2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Scheduling of Robotic Cellular Manufacturing Systems with Timed Petri Nets and Reinforcement Learning

ZhuTao Yao, Bo Huang, JianYong Lv, XiaoYu Lu, MeiJi Cui, and ShaoHua Yu

***Abstract*— This paper proposes a new Petri-net-based Q-learning scheduling method to schedule robotic cellular manufacturing (RCM) systems efficiently. First, we use generalized and place-timed Petri nets to model RCM systems. Then, we design a reinforcement learning method with a sparse Q-table to evaluate state-transition pairs of the net's reachability graph. It uses the negative transition firing time as a reward for an action selection and adopts a large penalty for any encountered deadlock. In addition, it balances the state space exploration and the experience exploitation by using a dynamic ϵ-greedy policy to update the state values with an accumulative reward. Three different dynamic ϵ-greedy policies are designed for different application scenarios. Some benchmark RCM systems are tested with the proposed method and several popular PN-based online dispatching rules, such as FIFO and SRPT. Simulation results demonstrate that our method schedules RCM systems as quickly as the online dispatching rules while outperforming them in terms of schedule makespan. For readers' reference, our source code and test data are available at https://github.com/PNOptimizer/PNQL.**

I. INTRODUCTION

Robotic cellular manufacturing (RCM) systems are prevalent across various manufacturing industry sectors, including semiconductor wafer fabrication, automobile manufacturing, and intelligent transportation. These systems typically manage multiple concurrent jobs and operations, often contending for finite shared resources like robots and machines. They often involve a wide range of resource options and processing routes, posing a challenging problem of efficiently allocating various resources to different operations while scheduling operations to maximize production efficiency. Optimally scheduling these systems is categorized as an NP-hard problem [1].

A Petri net (PN) [2] is a mathematical tool that is well-suited for modeling, analyzing, and controlling discrete event systems (DESs), particularly effective for representing structures such as sequence, concurrency, conflict, and synchronization within DESs. Additionally, the reachability graph (RG) derived from a PN comprehensively represents the system's behavior.

Reinforcement learning (RL) [3] is a powerful intelligent method to learn the optimal behavior in an environment to obtain a maximum accumulative reward. This optimal behavior is typically learned by an agent through interactions with the environment and observations of how the environment responds.

In recent years, some advancements have been made in applying RL techniques to PNs to optimize system scheduling. For example, Drakaki and Tzionas [4] introduced a scheduling method for manufacturing systems using timed colored PNs and RL, demonstrating its efficiency through a warehouse order-picking case study. Hu *et al.* [5] proposed a method to tackle the dynamic scheduling problem in flexible manufacturing systems using deep RL. Similarly, Lee and Kim [6] developed an RL algorithm for Petri net models to schedule robotic flow shop systems. However, existing RL-based methods still encounter challenges in effectively resolving scheduling issues in generalized PNs with deadlocks, primarily due to the inadequate and imprecise utilization of PN dynamic information within the RL framework. To address this, this paper proposes a timed PN-based Q-learning (QL) scheduling method for RCM systems, offering the following contributions:

1) This method is the first to integrate generalized and place-timed Petri nets with reinforcement learning for scheduling applications.
2) It assigns a negative value to the time required to fire an enabled transition at a state as a reward, imposes severe penalties for any encountered deadlocks and utilizes a sparse Q-Table with a dynamic ϵ-greedy strategy. This strategy aids in efficiently updating action values to balance the algorithm's exploration and exploitation. Moreover, it includes three dynamic ϵ-greedy policies tailored for various application scenarios.
3) It matches the speed of commonly used online dispatching rules and surpasses them in delivering superior scheduling quality.

II. PRELIMINARIES

This section provides an overview of PN, place-timed PN for scheduling, and the RL method. For further information on them, readers are encouraged to consult [7], [8].

A. Petri Net

Definition 1: PNs are defined as a four-tuple $N = (P, T, F, W)$, where P and T respectively represent finite sets of places and transitions with $P \cap T = \emptyset$. The set $F \subseteq (P \times T) \cup (T \times P)$ includes directed arcs linking places and transitions. The function $W : F \to \mathbb{Z}^+$ assigns positive weights to all arcs. A PN N is termed ordinary if, for every $(x, y) \in F$, $W(x, y) = 1$. On the other hand, N is categorized as generalized if $\exists(x, y) \in F$, $W(x, y) > 1$. For a transition $t \in T$, its input places are denoted as ${}^\bullet t = \{p \in P | (p, t) \in F\}$, and its output places are

The authors are with the School of Computer Science & Engineering, Nanjing University of Science & Technology, Nanjing 210094, China (email: {zhutaoyao, huangbo, lv_jy, xiaoyu.lu, cui_mj, shaohua.yu}@njust.edu.cn).

represented as $t^{\bullet} = \{p \in P | (t, p) \in F\}$. Similarly, for any place $p \in P$, its input transitions are expressed as ${}^{\bullet}p = \{t \in T | (t, p) \in F\}$, and its output transitions are denoted as $p^{\bullet} = \{t \in T | (p, t) \in F\}$.

Definition 2: In a PN N, the marking $M : P \to \mathbb{Z}$ represents the token distribution in places. The initial marking of N is denoted as M_0. At a given marking M, a transition t is enabled if, for all input places $p \in {}^{\bullet}t$, the token count $M(p)$ exceeds or equals the weight $W(p, t)$. Firing an enabled transition t at marking M produces a new marking M', where tokens are transferred according to the transition weights. The set of all reachable markings from M is denoted as $\mathcal{R}(N, M)$.

Definition 3: A pair (N, M_0) constitutes a net system. A net system (N, M_0) is considered k-bounded if $\forall M \in \mathcal{R}(N, M_0)$ and $\forall p \in P$, there exists a positive integer k such that $M(p) \le k$. For the net system (N, M_0), a place invariant $I : P \to \mathbb{Z}$ is defined as a non-zero $|P|$-dimensional integer vector satisfying $I^T[N] = \mathbf{0}^T$ and, for all reachable markings $M \in \mathcal{R}(N, M_0)$, $I^T M = I^T M_0$. Here, $[N] = [N]^+ - [N]^-$ represents the incidence matrix of N, with $[N]^+(p, t) = W(t, p)$ and $[N]^-(p, t) = W(p, t)$, and $\mathbf{0}$ denotes a zero vector. A place invariant I is termed a P-semiflow if all its elements are non-negative.

B. Place-timed PNs for System Scheduling

In this subsection, we examine the definition of a deterministic place-timed PN, referred to as an RSP-net, utilized for addressing the system scheduling problem [7].

Definition 4: A place-timed PN system for scheduling is $(\mathcal{N}, S_0)$, where S_0 is an initial state and $\mathcal{N} = (P_S \cup P_E \cup P_R \cup P_A, T, F, W, D)$ represents a generalized place-timed PN including process subnets $\mathcal{N}^x = (P_S^x \cup P_E^x \cup P_R^x \cup P_A^x, T^x, F^x, W^x, D^x)$ that share certain places, i.e., $\mathcal{N} = \cup_{x \in \mathbb{N}_J} \mathcal{N}^x$ with $\mathbb{N}_J = \{i \in \mathbb{Z}^+ | \mathcal{N}^i$ is the ith subnet of $\mathcal{N}\}$, and the following conditions are satisfied.

1) For a process subnet $\mathcal{N}^x$, $P^x = P_S^x \cup P_E^x \cup P_A^x \cup P_R^x$ where P_S^x, P_E^x, P_A^x, and P_R^x are the sets representing start places, end places, activity places, and resource places of $\mathcal{N}^x$, respectively. $\forall x \in \mathbb{N}_J$, $|P_S^x| = |P_E^x| = 1$.
2) $P_A^x \neq \emptyset$, $P_R^x \neq \emptyset$, $P_S^x \cap P_E^x = \emptyset$, $(P_S^x \cup P_E^x) \cap P_A^x = \emptyset$, and $(P_S^x \cup P_E^x \cup P_A^x) \cap P_R^x = \emptyset$.
3) $W = W_A \cup W_R$ with $W_A : F \cap ((P_A \cup P_S \cup P_E) \times T) \cup (T \times (P_A \cup P_S \cup P_E)) \to \{1\}$ and $W_R : F \cap ((P_R \times T) \cup (T \times P_R)) \to \mathbb{Z}^+$.
4) $\forall r \in P_R$, there exists a unique minimum P-semiflow I_r satisfying that $\|I_r\| \cap P_R = \{r\}$, $\|I_r\| \cap (P_A \cup P_S \cup P_E) \neq \emptyset$, and $I_r(r) = 1$. In addition, $P_A \subseteq \bigcup_{r \in P_R} (\|I_r\| \setminus \{r\})$.
5) The places shared by $\mathcal{N}^x$ and $\mathcal{N}^y (x \neq y)$ constitute a subset of P_R.
6) $D : P_A \to \mathbb{Z}$ denotes the assignment of deterministic operation time to P_A.

Based on such a PN, we can model an RCM system for scheduling by using a bottom-up method. For instance, we examine an RCM system [9] with two robots $\mathrm{R}_1 - \mathrm{R}_2$, four types of machines $\mathrm{M}_1 - \mathrm{M}_4$, two loading buffers $\mathrm{I}_1 - \mathrm{I}_2$, and two unloading buffers $\mathrm{O}_1 - \mathrm{O}_2$. It handles two types of parts (P_1 and P_2) following the specified routes.

$$\mathrm{P}_1 : \mathrm{I}_1 \to \mathrm{R}_1(3) \to \mathrm{M}_1(4) \to \mathrm{R}_1(4) \to \mathrm{M}_3(3) \to \mathrm{R}_2(5) \to \mathrm{O}_1$$
$$\text{or } \mathrm{I}_1 \to \mathrm{R}_1(3) \to \mathrm{M}_2(2) \to \mathrm{R}_1(4) \to \mathrm{M}_3(3) \to \mathrm{R}_2(5) \to \mathrm{O}_1$$
$$\mathrm{P}_2 : \mathrm{I}_2 \to \mathrm{R}_2(2) \to \mathrm{M}_4(4) \to \mathrm{R}_1(4) \to \mathrm{M}_2(3) \to \mathrm{R}_1(5) \to \mathrm{O}_2$$

The bottom-up modeling method starts by constructing different process subnets individually, ignoring their interconnections. Then, subnets are merged into a larger net through shared resource places. Finally, initial tokens and time information are added to start places and activity places, respectively. Its final RSP-net is shown in Fig. 1, which has 14 transitions and 21 places.

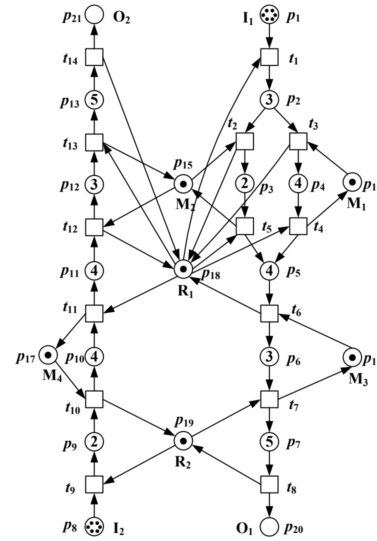

Fig. 1. The RSP-net of the example RCM system.

C. Reinforcement Learning

Reinforcement learning (RL) aims to address how one or more agents can maximize cumulative rewards in a complex and uncertain environment. An RL problem consists of the following four main elements to build its model:

1) Agent. An agent in RL refers to the decision-making entity in RL that interacts with the environment.
2) Environment. An environment provides the state information to the agent and determines the impact of actions through state transitions.
3) State space $\mathcal{S}$ and action space $\mathcal{A}$. They define all possible states and actions in the problem domain.
4) Reward. The reward r represents scalar feedback provided by the environment to assess the action taken by the agent.

The interaction between the agent and the environment can be formalized as a Markov Decision Process (MDP) [10], which unfolds as follows at each time step τ: The agent receives the current state s_τ from the environment. Based on this state, the agent picks an action a_τ. Subsequently, the

environment transitions to the next state $s_{\tau+1}$ and issues a reward $r_{\tau+1}$. This interaction persists indefinitely or until a goal state is attained. The MDP modeling encapsulates the essence of sequential decision-making problems. An interaction trajectory from the initial state s_0 to the goal state s_G can be represented as $(s_0, a_0, r_0; s_1, a_1, r_1; \ldots, s_n, a_n, s_G)$.

The agent adopts a policy π to determine its next action. A policy $\pi(a|s)$ represents the function that an agent uses to select actions based on a state s. The ultimate goal of an agent of RL is to learn an optimal policy π^* through the interactions with the environment, allowing the agent to make optimized decisions in a changing environment and maximize cumulative expected discounted return:

$$G_\tau = r_{\tau+1} + \gamma r_{\tau+2} + \gamma^2 r_{\tau+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{\tau+k+1}, \quad (1)$$

where $0 \leq \gamma \leq 1$ represents the discount rate that indicates the influence of future rewards. Note that a lower γ value makes the agent more focused on immediate rewards and a higher γ considers future rewards more strongly.

The state value function gives the reward the agent can get if it adopts a policy π from a state s, and it can be described as below:

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{\tau+k+1} \mid s_\tau = s \right]. \quad (2)$$

III. System Scheduling with PN and RL

This section proposes an RCM scheduling algorithm leveraging RSP-net and Q-learning (QL) [8]. QL is a model-free and off-policy reinforcement learning method, designed to obtain the optimal action based on the agent's current state. Model-free implies that the agent (agent) does not need prior knowledge of the dynamic properties of the environment during the training process, but learns the optimal behavior by interacting with the environment. QL aims to identify the most favorable action given its current state, often generating its own rules or deviating from prescribed policies, hence said to be off-policy. QL employs Q-values $Q(s, a)$ to assess the value of an action a taken at a specific state s.

In the RG of a place-timed RSP-net for an RCM system, a state s consists of two elements: the current marking M and the remaining token time matrix R for all activity places. The state space $\mathcal{S}$ is thus the set of all states in the RG. Furthermore, the possible actions at a state s are the transitions enabled at s, denoted by $\mathcal{E}_s$. Consequently, the action space $\mathcal{A}$ is the set of all transitions that can be enabled and fire at a specific state. With the MDP environment, state space, and action space, the system executor can use a QL method and act as an agent to perform an action at a state s, transitioning to another state and receiving a reward from the environment. Building on this foundation, the proposed RCM scheduling algorithm, which integrates the RSP-net with QL and is referred to as PNQL, is detailed in Algorithm 1.

The Q-Table Q plays a crucial role in the PNQL method. Traditional QL algorithms usually initialize the whole Q-Table with zeros [11]. However, for this scheduling problem, only some legal actions (i.e., enabled transitions) are available for a given state. So, many state-action pairs in the table may have no values. For example, in Fig. 1, only two transitions, t_1 and t_9, are enabled at the current state s_0, but the others are not enabled at the state. Thus, we have $Q(s_0, t_1) = Q(s_0, t_9) = 0$ and the other entries about s_0 in the Q-Table have no values, and we adopt such a sparse Q-Table for an efficient implementation in our PNQL method.

Then, the Q-Table entries with values may be iteratively

Algorithm 1 PNQL

Input: An RSP-net $\mathcal{N} = (P, T, F, W, D)$ with an initial state $s_0 = (M_0, R_0)$, a goal state $s_G = (M_G, R_G)$, the number of episodes Epi, the learning rate α, and the discount rate γ.

Output: A state-transition sequence from s_0 to s_G.

```
1:  /*Training phase.*/
2:  Q ← ∅;
3:  for episode ← 1 to Epi do
4:      s ← s_0;
5:      while s ≠ s_G do
6:          C(s) ← ∅;
7:          for each t enabled at s do
8:              if Q(s,t) = Null then
9:                  Q(s,t) ← 0 ;
10:             end if
11:             Fire t at s to generate s';
12:             C(s) ← C(s) ∪ {s'};
13:         end for
14:         Generate a random number η ∈ [0,1];
15:         if η < ε then
16:             Randomly select s' ∈ C(s);
17:         else
18:             Pick the state s' ∈ C(s) such that s[t⟩s' and
                t ← arg max_t Q(s,t);
19:         end if
20:         Q(s,t) ← Q(s,t) + α[r + γ(max_t Q(s',t) −
                Q(s,t))]; /*If ∀t ∈ T, Q(s',t) has no value, then
                max_{t∈T} Q(s',t) ← 0.*/
21:         if r = −Γ then
22:             break;
23:         end if
24:         s ← s';
25:     end while
26: end for
27: /*Execution phase.*/
28: Path ← ∅; s ← s_0;
29: while s ≠ s_G do
30:     Find a pair (s,t) with the maximal Q-value in Q;
31:     Path ← Path ∪ {(s,t)};
32:     Generate s' by firing t at s; s ← s';
33: end while
34: Return Path;
```

$$s' = \begin{cases} s' \text{ such that } s[t\rangle s' \text{ and } t = \arg\max_{t \in \mathcal{E}_s} Q(s,t), & \text{with probability } (1-\epsilon); \\ \text{Random } s' \in \mathcal{C}(s), & \text{with probability } \epsilon. \end{cases} \quad (9)$$

updated as follows:

$$Q(s,t) = Q(s,t) + \alpha[r + \gamma(\max_t Q(s',t) - Q(s,t))], \quad (3)$$

where s' is the successor state of s by firing t. Note that since our Q-Table is sparse, Thus, if for any transition t enabled at s', $Q(s',t)$ has no value, we let $\max_t Q(s',t) = 0$.

In an RCM scheduling problem, the reward r can be assigned based on the actual scheduling objectives, such as minimizing makespan, maximizing resource utilization, and minimizing tardiness. In this paper, we focus on minimizing makespan as our scheduling objective and define the following reward structure for the agent when firing transition t at state s in the PNQL method:

$$r = \begin{cases} -\Gamma, & \text{if } s \text{ is a deadlock state} \\ -\delta_{s,t}, & \text{otherwise,} \end{cases} \quad (4)$$

where Γ represents a large positive integer used as a penalty for encountering a deadlocked state in the PNQL method, and $\delta_{s,t}$ denotes the time required for the current state s to reach its successor state by firing transition t.

In addition, the PNQL method adopts a dynamic ϵ-greedy strategy to balance exploration and exploitation in the training process. In the ϵ-greedy strategy, the agent randomly selects a transition enabled at s (i.e., a permitted action at s) with probability ϵ and picks a transition with the highest Q-value at s with probability $1-\epsilon$. In our approach, we consider the following three dynamics ϵ-values, which decrease from 1 to 0.01 in different styles and at different speeds. But all of them encourage exploration in the early training stage and then a gradual shift toward exploitation. They are given as follows.

$$\epsilon_1 = 0.01^{\frac{episode}{Epi}}. \quad (5)$$

$$\epsilon_2 = 1 - 0.99 \times \frac{episode}{Epi}. \quad (6)$$

$$\epsilon_3 = -0.01^{\frac{Epi-episode}{Epi}} + 1.01. \quad (7)$$

The dynamic ϵ-greedy strategies that select a transition for firing at a given state can described as

$$t = \begin{cases} \arg\max_{t \in \mathcal{E}_s} Q(s,t), & \text{with probability } (1-\epsilon) \\ \text{Random } t \in \mathcal{E}_s, & \text{with probability } \epsilon. \end{cases} \quad (8)$$

Note that for a state s, there is a consistent one-to-one match between each transition enabled at s and each successor state of s. In addition, the list $\mathcal{C}(s)$ stores all successor states of s. Thus, we can use a successor state s' of s to replace the enabled transition t selected at s, and the resulting dynamic ϵ-greedy strategy can be given as Eq. (9).

In each episode of Algorithm 1, Q-values are updated based on the observed rewards and the maximum Q-value of the next state, until the goal state s_G or a deadlocked state is reached. After the agent learns for certain episodes, the accumulative reward may converge and an optimal or a near-optimal policy π^* can be obtained with the obtained action values Q^* using the following equation $\pi^* = \text{argmax}_{a \in \mathcal{A}} Q^*(s,a)$, i.e., the policy selects an action with the highest Q-value for any given state. In addition, PNQL can quickly obtain an executable schedule from s_0 to s_G in the RG of the system model. Such a process is presented in the execution phase of Algorithm 1.

Overall, PNQL is designed to derive an optimal or near-optimal transition firing sequence from the initial state s_0 to a goal state s_G in an RSP-net $\mathcal{N}$. This sequence corresponds to a series of operations within the underlying RCM system. The algorithm consists of two phases: training and execution. The training phase is a nested iterative process involving a specified number of episodes. Each episode initializes the current state s to s_0 and progresses through a series of decisions, influenced by a dynamic ϵ-greedy strategy employed throughout the training phase to enhance exploration. This strategy utilizes the specially designed reward system and a sparse Q-Table to update the Q-values. The process continues until the algorithm either reaches the goal state s_G or encounters a deadlock, prompting the start of a new episode. After completing a predetermined number of episodes, the algorithm leverages the obtained Q-Table to quickly make decisions for any given state or efficiently construct an execution path from s_0 to s_G in the RG of the PN, thereby guiding the underlying system.

IV. SIMULATION STUDY

In this section, we evaluate the efficacy of PNQL with three different ϵ settings by testing it on various benchmark RSP-nets designed for RCM systems, which are tested with our method and two commonly used online dispatching rules, i.e., FIFO (First In First Out) and SRPT (Shortest Remaining Processing Time) [12]. The FIFO method schedules tasks based on their arrival order, i.e., the task that arrives first is processed first. The SRPT method prioritizes tasks based on their remaining processing time, handling the task with the shortest remaining processing time first. The scheduling platform and all methods are implemented in the C# programming language and tested on a PC with an AMD CPU (3.2 GHz) and 16 GB of RAM. The hyperparameters for our PNQL are unified as follows: the deadlock penalty $\Gamma = 10000$, the discount rate $\gamma = 0.3$, and the learning rate $\alpha = 0.9$.

Then, we use three benchmark RCM systems in the literature as examples for comparing different methods. The

first one is the aforementioned net in Fig. 1. The second one is from [13], whose production sequences are:

$$\begin{aligned}
\mathrm{P}_1 : \mathrm{I}_1 &\rightarrow \mathrm{R}_1(3) \rightarrow \mathrm{M}_1(2) \rightarrow \mathrm{R}_2(6) \rightarrow \mathrm{M}_2(2) \rightarrow \mathrm{R}_3(4) \rightarrow \mathrm{O}_1 \\
\text{or} \ \ \mathrm{I}_1 &\rightarrow \mathrm{R}_1(3) \rightarrow \mathrm{M}_3(1) \rightarrow \mathrm{R}_2(3) \rightarrow \mathrm{M}_4(4) \rightarrow \mathrm{R}_3(4) \rightarrow \mathrm{O}_1 \\
\mathrm{P}_2 : \mathrm{I}_2 &\rightarrow \mathrm{R}_2(2) \rightarrow \mathrm{M}_2(4) \rightarrow \mathrm{R}_2(5) \rightarrow \mathrm{O}_2 \\
\mathrm{P}_3 : \mathrm{I}_3 &\rightarrow \mathrm{R}_3(6) \rightarrow \mathrm{M}_4(3) \rightarrow \mathrm{R}_2(4) \rightarrow \mathrm{M}_3(6) \rightarrow \mathrm{R}_1(2) \rightarrow \mathrm{O}_3
\end{aligned}$$

Its corresponding RSP-net is given in Fig. 2 with 20 transitions and 29 places. The last one comes from [14] with the following production sequences for four kinds of parts:

$$\begin{aligned}
\mathrm{P}_1 : \mathrm{I}_1 \rightarrow \mathrm{R}_1(5) \rightarrow \mathrm{R}_2(4) \rightarrow \mathrm{R}_3(4) \rightarrow \mathrm{O}_1 \\
\mathrm{P}_2 : \mathrm{I}_2 \rightarrow \mathrm{R}_2(2) \rightarrow \mathrm{R}_3(5) \rightarrow \mathrm{R}_1(2) \rightarrow \mathrm{O}_2 \\
\mathrm{P}_3 : \mathrm{I}_3 \rightarrow \mathrm{R}_3(5) \rightarrow \mathrm{R}_1(2) \rightarrow \mathrm{R}_2(5) \rightarrow \mathrm{O}_3 \\
\mathrm{P}_4 : \mathrm{I}_4 \rightarrow \mathrm{R}_3(2) \rightarrow \mathrm{R}_2(4) \rightarrow \mathrm{R}_1(2) \rightarrow \mathrm{O}_4
\end{aligned}$$

The corresponding RSP-net is given in Fig. 3 with 24 transitions and 31 places.

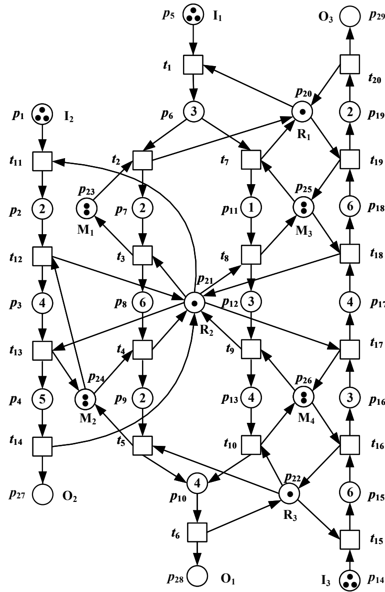

Fig. 2. The RSP-net of the second RCM system from [13].

Based on these three RSP-nets for RCM systems, we tested our proposed PNQL, the PN-based A* search (PNA*), as well as the PN-based online dispatching rules, i.e., FIFO and SRPT. Fig. 4 illustrates that the average cumulative reward for PNQL-ϵ_1 converges quickly in the initial stages but slows down significantly towards the end. In contrast, the convergence trend for PNQL-ϵ_2 is the reverse. Given that PNQL-ϵ_3 maintains a high exploration rate throughout most of the training, its trajectory remains low for an extended period before abruptly matching the performance levels of the first two methods in the final stages. Fig. 5 depicts the trajectories of the average total number of deadlocks encountered in each episode group for PNQL-ϵ_1, PNQL-ϵ_2, and PNQL-ϵ_3. Initially, all methods encounter frequent deadlocks due to limited experience, which can paralyze the

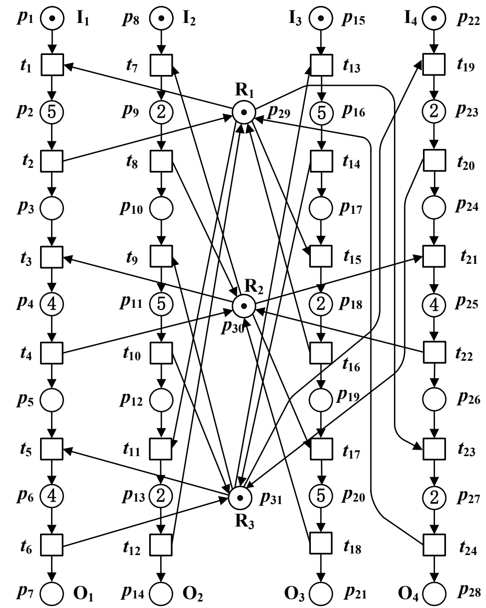

Fig. 3. The RSP-net of the third RCM system from [14].

entire system or parts of it. As training progresses, each method gains experience and increasingly avoids deadlocks. Eventually, all trajectories converge to zero, indicating that they have learned to generate a deadlock-free schedule for the modeled system. The trajectories also reveal varying convergence speeds across different stages.

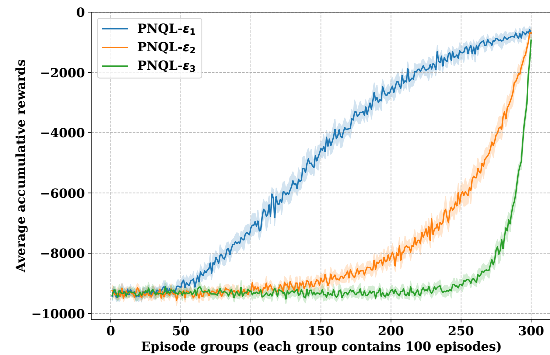

Fig. 4. Average accumulative reward trajectories of the PNQL with different ϵ settings.

Table I presents the simulation results of different methods. In the table, different lot sizes for each RSP-net have been tested with the scheduling methods to pursue minimized makespans. Note that PNA* is a time-consuming offline method, but its obtained makespans are optimal since it adopts an admissible heuristic function [13]. Thus, the scheduling results obtained by the online methods, including ours, are compared with such optimal schedules. The table shows the schedules' makespans obtained by each method and their deviations from the optimal ones, which are the average values of successful runs. A value in parenthesizes indicates the percentage of successful runs,

TABLE I

SCHEDULING RESULTS OF DIFFERENT METHODS FOR THE THREE PNS WITH DIFFERENT LOT SIZES.

RSP-net	Lot Size	PNA*	FIFO	dev(%)	SRPT	dev(%)	PNQL-ϵ_1	dev(%)	T_t(s)	T_e(s)	PNQL-ϵ_2	dev(%)	T_t(s)	T_e(s)	PNQL-ϵ_3	dev(%)	T_t(s)	T_e(s)
Fig. 1	(1, 1)	21.00	35.00	66.67	22.00	4.76	21.00	0.00	1.29	0.02	21.00	0.00	1.16	0.03	21.00	0.00	1.09	0.02
	(2, 2)	35.00	57.00	62.86	37.00	5.71	35.00	0.00	12.98	0.05	35.00	0.00	9.64	0.05	35.00	0.00	10.98	0.06
	(3, 3)	51.00	80.00	56.86	61.00	19.61	54.00	5.88	100.16	0.09	54.00	5.88	102.62	0.11	54.00	5.88	102.77	0.12
	(4, 4)	67.00	103.00	53.73	84.00	25.37	70.00	4.48	999.27	0.25	70.00	4.48	1095.92	0.24	70.00	4.48	793.62	0.25
	(5, 5)	83.00	126.00	51.81	101.00	21.69	88.00 (50%)	6.02	10875.21	0.32	87.70 (60%)	5.66	6694.51	0.32	86.90 (80%)	4.70	3616.73	0.31
Fig. 2	(1, 1, 1)	21.00	49.00	133.33	24.00	14.29	24.00	14.29	7.01	0.05	24.00	14.29	5.43	0.04	24.00	14.29	4.68	0.05
	(2, 1, 1)	23.00	54.00	134.78	24.00	4.35	24.00	4.35	40.03	0.06	24.00	4.35	35.98	0.05	24.00	4.35	34.20	0.05
	(2, 2, 1)	25.00	63.00	152.00	32.00	28.00	25.00	0.00	1683.47	0.19	25.00	0.00	1007.66	0.21	25.00	0.00	874.20	0.24
	(2, 2, 2)	30.00	70.00	133.33	38.00	26.66	35.60	18.67	15659.36	0.72	34.80	16.00	16834.88	0.75	34.40	14.67	16683.46	0.78
Fig. 3	(1, 1, 1, 1)	16.00	42.00	162.50	22.00	37.50	18.20	13.75	49.08	0.07	17.20	7.50	46.29	0.06	17.00	6.25	49.72	0.07
	(2, 1, 1, 1)	20.00	55.00	175.00	26.00	30.00	23.60	18.00	463.91	0.15	21.00	5.00	482.73	0.13	21.00	5.00	619.33	0.19
	(2, 2, 1, 1)	25.00	64.00	156.00	31.00	24.00	30.30	21.20	5568.48	0.35	28.60	14.40	5096.37	0.32	26.00	4.00	5887.37	0.45
	(2, 2, 2, 1)	30.00	76.00	153.33	41.00	36.66	42.30	41.00	21194.83	1.46	40.70	35.67	24561.68	1.35	36.10	20.33	26324.52	1.25

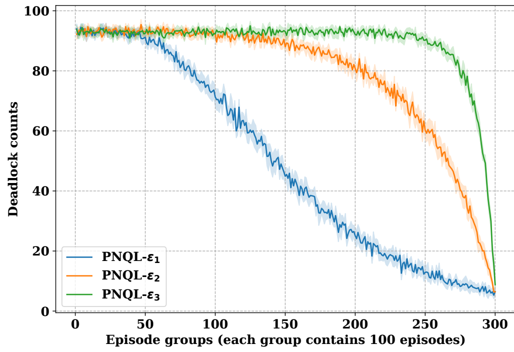

Fig. 5. Average counts of deadlocks encountered during the training of PNQL with different ϵ settings.

and no parenthesizes means that all runs are successful. In addition, the average training time T_t and average execution time T_e for PNQL-ϵ_1, PNQL-ϵ_2, and PNQL-ϵ_3 are given.

From the table, we can see that for almost all cases of the system models, our proposed PNQL outperforms FIFO and SRPT in terms of obtained schedule makespans. In addition, our method can be executed very quickly after they have been trained. For example, for the net in Fig. 1 with lot size $(2, 2)$, the makespans obtained by our PNQL-ϵ_1, PNQL-ϵ_2 and PNQL-ϵ_3 are 35, which is the same as the optimal one obtained by PNA*. In comparison, the ones obtained by FIFO and SRPT are 57 and 37, respectively. The execution of PNQL with any ϵ to find such a whole schedule is rapid, which makes it applicable to online scheduling scenarios. As the lot size increases, our methods still beat all the others except for one case regarding the deviations from the optimal results, while the execution time remains low. Furthermore, our PNQL methods with different ϵ demonstrate different features. In general, PNQL-ϵ_1 may need less training time, PNQL-ϵ_3 may generate better decisions, and PNQL-ϵ_2 is more or less in-between. Therefore, the readers can select an appropriate one according to their application objectives.

V. CONCLUSION

In this paper, we propose a new scheduling approach, PNQL, that combines Q-learning techniques with PNs to model and schedule RCM Systems. By incorporating Q-learning into the PN-based scheduling framework, PNQL achieves efficient and effective solutions. Experimental results demonstrate that PNQL outperforms traditional online dispatching rules such as FIFO and SRPT regarding accuracy and efficiency. This highlights the potential of our algorithm to handle complex scheduling problems and enhance the performance of RCM systems.

Looking ahead, we plan to integrate deep reinforcement learning into our framework to further improve scheduling performance and provide more robust solutions for large and complex RCM systems.

REFERENCES

[1] C. Ferreira, G. Figueira, and P. Amorim, "Scheduling human-robot teams in collaborative working cells," *International Journal of Production Economics*, vol. 235, p. 108094, 2021.

[2] J. Wang, "Patient flow modeling and optimal staffing for emergency departments: A Petri net approach," *IEEE Trans. Comput. Social Syst.*, vol. 10, no. 4, pp. 2022–2032, Aug. 2023.

[3] S. E. Li, "Deep reinforcement learning," in *Reinforcement Learning for Sequential Decision and Optimal Control*. Springer, 2023, pp. 365–402.

[4] M. Drakaki and P. Tzionas, "Manufacturing scheduling using colored petri nets and reinforcement learning," *Applied Sciences*, vol. 7, no. 2, p. 136, 2017.

[5] L. Hu, Z. Liu, W. Hu, Y. Wang, J. Tan, and F. Wu, "Petri-net-based dynamic scheduling of flexible manufacturing system via deep reinforcement learning with graph convolutional network," *Journal of Manufacturing Systems*, vol. 55, pp. 1–14, 2020.

[6] J.-H. Lee and H.-J. Kim, "Reinforcement learning for robotic flow shop scheduling with processing time variations," *International Journal of Production Research*, vol. 60, no. 7, pp. 2346–2368, 2022.

[7] B. Huang, M. Zhou, X. S. Lu, and A. Abusorrah, "Scheduling of resource allocation systems with timed Petri nets: A survey," *ACM Computing Surveys*, vol. 55, no. 11, pp. 1–27, 2023.

[8] J. Clifton and E. Laber, "Q-learning: Theory and applications," *Annual Review of Statistics and Its Application*, vol. 7, pp. 279–301, 2020.

[9] J. Lv and B. Huang, "A Petri-net-based anytime A* search for scheduling resource allocation systems," *IEEE Trans. Ind. Informat.*, vol. 20, no. 2, pp. 2865–2872, 2024.

[10] M. Van Otterlo and M. Wiering, "Reinforcement learning and markov decision processes," in *Reinforcement learning: State-of-the-art*. Springer, 2012, pp. 3–42.

[11] H.-J. Kim and J.-H. Lee, "Scheduling of dual-gripper robotic cells with reinforcement learning," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 2, pp. 1120–1136, 2022.

[12] Y. Lee, Z. Jiang, and H. Liu, "Multiple-objective scheduling and real-time dispatching for the semiconductor manufacturing system," *Computers & Operations Research*, vol. 36, no. 3, pp. 866–884, 2009.

[13] B. Huang, M. Zhou, A. Abusorrah, and K. Sedraoui, "Scheduling robotic cellular manufacturing systems with timed Petri net, A* search, and admissible heuristic function," *IEEE Trans. Autom. Sci. Eng.*, vol. 19, no. 1, pp. 243–250, Jan. 2022.

[14] B. Huang and M. Zhou, "Symbolic scheduling of robotic cellular manufacturing systems with timed Petri nets," *IEEE Trans. Control Syst. Technol.*, vol. 30, no. 5, pp. 1876–1887, Sep. 2022.