. It is a model-free, off-policy control algorithm. It is model-free because it purely samples from experience, i.e., it relies on real samples from an environment. Unlike the DP algorithms, it never uses generated predictions of the next state and next reward to alter behavior. It is off-policy, because it samples experiences by following a behavior policy that is different from the agent's learned policy. For details on the implementation of the Q-learning algorithm, we refer to Sutton and Barto [42]. The Q-learning algorithm learns an optimal policy π^* by finding the best Q-values. There is a Q-value for each state–action pair which essentially describes the expected reward when selecting an action in a given state. In Q-learning, a large table is utilized to find the Q-values for each state–action pair, $Q(S, A)$, where each entry of the table represents a state–action pair combination. Then, the best action to select in a particular state according to the optimal policy is given by the action which has the highest Q-value for that state:

$$\pi^*(S) = \underset{A' \in A(S)}{\arg\max}\ Q(S, A').$$

The problem with methods that utilize such a table is the lack of scalability as they cannot be applied to larger state spaces. Not only the physical memory required to store the table is a problem, but also the training time to find accurate values for the state–action pairs explodes. Suppose that for our problem we choose to include 2 machines with an intermediate buffer in the state space, buffer levels ranging from [0,20] and flexibility window ranges from [0,50]. The state space size as defined in then becomes approximately 1×10^{18}, which is simply unachievable with tabular-based methods. Fortunately, the scalability problem has been well addressed in recent years by implementing machine learning methods for already existing RL algorithms. The groundbreaking study by Mnih et al. [44] presented the first machine-learning equipped Q-learning algorithm, named Deep Q-Network (DQN). In DQN, the Q-values are approximated with a neural network θ, i.e., $Q(S, A, \theta) \approx Q(S, A)$. The two main elements of DQN are the experience replay and the target network θ^-. 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 $\mathcal{B}$. Mini-batches are sampled from $\mathcal{B}$ to train the neural network θ. For each sample j from the mini-batch, the Q-values are predicted by both the neural network θ and the target network θ^-, as follows:

$$\begin{aligned} Q_j &= Q(S_j, A_j, \theta) \\ Q_j^- &= R_j + \gamma \underset{A' \in A(S_{j+1})}{\arg\max}\ Q(S_{j+1}, A', \theta^-), \end{aligned} \tag{5}$$

where γ is a discount factor, which allows to make a trade-off between immediate and future rewards. Then, the difference between the values of Q_j and Q_j^- is computed as a loss and the neural network θ is updated with gradient descent. The loss considered in this work is the mean-squared error loss (MSE):

$$MSE = \frac{1}{b} \sum (Q_j - Q_j^-)^2 \tag{6}$$

Here, b is the size of the mini-batch. The weights of the target network θ^- get updated as well. For instance, by copying the weights of the neural network θ to the target network θ^-. After training,

new experiences are collected, added to the replay memory and older experiences are removed. DQN seeks to iteratively update the neural network parameters θ, which could well approximate the Q-values, until the ultimate policy π^* is obtained.

Since its publication, many improvements for the DQN have been presented of which the most significant is the Double Deep Q-Network (DDQN) by Hasselt et al. [45]. The main goal of DDQN is to overcome the overestimation of the action values in the DQN algorithm. The idea behind this is that in Q-learning and DQN, there exists a maximization bias. If the Q-values calculated in Eq. (5) are slightly overestimated, then this error gets compounded [42]. The max operator uses the same values both to select and to evaluate an action. This makes it more likely to select overestimated values, resulting in overoptimistic value estimates. To prevent this, DDQN proposes to decouple the selection from the evaluation. This is done by letting the neural network θ select the actions, but using θ^- to evaluate the values, when calculating the target Q-values. We refer to Hasselt et al. [45] for more details on the original DDQN algorithm. Consequently, Eq. (5) becomes:

$$Q_j^- = R_j + \gamma Q(S_{j+1}, \underset{A' \in A(S_{j+1})}{\arg\max}\ Q(S_{j+1}, A', \theta); \theta^-). \tag{7}$$

Both DQN and DDQN have been used successfully to solve maintenance problems by Wang and Qi [12] in a two-machine-one-buffer production line and by Huang et al. [33] in a multi-machine-multi-buffer production line. This strengthens the choice to use Q-network based algorithms for our problem as well. Though, as explained in Sections 2 and 3, an entirely different PM problem is considered in this work as maintenance is executed based on buffer contents and flexibility windows.

For our PM problem, the number of maintenance cycles (length of an episode) to train is an important parameter that needs to be defined carefully. Choosing it too low will most likely result in a policy that cannot capture the behavior of maintenance actions on the system throughput. For example, in the extreme case of 1 maintenance cycle, the effect of executing maintenance on the upstream machines after maintenance has been completed, will not be considered. At the same time, choosing it too large will require the discount factor γ to be large as well, in order to effectively trace back the effects of past actions. This might result in longer training times.

Unfortunately, preliminary experiments showed that applying DDQN in this discounted episodic setting results in unsatisfactory policies that are not better than the base-line policy from Section 5.1. Neither hyper-parameter tuning nor implementing any DQN-related enhancement from literature had any effect on the resulting policy. In addition, we also applied the Proximal Policy Optimization (PPO) algorithm by Schulman et al. [46] to verify whether the cause was algorithm-related, however with similar bad results. This lead to the idea that for this particular problem, discounted/episodic class of algorithms are not well-suited. For this reason, both DDQN and PPO are not applied for the experiments in Section 6. In Geurtsen et al. [41], a similar PM problem is studied. Among general DP methods, an average-reward Q-learning algorithm shows promising results. In the average-reward setting, experience is continuous and cannot be broken up into episodes. In this setting, an agent seeks to maximize the average reward per step, or reward rate, where immediate and delayed rewards are equally important. The nature of the PM problem studied here is well-suited for algorithms that aim to optimize the long-run average reward directly. Since, in this study we are interested in the throughput of the production line over an infinite time horizon. In addition, optimizing the long-run average reward eliminates decisions regarding the size of an episode and discount factor γ. The average reward Q-learning algorithm applied in Geurtsen et al. [41] is the Differential Q-learning algorithm proposed by Wan et al. [47], shown in Algorithm 2.

Algorithm 2: Differential Q-learning (one-step off-policy control)

Input: The policy b to be used (e.g., ϵ-greedy)
Algorithm parameters: step-size parameters α, η
1 Initialize $Q(s,a)\ \forall s,a$; $\bar{R}$ arbitrarily (e.g., to zero)
2 Obtain initial S
3 **while** *still time to train* **do**
4 A $\leftarrow$ action given by b for S
5 Take action A, observe R, S'
6 $\delta = R - \bar{R} + \max_a Q(S', a) - Q(S, A)$
7 $Q(S, A) = Q(S, A) + \alpha\delta$
8 $\bar{R} = \bar{R} + \eta\alpha\delta$
9 $S = S'$
10 **end**
11 return Q

Algorithm 2 extends the tabular Q-learning algorithm mentioned earlier to the average-reward setting. The main idea of the algorithm is to estimate an average reward rate and use this rate in the Temporal Difference (TD) error calculation, which can be seen in line 6 of Algorithm 2. This TD error calculation is similar to Eq. (5), but applied to tabular Q-learning. The algorithm is guaranteed to converge to a differential value function, which is the expected differential return under a policy from a given state or state–action pair. This differential value function captures how much more reward the agent gets by starting in a particular state than it would get on average over all states if it followed a fixed policy.

Wan et al. [47] also apply the same algorithm in the linear function approximation setting. The linear function approximation setting has some resemblance to the neural network setting, as it also attempts to reduce the size of the state space with the use of an approximator. They show that the idea of Differential Q-learning also works in the linear function approximation setting. Therefore, we chose to extend the DDQN with the concept of the average-reward algorithm of Wan et al. [47].

To convert the standard DDQN to a DDQN for the average reward setting, the first change required is to adapt Eq. (7):

$$Q_j^- = R_j - \bar{R} + Q(S_{j+1}, \underset{A' \in A(S_{j+1})}{\arg\max}\ Q(S_{j+1}, A', \theta); \theta^-)$$

Here, $\bar{R}$ is the value for the average reward. Notice that the discount factor γ has been removed. Then, we need to establish a formula that estimates the average reward $\bar{R}$. Similar to line 7 in Algorithm 2, we define an error using the Q-values of the neural network θ and the Q-values of the target network θ^-:

$$\delta = \frac{1}{b}\sum_{j=1}^{b}(Q_j^- - Q_j)$$

We take the sum over all samples of the mini-batch, and compute the average. Notice that we explicitly do not use the MSE loss in Eq. (6), as it contains a squared term which would make δ always positive. This would result in an ever-increasing average reward. With δ defined, $\bar{R}$ can be estimated as follows:

$$\bar{R} = \eta \cdot \delta$$

Here, η is a positive constant that controls how much the average reward $\bar{R}$ will be changed. With all modifications for the average-reward setting described, the full algorithm is shown in Algorithm 3. A flow chart of the algorithm is depicted in Fig. 11. From this point onwards, the algorithm is referred to as the Average-reward DQN (ADQN). Notice that in the action-selection mechanism, not a standard ϵ-greedy policy is used as there is not a random action selected in

case the sampled random number is below ϵ. Instead, we apply an exploration procedure that is specifically suited for the PM problem considered in this work. As opposed to selecting a random action, we choose to select either action 0 (do nothing) or action 1 (perform PM) for K consecutive times. The idea originates from the characteristic of the flexibility window. In order to explore the effect of delaying the execution of PM until the PM limit, we need to be able to reach this limit, which can only be achieved if action 0 is selected multiple times in sequence. This means that the full flexibility windows needs to be explored which cannot be accomplished by purely selecting a random action.

Algorithm 3: ADQN

Input: N_b, b, η, C, ϵ_0, ϵ_d, ϵ_f, α, U, K
Output: θ

Initialize replay memory $\mathcal{B}$ to capacity N_b
Randomly initialize neural network θ
Initialize target network with weight $\theta^- = \theta$
Initialize average reward $\bar{R} = 0$
Initialize ϵ to ϵ_0
for *t=0,1,....* ∞ **do**
  Every C steps, set $\theta^- \leftarrow \alpha \cdot \theta$
  Every K steps, alternate τ between values from set {0,1}
  Set $\epsilon \leftarrow \text{Max}(\epsilon_d \cdot \epsilon, \epsilon_f)$
  Find legal action set $A(S_t)$
  Draw a random number $\xi \sim \text{Uniform}(0,1)$
  if $\xi > \epsilon$ **then**
    Select $A_t = \underset{A' \in A(S_t)}{\arg\max}\ Q(S_t, A', \theta)$
  else
    Select $A_t = \tau$ if $\tau \in A(S_t)$
  end
  Provide action A_t to the environment
  Run the environment for one step
  Observe R_t, S_{t+1}
  Store transition sample (S_t, A_t, R_t, S_{t+1}) in replay memory $\mathcal{B}$
  case *Every U steps*
    Sample a mini-batch of transitions (S_j, A_j, R_j, S_{j+1}) of size b from $\mathcal{B}$
    Set $Q_j^- = R_j - \bar{R} + Q(S_{j+1}, \underset{A' \in A(S_{j+1})}{\arg\max}\ Q(S_{j+1}, A', \theta); \theta^-)$
    Set $Q_j = Q(S_j, A_j, \theta)$
    Perform a gradient descent step on $\left(Q_j^- - Q_j\right)^2$ w.r.t. neural network θ
    Set $\bar{R} = \eta \cdot \frac{1}{b} \sum_{j=1}^{b} (Q_j^- - Q_j)$
  endsw
end

As can be seen in Algorithm 3, there are multiple hyper-parameters that can be tuned. In addition, the size of the neural network can be adjusted. The main properties of a conventional neural network consist of the number of layers, L, and the size of the layers, h_l. A full list of all parameters is given in Table 3.

6. Experiments

In this section, policies generated by the OBT and ADQN algorithms described in Section 5 are compared against the policy currently employed by the factory. First, the real-world production lines and the related data are presented in Section 6.1. Next, Section 6.2 describes the parameter selection for each algorithm. Then, the training process of the ADQN agent is analyzed in Section 6.3. Finally, the results are shown in Section 6.4.

6.1. Real-world production lines

Experiments are conducted with a simulation model using real data as input. All the considered production lines have a similar configuration in terms of number of machines and sizes of buffers. Table 4 summarizes the complete configuration. Each machine in each line has different behavior in terms of the up and down states and the machine speed, as explained in Section 4. A total of 11 different production lines are examined. For the sake of overview, the average (μ) and standard deviation (σ) over all lines is provided in Table 4. However, for the experiments, policies are generated for each line individually. Since each line is a unique environment, an agent is trained for each line separately, thereby generating in total 11 unique agents and consequently 11 different policies.

6.2. Settings

Before applying the OBT algorithm and training the ADQN algorithm, the settings for each algorithm have to be defined. Additionally, the general settings for the simulation need to be specified as well.

6.2.1. Simulation settings

The main setting for the simulation that needs to be defined is the time step T. This parameter determines how many virtual simulation seconds elapse until an action is requested by the agent. Setting it too low might slow down the training process since more training steps are required for the same time frame, compared to using a much larger time step. However, setting it too large might result in missed opportunities since, with a larger time step, occasions of ideal maintenance executions might be lost. Therefore, in an attempt to find the right balance between training time and loss of opportunities, the time step T is set to 100 s. With an average production speed of 26 products/second for the last machine, obtained from Table 4, the maximum reward and therefore the maximum step in terms of production and buffer increase/decrease are 2600 products. In addition, when the threshold for executing maintenance is set to e.g. 1.2 million products, the minimum number of possible steps to complete one maintenance cycle is equal to 1.2 million divided by 2600 products, which is roughly 462 steps. Preliminary experiments showed that with a time step T of 100 s, the best trade-off between training steps per maintenance cycle and frequency for maintenance opportunities is realized.

Furthermore, we need to model the up-and-down behavior of the machines benefiting from the real-world data. For each machine, multiple thousands of samples are obtained. Therefore, the choice is made to keep it simple and adopt an empirical distribution, thereby randomly selecting up and down times from the set of samples for each machine.

At last, the size of the periodic interval for PM, P, and the size of the flexibility window for PM, F, need to be defined. The value for P is based on the policy currently employed by the factory. This policy is a time-based policy, where PM is performed every 12 h. Then, the number of produced products between two consecutive maintenance activities can be extracted from the production data. By doing this for many PM activities conducted over the past year, a distribution is created. The value for P is then set to the average of this distribution. This average is found to be approximately 1.000.000 products. For the flexibility window, multiple options are examined starting from a window size of 100.000 products until 600.000 products, with step size of 100.000 products. Therefore, in total, six flexibility levels are investigated to evaluate the effect of increased flexibility.

6.2.2. OBT algorithm

In the OBT algorithm, only the final two machines and the buffer in-between are considered. Therefore, the only choice to make is the step size of the buffer thresholds. The step size of the buffer threshold is chosen to be 5.000 products, which is well above the maximum increase/decrease of the buffer in one time step. Table 4 shows that the size of the buffer equals 120.000 products for each production line instance. With a step size of 5.000 products, 24 buffer thresholds will be explored, for each machine-state pair.

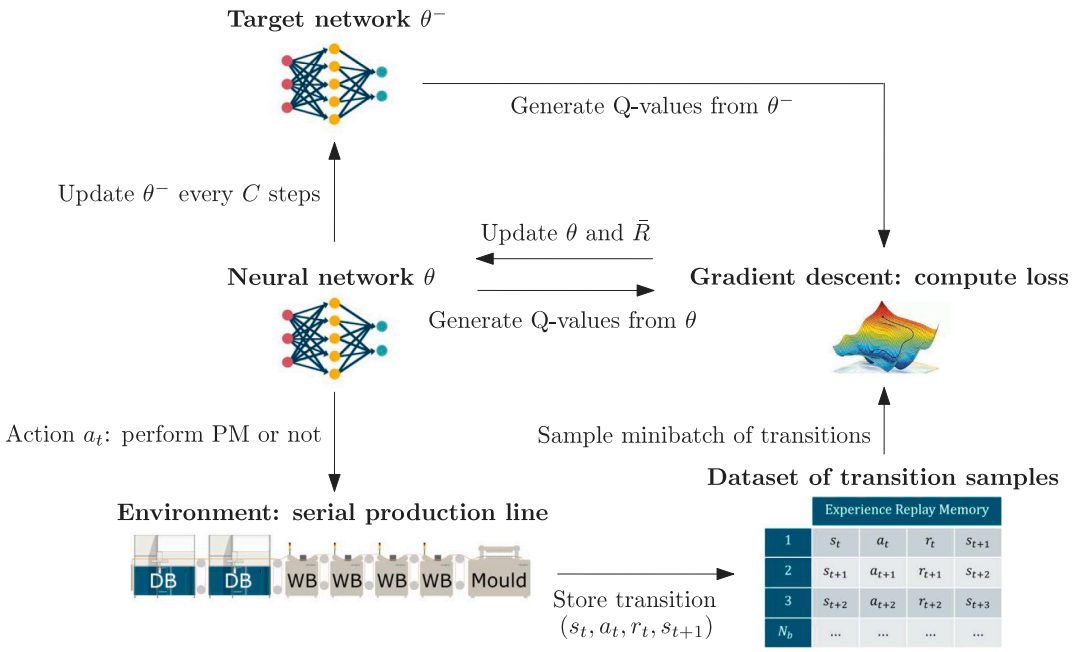


Fig. 11. Flow chart of ADQN.

Table 3
Hyper-parameters for Algorithm 3.

Notation	Description	Value
N_b	Capacity of the replay memory	200.000
b	Size of the mini-batch of samples	64
η	Parameter to control the adjustment of the average-reward	100
C	Step interval to change target network θ^- weights to the weights of neural network θ	5
α	Parameter to control how much to change target network θ^- weights to the weights of neural network θ	0.1
U	Step interval to train the neural network θ	4
K	Step interval to change the random action	1000
ϵ_0	Initial value for ϵ, defining the threshold to take a random or greedy action	1.0
ϵ_d	Parameter to control how much to decrease ϵ	0.999999
ϵ_f	Minimum value for ϵ	0.05
L	Number of hidden layers in the neural network	2
h_l	Number of hidden units in layer l, $l = 1, \ldots, L$	64

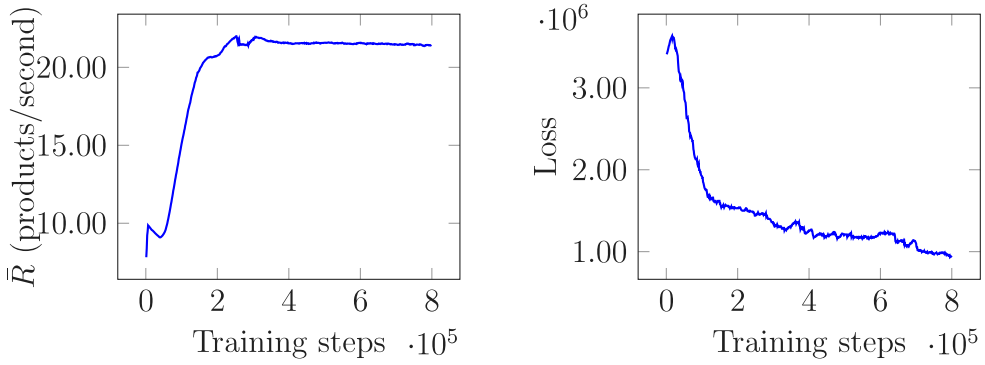
Table 4
Real-world production line instances.

Properties		Machines						
		M_1	M_2	M_3	M_4	M_5	M_6	M_7
Speed v_i^{max} (products/second)	μ	25.57	25.59	26.35	26.44	26.13	26.34	26.66
	σ	0.23	0.27	0.13	0.17	0.16	0.19	0.22
Up-time X_u (seconds)	μ	1114.44	1255.20	2001.08	1843.70	1825.65	1911.86	944.68
	σ	1646.76	2295.83	3695.61	3296.69	3520.86	3789.51	1638.07
Down-time X_d (seconds)	μ	68.53	68.45	103.22	73.33	103.46	76.13	105.14
	σ	342.78	404.00	699.01	551.24	635.98	546.51	429.85
PM duration d_{PM} (seconds)	μ							3544.72
	σ							601.43
		Buffers						
		B_1	B_2	B_3	B_4	B_5	B_6	
Buffer capacity B_i^{max} (products)		25.000	9000	25.000	25.000	25.000	120.000	

6.2.3. ADQN algorithm

In the PM problem considered in this study, the state space is characterized by the state of the machines in the line, $s_i(t)$, the buffer levels $l_j(t)$ and the active number of products left in the flexibility window until the PM limit is reached, $r(t)$. Accordingly, the decisions to be made are machines and buffers to include, the step size of the buffer thresholds and the step intervals for the flexibility window.

Preliminary analysis showed that including only the last two machines and the buffer in-between, similar to the OBT algorithm, is enough to capture most of the behavior of the production line for this



(a) Average reward $\bar{R}$ (b) Training loss

Fig. 12. Average reward $\bar{R}$ and loss during training.

particular PM problem. Adding more machines and buffers resulted only in a negligible improvement. Including the last two machines and the final buffer only makes it possible to graphically show the trained policy. Therefore, for this PM problem, the choice is made to include the final two machines and the buffer in-between only. Of course, in case maintenance would take place on other machines in the line as well, it may be important to also include these machines and buffers in the state space. Then, the step size of the buffer thresholds is chosen to be similar as the OBT algorithm, which is 5000 products. The step intervals for the flexibility window are taken to be 10.000 products. As an example, for a flexibility window of 400.000 products, a bin of 40 flexibility intervals will be explored.

In addition, the values for the hyper-parameters in Table 3 need to be selected. An overview of the parameters is shown in the same table under the final column. The selection has been done by means of manual tuning.

The size of the input layer for the neural network is equal to the size of the state space. In case of the 2 final machines, one buffer, 24 buffer thresholds and 40 flexibility intervals, the total size equals $24 + 40 + 4 + 4 = 72$. Here, both the last and the second-last machine can have 4 different machine states. The size of the output layer is equal to the number of possible actions, which is equal to 2 for this PM problem. The optimizer used for the gradient descent step is Adam [48], with default settings, except for the learning rate $\rho = 1 \times 10^{-6}$. The algorithm is implemented in PyTorch (1.13.0 with CUDA 11.6), the discrete event simulation is implemented in C# (.NET 5) and both run on a PC with CPU: AMD Ryzen™ Threadripper™ 3970X Processor, and GPU: 2 × Nvidia 2080 Ti. The total training time for each production line instance is set to be 1 h.

6.3. Training process

To evaluate the training process, the average reward and the trained policy can be examined. An example of the average reward $\bar{R}$ during training is shown in Fig. 12(a). Since $\bar{R}$ is initialized at zero, the average reward increases during the first part of learning and converges at the end. Once the average reward has converged, it does not necessarily mean that learning has stopped. This can be seen in Fig. 12(b), which shows that the training loss is still decreasing in the last phase of learning as well.

The trained policy can be examined by analyzing the buffer thresholds for each point in the flexibility window for every machine state-combination, similar to Fig. 9 in Section 5. This is possible since the choice is made to only include the final two machines and the buffer in-between in the state. Therefore, such a 2D plot makes it possible to represent the entire state space. An example of a policy is depicted in Fig. 13. Here, a policy for the smallest flexibility window of 100.000 products is shown. Different from the figure of the OBT policy from Section 5, buffer thresholds are now defined for every single point in the flexibility window. This creates unique regions per machine-state combination, which defines an area under which it is a good time (in terms of the number of products left in the flexibility window) to initiate maintenance. The main observation from the policy shown in Fig. 13 is the tendency to postpone the execution of maintenance at the beginning of the flexibility window. Then, when the end of the flexibility window is approaching, acceptance to perform maintenance at higher buffer levels increases. The behavior of the shown policy is the intended behavior as performing maintenance already at a very high buffer level at the start of the flexibility window would not make sense.

It is interesting to notice the significant influence of the machine states on the policy. In case machine M_{N-1} is up and machine M_N is down, the policy shows it is best to perform PM only when the buffer level is extremely low and preferably at the end of the window. This makes sense, as performing PM outside of the advised region for this machine state pair, at higher buffer levels or earlier in the flexibility window, might results in significant congestion upstream in the assembly line. Also, in case machine M_{N-1} is starved and machine M_N is up, PM may be performed at high buffer levels. This is intended behavior, as the machine is in a state where it cannot produce and therefore is unable to fill the buffer in case PM would be initiated. The buffer levels for this state combination are higher compared to a similar state, where machine M_{N-1} is down and machine M_N is up. This might indicate that the starved state of machine M_{N-1} usually lasts longer than a down state, resulting in a more opportune moment to initiate PM.

6.4. Results

To determine the performance of the learned policy, it is evaluated against two other policies: (1) the current policy at the case study company (practitioner policy) and (2) the policy obtained by the OBT algorithm from Section 5.1. As mentioned briefly in Section 6.2, the practitioner policy is the policy currently employed by the factory. It is a time-based policy, where PM is performed every 12 h. For each of the 11 production lines, a policy is trained. In addition, a policy is trained for each of the 6 flexibility levels. Results are averaged over all 11 instances. Fig. 14 shows the improvement in throughput of the policies obtained by the OBT and ADQN algorithm with respect to the practitioner policy; both outperform the practitioner policy. By fully utilizing the flexibility window, the ADQN agent is able to generate policies that improve the long-run average throughput and outperform the OBT policy by an extra 0.15%. Additionally, the more flexibility is added, the more improvement is realized. This makes sense as larger flexibility windows result in higher chances of good opportunities for executing maintenance. Typically, the company considered in this use case, Nexperia, produces hundreds of billions of products per year.

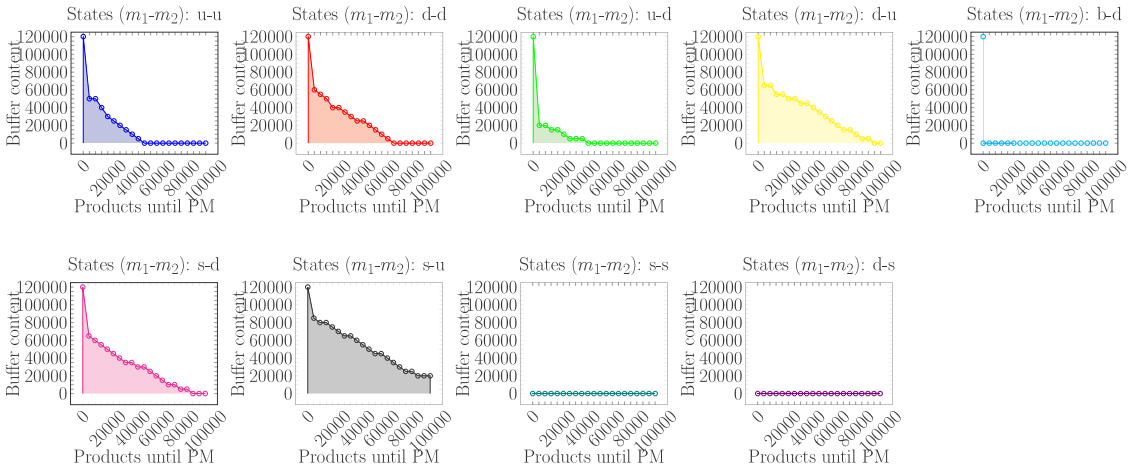


Fig. 13. Policy describing what action to take, given a pair of machines with a buffer in-between. The decision depends on the buffer content, depicted on the y-axis, and how many products are left in the flexibility window until the limit is reached, depicted on the x-axis. The shaded areas shows the region under which it is allowed to perform PM.

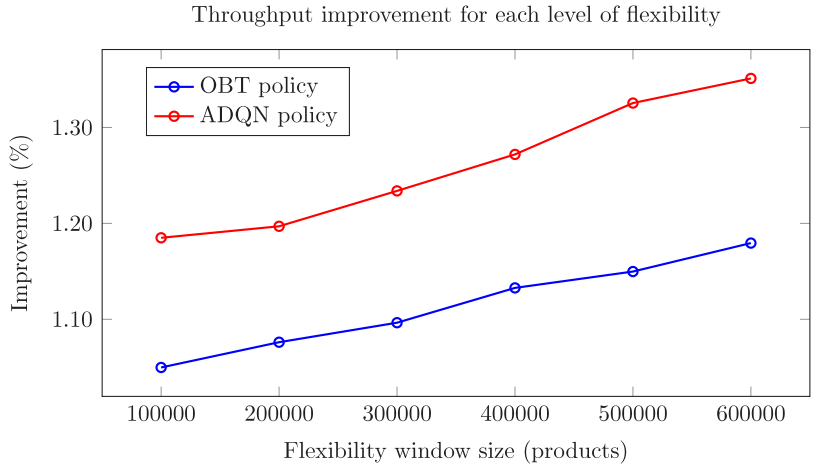


Fig. 14. Improvement in throughput of the OBT and ADQN policy, w.r.t. the practitioner time-based policy.

Being able to improve by more than 1% means that multiple additional billions of products can be produced, without the need to invest in additional assembly lines, factory space and personnel.

To better understand how these improvements are realized, additional analyses on key statistics are performed. An increase in throughput due to better timing of maintenance can result from lower blocking states of the upstream machines in the production line. Fig. 15(a) shows the reduction of the blocking states percentages with respect to the practitioner policy, for each machine in the line. The smallest and largest flexibility levels are presented. It shows that ADQN is able to reduce the total blocking state percentages for each machine further than the OBT policy. Additionally, the more flexibility is added, the larger the reductions. Fig. 15(c) shows the average buffer content at the exact moment that maintenance is initiated, i.e., the content of the buffer at the start of a maintenance activity. ADQN is able to reduce this start content further than OBT, which most likely contributes to the increase in throughput. Similarly, Fig. 15(d) shows the number of times the buffer reaches its maximum capacity during a maintenance activity. Again, ADQN outperforms the OBT policy, which might explain the increase in throughput and further reduction of the blocking states of the other machines in the production line.

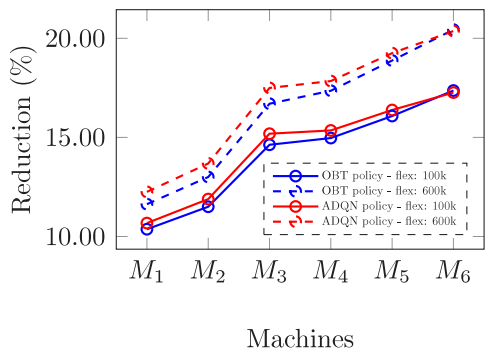
Interestingly, Fig. 15(b) shows that the overall reduction of the average buffer content of the final buffer is reduced less by ADQN, compared to the OBT policy. Intuitively, this seems strange. However, the same figure also displays the same statistic exclusively for the period during maintenance. For this period, the average buffer content is indeed reduced more by ADQN than by OBT. The reason for the further reduction for the overall buffer content by OBT could be explained by the combination of reduced blocking states, lower initial buffer content at execution and higher throughput. Due to the higher throughput, it might be the case that the work in progress (WIP) in the final buffer is slightly higher with ADQN, resulting in a smaller reduction compared to OBT.

6.5. Sensitivity analysis & managerial insights

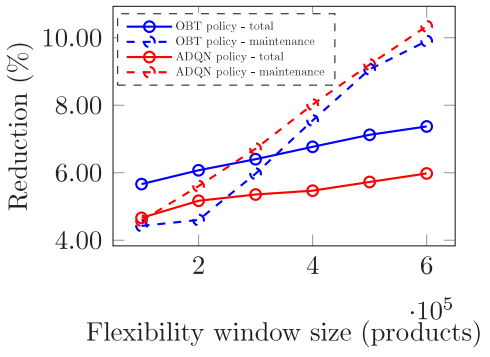
To better understand the effect the assembly characteristics have on the results and to provide more insights into the benefits of the improvements for the shop floor, more detailed experiments are carried out. First, the previous throughput results are analyzed for individual assembly lines. Then, the impact of the buffer size and the duration of PM on the improvements is examined.

6.5.1. Line to line comparison

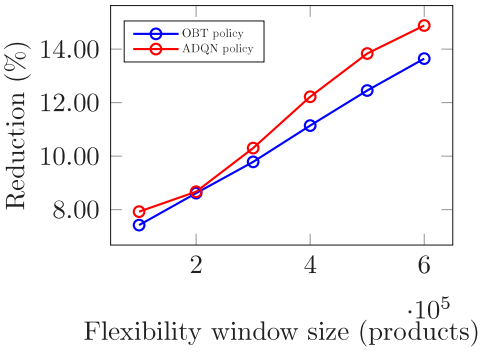
The throughput results in Fig. 14 are averaged over all 11 assembly lines, as stated in Table 4. Although they constitute the same number of buffers and machines, their performance can differ significantly. This arises from the variations in the speed and up and down behavior of each individual machine in the production line. We illustrate the impact of these characteristics in Fig. 16, where the results of the throughput improvement of ADQN per assembly line, for the lowest and highest flexibility level are depicted.



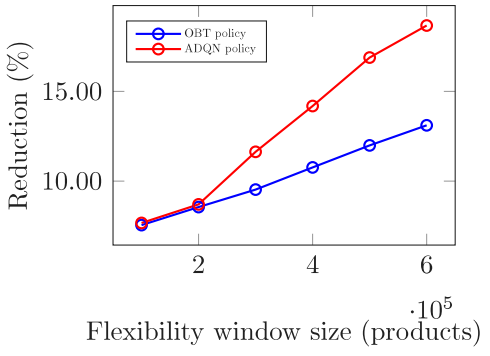
(a) Reduction in % of fraction of time machine M_i is blocked for two different levels of flexibility



(b) Average buffer content reduction during maintenance and during operation + maintenance together



(c) Buffer content at the moment maintenance is started for each level of flexibility



(d) Occurrences of full buffer during maintenance for each level of flexibility

Fig. 15. Production line statistics.

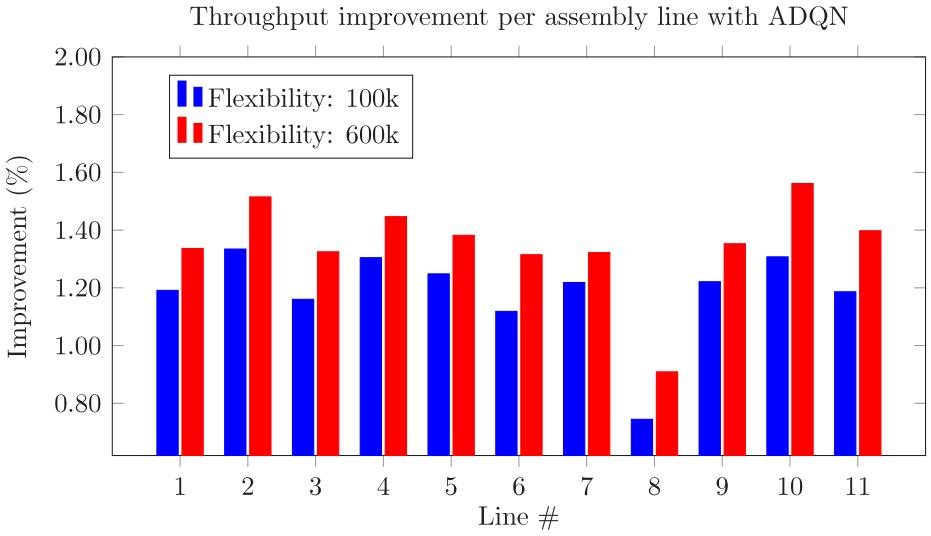


Fig. 16. Improvement in throughput of the ADQN policy, w.r.t. the practitioner time-based policy, for each assembly line instance.

Interestingly, there is a large difference in throughput improvement, differing almost by 0.60% between the worst and best performing assembly lines, line 8 and 10, respectively. To understand the origin of these differences, the characteristics of these worst and best performing assembly lines, together with an average assembly line (e.g. line 5), is depicted in Fig. 17. The differences are clear and profound. The speed of machines is higher for line 10, compared to line 8, except for the last machine in the line, which coincidentally is the PM machine. Since the last machine in line 10 has a lower speed compared to its upstream machines, the average content of the buffer before the last machine will be higher compared to line 8, where the opposite trend in machine speed is observed. In addition, in case the last machine stops production due to either a down state or a planned PM, the buffer in line 10 will be filled more quickly compared to line 8, resulting in faster congestion of upstream machines. Therefore, logically, improvements for line 8 are lower since the impact of the algorithm is less pronounced

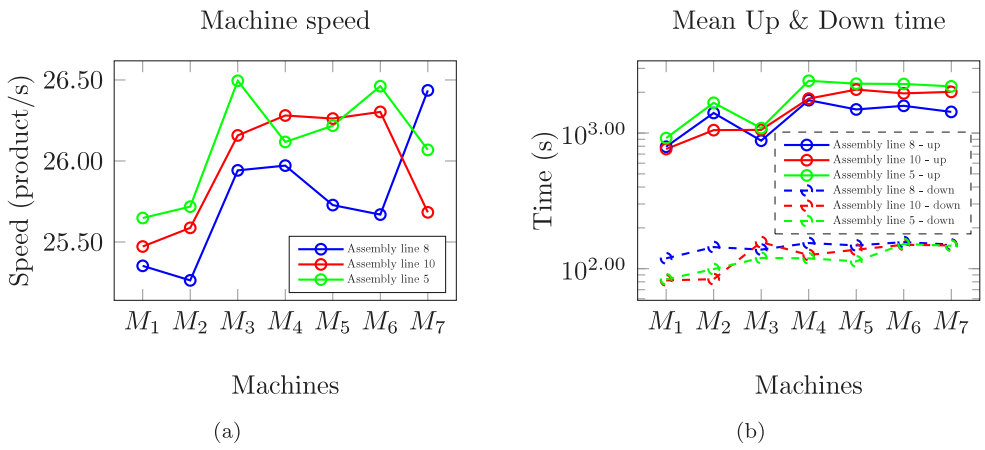


Fig. 17. Assembly line characteristics for assembly lines #8 and #10.

for cases where the buffer is already at lower levels and the fill rate of the buffer is lower. Line 5 has a similar behavior for the machine speed as line 10, although the difference between the speed of the last machine and the speed of the machines upstream is larger for line 10. In addition, the up times are larger and the down times are slightly lower for line 5, compared to line 10. Larger up times and lower down times should result in more throughput improvements with an algorithm such as ADQN, as the buffer can be filled more rapidly during PM. However, larger throughput improvements are observed for line 10, which indicate that the machine speed seems to be more important, as this is in favor of line 5.

The comparison of assembly lines highlights the significance of a well-balanced assembly line and the effect that maintenance can have on the level of improvement. The similarity between assembly lines in terms of speed and up-and-down behavior enhances the predictability of maintenance strategies' effectiveness.

6.5.2. Varying buffer capacity

Another interesting element to study is the effect of the maximum buffer capacity of the buffer prior to the PM machine on the throughput. Fig. 18(a) shows that the throughout improves with increasing buffer capacity. Nonetheless, the rate of improvement slows down as buffer capacity increases. This trend is anticipated, as an increase in buffer size would lead to less congestion upstream due to the downtime of machines caused by either corrective maintenance or preventive maintenance.

Fig. 18(b) illustrates the comparison of the OBT and ADQN algorithms with the practitioner policy for each level of buffer capacity. Interestingly, smaller buffer sizes result in greater improvements. This observation suggests that larger buffer capacities have a lower impact on algorithms that employ buffer size as an input for PM decision making. In contrast, when the buffer capacities are small, algorithms are more beneficial, as upstream machines can become congested more quickly.

The differences in improvement between the lowest and highest flexibility windows are larger for lower levels of buffer capacity and decrease when the buffer capacity increases. This trend is observed for both the ADQN and OBT algorithms. This suggests that the benefit that an increase in buffer capacity provides makes up for the increase in flexibility window, i.e., it becomes less important to have more flexibility since there is more space to absorb the impact of PM with a larger buffer capacity. Interestingly, the differences in throughput improvement between ADQN and OBT for both the lower and higher flexibility windows increase as the buffer capacity increases. This indicates that the ADQN algorithm is able to utilize the increase in buffer capacity better. Most likely it is better able to define regions of opportunity as illustrated in Fig. 13. Also, the ADQN algorithm with the smallest flexibility window performs almost as good as the OBT algorithm with the highest flexibility window. The learning for managers is that with higher buffer capacities, the decision when to perform PM can be postponed further towards the end of the flexibility window. This makes sense as the chances of having an empty enough buffer to compensate for the impact of PM increases. Increasing buffer size may seem like a simple decision. However, it may not always be feasible to add a larger buffer to an assembly line due to the associated costs and limited physical space available on the shop floor. Consequently, it is crucial for the practitioners to employ intelligent algorithms to guide PM decision making.

6.5.3. Varying maintenance duration

The duration of PM is a variable that can significantly impact throughput, and therefore warrants careful examination. To investigate the effect of PM duration, we vary the duration of PM listed in Table 4. This is done by shifting the mean duration from Table 4 by adding or subtracting a fixed value to the mean. We use increments of 10 min. The impact on throughput is shown in Fig. 19. As expected, longer PM duration results in decreased throughput (Fig. 19(a)). Conversely, the throughput improvements resulting from the OBT and ADQN policies increase with longer PM duration 19(b). This is intuitive, since longer PM duration increases the likelihood of congestion on upstream machines, ultimately leading to reduced throughput. Therefore, the importance of initiating PM at the appropriate moment is amplified, resulting in greater throughput improvements when using a policy generated by either OBT or ADQN. Notably, the rate of improvement increases with PM duration, and the difference in improvement between the lowest and highest flexibility windows also increases with PM duration. These trends suggest that flexibility windows become more critical as PM duration increases, a finding consistent with our previous analysis of buffer capacity, which indicated that flexibility windows become increasingly important as buffer capacity decreases. From a managerial perspective, there is a significant advantage in avoiding PM from exceeding the average duration, as the impact is more substantial in comparison to increasing the maximum buffer capacities.

7. Conclusions and future work

In this study, a problem is considered where maintenance must be scheduled on the last machine of a serial production line. Three elements may determine the optimal timing to execute maintenance: (1) the production state of the machines, (2) the content of the buffers and (3) the flexibility given to the maintenance activity. These three elements provide a unique problem which has not been studied before in the context of scheduling maintenance on a single machine of a serial production line. Given these three elements, the state space of the

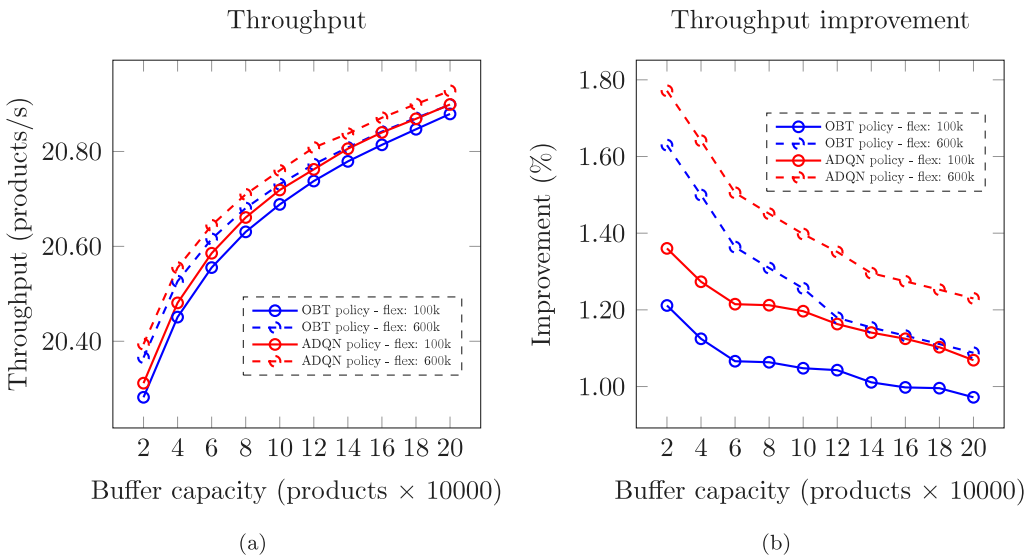


Fig. 18. Throughput and improvement in throughput of the ADQN and OBT policy, w.r.t. the practitioner time-based policy for different levels of buffer capacity.

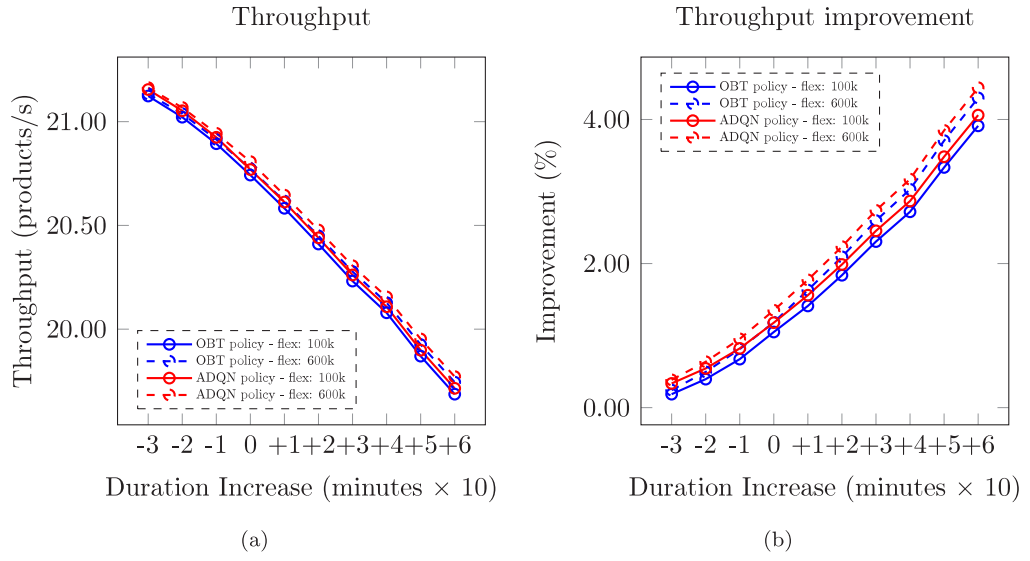


Fig. 19. Throughput and improvement in throughput of the ADQN policy, w.r.t. the practitioner time-based policy for different levels of PM duration.

problem quickly explodes. For this reason, a novel deep reinforcement learning approach, named ADQN, is presented which aims to find optimal policies for the long-run average reward. Numerical experiments are performed in a discrete event simulation model of the production line consisting of multiple machines and buffers in series that uses real-world data as input. The experiments show that the ADQN policy outperforms both the time-based policy currently employed by the factory and a benchmark policy.

This study considers maintenance on the last machine of the assembly line. For future work, it would be interesting to extend the problem to the more general case of performing maintenance on any machine of the assembly line. At the moment, deterioration is not modeled as part of the problem. If deterioration would also be considered in the problem, more improvements could be realized since maintenance could be initiated based on the deterioration which would result in higher availability of the machine. It would be interesting to study the effect of such deterioration models. In addition, maintenance of different types on the other machines in the production line might be considered as well, such as maintenance activities that flow through the line from one machine to the next. Extending the problem to multiple parallel production lines with resource constraints would also be an interesting and challenging research topic.

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

Funding: This work is partially supported by the ECSEL Joint Undertaking under grant agreement number 101007311 (IMOCO4.E).

References

[1] Allahverdi A. The third comprehensive survey on scheduling problems with setup times/costs. European J Oper Res 2015;246(2):345–78.

[2] Budai G, Dekker R, Nicolai RP. Maintenance and production: A review of planning models. In: Complex system maintenance handbook. London: Springer London; 2008, p. 321–44. http://dx.doi.org/10.1007/978-1-84800-011-7_13.

[3] Geurtsen M, Didden JB, Adan J, Atan Z, Adan I. Production, maintenance and resource scheduling: A review. European J Oper Res 2022. http://dx.doi.org/10.1016/j.ejor.2022.03.045, URL https://www.sciencedirect.com/science/article/pii/S0377221722002673.

[4] Van der Duyn Schouten F, Vanneste S. Maintenance optimization of a production system with buffer capacity. European J Oper Res 1995;82(2):323–38. http://dx.doi.org/10.1016/0377-2217(94)00267-G, URL https://www.sciencedirect.com/science/article/pii/037722179400267G.

[5] Kyriakidis E, Dimitrakos T. Optimal preventive maintenance of a production system with an intermediate buffer. European J Oper Res 2006;168(1):86–99. http://dx.doi.org/10.1016/j.ejor.2004.01.052, URL https://www.sciencedirect.com/science/article/pii/S0377221704003091.

[6] Karamatsoukis C, Kyriakidis E. Optimal maintenance of two stochastically deteriorating machines with an intermediate buffer. European J Oper Res 2010;207(1):297–308. http://dx.doi.org/10.1016/j.ejor.2010.04.022, URL https://www.sciencedirect.com/science/article/pii/S0377221710003413.

[7] Meller RD, Kim DS. The impact of preventive maintenance on system cost and buffer size. European J Oper Res 1996;95(3):577–91. http://dx.doi.org/10.1016/0377-2217(95)00313-4, URL https://www.sciencedirect.com/science/article/pii/0377221795003134.

[8] Wang H. A survey of maintenance policies of deteriorating systems. European J Oper Res 2002;139(3):469–89. http://dx.doi.org/10.1016/S0377-2217(01)00197-7, URL https://www.sciencedirect.com/science/article/pii/S0377221701001977.

[9] Liu Q, Dong M, Frank Chen F, Liu W, Ye C. Multi-objective imperfect maintenance optimization for production system with an intermediate buffer. J Manuf Syst 2020;56:452–62. http://dx.doi.org/10.1016/j.jmsy.2020.07.002, URL https://www.sciencedirect.com/science/article/pii/S0278612520301126.

[10] Fitouhi M-C, Nourelfath M, Gershwin SB. Performance evaluation of a two-machine line with a finite buffer and condition-based maintenance. Reliab Eng Syst Saf 2017;166:61–72. http://dx.doi.org/10.1016/j.ress.2017.03.034, URL https://www.sciencedirect.com/science/article/pii/S0951832017303733, Reliability and Performance of Multi-State Systems.

[11] Zhou Y, Zhang Z. Optimal maintenance of a series production system with two multi-component subsystems and an intermediate buffer; [optymalna strategia utrzymania ruchu dla seryjnego systemu produkcji złożonego z dwóch podsystemów wieloskładnikowych oraz buforu pośredniego]. Eksploatacja I Niezawodnosc 2015;17(2):314–25. http://dx.doi.org/10.17531/ein.2015.2.20, URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84925945296&doi=10.17531%2fein.2015.2.20&partnerID=40&md5=10043bf4a833f4ea0c95cc441acb67db, Cited by: 6; All Open Access, Gold Open Access.

[12] Wang X, Qi C. Multi-agent reinforcement learning based maintenance policy for a resource constrained flow line system. J Intell Manuf 2016;27:325–33. http://dx.doi.org/10.1007/s10845-013-0864-5.

[13] Li B, Zhou Y. Multi-component maintenance optimization: an approach combining genetic algorithm and multiagent reinforcement learning. In: 2020 Global reliability and prognostics and health management. 2020, p. 1–7. http://dx.doi.org/10.1109/PHM-Shanghai49105.2020.9280997.

[14] Gu X, Guo W, Jin X. Performance evaluation for manufacturing systems under control-limit maintenance policy. J Manuf Syst 2020;55:221–32. http://dx.doi.org/10.1016/j.jmsy.2020.03.003, URL https://www.sciencedirect.com/science/article/pii/S0278612520300364.

[15] Arab A, Ismail N, Lee LS. Maintenance scheduling incorporating dynamics of production system and real-time information from workstations. J Intell Manuf 2013;24(4):695–705. http://dx.doi.org/10.1007/s10845-011-0616-3, URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84880285817&doi=10.1007%2fs10845-011-0616-3&partnerID=40&md5=3333536dfd5b1da5883084fe0ea6ff18, Cited by: 36.

[16] Zequeira RI, Valdes JE, Berenguer C. Optimal buffer inventory and opportunistic preventive maintenance under random production capacity availability. Int J Prod Econ 2008;111(2):686–96. http://dx.doi.org/10.1016/j.ijpe.2007.02.037, URL https://www.sciencedirect.com/science/article/pii/S0925527307001557, Special Section on Sustainable Supply Chain.

[17] Magnanini MC, Tolio T. Switching- and hedging- point policy for preventive maintenance with degrading machines: application to a two-machine line. Flex Serv Manuf J 2020;32(2):241–71. http://dx.doi.org/10.1007/s10696-019-09370-7, URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85073998192&doi=10.1007%2fs10696-019-09370-7&partnerID=40&md5=ac07f4387eaad3df0b9519844aa92504, Cited by: 5; All Open Access, Green Open Access.

[18] Langer R, Li J, Biller S, Chang Q, Huang N, Xiao G. Simulation study of a bottleneck-based dispatching policy for a maintenance workforce. Int J Prod Res 2010;48(6):1745–63. http://dx.doi.org/10.1080/00207540802555769.

[19] Li L, Chang Q, Ni J. Data driven bottleneck detection of manufacturing systems. Int J Prod Res 2009;47(18):5019–36. http://dx.doi.org/10.1080/00207540701881860.

[20] Gopalakrishnan M, Skoogh A, Laroque C. Buffer utilization based scheduling of maintenance activities by a shifting priority approach - a simulation study. In: 2016 Winter simulation conference. 2016, p. 2797–808. http://dx.doi.org/10.1109/WSC.2016.7822316.

[21] Lu L, Liu Y, Li J, Chang C, Biller S, Xiao G. A real-time maintenance scheduling policy in serial production lines. In: 2011 9th world congress on intelligent control and automation. 2011, p. 36–41. http://dx.doi.org/10.1109/WCICA.2011.5970578.

[22] Nicolai RP, Dekker R. Optimal maintenance of multi-component systems: A review. In: Complex system maintenance handbook. London: Springer London; 2008, p. 263–86. http://dx.doi.org/10.1007/978-1-84800-011-7_11.

[23] Ab-Samat H, Kamaruddin S. Opportunistic maintenance (OM) as a new advancement in maintenance approaches: A review. J Qual Maint Eng 2014;20(2). http://dx.doi.org/10.1108/JQME-04-2013-0018, URL https://www.emerald.com/insight/content/doi/10.1108/JQME-04-2013-0018/full/html.

[24] Werbińska-Wojciechowska S. Preventive maintenance models for technical systems. Springer Ser Reliab Eng 2019;21–100. http://dx.doi.org/10.1007/978-3-030-10788-8_2, URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85061117553&doi=10.1007%2f978-3-030-10788-8_2&partnerID=40&md5=c69c1fcd83db2899e1c873666c4d761c.

[25] van der Duyn Schouten F, Vanneste S. Analysis and computation of (n, N)-strategies for maintenance of a two-component system. European J Oper Res 1990;48(2):260–74. http://dx.doi.org/10.1016/0377-2217(90)90379-P, URL https://www.sciencedirect.com/science/article/pii/037722179090379P.

[26] Laggoune R, Chateauneuf A, Aissani D. Opportunistic policy for optimal preventive maintenance of a multi-component system in continuous operating units. Comput Chem Eng 2009;33(9):1499–510. http://dx.doi.org/10.1016/j.compchemeng.2009.03.003, URL https://www.sciencedirect.com/science/article/pii/S0098135409000696.

[27] Sarker BR, Faiz TI. Minimizing maintenance cost for offshore wind turbines following multi-level opportunistic preventive strategy. Renew Energy 2016;85:104–13. http://dx.doi.org/10.1016/j.renene.2015.06.030, URL https://www.sciencedirect.com/science/article/pii/S0960148115300549.

[28] Gunn EA, Diallo C. Optimal opportunistic indirect grouping of preventive replacements in multicomponent systems. Comput Ind Eng 2015;90:281–91. http://dx.doi.org/10.1016/j.cie.2015.09.013, URL https://www.sciencedirect.com/science/article/pii/S0360835215003939.

[29] Zhou X, Huang K, Xi L, Lee J. Preventive maintenance modeling for multi-component systems with considering stochastic failures and disassembly sequence. Reliab Eng Syst Saf 2015;142:231–7. http://dx.doi.org/10.1016/j.ress.2015.05.005, URL https://www.sciencedirect.com/science/article/pii/S0951832015001465.

[30] Ferreira Neto WA, Cavalcante CA, Santos AC, Araújo LH, Alberti AR, Lima HB. An inspection policy for shredder equipment used in steel production lines considering buffer level and operating time. J Manuf Syst 2021;60:640–51. http://dx.doi.org/10.1016/j.jmsy.2021.06.013, URL https://www.sciencedirect.com/science/article/pii/S0278612521001370.

[31] Wu T, Ma X, Yang L, Zhao Y. Proactive maintenance scheduling in consideration of imperfect repairs and production wait time. J Manuf Syst 2019;53:183–94. http://dx.doi.org/10.1016/j.jmsy.2019.09.011, URL https://www.sciencedirect.com/science/article/pii/S0278612519300834.

[32] Yang L, Zhao Y, Peng R, Ma X. Opportunistic maintenance of production systems subject to random wait time and multiple control limits. J Manuf Syst 2018;47:12–34. http://dx.doi.org/10.1016/j.jmsy.2018.02.003, URL https://www.sciencedirect.com/science/article/pii/S0278612518300141.

[33] Huang J, Chang Q, Arinez J. Deep reinforcement learning based preventive maintenance policy for serial production lines. Expert Syst Appl 2020;160:113701. http://dx.doi.org/10.1016/j.eswa.2020.113701, URL https://www.sciencedirect.com/science/article/pii/S095741742030525X.

[34] Valet A, Altenmüller T, Waschneck B, May MC, Kuhnle A, Lanza G. Opportunistic maintenance scheduling with deep reinforcement learning. J Manuf Syst 2022;64:518–34. http://dx.doi.org/10.1016/j.jmsy.2022.07.016, URL https://www.sciencedirect.com/science/article/pii/S0278612522001285.

[35] Kuhnle A, Jakubik J, Lanza G. Reinforcement learning for opportunistic maintenance optimization. Prod Eng 2019;13(1):33–41. http://dx.doi.org/10.1007/s11740-018-0855-7, URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85055991618&doi=10.1007%2fs11740-018-0855-7&partnerID=40&md5=c710a9a712200b8a697ba8f739a0533d, Cited by: 38.

[36] Zhang N, Qi F, Zhang C, Zhou H. Joint optimization of condition-based maintenance policy and buffer capacity for a two-unit series system. Reliab Eng Syst Saf 2022;219:108232. http://dx.doi.org/10.1016/j.ress.2021.108232, URL https://www.sciencedirect.com/science/article/pii/S0951832021007109.

[37] Zhou B, Yu J, Shao J, Trentesaux D. Bottleneck-based opportunistic maintenance model for series production systems. J Qual Maint Eng 2015;21(1):70–88. http://dx.doi.org/10.1108/JQME-09-2013-0059, URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84923929158&doi=10.1108%2fJQME-09-2013-0059&partnerID=40&md5=ea120c565d916a4770ccf31de33dc6d2, Cited by: 21.

[38] Chang Q, Ni J, Bandyopadhyay P, Biller S, Xiao G. Maintenance opportunity planning system. J Manuf Sci Eng 2007;129(3):661–8. http://dx.doi.org/10.1115/1.2716713, URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-34547416651&doi=10.1115%2f1.2716713&partnerID=40&md5=7b8659c7a17ef59434ac511379076fca, Cited by: 88.

[39] Gu X, Jin X, Ni J. Prediction of passive maintenance opportunity windows on bottleneck machines in complex manufacturing systems. Trans ASME, J Manuf Sci Eng 2015;137(3). http://dx.doi.org/10.1115/1.4029906, URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84924675023&doi=10.1115%2f1.4029906&partnerID=40&md5=2cfab077f3794dc81f124b6d6fdd00fe, Cited by: 42.

[40] Gu X, Jin X, Guo W, Ni J. Estimation of active maintenance opportunity windows in Bernoulli production lines. J Manuf Syst 2017;45:109–20. http://dx.doi.org/10.1016/j.jmsy.2017.08.005, URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85029688456&doi=10.1016%2fj.jmsy.2017.08.005&partnerID=40&md5=26c4a20b2a7f33148a89794759019abb, Cited by: 21.

[41] Geurtsen M, Atan Z, Adan IJ. Dynamic scheduling of maintenance by a reinforcement learning approach - a semiconductor simulation study. In: 2022 Winter simulation conference. 2022, URL https://ieeexplore.ieee.org/document/10015402/.

[42] Sutton RS, Barto AG. Reinforcement learning: An introduction. MIT Press; 2018.

[43] Watkins CJCH. Learning from delayed rewards [Ph.D. thesis], King's College, Oxford; 1989.

[44] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, et al. Playing atari with deep reinforcement learning. 2013, http://dx.doi.org/10.48550/ARXIV.1312.5602, URL https://arxiv.org/abs/1312.5602.

[45] Hasselt Hv, Guez A, Silver D. Deep reinforcement learning with double Q-learning. In: Proceedings of the thirtieth AAAI conference on artificial intelligence. AAAI Press; 2016, p. 2094–100.

[46] Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. 2017, http://dx.doi.org/10.48550/ARXIV.1707.06347, URL https://arxiv.org/abs/1707.06347.

[47] Wan Y, Naik A, Sutton RS. Learning and planning in average-reward Markov decision processes. In: Meila M, Zhang T, editors. Proceedings of the 38th international conference on machine learning. Proceedings of machine learning research, vol. 139, PMLR; 2021, p. 10653–62, URL https://proceedings.mlr.press/v139/wan21a.html.

[48] Kingma DP, Ba J. Adam: A method for stochastic optimization. 2014, http://dx.doi.org/10.48550/ARXIV.1412.6980, URL https://arxiv.org/abs/1412.6980.