Journal of Manufacturing Systems 55 (2020) 1–14

Contents lists available at ScienceDirect

Journal of Manufacturing Systems

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

Technical Paper

Petri-net-based dynamic scheduling of flexible manufacturing system via deep reinforcement learning with graph convolutional network

Liang Hu[a], Zhenyu Liu[a,*], Weifei Hu[a], Yueyang Wang[b], Jianrong Tan[a], Fei Wu[c]

[a] State Key Lab of CAD&CG, Zhejiang University, Hangzhou, China
[b] School of Big Data & Software Engineering, Chongqing University, Chongqing, China
[c] College of Computer Science and Technology, Zhejiang University, Hangzhou, China

ARTICLE INFO

Keywords:
Dynamic scheduling
Petri nets
Deep reinforcement learning
Graph convolutional networks
Digital twin

ABSTRACT

To benefit from the accurate simulation and high-throughput data contributed by advanced digital twin technologies in modern smart plants, the deep reinforcement learning (DRL) method is an appropriate choice to generate a self-optimizing scheduling policy. This study employs the deep Q-network (DQN), which is a successful DRL method, to solve the dynamic scheduling problem of flexible manufacturing systems (FMSs) involving shared resources, route flexibility, and stochastic arrivals of raw products. To model the system in consideration of both manufacturing efficiency and deadlock avoidance, we use a class of Petri nets combining timed-place Petri nets and a system of simple sequential processes with resources (S^3PR), which is named as the timed S^3PR. The dynamic scheduling problem of the timed S^3PR is defined as a Markov decision process (MDP) that can be solved by the DQN. For constructing deep neural networks to approximate the DQN action-value function that maps the timed S^3PR states to scheduling rewards, we innovatively employ a graph convolutional network (GCN) as the timed S^3PR state approximator by proposing a novel graph convolution layer called a Petri-net convolution (PNC) layer. The PNC layer uses the input and output matrices of the timed S^3PR to compute the propagation of features from places to transitions and from transitions to places, thereby reducing the number of parameters to be trained and ensuring robust convergence of the learning process. Experimental results verify that the proposed DQN with a PNC network can provide better solutions for dynamic scheduling problems in terms of manufacturing performance, computational efficiency, and adaptability compared with heuristic methods and a DQN with basic multilayer perceptrons.

1. Introduction

Scheduling is a classical topic in manufacturing control, which is of critical importance for improving the resource utilization and manufacturing efficiency of flexible manufacturing systems (FMSs) [1–4]. The recent applications of advanced digital twin technologies [5–8] in modern smart plants improve the simulation accuracy and the configuration flexibility of FMSs, which however raise the following questions regarding to the manufacturing scheduling: (1) How can we use actual manufacturing information acquired by digital twin technologies to optimize scheduling policies? (2) How can we realize real-time scheduling to address frequent variations in production plans owing to the customization of modern smart plants? (3) How can we ensure consistency of a scheduling policy between a simulation environment and a real plant? Answering these questions requires the development of smart dynamic scheduling methods.

As an important research field of artificial intelligence, reinforcement learning (RL) methods, especially deep reinforcement learning (DRL) methods, have witnessed significant breakthroughs in recent research endeavors [9,10], and they have been successfully adopted in several fields such as gaming, simulation, and cybernetics [11–13]. RL is applied to solve the problem of learning from interaction to achieve a goal. Generally, this problem is mathematically formalized by a Markov decision process (MDP) [14]. In an MDP, the learner (or the decision maker) is called the agent. The thing that the agent interacts with, comprising everything outside the agent, is called the environment. The agent selects actions and observes the rewards and new states presented by the environment continually. This process has the Markov property that the conditional probability distribution of future states depends only upon the present state and the action. RL enables the agent to learn optimal policy by maximizing an action-value function that estimates the long-term accumulated reward of each action in each state. As a

* Corresponding author.
E-mail address: liuzy@zju.edu.cn (Z. Liu).

https://doi.org/10.1016/j.jmsy.2020.02.004
Received 6 December 2019; Received in revised form 16 February 2020; Accepted 16 February 2020
Available online 21 February 2020
0278-6125/ © 2020 The Society of Manufacturing Engineers. Published by Elsevier Ltd. All rights reserved.

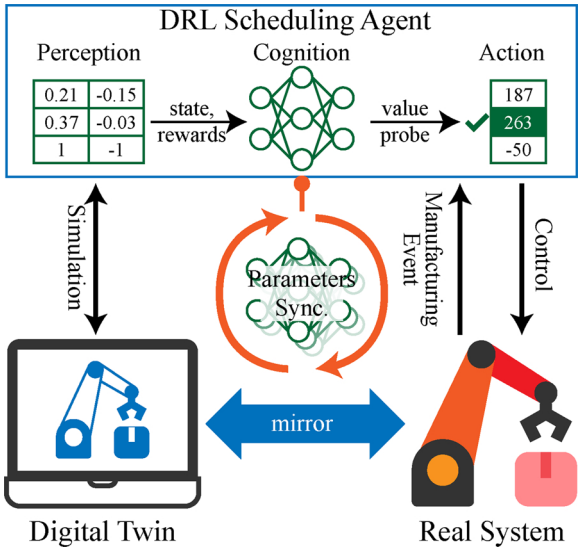

Fig. 1. Setup of DRL scheduling agent in smart plant and its digital twin.

classical research subject in the field of RL, the approximation of the action-value function is a critical point affecting the performance of RL. DRL introduces deep neural networks to implement the approximation and outperforms traditional RL methods when applied in complex systems.

DRL has appropriate characteristics that match the requirements of smart scheduling: (1) It can interact with the environment and use feedback data for policy optimization. (2) As with other machine learning methods, DRL involves computationally expensive offline training; nevertheless, it is efficient when executing. (3) The synchronization of the deep neural network's parameters exploits the consistency of the scheduling policy between the simulation environment and a real plant. DRL can be integrated with digital twin technologies of the smart plant to achieve a self-learning and self-optimizing system, which has been discussed by Waschneck et al. [15] as shown in Fig. 1. The digital twin of a manufacturing system offers a high-fidelity simulation and visualization environment which the DRL scheduling agent interacts with [16,17]. Therefore, the computationally expensive training of DRL can be conducted offline without interference on execution of the real system. When the training is converged, the agent is deployed to the real system for online scheduling. As the execution of deep neural networks is fast, the scheduling is real-time. The DRL scheduling agent with the self-adaptive capability consists of perception, cognition, and action modules [18,19]. By taking advantages of data-accessing and data-processing technologies of cyber-physical systems in modern smart plants [20–22], the FMS states can be mapped to the perception of the agent and the agent's actions can also be transformed to manufacturing instructions. Therefore, a closed-loop control is established. The digital twin records the manufacturing data and uses them to keep training the agent to remedy deviations between simulation and reality. The cognition of the agent is the deep neural network of DRL, and therefore the synchronization of the trainable parameters of the neural network guarantees the consistency of the scheduling policy between the real system and the digital twin.

Although some studies have used RL and DRL for job shop and FMS scheduling [15,18,23,24], there remains scope for improving the scheduling performance from the following two aspects: (1) Early RL methods did not work well owing to experience correlations caused by the observation and training mode, until Mnih et al. [9] proposed the deep Q-network (DQN) with two novel mechanisms, namely experience replay and double action-value functions, to reduce algorithm instabilities. DQN makes DRL practical and is further improved by other features such as prioritized experience replay [25] and dueling network architectures [26]. (2) Previous studies involved basic action-value function approximators, such as linear functions and multilayer perceptrons (MLP, or fully connected neural networks), which map system states to scheduling rewards. This is because the states of a manufacturing system are usually represented as regular tensors by arranging the state values of individual products, machines, and other elements; however, these tensors have no local spatial information which can be successfully handled by deep convolutional neural networks (CNNs). Furthermore, the constraint information in the manufacturing process, such as the sequential order of the operation steps and the usage/release of limited resources, is lost when constructing these tensors.

Compared to regular tensors, graphs or nets with topological structures that can model constraints and relationships are more suitable for modeling manufacturing processes. A Petri net is a common process modeling technique that tracks a system's states, dynamics, and constraints according to places, transitions, tokens, and arcs through a dual form of a graphical tool and a mathematical object [27–29]. A special class of Petri nets, namely a system of simple sequential processes with resources (S^3PR) [30], is used to model FMS processes. S^3PR can identify a deadlock in an FMS process, as its liveness is closely related to specific structures called siphons. There are well-developed methods for calculating siphons of S^3PR [31,32] and avoiding deadlocks on the basis of siphons [33,34]. Some researchers have appended time attributes to Petri nets for representing the manufacturing efficiency of FMSs, namely timed-place Petri nets [35], which are also used for scheduling [36,37]. Luo et al. [38] performed a hybrid heuristic search to solve the static scheduling problem of an FMS through a combination of S^3PR and timed-place Petri nets.

The states of Petri nets, i.e., markings distributed in topologically complicated net structures, are defined on non-Euclidean spaces, which are significantly different from regular Euclidean data such as images or videos. Therefore, neural networks designed for regular Euclidean data are not suitable for Petri net states. In other fields of machine learning, such as social network embedding [39], geographic information mining [40], and molecular fingerprint learning [41], researchers have attempted to address complicated graphical data by developing novel graph convolutional networks (GCNs) based on signal processing of graphs and graph Fourier transforms [42–44]. In general, GCNs can be classified into two categories: (1) vertex-domain methods that calculate feature propagation in graphs via filters defined on the vertex neighborhood [45] and (2) spectrum-domain methods that convert a convolution of a feature and a kernel in a graph into the product of their graph Fourier transforms [46]. Although GCNs have achieved significant progress in many applications, it is challenging for existing GCN methods to handle complex Petri nets that are directed, bipartite, and heterogeneous.

This study considers the dynamic scheduling problem of an FMS with shared resources, route flexibility, and stochastic arrivals of raw products. First, a class of Petri nets, namely timed S^3PR, which combines timed-place Petri nets and S^3PR, is used to model FMS process in consideration of both manufacturing efficiency and deadlock avoidance. Then, a novel GCN layer, namely a Petri-net convolution (PNC) layer, is proposed for the mapping approximation of the timed S^3PR state. A PNC layer contains two sub-layers that use the input and output matrices of the timed S^3PR for feature propagation computation from places to transitions (P2T) and from transitions to places (T2P), respectively. Finally, by defining the scheduling problem of the timed S^3PR as an MDP, we use a DQN with a PNC network for smart scheduling. Experimental results verify the following advantages of the proposed method: (1) It is more efficient in handling timed S^3PR states compared to basic MLP. (2) Its scheduling performance, learning convergence, and robustness are better than those of DQN with basic MLP, and it can achieve similar dynamic scheduling performance with much faster online computation compared to a heuristic search method. (3) It has better environmental adaptability than heuristic methods.

The remainder of this paper is organized as follows. Section 2

reviews the definitions and notations associated with Petri nets and introduces the timed S^{3}PR of an FMS. Section 3 describes the design of the two PNC sub-layers, namely the P2T layer and T2P layer, on the basis of a detailed analysis of timed S^{3}PR. Section 4 introduces MDP and DRL for dynamic scheduling. Section 5 presents and discusses the experimental results. Finally, Section 6 concludes the paper and briefly explores directions for future work.

2. Preliminaries

This section reviews the fundamentals of Petri nets and the definition of a timed S^{3}PR.

Definition 1. Petri Nets [47]

A *Petri net* (place/transition net) is a 3-tuple $\mathcal{N} = \langle P, T, F\rangle$, where P and T are two nonempty finite disjoint sets representing places and transitions, respectively. $F \subseteq (P \times T) \cup (T \times P)$ is a finite set of arcs. Thus, a Petri net is a directed bipartite graph. F can be represented by two incidence matrices $\boldsymbol{I}, \boldsymbol{O} \in \{0, 1\}^{|P|\times|T|}$, called the *input matrix* and *output matrix* of places, respectively, where $\boldsymbol{I}[p, t] = \begin{cases} 1 & (p, t) \in F \\ 0 & (p, t) \notin F \end{cases}$ and $\boldsymbol{O}[p, t] = \begin{cases} 1 & (t, p) \in F \\ 0 & (t, p) \notin F \end{cases}$. Given a note $x \in P \cup T$, $\bullet x = \{y \in P \cup T | (y, x) \in F\}$ is the *pre-set* of x, while $x\bullet = \{y \in P \cup T | (x, y) \in F\}$ is the post-set of x. $\forall\, X \subseteq P \cup T$, $\bullet X = \bigcup_{x\in X} \bullet x$ and $X\bullet = \bigcup_{x\in X} x\bullet$. $\mathcal{N} = \langle P, T, F\rangle$ is called a *state machine* if $\forall\, t \in T$, $|\bullet t| = |t\bullet| = 1$.

Definition 2. Marking and Firing [47]

Places in a Petri net contain a discrete number of marks called *tokens*. A distribution of tokens over the places represents a state of the Petri net called a *marking* $\boldsymbol{m} \in \mathbb{N}^{|P|}$. With regard to a set of places $X \subseteq P$, $\boldsymbol{m}(X)$ is the restriction of $\boldsymbol{m}$ to X. The pair $\langle \mathcal{N}, \boldsymbol{m}_0\rangle$, where $\mathcal{N}$ is a Petri net and $\boldsymbol{m}_0$ is an (initial) marking, is called a marked Petri net (or net system). A transition t is *enabled* and can be fired if $\forall\, p \in \bullet t$, $\boldsymbol{m}(p) > 0$, denoted as $\boldsymbol{m}[t\rangle$. *Firing* of a transition t gives a new marking $\boldsymbol{m}' = \boldsymbol{m} + \boldsymbol{O}[\cdot, t] - \boldsymbol{I}[\cdot, t]$, denoted as $\boldsymbol{m}[t\rangle\boldsymbol{m}'$. Transitions t and t' are *conflicted* under $\boldsymbol{m}$ if $\boldsymbol{m}[t\rangle \wedge \boldsymbol{m}[t'\rangle$. A marking $\boldsymbol{m}'$ is *reachable* from another marking $\boldsymbol{m}$ iff there exists a firing sequence $\boldsymbol{t} = t_1 t_2 ... t_n$ such that $\boldsymbol{m}[t_1\rangle\boldsymbol{m}_1[t_2\rangle\boldsymbol{m}_2 ... [t_n\rangle\boldsymbol{m}'$. The set of markings reachable from $\boldsymbol{m}$ in $\mathcal{N}$ is denoted as $\mathcal{R}(\mathcal{N}, \boldsymbol{m})$.

Definition 3. P-invariant and Strict Minimal Siphon [31]

Given a P-vector $\boldsymbol{y}$: $P \to \mathbb{Z}$ indexed by P, $\boldsymbol{y}$ is a *P-invariant* if $\boldsymbol{y} \neq \boldsymbol{0}$ and $\boldsymbol{y} \cdot (\boldsymbol{O} - \boldsymbol{I}) = \boldsymbol{0}$ hold. $\|\boldsymbol{y}\| = \{p \in P | \boldsymbol{y}(p) \neq 0\}$ is called the support of $\boldsymbol{y}$. A nonempty set $S \subseteq P$ is a *siphon* if $\bullet S \subseteq S\bullet$. A siphon is minimal if it contains no siphon as a proper subset. A minimal siphon that does not contain the support of any P-invariant is called a *strict minimal siphon* (SMS).

Definition 4. System of Simple Sequential Processes with Resources [30,31]

A *system of simple sequential processes with resources* (S^{3}PR) $\mathcal{N} = \bigcirc_{i=1}^{q} \mathcal{N}_i = \langle P_A \cup P_0 \cup P_R, T, F\rangle$ is defined as the union of a set of nets $\mathcal{N}_i = \langle P_{A_i} \cup \{p_{0_i}\} \cup P_{R_i}, T_i, F_i\rangle$ sharing common places, where following statements are true:

1. p_{0_i} is called the process idle place of $\mathcal{N}_i$. The elements of P_{A_i} and P_{R_i} are called activity places and resource places, respectively.
2. $P_{A_i} \neq \emptyset$, $P_{R_i} \neq \emptyset$, $p_{0_i} \notin P_{A_i}$, and $(P_{A_i} \cup \{p_{0_i}\}) \cap P_{R_i} = \emptyset$.
3. $\forall\, p \in P_{A_i}$, $\forall\, t \in \bullet p$, $\forall\, t' \in p\bullet, \bullet t \cap P_{R_i} = t'\bullet \cap P_{R_i}$ and $|\bullet t \cap P_{R_i}| = 1$.
4. $\forall\, r \in P_{R_i}$, $\bullet\bullet r \cap P_{A_i} = r\bullet\bullet \cap P_{A_i} \neq \emptyset$ and $\bullet r \cap r\bullet = \emptyset$.
5. $\bullet\bullet p_{0_i} \cap P_{R_i} = p_{0_i}\bullet\bullet \cap P_{R_i} = \emptyset$.
6. The subset $\mathcal{N}_i^- = \langle P_{A_i} \cup \{p_{0_i}\}, T_i, F_i^-\rangle$ generated by removing P_{R_i} and related arcs from $\mathcal{N}_i$ is a strongly connected state machine, and every circuit of $\mathcal{N}_i^-$ contains place p_{0_i}.
7. For $r \in P_R$, $\bullet\bullet r \cap P_A$ is the set of activity places that use r, which are called holders of r.
8. For $p \in P_A$, $\bullet\bullet p \cap P_R = \{r\}$ is the unique resource used by p.

The S^{3}PR has a useful behavioral property in that its liveness is associated with its SMS: the existence of an empty SMS ($\boldsymbol{m}(S) = \boldsymbol{0}$) under a reachable marking $\boldsymbol{m} \in \mathcal{R}(\mathcal{N}, \boldsymbol{m}_0)$ indicates a system deadlock. Classical S^{3}PR control methods append control places to the original S^{3}PR to prevent the existence of an empty SMS [33].

Here, we adjust some notations of the S^{3}PR so that it can accommodate the dynamic scheduling problem. First, p_{0_i} in an S^{3}PR is split into two places p_{B_i} and p_{E_i} representing the beginning and the end of the state machine $\mathcal{N}_i^-$ (or operation sequence), respectively. The sets $\{p_{B_1}, p_{B_2}, ..., p_{B_q}\}$ and $\{p_{E_1}, p_{E_2}, ..., p_{E_q}\}$ are denoted as P_B and P_E, respectively. Places in the set $P_O = P_B \cup P_A$ are called operation places in this paper. Second, for representing the manufacturing efficiency of a system, time attributes are added to the adjusted S^{3}PR to construct timed S^{3}PR:

Definition 5. Timed S^{3}PR

A marked *timed S^{3}PR* is denoted as $\langle \mathcal{N}^*, D, \boldsymbol{m}_0\rangle$, where

1. $\mathcal{N}^* = \langle P_O \cup P_R \cup P_E, T, F\rangle$ is an adjusted S^{3}PR with $P_O = P_A \cup P_B$ introduced above.
2. D: $P_O \to \mathbb{R}^+ \cup \{0\}$ is a mapping that specifies a deterministic token holding time at an operation place. Thus, if a token is put into $p_O \in P_O$ at time τ, it can be used to enable the transitions in $p_O\bullet$ after time $\tau + D(p_O)$ (hereafter, for simplicity, we say that this token is enabled after time $\tau + D(p_O)$).
3. $\boldsymbol{m}_0$ is the initial marking with $\boldsymbol{m}_0(P_O) = \boldsymbol{0}$, $\boldsymbol{m}_0(P_E) = \boldsymbol{0}$, and $\forall\, p \in P_R$, $\boldsymbol{m}_0(p) > 0$ indicating the counts of the resources.

Fig. 2 shows a marked timed S^{3}PR derived from a well-known FMS example involving three robots, four machines, and three product types [30,38]. In this model, $P_B = \{p_1, p_2, p_3\}$ marked in green represents the arrival of three types of raw products, $P_E = \{p_4, p_5, p_6\}$ marked in red represents the end of the products, $P_R = \{p_{23}, p_{24}, p_{25}, p_{26}, p_{27}, p_{28}, p_{29}\}$ marked in orange are the resources, including the three robots and four machines, and the remaining black circles constitute P_A. The initial marking of P_R is labeled inside the circles and the holding times of P_O are added in parentheses. t_3, t_{13}, and t_{19} are autonomous transitions, while the others are schedulable ones. The liveness of the timed S^{3}PR can be evaluated on the basis of the SMSs of the original S^{3}PR (listed in Table 1), which can be calculated efficiently according to [31,32].

The dynamic scheduling problem in the formulation of the timed S^{3}PR is explained as follows: When a raw product of type i arrives, a token is fed into p_{B_i}. Then, the scheduling agent fires enabled

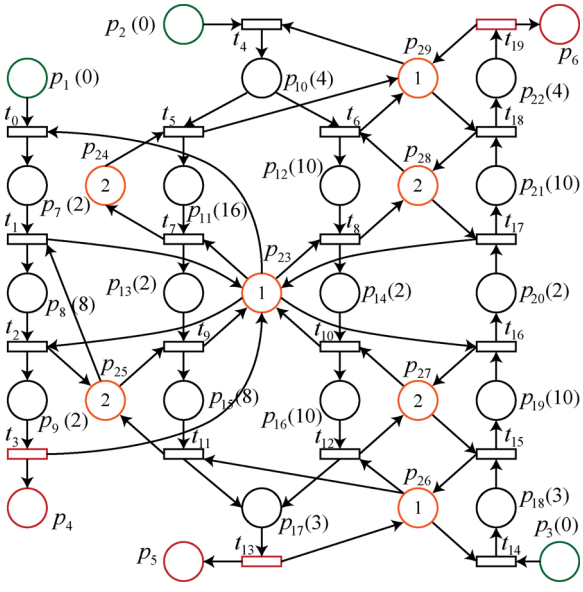

Fig. 2. Timed S^{3}PR of a well-known FMS example [30].

Table 1
SMSs of the original S^{3}PR [30].

No.	SMS
1	$p_9, p_{17}, p_{21}, p_{23}, p_{25}, p_{26}, p_{27}, p_{28}$
2	$p_7, p_9, p_{13}, p_{14}, p_{22}, p_{23}, p_{24}, p_{28}, p_{29}$
3	$p_7, p_9, p_{13}, p_{17}, p_{20}, p_{23}, p_{26}, p_{27}$
4	$p_9, p_{15}, p_{16}, p_{22}, p_{23}, p_{24}, p_{25}, p_{27}, p_{28}, p_{29}$
5	$p_7, p_9, p_{13}, p_{16}, p_{20}, p_{23}, p_{27}$
6	$p_7, p_9, p_{13}, p_{17}, p_{21}, p_{23}, p_{26}, p_{27}, p_{28}$
7	$p_7, p_9, p_{13}, p_{16}, p_{21}, p_{23}, p_{27}, p_{28}$
8	$p_9, p_{14}, p_{15}, p_{22}, p_{23}, p_{24}, p_{25}, p_{28}, p_{29}$
9	$p_7, p_9, p_{13}, p_{14}, p_{21}, p_{23}, p_{28}$
10	$p_9, p_{15}, p_{16}, p_{20}, p_{23}, p_{25}, p_{27}$
11	$p_9, p_{15}, p_{16}, p_{21}, p_{23}, p_{25}, p_{27}, p_{28}$
12	$p_{17}, p_{19}, p_{26}, p_{27}$
13	$p_9, p_{14}, p_{15}, p_{20}, p_{23}, p_{25}$
14	$p_9, p_{17}, p_{22}, p_{23}, p_{24}, p_{25}, p_{26}, p_{27}, p_{28}, p_{29}$
15	$p_9, p_{14}, p_{15}, p_{21}, p_{23}, p_{25}, p_{28}$
16	$p_9, p_{17}, p_{20}, p_{23}, p_{25}, p_{26}, p_{27}$
17	$p_7, p_9, p_{13}, p_{16}, p_{22}, p_{23}, p_{24}, p_{27}, p_{28}, p_{29}$
18	$p_7, p_9, p_{13}, p_{17}, p_{22}, p_{23}, p_{24}, p_{26}, p_{27}, p_{28}, p_{29}$

transitions in a suitable order to make the token finally arrive at p_{E_i}, indicating that the processing of this product has been completed. Because several products are processed concurrently in the system and new raw products can arrive at any time during the system processing, the scheduling agent should dynamically make decisions based on the states of the system to optimize the manufacturing performance. The purpose of such decision making is to fire a proper transition when several transitions are conflicted. For example, if t_{12} in Fig. 2 is enabled and a new raw product of type 3 arrives, which enables t_{14}, then and t_{14} are conflicted, and the scheduling agent should decide which one will use the resource of p_{26} first.

3. GCN layers for timed S^{3}PR

A timed S^{3}PR is a typical directed bipartite graph, and existing GCN methods, which are generally designed for homogenous undirected graphs, cannot handle the timed S^{3}PR. To mine the hidden information from the state features and structural properties of the timed S^{3}PR, we construct two special graph convolution layers to compute the feature propagation from places to transitions and from transitions to places. Because the timed S^{3}PR is a directed graph, only the influence of the pre-order place (or transition) is considered for a transition (or a place).

3.1. P2T layer: feature propagation from places to transitions

Generalizing the classical CNN layer from regular Euclidean spaces to graphs is challenging in two aspects: (1) the number of neighbors of each node in a graph usually varies, and (2) the correspondence between the filter and the neighborhood of each node is ambiguous. We can overcome these challenges by exploring the neighbor structure of transitions in the timed S^{3}PR as explained below.

From the previous section, we know that there are two types of transitions in the timed S^{3}PR: schedulable and autonomous. A schedulable transition has two pre-order places, i.e., one operation place and one resource place, while an autonomous transition has only one pre-order operation place. Adding a dummy resource place containing a token that is always enabled before an autonomous transition will make the pre-order place structure of each transition isomorphic. Then, following the classical CNN, we can construct a trainable filter (Fig. 3) containing two units to calculate the weighted sum of features of two pre-order places and treat the result as the transition feature f_t:

$$f_t = \begin{cases} \boldsymbol{w}_P f_{p_O} + \boldsymbol{w}_R f_{p_R} + b & t \in T_S \\ \boldsymbol{w}_P f_{p_O} + \boldsymbol{w}_R \boldsymbol{1} + b & t \in T_A \end{cases} \tag{1}$$

where f_{p_O}, f_{p_R}, and $\boldsymbol{1}$ are d-dimensional features of the pre-order operation place, resource place, and dummy resource place, respectively. Further, $\boldsymbol{w}_P$ and $\boldsymbol{w}_R$ are $d \times d$ trainable weight matrices, while b is a d-dimensional trainable bias. Thus, f_t is also d-dimensional and it will be used in the T2P layer as discussed in Section 3.2.

The convolution in the P2T layer is calculated on the basis of the timed S^{3}PR input matrix as shown in Fig. 4. First, the input matrix $\boldsymbol{I}$ of the timed S^{3}PR is transposed and transformed into a new location matrix $\boldsymbol{I}' \in \{0, 1\}^{(2|T|)\times(|P|+1)}$, whose odd rows represent the location of a transition's pre-order operation place and even rows represent the location of its pre-order resource place. Thus, each row of $\boldsymbol{I}'$ contains only one non-zero element. Note that $\boldsymbol{I}'$ is used as a static parameter of the P2T layer. Then, the input feature matrix indexed by places $\boldsymbol{F}_P = (f_{p_1}, f_{p_2}, ...f_{p_n})^{\mathrm{T}} \in \mathbb{R}^{|P|\times d}$ is extended by an all-1 row representing the feature of the dummy resource place, denoted as $\boldsymbol{F}'_P \in \mathbb{R}^{(|P|+1)\times d}$. The product of $\boldsymbol{I}'$ and $\boldsymbol{F}'_P$ generates a new feature matrix $\boldsymbol{F}''_P \in \mathbb{R}^{(2|T|)\times d}$. Finally, the convolution of $\boldsymbol{F}''_P$ and a trainable $2 \times 1 \times d \times d$ shared kernel is calculated as the classical convolution layer[1] with a stride of 2. The convolution generates the expected feature matrix of transitions $\boldsymbol{F}_T \in \mathbb{R}^{|T|\times d}$. $\boldsymbol{F}_P$ and $\boldsymbol{F}_T$ are passed to the next layer.

3.2. T2P layer: feature propagation from transitions to places

Sandryhaila and Moura [42] proposed a type of linear, shift-invariant filters on graphs:

$$\hat{\boldsymbol{F}} = \boldsymbol{H}\boldsymbol{F} = \sum_{k=0}^{K} h_k \boldsymbol{A}^k \boldsymbol{F}$$

where $\boldsymbol{F}$ and $\hat{\boldsymbol{F}}$ are the original feature and the filtered feature on the graph, respectively, $\boldsymbol{H} = \sum_{k=0}^{K} h_k$ is the filter, $\boldsymbol{A}$ is the adjacency matrix of a weighted directed graph, h_k is the filter coefficient, and K is the order of the filter. This filter ensures that the filtered feature of a node in the graph is influenced only by its neighbors with a maximum order of K (similar to Lemma 5.2 in [48]).

According to this principle, feature propagation in the T2P layer is defined as

$$\hat{\boldsymbol{F}}_P = [h_0 \odot \boldsymbol{F}_P + h_1 \odot (\tilde{\boldsymbol{O}}\boldsymbol{F}_T)]\boldsymbol{W} + \boldsymbol{B} \tag{2}$$

The filter order K is limited to 1; hence, the filtered feature of a place is related only to its original feature and its pre-order transitions. The propagation from places to transitions will be included if $K \geq 2$; however, this is implemented by the P2T layer in our framework. Because $\boldsymbol{H}$ is a polynomial of $\boldsymbol{A}$, arbitrary-order filters can be approximated by stacking the basic 1-order filter in deep neural networks [49]. Further, h_0 and h_1 are the d-dimensional trainable filter coefficients. The operation $\odot$ is the Hadamard product of a vector and a matrix, yielding another matrix in which each element i, j is the product of the element j of the original vector and the element i, j of the original matrix. Through the operation $\odot$, the filter coefficient of each feature channel can be learned independently. $\tilde{\boldsymbol{O}}$ is the normalized output matrix of the timed S^{3}PR, i.e., $\tilde{\boldsymbol{O}}_{\mathrm{ij}} = \boldsymbol{O}_{\mathrm{ij}}/(\sum_{k=1}^{|T|} \boldsymbol{O}_{\mathrm{ik}})$. $\tilde{\boldsymbol{O}}$ is $|P| \times |T|$ and features of both the place and the transition are d-dimensional; hence, the addition in the parentheses holds. $\boldsymbol{W} \in \mathbb{R}^{d\times d'}$ is the trainable weight for mapping features from d channels to d' channels, and $\boldsymbol{B}$ is the trainable bias.

3.3. Analysis about PNC layer

Combining the P2T layer with the T2P layer gives a PNC layer as

[1] In our experiments, the convolution is implemented by tf.nn.conv2d (https://www.tensorflow.org/api_docs/python/tf/nn/conv2d).

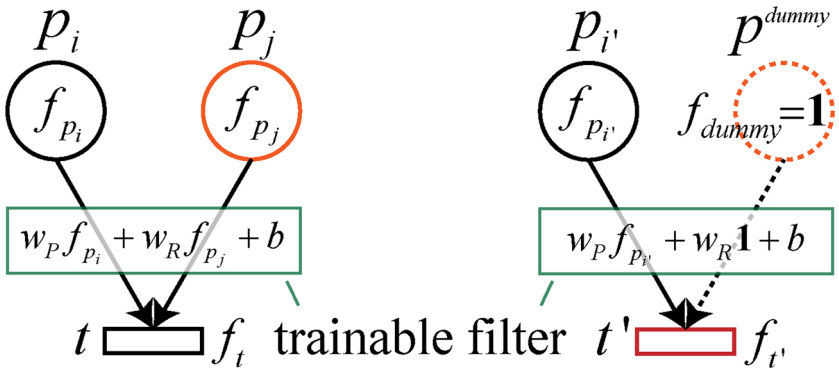

Fig. 3. Trainable filter of the P2T layer.

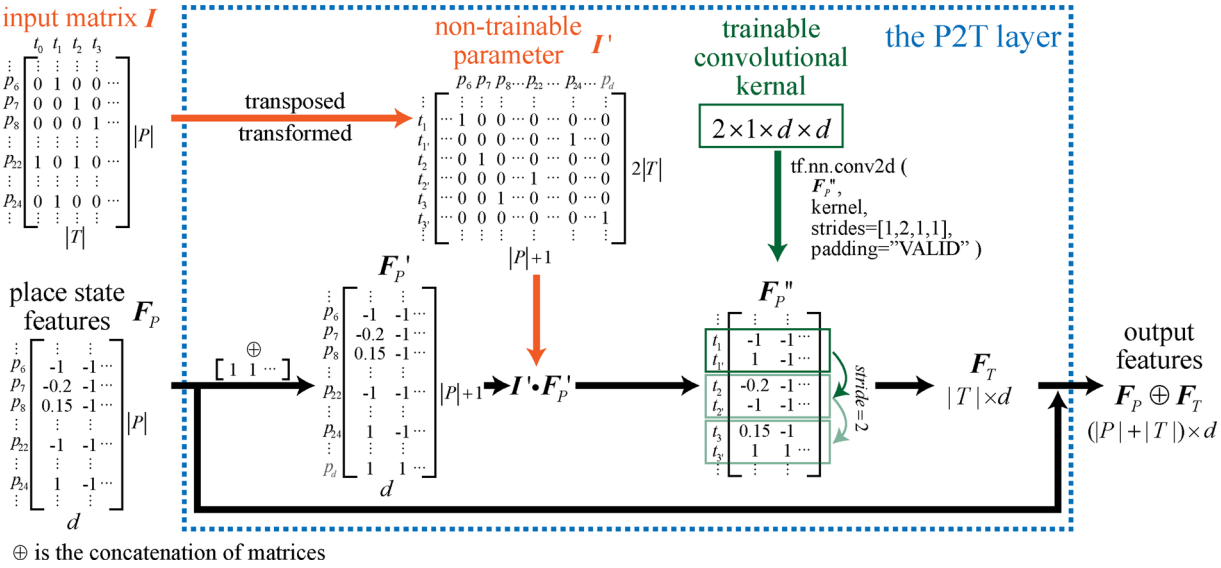

Fig. 4. P2T layer of the timed S^{3}PR.

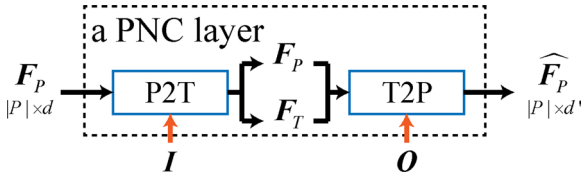

Fig. 5. PNC layer.

shown in Fig. 5. As the purpose of Petri nets, a timed S^{3}PR represents a state of an FMS process by its net structure and its place state. The net structure models the sequential order of the process and the usage/release of resources. The place state identifies the current state of each manufacturing task and each resource by the token distribution in places of the timed S^{3}PR. From the definitions of the P2T and T2P layers, we know that the PNC layer can handle both the two aspects of the timed S^{3}PR: The net structure is embedded into the PNC layer by taking the input matrix $\boldsymbol{I}$ and the output matrix $\boldsymbol{O}$ of the timed S^{3}PR as non-trainable parameters of the layer. The place state is transformed to the place feature matrix $\boldsymbol{F}_P$ as the input of the first PNC layer in a PNC network. The input of each other PNC layer in the PNC network is the filtered feature $\hat{\boldsymbol{F}}_P$ of its preceding PNC layer. The construction of $\boldsymbol{F}_P$ from the place state will be discussed in Section 4.1.

More analysis about the design motivation for the P2T and T2P layers from an intuitive perspective is as explained below. For a transition, its pre-order operation place and resource place are of different types and their influences should be distinguished; hence, the shared convolution kernel contains two units, w_P and w_R. Similarly, for a place, h_0 and h_1 distinguish the influences from the place itself and its pre-order transitions. Further details about $\tilde{\boldsymbol{O}}\boldsymbol{F}_T$ in Eq. (2) merit closer attention. For all p_O and p_E, it is easy to understand that Eq. (2) degenerates into $\tilde{f}_p = (h_0 \circ f_p + h_1 \circ f_t)\boldsymbol{W} + \boldsymbol{B}$, where $\circ$ is the Hadamard product of two vectors, because each of them has only one pre-order transition. However, for all p_R, $\tilde{\boldsymbol{O}}\boldsymbol{F}_T$ generates mean values of their pre-order transitions' features, which is consistent with the assumption that the transitions' features of a resource place embed the hidden information of resource release and they should be uniformly propagated to the filtered feature of the resource place. Eq. (2) can also be understood as a spectrum-domain GCN method when considering the graph Fourier transform based on the Jordan decomposition of the adjacency matrix [43] and eigenvalue matrix approximations [46].

From the definitions of the P2T and T2P layers, the number of trainable parameters of a PNC layer is given by:

$$\underbrace{2d^2}_{\text{P2Tkernel}} + \underbrace{d}_{\text{P2Tbias}} + \underbrace{2d}_{h_0 \text{ and } h_1} + \underbrace{d \times d'}_{\text{T2Pweight}} + \underbrace{d'}_{\text{T2Pbias}}$$

Clearly, the number of trainable parameters of a PNC layer is related only to the number of channels of the input and output features and not to the scale of the timed S^{3}PR. Thus, by using the PNC layer, we can overcome the explosion problem of trainable parameters when building deep neural networks.

In addition, we can derive that the time complexity of a PNC layer is $O(|P| \times |T| \times d + |T| \times d^2 + |P| \times d \times d')$. For a given set of channel numbers and depth of a PNC network, its time complexity is quadratic

to the scale of the timed S^{3}PR, as the numbers of places and transitions in a timed S^{3}PR are similar. However, in practical applications, to achieve stronger approximation for larger timed S^{3}PR, the PNC network will be established with deeper layers and more channels. Therefore, the complexity of the PNC network is polynomial-time. The majority of computations of a PNC network are matrix manipulations which can be significantly accelerated by advanced machine learning libraries with hardware boosting.

4. DRL methods for timed S^{3}PR scheduling

In this section, the dynamic scheduling problem of the timed S^{3}PR is defined as an MDP and then solved by DRL.

4.1. MDP of timed S^{3}PR scheduling

Using DRL to solve the dynamic scheduling problem requires the construction of an environment in which a DRL scheduling agent can take actions and gain rewards. The environment must satisfy the criterion of an MDP, i.e., all relevant information for the action decision is encoded in the state information.

The MDP framework describes the environment with a 5-tuple $\langle \mathcal{S}, \mathcal{A}, \Phi, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ denotes the action space, Φ denotes the dynamics of a state transition, $\mathcal{R}$ denotes the reward, and γ denotes the discount rate. Now, the specific meanings of the elements in the 5-tuple are defined in the dynamic scheduling scenario of the timed S^{3}PR:

- Given a feature dimension d, the state of the MDP is a matrix $\boldsymbol{F}_P \in \mathbb{R}^{|P|\times d}$ in which each row is a *state feature* of a place constructed as follows:
  (a) For an operation place p_O, each element of its state feature is the *time feature* of a token: $f_{p_O}(i) = \begin{cases} \rho(\tau - \tau_i^e) & i \le \boldsymbol{m}_\tau(p_O) \\ -1 & i > \boldsymbol{m}_\tau(p_O) \end{cases}$, where $1 \le i \le d$, τ is the current simulation clock, $\boldsymbol{m}_\tau(p_O)$ is the current marking, τ_i^e is the enabled time of token i, and ρ is a normalization function that normalizes the time feature into $(-1, 1)$. Thus, $f_{p_O}(i) \ge 0$ implies that the token is waiting for the next operation, and $-1 < f_{p_O}(i) < 0$ implies that the token is being processed. If $\boldsymbol{m}_\tau(p_O) < d$, the state feature is padded with -1. The time features in the state feature are sorted in descending order.
  (b) For a resource place p_R, its state feature is defined such that its first $\boldsymbol{m}_\tau(p_R)$ elements are 1 and last $d - \boldsymbol{m}_\tau(p_R)$ elements are -1, where $\boldsymbol{m}_\tau(p_R)$ is the number of available resources. For the dummy resource place introduced in Section 3.1, its state feature is defined as an all-1 vector that represents always available dummy resources.
  (c) For a final place p_E, its state feature is a d-dimensional all-(-1) vector that indicates that its state has no influence on the scheduling of the timed S^{3}PR.
  Because the state features of all places encode both the marking of the timed S^{3}PR and the time features of all tokens, the choice of conflicted transitions depends only on the state and not on the previous transition firing sequence, i.e., the evolution of the timed S^{3}PR fulfills the MDP condition.
- The scheduling of the timed S^{3}PR is an alternative firing problem of conflicted schedulable transitions. Therefore, firing a specific transition in T_S in a scheduling step corresponds to an action in the action spate of the MDP. In addition, the action space also contains an idle action which means firing no transitions in T_S in this step. To summarize, the action space of the MDP is the set $T_S \cup$ idle, with a size of $|T_S| + 1$. Because the number of enabled transitions is usually limited during the scheduling of the timed S^{3}PR, we introduce a *mask* $\xi = \{t | t \text{ isavalidactioninthecurrentstate}\}$. This mask can significantly reduce the trial-and-error operation of a DRL scheduling agent during the learning process and improve the learning efficiency. It is worth noting that the idle action should not always be valid during scheduling; otherwise, the DRL scheduling agent may easily be trapped in an eccentric policy, always choosing the idle action to avoid deadlock. The motivation for the idle action has two aspects: (1) firing the only enabled schedulable transition will lead to a deadlock such that nothing can be done until any operation releases the relevant resource; (2) waiting for a reasonable resource in use may be more efficient than immediately choosing an available one in some cases. In summary, the idle action is used to wait for any operation to be completed. In other words, if there are tokens being processed, then the idle action is valid, i.e., if $\exists\, p_O \in P_O, \exists\, i \in \{1, 2, ..., d\}, -1 < f_{p_O}(i) < 0$, then $\xi = \{t \mid t \text{ is an enabled schedulable transition}\} \cup \{\text{idle action}\}$.
- The dynamics of the MDP are implicitly defined by the discrete event simulation (DES) of the timed S^{3}PR shown in Algorithm 1. From Algorithm 1, we can see that a state transition of the MDP (i.e., a procedure of the DES) involves performing a selected action, firing several autonomous transitions, and feeding $p_B \in P_B$ with tokens that represent arrivals of raw products. Therefore, the randomness of the MDP is introduced by the stochastic arrivals of raw products.
- A reward can be identified flexibly on the basis of the objective of a scheduling. Here, we give an ordinary reward r to optimize the makespan of several products with deadlock avoidance:

$$r_\tau = \begin{cases} -\theta_0 & \text{exists a SMS } \boldsymbol{s},\ \boldsymbol{m}(\boldsymbol{s}) = \boldsymbol{0} \\ \alpha_\tau + \sum \beta_\tau & \textbf{otherwise} \end{cases}$$

  where θ_0 is a relatively large positive number such that a severe penalty is imposed on the DRL scheduling agent when the timed S^{3}PR is deadlocked, i.e., when there exists at least one empty SMS under the current marking. Further, α_τ is the average waiting time of tokens in this clock τ:

$$\alpha_\tau = -\rho\left(\frac{\sum \mathrm{Relu}(\tau - \tau^e)}{\sum \boldsymbol{m}(p_O)}\right)$$

  where $\mathrm{Relu}(\cdot) = \max(\cdot, 0)$ ignores the time feature of in-processing tokens and ρ is a normalization function that restricts α_τ within $(-1, 0]$. α_τ imposes a penalty on the DRL scheduling agent to prevent situations in which too many products are waiting at the same time. β_τ is the reward for each completed product in the current scheduling step:

$$\beta_\tau = \theta_1 + \theta_1[1 - \rho(\mathrm{ms} - \mathrm{mms})]$$

  where ms is the actual processing time of a completed product, i.e., the interval of the token from p_B to p_E, and mms is the minimal processing time of this product type. For example, the mms of Type 1, 2, and 3 products in Fig. 2 are 12, 29, and 29, respectively. θ_1 denotes a reward magnification such that $\beta_\tau \gg |\alpha_\tau|$; thus, the DRL scheduling agent receive positive feedback upon product completion.
- The discount rate γ determines whether the DRL scheduling agent focuses on a short-term revenue or a long-term accumulated reward. It strongly influences the policy learned by the DRL scheduling agent. A small γ, e.g., 0.85, results in a low-parallelism policy that makes all other tokens wait in P_B until the manufacturing process of the current token is completed, because β_τ can be obtained in the minimum number of steps. Therefore, γ should be close to 1 to optimize the makespan; however, this makes the convergence of an action-value approximator problematic and thus requires more suitable deep neural networks.

4.2. DRL with PNC networks

The MDP derived from the timed S^{3}PR scheduling problem can be

solved by standard RL methods. In this study, we consider the Q-learning framework that optimizes the action selection policy of the DRL scheduling agent such that the expected sum of future rewards is maximized. An action-value function $Q_\pi(s, a)$ is defined to assess the performance of taking an action a given a state s under a policy π:

$$Q_\pi(s, a) = \mathbb{E}_\pi[\sum_k \gamma^k r_{\tau+k} | s_\tau = s, a_\tau = a]$$

where k is the future step and $\mathbb{E}[\cdot]$ denotes the expected value. The optimal action-value function $Q^*(s, a)$ obeys the Bellman equation, which can be used as an iterative update approximation in RL algorithms:

$$Q^*(s, a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^*(s', a') | s, a]$$

where s' and a' are the state and the possible action in the next time step, respectively. In practice, the action-value function is commonly estimated by a function approximator, which is often constructed by deep neural networks owing to their flexibility and robustness.

In this study, we use a DQN [9] with the prioritized experience replay [25] as the DRL algorithm, and a deep PNC network as the approximator of the action-value function.

In the dynamic scheduling scenario of the timed S^{3}PR, the role of the prioritized experience replay involves two aspects. (1) Because a large penalty is imposed on the DRL scheduling agent when deadlock occurs, the deadlock states will be replayed with high probability in the early learning stages. Hence, the agent will quickly learn to avoid deadlock and focus on improving the manufacturing efficiency in the later stages. (2) States with $\beta \neq 0$ will be replayed frequently to improve the agent sensitivity to the processing-time variation of the products.

We further extend DQN using the MDP action mask to avoid trial-and-error execution of invalid actions and achieve rapid learning convergence. The DRL algorithm is shown in Algorithm 2[2]. Owing to space limitations, we suggest that readers refer to Algorithm 1 in [9] and Algorithm 1 in [25] for the details of this algorithm.

A PNC network (Fig. 6) with seven PNC layers and one fully connected layer (FC layer) is built for the timed S^{3}PR as shown in Fig. 2 to map the MDP states to action values. From the definition of a PNC layer in Section 3, we know that one PNC layer propagates the hidden feature from each place p to its neighboring post places $p\bullet\bullet$ in the timed S^{3}PR. Because the longest route in the timed S^{3}PR in Fig. 2 contains seven places, the depth of PNC layers in the PNC network is chosen as seven. Every PNC layer uses a batch normalization regularization [50] and a LeakyRelu activation [51] to improve the neural network's robustness and non-linearity. The PNC layers are followed by the FC layer which maps the hidden place feature to a $(|T_S| + 1)$-dimensional vector in the action space. This 8-layer PNC network performs well for the timed S^{3}PR shown in Fig. 2. In addition, for more complicated timed S^{3}PR, PNC networks with deeper PNC layers and more feature channels are required. However, owing to the reasonable computation complexity of a PNC layer, PNC networks are scalable for larger processes.

5. Experiments

Three experiments are designed to verify the effects of the proposed PNC method on the dynamic scheduling of the timed S^{3}PR. The first one is a supervised learning experiment to compare the performances of a PNC network and MLP in handling the timed S^{3}PR state. The second one is the main experiment that verifies the effectiveness of the proposed method, i.e., the masked DQN with the PNC network, in the dynamic scheduling of the timed S^{3}PR by comparing it with three other methods. The third experiment is conducted on the basis of the second one to compare the environmental adaptabilities of the scheduling methods.

5.1. Empty SMS recognition

Owing to the strong association between an empty SMS and the liveness of the timed S^{3}PR, we conduct a basic supervised learning experiment on empty SMS recognition by neural networks. It is worth noting that this experiment is conducted not to introduce a new deadlock recognition method to replace the advanced siphon-based methods but to compare the efficiency and robustness of the PNC network with those of MLP when handling the timed S^{3}PR states.

A deadlock dataset of the timed S^{3}PR shown in Fig. 2 is first constructed for training as follows. Randomly generate 10,000 legal timed S^{3}PR states with markings $\boldsymbol{m} \in \mathcal{R}(\mathcal{N}, \boldsymbol{m}_0)$ ($\boldsymbol{m}_0$: p_1(randompositiveinteger), p_2(randompositiveinteger), p_3(randompositiveinteger), $p_{23}(1)$, $p_{24}(2)$, $p_{25}(2)$, $p_{26}(1)$, $p_{27}(2)$, $p_{28}(2)$, $p_{29}(1)$, and other places are all 0) and time features $f_p \in [-1, 1]$. Each of these states is labeled with 1 in the case of an existing empty SMS and 0 in the absence of an empty SMS. The numbers of deadlocked and alive states are balanced. Then, the 10,000 states are split into two subsets, i.e., the training set containing 8000 states and the test set containing the remaining states.

We use TensorFlow [52] and Keras [53] to implement a PNC network, denoted as PNCN here, and two MLPs as baselines, denoted as MLP1 and MLP2 (Fig. 7). Because this task is relatively simple, all three neural networks contain only four layers without batch normalization, dropout, or other regularization strategies. The training settings are as follows: the binary cross-entropy $-(y\log(\hat{y}) + (1-y)\log(1-\hat{y}))$, where y is the label and $\hat{y}$ is the prediction, is used as the loss function; Adam [54] with a learning rate of 0.001 is used as the optimizer; and the state feature dimension $d = 5$. For 20 training epochs with a batch size of 8, the training accuracy, test accuracy, training loss, and test loss are shown in Fig. 8(a)–(d), respectively.

As shown in Fig. 8(a) and (b), PNCN achieves accuracies of 99.88% and 99.90% on the training set and test set, respectively, which verifies that the PNC network can precisely recognize deadlock states of the timed S^{3}PR. In this experiment, a larger number of hidden nodes in the MLP leads to better performance. However, MLP2 is less effective than PNCN, even though the former has 40 times more trainable parameters than the latter. From Fig. 8, we can see that the accuracy and loss trajectories of all three networks are smooth for the training set, but only PNCN achieves smoothness on the test set. This implies that PNCN is more stable than MLP. In summary, PNCN shows the best performance in terms of both accuracy and convergence.

Although latent features generated by hidden layers of a deep neural network are usually obscure, we attempt to understand how PNCN works by analyzing the differences in the latent features generated by PNCN between pairs of similar timed S^{3}PR states. The differences are used for qualitatively marking the important places when identifying deadlock. Groups of similar timed S^{3}PR states are selected. Each group consists of one deadlock state and two alive states with a few variations in the markings. For pairs of states in each group, the differences in the latent features in the FC layer of PNCN are calculated and mapped to colors to fill the circles of the corresponding places as shown in Fig. 9(a). All the selected groups produce the same result as shown in Fig. 9(b): The most significant differences in the latent features between deadlock states and similar alive states are reflected in some operation places that belong to the relevant SMS. However, between two similar alive states, the differences in all the places are not obvious. This result indicates that PNCN can identify deadlock by feature propagation in local structures of the timed S^{3}PR.

5.2. Dynamic scheduling

In this experiment, the timed S^{3}PR DES environment (Algorithm 1) and the DRL scheduling agent (Algorithm 2) are developed on the basis

[2] ξ in Algorithm 2 is the *mask vector* indexed by actions derived from the mask introduced in Section 4.1: $\xi(a) = 0$ if a is a valid action; otherwise, $\xi(a) = -\infty$.

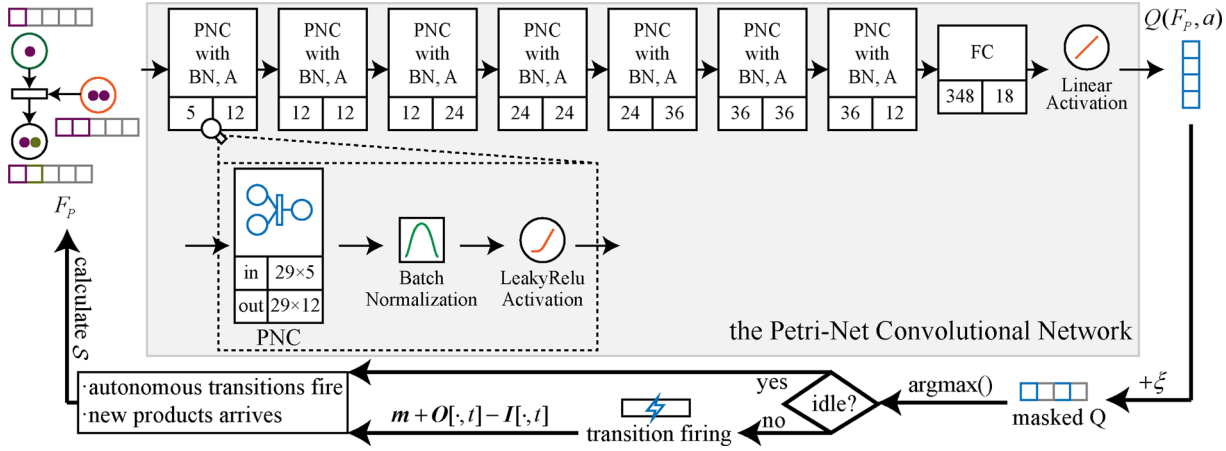

Fig. 6. 8-layer PNC network for the timed S^{3}PR shown in Fig. 2.

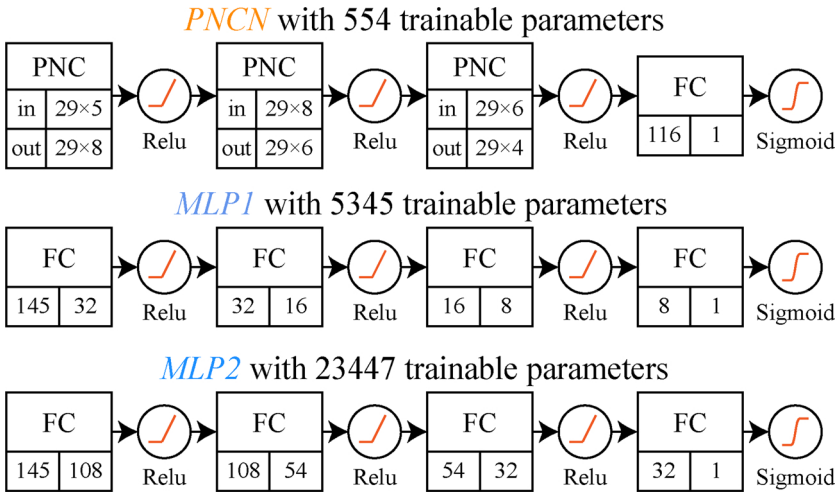

Fig. 7. Three neural network structures for deadlock recognition.

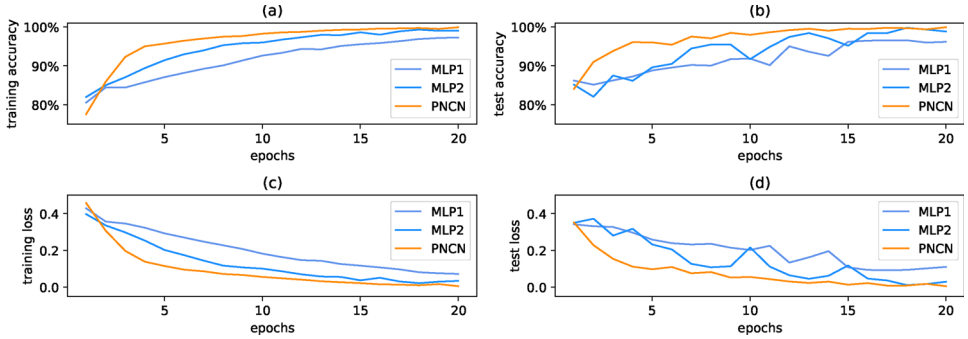

Fig. 8. Results of the deadlock recognition experiment.

of OpenAI Gym [55], OpenAI Baselines [56], and Keras-rl [57].The performance of the proposed method, i.e., the masked DQN with the 8-layer PNC network, denoted as PNCQ here, in the dynamic scheduling of the timed S^{3}PR is compared that of three other methods:

1. *FCFS+*: FCFS+ is a heuristic rule method that combines the first come, first served (FCFS) strategy with deadlock-free control. It fires the transition to handle the token with the longest waiting time in every step under the constraint of seven extra control places (calculated by [33]) to avoid deadlock.
2. *D^{2}WS* [38]: D^{2}WS is a reachable tree-based deadlock-free heuristic search method for static scheduling of the timed S^{3}PR. It uses an elaborate heuristic function to selectively search the branches of a timed S^{3}PR's reachable tree for the optimal or nearly optimal scheduling scheme. It can achieve high manufacturing efficiency; however, it is computationally expensive. D^{2}WS is invoked several times in a dynamic scheduling process. In every step, D^{2}WS is executed once for rescheduling to update the transition firing sequence if a new token is put into P_B, or sequential transitions are fired in turn. In this experiment, h_3 from [38] is selected as the heuristic function, and the parameters are accordingly set as (high, max_size, max_vertexes, max_top) = (3, 2, 2, 6) to achieve a trade-off between performance and time consumption.
3. *MLPQ*: MLPQ is a DRL baseline method that combines the masked

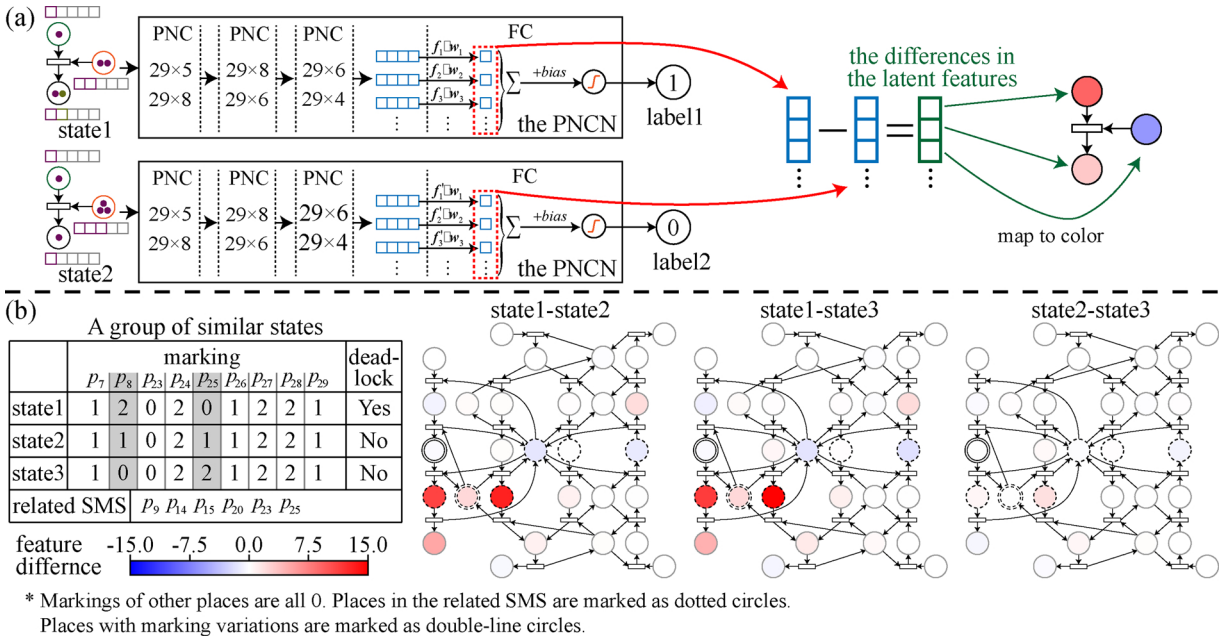

Fig. 9. Visualization of (a) latent feature differences and (b) a case study of groups of similar states.

DQN with an MLP designed by us to compare the effects of different neural networks. Using the same DRL scheduling agent as the proposed method, we construct an 8-layer MLP, containing 124, 64, 48, 32, 24, and 12 hidden nodes, as the action-value function approximator. PNCQ and MLPQ contain 18971 and 33644 trainable parameters, respectively.

The settings of the timed S^{3}PR's MDP and DES are as follows. State feature dimension $d = 5$, reward magnifications $\theta_0 = 50$ and $\theta_1 = 10$, and normalization function $\rho(x) = 2/(1 + e^{-0.05x}) - 1$. Let $\gamma = 0.99$ so that the DRL scheduling agent can optimize the makespan. Completing 20 products or being trapped in a deadlock is treated as an episode, and an episode reward is an accumulation of the rewards of every step in this episode. The arrival intervals of the raw products follow an exponential distribution with $\lambda = 3$, and the probabilities of the three product types are equal.

The settings of the masked DQN are as follows. The prioritized experience replay memory is warmed up by FCFS+ for 10,000 steps. To achieve a trade-off between exploration and exploitation, ϵ linearly falls from 1.0 to 0.1 during the first 200,000 learning steps and is then maintained at 0.1. Back-propagation fitting of the neural networks is conducted in every learning step by the Adam optimizer with a learning rate of 0.0001.The target neural network is updated every 1000 steps. After every 10,000 steps, a performance evaluation is performed using the current neural network to schedule 100 episodes and recording the average episode reward as well as the deadlock rate. Meanwhile, the same number of episodes is also scheduled by FCFS+ and D^{2}WS. A learning process is run over 3,000,000 steps; the experimental results are shown in Fig. 10.

From results of the episode rewards shown in Fig. 10(a), we can see that the trajectory of PNCQ is higher overall than that of MLPQ, which indicates that PNCQ achieves faster convergence and more efficient scheduling performance than MLPQ. Furthermore, the performance of PNCQ is close to that of FCFS+ after around 500,000 learning steps, while MLPQ requires more than 1,300,000 steps to achieve similar performance. The performance of PNCQ approaches that of D^{2}WS after around 800,000 steps, and it remains stable after 1,900,000 steps. The Q value shown in Fig. 10(c) and the loss shown in Fig. 10(d) also indicate that PNCQ achieves faster convergence than MLPQ. Noteworthily, as illustrated by the trajectories of the losses in Fig. 10(d), the approximation loss of PNCQ is much lower than that of MLPQ after convergence, which indicates that, compared to MLPQ, PNCQ achieves better accuracy when mapping the timed S^{3}PR state to the Q value. The reward statistics of the last 100 evolution episodes are shown in Fig. 11. The final average episode reward of PNCQ is around 309.95. Thus, PNCQ outperforms FCFS+ by approximately 12.3% and MLPQ by approximately 7.0%; however, it is slightly inferior to D^{2}WS by approximately 1.3%.

To achieve the best scheduling performance, D^{2}WS, which is a heuristic search method, needs to traverse numerous timed S^{3}PR states; consequently, its time consumption is extremely high, which may be unacceptable in some real-time scenarios. We conduct an additional experiment to compare the running times of the above-mentioned methods on a computer with an Intel I5 (3.4 GHz) CPU and 8GB of RAM (GPU boosting of TensorFlow is disabled for fair comparison). The running time of each method is tested over 100 episodes; the results are listed in Table 2. The slight performance superiority of D^{2}WS over PNCQ is based on the fact that the computational burden of the former is two orders of magnitude greater than that of latter. Owing to extremely short computational times of PNCQ, MLPQ, and FCFS+, we compare them by focusing on their scheduling performances.

The results of the deadlock rates shown in Fig. 10(b) reflect another advantage of the proposed method in that it can realize deadlock-resistant scheduling without any extrinsic control policy. Following convergence after 200,000 learning steps, only 9 out of 28,000 subsequent evaluation episodes scheduled by PNCQ are deadlocked (0.03% probability). By contrast, the trajectory of MLPQ's deadlock rate fluctuates during the entire learning process (655 out of 28,000) owing to its instability. In real-world applications, PNCQ can be integrated with SMS control policies [33] to achieve absolute deadlock-free scheduling.

In summary, the proposed method can achieve excellent scheduling performance comparable to that of the heuristic search method but with much faster computation. Compared to basic MLP, PNCQ is more robust and achieves faster convergence.

5.3. Environmental adaptability

In practice, there usually is a gap between the real-world environment and the assumed simulation environment, e.g., the real arrival rule of raw products may not exactly follow the distribution assumed in

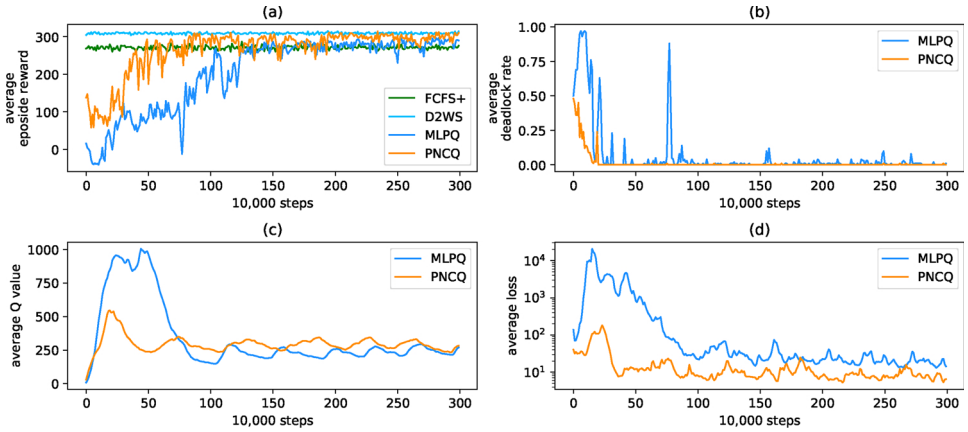

Fig. 10. Results of the dynamic scheduling experiment.

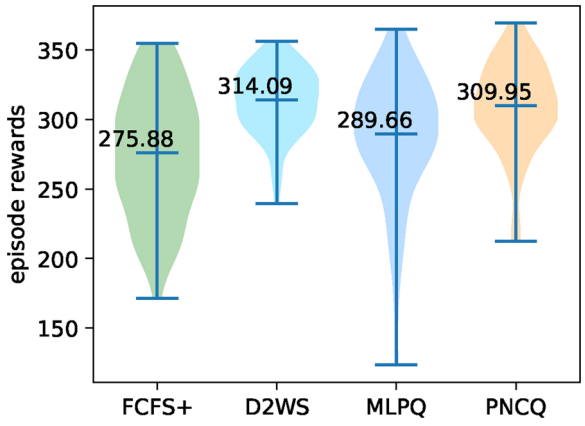

Fig. 11. Episode rewards of the last 100 evaluation episodes.

Table 2
Running time of an episode.

	PNCQ	FCFS +	D^2WS	MLPQ
Mean	135 ms	82 ms	34381 ms	48 ms
Min	115 ms	61 ms	13820 ms	38 ms
Max	171 ms	116 ms	59335 ms	62 ms

a simulation. A smart scheduling method should be adaptive to bridge the aforementioned gap. To this end, the equal probabilities (1:1:1) of the three product types in the second experiment are modified into an extreme non-uniform distribution (1:10:10) in this experiment, with the arrival interval distribution remaining unchanged. This adjustment leads to a more severe shortage of resources p_{26}, p_{27}, p_{28}, and p_{29}, and the experiment is conducted to verify the adaptability of the scheduling methods to this adjustment.

After 3,000,000 learning steps of the second experiment, the proposed method continues the training process in the adjusted environment for another 250,000 steps with $\epsilon = 0.1$, and the other settings are the same as those of the second experiment. Evaluations of PNCQ, FCFS +, and D^2WS are also conducted every 10,000 steps; the results are shown in Fig. 12.

Owing to the greater resource shortage after the arrival rule change, the episode rewards obviously decrease (around 196.6 for FCFS + and 257.5 for D^2WS) as shown in Fig. 12(a). PNCQ requires 150,000 learning steps to achieve its optimal performance, i.e., 277.9, which is 41.4% and 7.9% higher than those of FCFS + and D^2WS, respectively.Fig. 12(b) indicates the convergence of PNCQ.

We analyze the possible reason for the superiority of PNCQ over the heuristic search method D^2WS in this scenario. The heuristic function of D^2WS determines that this method tends to select the operation route with a shorter processing time, e.g., $p_{12}p_{14}p_{16}$ but not $p_{11}p_{13}p_{15}$ for Type 2 products. However, because few Type 1 products exist in the system after the arrival rule change, the route $p_{11}p_{13}p_{15}$ reduces resource contention with Type 3. The proposed method can learn this pattern without supervision; however, the heuristic function in D^2WS is not adaptive to this change. A case study to demonstrate the adaptability of the proposed method is shown in Fig. 13; we can see that the Q values of the DRL scheduling agents trained in the second experiment are clearly different from those in this experiment. After fitting in this experiment, PNCQ tends to choose the route $p_{11}p_{13}p_{15}$.

In summary, the proposed method, i.e., the masked DQN with the PNC network, can continuously adjust the scheduling policy to bridge the aforementioned environmental gap.

6. Conclusions and future works

To solve the dynamic scheduling problem of an FMSs involving shared resources, route flexibility, and stochastic arrivals of raw products, this paper proposed a novel Petri-net-based dynamic scheduling approach via DQN with GCN.

First, the timed S^3PR was used to model an FMS in terms of the operation sequential order, resource utilization constraints, and processing time. Then, according to specific subnet structures of the timed S^3PR, a PNC layer was designed with two graph convolution sub-layers that implement feature propagation from places to transitions and from transitions to places, respectively. The advantage of the PNC layer is that the number of its trainable parameters is related only to the number of filter channels and not to the scale of the timed S^3PR; thus, it is possible to overcome the parameter explosion problem when building deep neural networks. Finally,

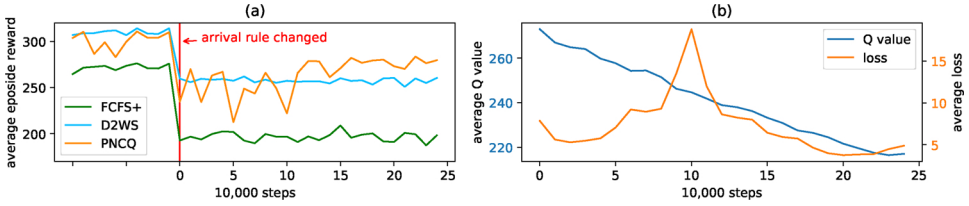

Fig. 12. Results of the adaptivity experiment.

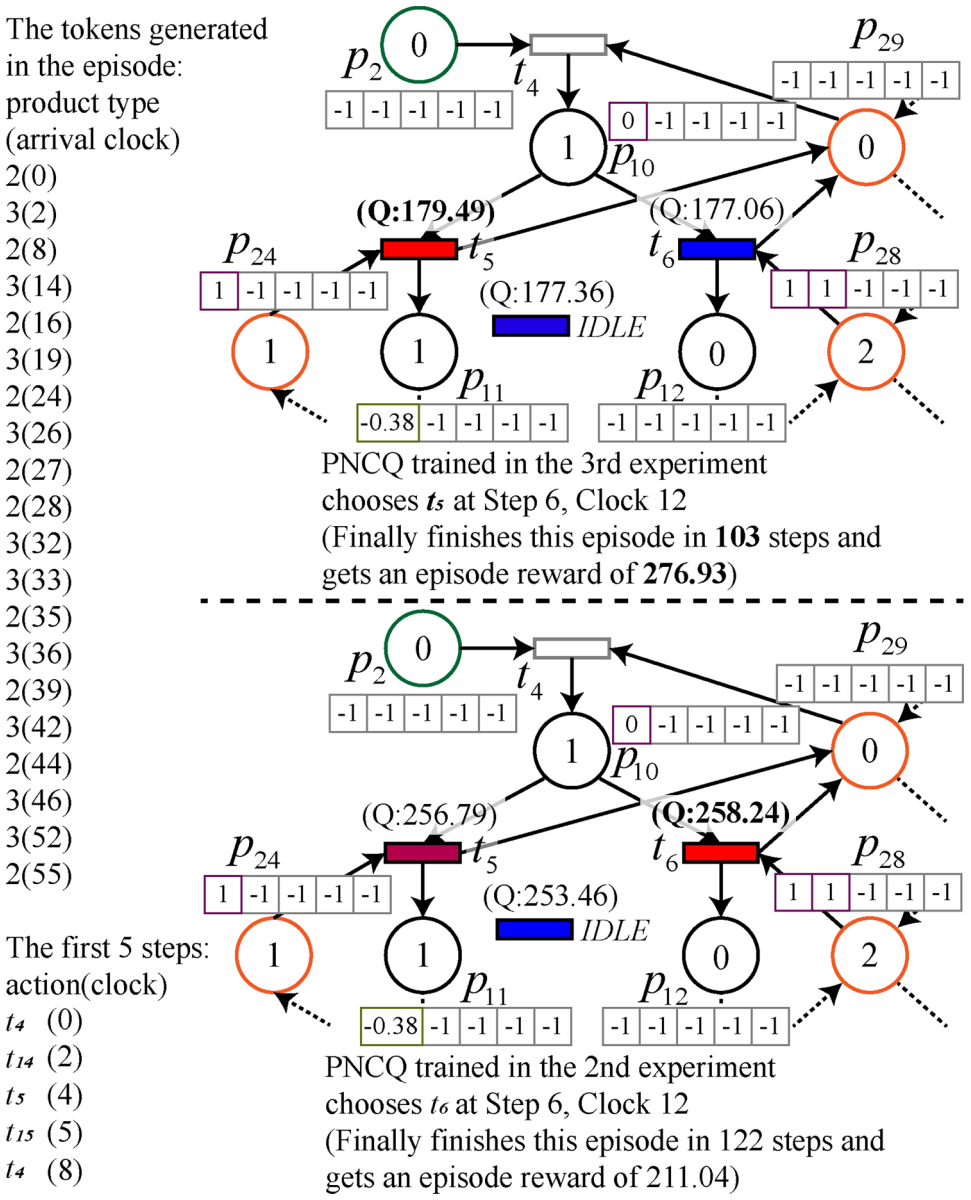

Fig. 13. Case study of the adaptivity experiment.

a masked DQN with the PNC network was employed for solving the timed S^3PR dynamic scheduling problem defined by the MDP.

Three experiments were designed to verify the advantages of the proposed method. The first experiment proved that even the simple PNC network can effectively handle the timed S^3PR states with better stability than an MLP. The second experiment showed that the proposed masked DQN with the PNC network can achieve similar dynamic scheduling performance with much faster online computation compared to the heuristic search method. In addition, the scheduling performance, learning convergence, and robustness of the proposed method were shown to be better than those of the MLP-based method. The third experiment verified that the proposed method is more adaptive to environmental changes than heuristic methods.

Finally, we identify three directions for future work: (1) insightful theoretical explorations about the PNC layer, e.g., the meanings of the hidden features, to guide the fine-tuning of the PNC network and the learning process, and further improve the accuracy of the proposed method; (2) dynamic scheduling problems of a more complex FMS model with stochastic machine breakdown and stochastic processing time; (3) smart cooperative multi-agent learning for distributed manufacturing systems.

Algorithm 1. DES procedure of the timed S^3PR

Input: selected action a

Output: state, mask, reward, done

if *a is a schedulable transition* **then**
  Fire(a) to update the marking and the token time features;
  if *there exists an empty SMS* **then**
    Trapped in deadlock and **return** *(Null, $\emptyset$, $-\theta_0$, True)*;
  end
else
  τ=next event time;
end
$r = 0$;
while *True* **do**
  if *products arrive at* τ **then**
    put tokens into the corresponding $p_B \in P_B$;
  end
  while *there exists an enabled autonomous transition* t' *at* τ **do**
    Fire(t') to update the marking and the token time features;
    Calculate the reward β_τ of the last completed token;
    $r = r + \beta_\tau$;
  end
  Calculate the mask ξ;
  if $|\xi| = 0$ **then**
    if *there exists a next event* **then**
      τ=next event time;
    else
      return *(Null, $\emptyset$, r, True)*;
    end
  else if $|\xi| = 1$ **then**
    Fire($t' \in \xi$) to update the marking and the token time features;
  else
    $r = r + \alpha(\boldsymbol{m}_\tau)$;
    Encode the marking and the token time features into a state s;
    return *(s, ξ, r, False)*;
  end
end

Algorithm 2. DQN with mask

Input: steps C, warmup steps C_0, train interval C_1, target update interval C_2, prioritized experience replay parameters K

Initialize prioritized experience replay memory $\mathcal{H} = \emptyset$ with K;
Warm up $\mathcal{H}$ for C_0 steps;
Initialize action-value function Q with random trainable parameters;
Initialize target action-value function $\hat{Q} = Q$;
Initialize timed S^{3}PR model and observe s, ξ;
for $i = 1$ *to* C **do**
  select action $a = \begin{cases} \text{a random one in } \xi \text{ with probability } \epsilon \\ \arg\max_{a^*}(Q(s, a^*) + \xi) \text{ otherwise} \end{cases}$;
  $(s', \xi', r, d) =$ execute Algorithm 1 by a;
  Store experience $(s, s', \xi, \xi', a, r, d)$ in $\mathcal{H}$ with maximal priority;
  if d=*True* **then**
    Initialize timed S^{3}PR model and observe s, ξ;
  else
    $s = s', \xi = \xi'$;
  end
  Discount ϵ;
  if $(i+1) \mod C_1 = 0$ **then**
    Sample random mini-batch of experiences $(\boldsymbol{s}, \boldsymbol{s}', \boldsymbol{\xi}, \boldsymbol{\xi}', \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{d})$ from $\mathcal{H}$ with priorities;
    Set $y = \begin{cases} r & d = True \\ r + \gamma \max_{a^*}(\hat{Q}(s', a^*) + \xi') & \text{otherwise} \end{cases}$;
    Compute TD-error $\delta = y - Q(s, a)$;
    Train Q by $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{y})$;
    Update experience priorities by $\boldsymbol{\delta}$;
  end
  if $(i+1) \mod C_2 = 0$ **then**
    Set $\hat{Q} = Q$;
  end
end

Acknowledgments

Support from the National Natural Science Foundation of China (grant Nos. 51935009, 51821093, 51490663) is gratefully acknowledged.

Conflicts of interest: The authors declare no conflicts of interest.

References

[1] Mejía G, Odrey NG. An approach using petri nets and improved heuristic search for manufacturing system scheduling. J Manuf Syst 2005;24(2):79–92. https://doi.org/10.1016/S0278-6125(05)80009-3.
[2] Uhlmann IR, Frazzon EM. Production rescheduling review: opportunities for industrial integration and practical applications. J Manuf Syst 2018;49:186–93. https://doi.org/10.1016/j.jmsy.2018.10.004.
[3] Denno P, Dickerson C, Harding JA. Dynamic production system identification for smart manufacturing systems. J Manuf Syst 2018;48:192–203. https://doi.org/10.1016/j.jmsy.2018.04.006.
[4] Leng J, Jiang P. Dynamic scheduling in RFID-driven discrete manufacturing system by using multi-layer network metrics as heuristic information. J Intell Manuf 2019;30(3):979–94. https://doi.org/10.1007/s10845-017-1301-y.
[5] Tao F, Qi Q, Liu A, Kusiak A. Data-driven smart manufacturing. J Manuf Syst 2018;48:157–69. https://doi.org/10.1016/j.jmsy.2018.01.006.
[6] Coronado PDU, Lynn R, Louhichi W, Parto M, Wescoat E, Kurfess T. Part data integration in the shop floor digital twin: mobile and cloud technologies to enable a manufacturing execution system. J Manuf Syst 2018;48:25–33. https://doi.org/10.1016/j.jmsy.2018.02.002 special Issue on Smart Manufacturing.
[7] Papacharalampopoulos A, Stavropoulos P. Towards a digital twin for thermal processes: control-centric approach. 7th CIRP global web conference 2019.
[8] Nikolakis N, Alexopoulos K, Xanthakis E, Chryssolouris G. The digital twin implementation for linking the virtual representation of human-based production tasks to their physical counterpart in the factory-floor. Int J Comput Integr Manuf 2019;32(1):1–12. https://doi.org/10.1080/0951192X.2018.1529430.
[9] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, et al. Human-level control through deep reinforcement learning. Nature 2015;518(7540):529–33. https://doi.org/10.1038/nature14236.
[10] Silver D, Hubert T, Schrittwieser J, Antonoglou I, Lai M, Guez A, et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science 2018;362(6419):1140–4. https://doi.org/10.1126/science.aar6404.
[11] Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. Mastering the game of go without human knowledge. Nature 2017;550(7676):354–9. https://doi.org/10.1038/nature24270.
[12] Ou X, Chang Q, Chakraborty N. Simulation study on reward function of reinforcement learning in gantry work cell scheduling. J Manuf Syst 2019;50:1–8. https://doi.org/10.1016/j.jmsy.2018.11.005.
[13] Duguleana M, Barbuceanu FG, Teirelbar A, Mogan G. Obstacle avoidance of redundant manipulators using neural networks based reinforcement learning. Robot Comput-Integr Manuf 2011. https://doi.org/10.1016/j.rcim.2011.07.004. S0736584511000962.
[14] Sutton RS, Barto AG. Reinforcement learning: an introduction. MIT Press; 2018.
[15] Waschneck B, Reichstaller A, Belzner L, Altenmüller T, Bauernhansl T, Knapp A, et al. Optimization of global production scheduling with deep reinforcement learning. Proc CIRP 2018;72:1264–9. https://doi.org/10.1016/j.procir.2018.03.212.
[16] Angrish A, Starly B, Lee Y-S, Cohen PH. A flexible data schema and system architecture for the virtualization of manufacturing machines (vmm). J Manuf Syst 2017;45:236–47. https://doi.org/10.1016/j.jmsy.2017.10.003.
[17] Hu L, Liu Z, Tan J. A vr simulation framework integrated with multisource cae analysis data for mechanical equipment working process. Comput Ind 2018;97:85–96. https://doi.org/10.1016/j.compind.2018.01.009.
[18] Aydin M, Öztemel E. Dynamic job-shop scheduling using reinforcement learning agents. Robot Auton Syst 2000;33(2–3):169–78. https://doi.org/10.1016/S0921-8890(00)00087-7.
[19] Hammami Z, Mouelhi W, Said LB. On-line self-adaptive framework for tailoring a neural-agent learning model addressing dynamic real-time scheduling problems. J Manuf Syst 2017;45:97–108. https://doi.org/10.1016/j.jmsy.2017.08.003.
[20] Leng J, Zhang H, Yan D, Liu Q, Chen X, Zhang D. Digital twin-driven manufacturing cyber-physical system for parallel controlling of smart workshop. J Ambient Intell Humaniz Comput 2019;10(3):1155–66. https://doi.org/10.1007/s12652-018-0881-5.
[21] Leng J, Jiang P, Liu C, Wang C. Contextual self-organizing of manufacturing process for mass individualization: a cyber-physical-social system approach. Enterp Inf Syst 2018:1–26. https://doi.org/10.1080/17517575.2018.1470259.
[22] Huang Z, Kim J, Sadri A, Dowey S, Dargusch MS. Industry 4.0: development of a multi-agent system for dynamic value stream mapping in smes. J Manuf Syst 2019;52:1–12. https://doi.org/10.1016/j.jmsy.2019.05.001.
[23] Qu S, Chu T, Wang J, Leckie J, Jian W. A centralized reinforcement learning approach for proactive scheduling in manufacturing. 2015 IEEE 20th conference on emerging technologies & factory automation (ETFA). 2015. p. 1–8.
[24] Shahrabi J, Adibi MA, Mahootchi M. A reinforcement learning approach to parameter estimation in dynamic job shop scheduling. Comput Ind Eng 2017;110:75–82. https://doi.org/10.1016/j.cie.2017.05.026.
[25] Schaul T, Quan J, Antonoglou I, Silver D. Prioritized experience replay. 2015arXiv:1511.05952.
[26] Wang Z, Schaul T, Hessel M, Van Hasselt H, Lanctot M, De Freitas N. Dueling network architectures for deep reinforcement learning. 2015arXiv:1511.06581.
[27] Petri C. Nets, time and space. Theor Comput Sci 1996;153(1–2):3–48. https://doi.org/10.1016/0304-3975(95)00116-6.
[28] Başak Ö, Albayrak YE. Petri net based decision system modeling in real-time scheduling and control of flexible automotive manufacturing systems. Comput Ind Eng 2015;86:116–26. https://doi.org/10.1016/j.cie.2014.09.024.
[29] Baruwa OT, Piera MA. Identifying fms repetitive patterns for efficient search-based scheduling algorithm: a colored petri net approach. J Manuf Syst 2015;35:120–35. https://doi.org/10.1016/j.jmsy.2014.11.009.
[30] Ezpeleta J, Colom J, Martinez J. A Petri net based deadlock prevention policy for flexible manufacturing systems. IEEE J Robot Autom 1995;11(2):173–84. https://doi.org/10.1109/70.370500.
[31] You D, Wang S, Zhou M. Computation of strict minimal siphons in a class of Petri nets based on problem decomposition. Inf Sci 2017;409–410:87–100. https://doi.org/10.1016/j.ins.2017.05.011.
[32] Wang S, Wang C, Zhou M, Li Z. A method to compute strict minimal siphons in a class of petri nets based on loop resource subsets. IEEE Trans Syst Man Cybern A 2012;42(1):226–37. https://doi.org/10.1109/TSMCA.2011.2159590.
[33] You D, Wang S, Zhou M. Synthesis of monitor-based liveness-enforcing supervisors for S^3PR with ξ-resources. IEEE Trans Syst Man Cybern Syst 2015;45(6):967–75. https://doi.org/10.1109/TSMC.2014.2376476.
[34] Chao DY. A new optimal control policy for a well-known S ^{3}PR (systems of simple sequential processes with resources). Int J Prod Res 2012;50(22):6259–71. https://doi.org/10.1080/00207543.2011.623725.
[35] Cheng C, Sun T, Fu L. Petri-Net based modeling and scheduling of a flexible manufacturing system. IEEE Comput Soc Press 1994:513–8. https://doi.org/10.1109/ROBOT.1994.351246.
[36] Kim H-J, Lee J-H, Lee T-E. Time-feasible reachability tree for noncyclic scheduling of timed petri nets. IEEE Trans Autom Sci Eng 2015;12(3):1007–16. https://doi.org/10.1109/TASE.2014.2313979.
[37] Baruwa OT, Piera MA, Guasch A. Deadlock-free scheduling method for flexible manufacturing systems based on timed colored petri nets and anytime heuristic search. IEEE Trans Syst Man Cybern Syst 2015;45(5):831–46. https://doi.org/10.1109/TSMC.2014.2376471.
[38] Luo J, Xing K, Zhou M, Li X, Wang X. Deadlock-free scheduling of automated manufacturing systems using petri nets and hybrid heuristic search. IEEE Trans Syst Man Cybern Syst 2015;45(3):530–41. https://doi.org/10.1109/TSMC.2014.2351375.
[39] Wang Y, Duan Z, Liao B, Wu F, Zhuang Y. Heterogeneous attributed network embedding with graph convolutional networks. 33rd AAAI conference on artificial intelligence. 2019. p. 10061–2.
[40] Sakiyama A, Tanaka Y, Tanaka T, Ortega A. Accelerated sensor position selection using graph localization operator. 2017 IEEE international conference on acoustics, speech and signal processing (ICASSP) 2017. p. 5890–4. https://doi.org/10.1109/ICASSP.2017.7953286.
[41] Duvenaud DK, Maclaurin D, Iparraguirre J, Bombarell R, Hirzel T, Aspuru-Guzik A, et al. Convolutional networks on graphs for learning molecular fingerprints. Advances in neural information processing systems. 2015. p. 2224–32.
[42] Sandryhaila A, Moura JMF. Discrete signal processing on graphs. IEEE Trans Signal Process 2013;61(7):1644–56. https://doi.org/10.1109/TSP.2013.2238935.
[43] Sandryhaila A, Moura JM. Discrete signal processing on graphs: graph fourier transform. ICASSP 2013:6167–70. https://doi.org/10.1109/ICASSP.2013.6638850.
[44] Shuman DI, Narang SK, Frossard P, Ortega A, Vandergheynst P. The emerging field of signal processing on graphs: extending high-dimensional data analysis to networks and other irregular domains. IEEE Signal Process Mag 2013;30(3):83–98. https://doi.org/10.1109/MSP.2012.2235192.
[45] Niepert M, Ahmed M, Kutzkov K. Learning convolutional neural networks for graphs. International conference on machine learning 2016:2014–23.
[46] Defferrard M, Bresson X, Vandergheynst V. Convolutional neural networks on graphs with fast localized spectral filtering. Lee DD, Sugiyama M, Luxburg UV, Guyon I, Garnett R, editors. Advances in neural information processing systems, 29. 2016. p. 3844–52.
[47] Murata T. Petri nets: properties, analysis and applications. Proc IEEE 1989;77(4):541–80. https://doi.org/10.1109/5.24143.
[48] Hammond DK, Vandergheynst P, Gribonval R. Wavelets on graphs via spectral graph theory. Appl Comput Harmon Anal 2011;30(2):129–50. https://doi.org/10.1016/j.acha.2010.04.005.
[49] Kipf TN, Welling M. Semi-supervised classification with graph convolutional networks. 2016arXiv:1609.02907.
[50] Ioffe S, Szegedy C. Batch normalization: accelerating deep network training by reducing internal covariate shift. 2015arXiv:1502.03167.
[51] Maas AL, Hannun AY, Ng AY. Rectifier nonlinearities improve neural network acoustic models. Proc ICML 2013;30:3.
[52] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, et al. TensorFlow: large-scale machine learning on heterogeneous systems, software available from tensorflow.org. 2015https://www.tensorflow.org/.
[53] Chollet F, et al. Keras. 2015https://github.com/fchollet/keras.
[54] Kingma DP, Ba J. Adam: a method for stochastic optimization. 2014arXiv:1412.6980.
[55] Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, et al. Openai gym. 2016arXiv:1606.01540.
[56] Dhariwal P, Hesse C, Klimov O, Nichol A, Plappert M, Radford A, et al. Openai baselines. 2017https://github.com/openai/baselines.
[57] Plappert M. keras-rl. 2016https://github.com/keras-rl/keras-rl.