Computers and Chemical Engineering 204 (2026) 109415

Contents lists available at ScienceDirect

Computers and Chemical Engineering

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

A hybrid deep Q-learning approach to online planning and rescheduling of single-stage multi-product continuous processes☆

Syu-Ning Johnn, Vassilis M. Charitopoulos *

Department of Chemical Engineering, Sargent Centre for Process Systems Engineering, University College London (UCL), London, WC1E 7JE, UK

ARTICLE INFO

Keywords:
Autonomous online scheduling
Reinforcement learning
Deep neural networks
Continuous processes
Process scheduling
Q-learning

ABSTRACT

Optimisation-based process scheduling methods lie at the core of the process supply chains, facilitating the efficient allocation of limited resources and ensuring profitable operations. The efficiency and adaptability of these methods are of paramount importance, especially when dealing with frequent modifications to existing scheduling plans, caused by uncertainties and unforeseen real-world disturbances. Compared to heuristic methods that heavily rely on instance-specific manual customisation and fine-tuning, reinforcement learning (RL) has the advantages of learning from existing experiments and generalising to unknown scenarios, thus automating the process with higher flexibility and adaptability. In this work, we propose an RL-based method that transforms a single-stage multi-product process scheduling problem, originally framed as a mixed-integer linear programming (MILP) problem, into a Markov decision process and trains the RL agent to identify the optimal production sequence. The trained agent is subsequently integrated into a simplified planning and scheduling linear programming (LP) framework to enable efficient decision-making for the re-optimised production sequence and time length. Results show that our proposed learning-based integrated decision-making framework demonstrates strong computational efficiency and adaptability, outperforming both the benchmark random agent and heuristic approaches with minimal deviation from the optimal solution achieved by the state-of-the-art MILP solvers.

1. Introduction

Growing market competition and demand volatility constitute two of the most prominent factors that endanger the profitability and resilience of process industries (Badejo and Ierapetritou, 2022). The aim to efficiently organise production and meet the long-term manufacturing demands of multiple products necessitates the accommodation of frequent modifications due to unforeseen fluctuations such as product demand or electricity prices (Gupta et al., 2016; Castro et al., 2018). Production management encompasses two key components: planning, which involves resource planning over long time horizons, and scheduling, which entails the detailed allocation of production tasks to specific resources (Perez et al., 2021). Optimising the coordination of planning and scheduling is imperative for enhancing the overall process supply chain efficiency. We therefore necessitate effective response mechanisms capable of addressing recurring optimisation needs and preventing financial losses (Kopanos and Puigjaner, 2019). In this work, we focus on the integrated planning and online rescheduling of multi-product continuous manufacturing systems through a novel hybrid deep Q-learning/linear programming (DQN/LP) framework with the goal of facilitating real-time implementation of rescheduling for processing systems.

1.1. Online reactive process scheduling

While the conventionally sequential approach to planning and scheduling may neglect their interdependence and lead to suboptimal solutions, simultaneous approaches for addressing the integrated planning and scheduling problem via mixed integer linear programming (MILP) models can be time-consuming. Their computational complexity stems from the combinatorial nature of the underlying formulations as well as the inherent multi-scale coordination that is sought.

In recent years, there has been considerable research interest in the effective integration of planning and scheduling within process industries. Many studies in the literature aim to reduce the computational complexity and derive more computationally efficient frameworks. Previous studies by Erdirik-Dogan and Grossmann (2006, 2008) and Sung and Maravelias (2007) introduced several decomposition-based approaches to mitigate the exponential increase in computing effort as

☆ This article is part of a Special issue entitled: 'ESCAPE - PSE 2024' published in Computers and Chemical Engineering.

* Corresponding author.
E-mail address: v.charitopoulos@ucl.ac.uk (V.M. Charitopoulos).

https://doi.org/10.1016/j.compchemeng.2025.109415
Received 28 January 2025; Received in revised form 29 July 2025; Accepted 22 September 2025
Available online 27 September 2025

instance size grows. Liu et al. (2008) and Charitopoulos et al. (2017, 2019) proposed a hybrid time representation approach that circumvents the direct handling of continuous-time formulation within each period when the planning horizon expands.

To navigate dynamic industrial environments effectively, timely and reactive responses are crucial to managing various uncertainties, such as equipment malfunctions and rush order arrivals. Nevertheless, existing exact methods encounter challenges in accommodating re-optimisation to address the evolving information promptly, thereby hindering their industrial applicability. For one, as suggested by Baker (1977), we inevitably need to address an infinite real-time process using decisions derived from a finite horizon planning model. The passive re-optimisation of models triggered by individual events is inherently difficult and inefficient in managing an ongoing system operating indefinitely, where the conditions and parameters influencing optimality can shift significantly over time (Harjunkoski et al., 2014). Furthermore, computational complexity associated with exact methods limits the feasibility of re-optimising and generating entire new solutions from scratch whenever new uncertainties arise. This complexity not only demands substantial computational resources but also introduces potential delays that can hinder the timely implementation of optimal decisions (Li and Ierapetritou, 2008).

Consequently, the shortcomings of traditional decision-making approaches underscore the need for an automated online system that can bolster decision-making efficiency to handle large datasets amidst rapid modifications based on uncertainties unfolding in a sequential and timely manner.

Many pertinent studies on online scheduling have been conducted that incorporate re-optimisation to address real-time information in critical characteristics such as demand and time uncertainties in an efficient manner. Framinan et al. (2019) highlighted the significance of frequently updating and maintaining the decision-making process based on event-driven rescheduling policies. Gupta and Maravelias (2019), Kopanos and Pistikopoulos (2014) and McAllister et al. (2019) demonstrated that carefully designed models and efficient algorithms with lower processing times can significantly enhance the implementation of the online decision-making system. However, their proposed models remain MILP, posing limitations in terms of practicality, particularly for problems that require rapid decision determination and updates when compared to LP.

1.2. Machine learning-based process scheduling

In recent years, the integration of machine learning (ML) into process scheduling has received increasing attention (Hubbs et al., 2020a), with recent studies showing very promising advancements and developments of the ML applications in the chemical industry (Chiang et al., 2022; Fuentes-Cortés et al., 2022). Reinforcement learning (RL), a branch of ML that learns optimal strategies to map system states to the optimal actions through trial-and-error by interacting with an uncertain and dynamic environment, demonstrates the potential for reactive online scheduling with lower computational costs (Hubbs et al., 2020b). Unlike traditional rule-based learning systems, RL is a popular training approach that exhibits the potential to discover decision-making strategies generalised from its learned experiences. Recent years have witnessed a notable surge in the application of RL techniques within the process industry to optimise complex industrial processes characterised by dynamic and stochastic behaviours. We refer the interested readers to Lee et al. (2018) and Nian et al. (2020) for reviews of RL applications within the field of industrial process systems.

Deep reinforcement learning (DRL) leverages the strengths of artificial neural networks to handle high-dimensional complex problems. The DRL models can be broadly categorised into policy-based methods and value-based methods. Many works with policy-based methods for process systems that directly estimate the probability of selecting a particular action from a given state, include actor–critic (Liu et al., 2020) advantage actor–critic (Hubbs et al., 2020a), deep deterministic policy gradient (Ma et al., 2019), and policy gradients (Petsagkourakis et al., 2020). In contrast to policy-based RL algorithms that optimise the agent's policy to guide its action, value-based RL algorithms focus on approximating the value of each action or particular state to maximise the expected reward function.

Deep Q-learning (DQN) is a popular value-based off-policy algorithm within the RL framework that utilises experience replay and target networks to ensure stable learning by estimating the quality of a selected action at each time step. It demonstrates strong end-to-end learning capability to learn from experiences generated by different policies and effectively handle high-dimensional state spaces with discrete action spaces.

The reasons for us to consider DQN are two-fold: firstly, our problem involves a finite set of possible actions corresponding to the selection of which products to schedule next in a production sequence, which forms a discrete action space that fits the architecture and algorithm designs of DQN. Secondly, DQN as an off-policy algorithm, owns greater sample efficiency compared to policy gradient methods due to its ability to utilise off-policy samples (Gu et al., 2016), thereby eliminating the necessity for on-policy sample collection.

Existing studies applying Q-learning to the scheduling problem include the work of Zheng and Chen (2024) on developing a DQN framework with an actor–critic architecture to optimise model policies for job sequencing and adjusting batch speeds in a single-machine batch scheduling problem. Pan et al. (2021) proposed an oracle-assisted constrained Q-learning algorithm to optimise a controller policy and ensure that a given set of chance constraints is satisfied with a high probability in a nonlinear stochastic optimal control system. More generally, Karimi-Mamaghan et al. (2022) integrated Q-learning into the iterated greedy metaheuristic as an efficient operator selection mechanism for the permutation flow-shop scheduling problem. Their results illustrate the superior performance of the proposed Q-learning framework that can effectively surpass the performance of heuristic methods in enhancing solution quality and commercial solvers in reducing computational time.

1.3. Contributions of this work

This work introduces a novel hybrid learning-based decomposition framework that integrates RL with a planning and scheduling reduced MILP model to actively and recurrently optimise the online rescheduling of production sequences for manufacturing processes. The proposed framework focuses on a single-machine, single-line continuous process setting. In scenarios where multiple identical parallel lines are assumed to operate independently with no inter-line interactions (e.g., material transfers between lines), the same learning-based algorithm can be deployed concurrently across the parallel lines. This configuration effectively forms a set of single-stage multi-product continuous machines that operate under no material transfers, each optimising its operations in isolation as noted by Liu et al. (2010). We employ RL and decompose the entire problem into two interdependent decision-making steps, ensuring a feasible computational time frame and allowing for timely updates to the solution as new data or uncertainties arise.

The proposed hybrid decomposition framework can be divided into two stages: model training and model implementation. During the model training stage, we opt for DQN to identify optimal production sequences for each period in a single-unit multi-product planning and scheduling problem. We frame the production sequencing as a Markov decision process (MDP), a mathematical model of decision-making characterised by discrete time steps involving states, actions and rewards that form the foundation of RL. The DQN agent, after constructing a production sequence, is rewarded proportionally for the enhancement of the solution quality. A more detailed description of the problem formulation can be found in Section 2.2.2.

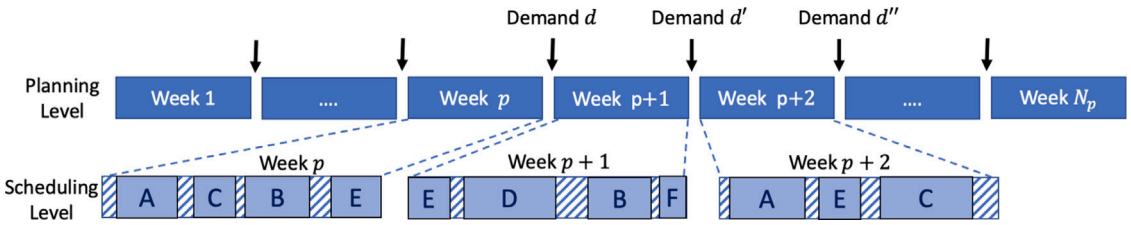

Fig. 1. A conceptual representation of the planning and scheduling problem.

During the subsequent model implementation stage for real-time rescheduling, the trained agent based on its refined strategy generates production sequences in an offline manner to support the online rescheduling of production sequences. For each round of rescheduling, the identified sequence is utilised as input for a linear programming (LP) model to determine the remaining decisions on production amount, backlog and inventory levels, and schedule length. The whole set of decisions is updated continuously as new information becomes available.

We demonstrate the effectiveness of our hybrid decomposition framework in the context of a multi-product continuous manufacturing process over multiple periods (weeks). Our findings reveal that the computational time is approximately halved compared to conventional MILP formulations with limited loss of optimality compared to the MILP global solution. Through a series of case studies, the DQN-based agent also demonstrates better performance on average in producing sequences with a higher profit than the random learning baselines and heuristic approach. By bridging the gap between RL and traditional optimisation techniques, this study provides an efficient and scalable decision-making framework for real-time continuous manufacturing processes.

To summarise, the primary contributions of this work are:

- We propose a novel hybrid decomposition framework that integrates DQN with MILP to efficiently handle decision-making on the re-optimised production sequence and process scheduling length in dynamic environments.
- We demonstrate the efficiency and flexibility of the proposed framework by implementing it to planning and scheduling problems with varying product counts under both single-period and multi-period scenarios in industrial process control.
- We benchmark the proposed framework against established methods, including random learning baselines, a heuristic approach, and state-of-the-art MILP solvers, showcasing competitive performance in both computational efficiency and solution quality provided by our approach.

The remainder of the article is structured as follows: Section 2 introduces the methodology and problem formulation of the MILP and DQN models. Section 3 presents the two case studies and describes relevant results. Lastly, Section 4 concludes the work and outlines future research directions.

2. Methodology

2.1. TSP-based MILP integrated planning and scheduling model

We consider a continuous-time representation with the total planning horizon divided into discrete periods (weeks). Production demands arrive at the beginning of each planning period (p). As shown in Fig. 1, the planning and scheduling problem involves determining the set and amount of products to be manufactured at the planning level, followed by constructing a production sequence, establishing production durations, inventory levels, backlogs, and inter-period changeovers within each subsequent period at the scheduling level.

To investigate the planning and scheduling problem, we present the complete MILP formulation outlined in the Appendix, which is derived from the framework introduced by Liu et al. (2008).

2.2. DQN framework for online scheduling

2.2.1. Markov decision process

RL is a learning method that maps states to actions to maximise the expected future rewards (Sutton and Barto, 2018). MDPs are a fundamental mathematical framework that model a sequential decision-making process, based on which we can apply the RL techniques. An MDP can be represented by a tuple (S, A, P, R, γ), containing the state space, action space, transition function, reward function and discount factor. A state $s \in S$ is a representation of the specific time step within an environment that the agent is in. At each non-terminal state s, the agent selects an action $a \in A(s)$ from the set of actions available at this state and receives a reward r according to a given reward function $R(s, a)$, which is the immediate benefit for selecting the state–action pair. Then, the agent reaches a new state based on the transition probability that describes the likelihood of moving from one state to another given an action. The key defining feature of a MDP is the Markov property, i.e. that the future state of the system is solely dependent on the preceding state–action pair, rendering the history of events irrelevant.

The agent engages with the environment through episodes, each of which encapsulates the agent's interactions over time as a sequential chain of states, actions, and rewards. Each episode of the agent–environment interaction can be defined as a trajectory in the form of $s_t \rightarrow a_t \rightarrow r_t \rightarrow s_{t+1}$ that terminates with a terminal state s^+. A conceptual representation of MDP is depicted in Fig. 2. By receiving feedback in the form of rewards from the environment, the agent is trained to refine its strategy with actions in dynamic states and formulate an optimal policy $\pi(a|s)$ that dictates the agent's behaviour to maximise the cumulative future rewards. Central to this learning is the state–action value function $Q(s, a)$, also denoted as Q-values, which quantifies the expected rewards associated with taking any particular state–action pair by following π.

2.2.2. Online scheduling formulation via MDP

We make several key assumptions to streamline the analysis of the online planning and scheduling problem. Firstly, we focus on a continuous manufacturing process with a single production unit that handles multiple products over a single or multiple periods, where each period is equivalent to one week. The single period setting can be treated as a special case of the multi-period setting. Demands for different sets of products become available at the beginning of each period. Each product can be produced at most once in each period, with a known changeover time to any other product and associated transition costs for switching between products.

The proposed hybrid decomposition framework utilises DQN to develop a learning-based decision system for online production sequence scheduling. The problem configuration that the hybrid framework is designed to handle is presented in Fig. 1. Specifically, our goal is to employ DQN to identify sequences of products that maximise the objective function (3a). The constructed sequence is subsequently incorporated into a simplified LP model, derived from the full MILP formulation (3), to determine the remaining decisions on production quantity and duration. The complete decision framework iteratively generates the production sequence determined by DQN and uses it

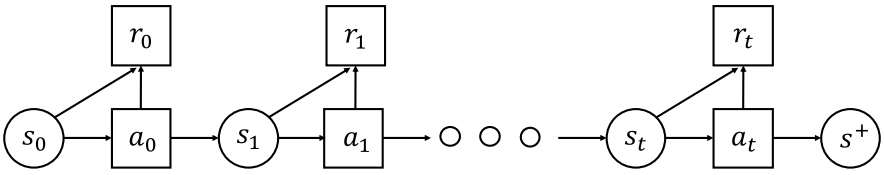

Fig. 2. A conceptual representation of MDP.

to calculate the production duration via the simplified LP model. To develop the learning-based framework, it is essential to first transform the production scheduling into a MDP for each period. The process for achieving this is outlined below.

The production scheduling task is episodic, each of which involves the agent completing a sequence of product selections. In this problem, each episode comprises a fixed number of periods indexed as $P = \{1, \dots, p\}$, with each period consisting of multiple time steps represented as $T = \{1, \dots, t\}$, where the number of time steps can differ between periods within the same episode. Each period is formulated as a MDP starting with an initial state s_0, which corresponds to the initial solution where no production sequence has yet been formed, along with a predefined sequence length limit, which is capped at the total number of products with nonzero demand that can be chosen and added to the sequence.

The agent is provided with two types of discrete actions that alternate sequentially within each period: (1) picking an available product with a nonzero demand to add to the end of the sequence during the *product selection* phase, and (2) making a binary choice of either continuing to include new product at the end of production sequence or terminating the product selection process during the *process termination* phase. Visualised in Fig. 3, the agent alternates between these two types of actions until either all the available products have been included in the sequence, or when the agent decides to quit and hence ends the sequence with only a subset of demanded products, thus defining the terminal state.

After an action is selected, the agent receives a reward based on the improvement in the objective function. Upon completion, the environment provides a reward and transitions the agent to a new state, updating its stored information accordingly to reflect the sequential progression. For the single-period scenario, when a terminal state is reached, the environment resets and the same process repeats with another randomly generated initial state s_0. For the multi-period setting in which each episode includes a fixed number of periods $p_1, p_2, \dots, p_n$, the terminal state of any non-final period $p_i \neq p_n$ transitions the agent to the initial time step of the subsequent period p_{i+1} within the same episode. This transition entails updating the inventory, backlog, and product demand for the sequential period while keeping the episode-based information including inter-product transition time and product sale price unchanged. At the end of the last period p_n within the episode, the agent transitions to begin a new episode, prompting a reset of the environment.

The production scheduling problem for each period can be represented as a directed complete graph $G = (V, E)$ where each node represents a product and each directed arc as the transition between a pair of products (Charitopoulos et al., 2017). A production sequence is represented by a path comprising nodes and arcs, where the stopping time at each node denotes the production time, and each arc represents the inter-product changeover time. Each MDP state can be represented by a feature matrix comprising a list of features either dependent or independent of the current production sequence. The independent features contain relevant information about each node, such as the product index, production demand, unit price, inventory level, and backlog level, regardless of whether this node has been included in the production sequence or not. Moreover, a list of additional static features captures state-independent information such as the transition time and cost between any pair of nodes. These features remain static throughout the period and are unaffected by the extension of the production sequence during execution. Conversely, the dynamic features of a state change as different products are selected, indexed by the time step t of each period within an episode. A detailed description of the feature matrix for the planning and scheduling problem is given in Table 1.

We denote the existing partial solution at state s_t as F_t, which contains the production sequence up to the most recently selected product. Additionally, a boolean indicator φ_t specifies the action phase: $\varphi_t = 0$ when the agent is at the product selection phase expanding the production sequence, and $\varphi_t = 1$ when it is at the process termination phase determining whether to continue or terminate the selection process. Furthermore, b_t indicates the remaining action capacity that is available to the agent at each step t. The total number of steps can assume any integer value between 1 and twice the product sizes and may vary across different periods within the same episode.

We now formulate each element of the MDP within the context of the production scheduling problem with the details below:

- *States* $\mathcal{S}$: each state s_t is a tuple $(G, \mathbf{X^n}, \mathbf{X^e}, F_t, b_t, \varphi_t, \eta_t)$, wherein the graph G and feature set $\mathbf{X^n}$ and $\mathbf{X^e}$ contain all the node and edge features, respectively. F_t represents the solution formed as a sequence of products at time t. The available action budget b_t is a non-negative integer not exceeding the total number of nodes. Boolean variable φ_t indicates the action phase at step t, whether it is permissible to select an additional product, or to choose to terminate the process or not. Boolean variable η_t indicates whether an episode reaches a terminal state.
- *Actions* A encompasses both the selection of an available product and the determination of whether to continue or terminate the selection process, together determined by the binary phase indicator φ_t of the state. When $\varphi_t = 0$, the available set of actions is the available products and are defined by the set $A(s_t) = \{i : i \notin F_t, D_i + B_i > 0\}$, where D_i and B_i corresponding to the demand and backlog level for the given period this time step t belongs to. When $\varphi_t = 1$, the action is to decide whether or not to terminate the process, with two available actions $A(s_t) = \{C, T\}$.
- *Transitions* P: when $\varphi_t = 0$, an available product not yet attached to the sequence is selected, decreasing the action budget b_t by 1. The selected product i_t is appended to the current production sequence $F_{t-1} = \{i_1, i_2, \dots, i_{t-1}\}$, updating it to $F_t = \{i_1, i_2, \dots, i_t\}$. The product i_t is then masked as unavailable within the feature set X_2 during the remaining steps until the environment resets to the sequential period or initiates a new episode if the current period is the last. Subsequently, φ_t assumes the value of 1 to start the process termination decision phase, during which the transition based on the chosen action either terminates the process, marking the state as terminated by setting η_t = True, or continues the process and alternates to the sequential product selection phase, setting $\varphi_t = 0$. φ_t alternates cyclically between the two values 0 and 1, with each value corresponding to the phase that follows immediately to the completion of the preceding phase.
- *Rewards* R are provided, either when the last available product has been inserted so that the action budget is exhausted with $b_t = 0$ at $\varphi_t = 0$, or when the "terminate" option is picked at $\varphi_t = 1$ so that a state becomes the terminal state and a reward is given. The reward takes the value of the improvement in solution quality that can be assessed via an objective function f. Concretely, $R_t = f(s_t)$ is computed using (3a) and equals 0 otherwise for all the non-terminal states.

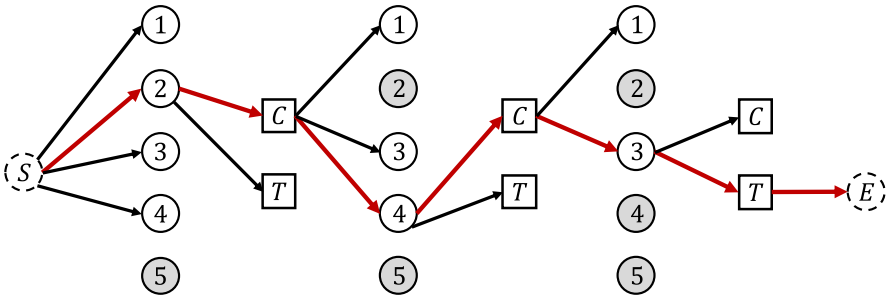

Fig. 3. Sequential decision process alternating between *product selection* and *process termination* phases, where the former phase provides an action set including products 1 to 5 and the latter includes only two actions "continue" and "terminate". Products without demand and previously selected products are excluded from the action set. Auxiliary start node **S** and end node **E** are included for clarity. The final production sequence is formed as $2 \rightarrow 4 \rightarrow 3$ with the MDP process terminated at the third iteration of the *process termination* phase, during which the agent terminates further selection process, leaving product 1 unselected for this particular period. A reward is given at the end point **E**.

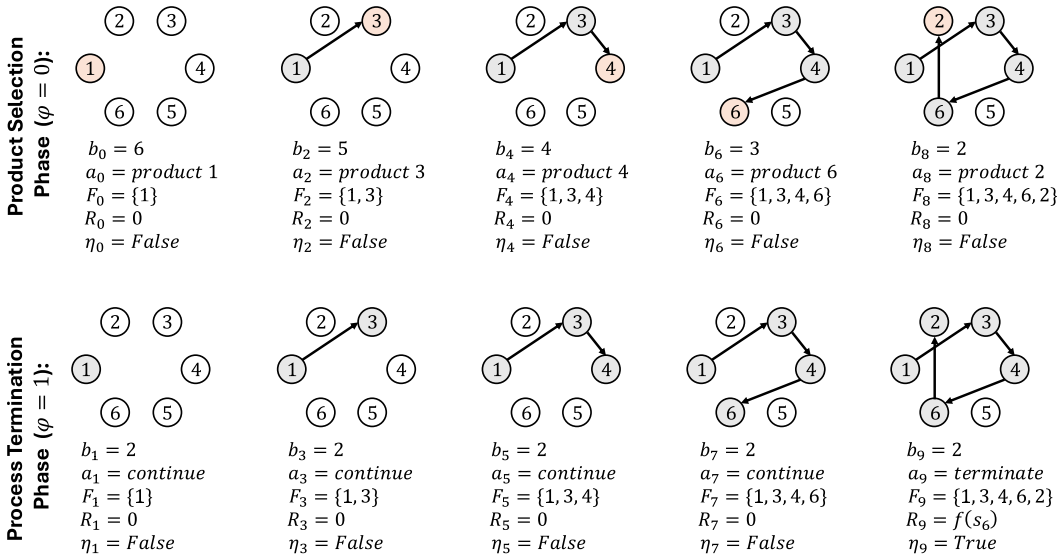

Fig. 4. Proposed hybrid decomposition framework to tackle the planning and scheduling problem. The upper row of graphs refers to the product selection phase, and the lower row to the process termination phase.

2.2.3. Motivational example

Fig. 4 showcases the visualisation of a MDP episode with a single period, which begins with a starting state s_0 and ends at the terminal state s^+, which is reached either when the agent selects the termination option at $\varphi_t = 1$ or when the selection budget is exhausted at $\varphi_t = 0$. We assume an initial state s_0 based on a randomly generated initial set of parameters, and all products 1 to 6 are associated with positive demands. For simplicity, each state does not explicitly display the phase indicator φ_t.

From the top left-hand graph in the figure, the agent starts at state s_0 with an initial budget $b_0 = 6$. The value of φ_0 is initialised to 0 to reflect the product selection phase, during which the agent selects from the action set (marked in white) its first action a_0 = "product 1" (marked in red) according to its trained strategy to add to the production sequence and transitioned to the sequential state s_1 in the bottom left graph. All infeasible actions in this state are marked in grey. In the subsequent state s_1, the indicator φ_1 assumes the value 1, and the agent selects a_1 = "Continue", thereby maintaining the sequencing process with η_1 = False and transitioning to the next state with phase indicator updated to $\varphi_2 = 0$. Moving to state s_2, the agent selects a_2 = "product 3" from the available actions, updating the production sequence to $F_2 = \{1, 3\}$. The product selection process continues as the agent iteratively selects actions a_4 = "product 4", a_6 = "product 6" and a_8 = "product 2", after choosing in their preceding states $a_3 = a_5 = a_7$ = "Continue" during the process termination phase. Eventually, the agent chooses a_9 = "Terminate", thereby terminating the process and forming a sequence $F_8 = \{1, 3, 4, 6, 2\}$ with 5 products scheduled inside the sequence, leaving product 5 unselected for the episode. The newly constructed production sequence F_8 is then integrated into the MILP model, with the discrete decision variables treated as redefined inputs, simplifying the MILP model into an LP problem on determining the remaining continuous decisions. This will be further described in Section 2.3.2.

2.3. Proposed hybrid learning framework

2.3.1. Deep Q-network

Q-learning (Watkins and Dayan, 1992) is a popular model-free RL approach employed to solve MDPs by iteratively updating the Q-values according to the rule:

$$Q(s,a) \leftarrow (1-\alpha_L) \cdot Q(s,a) + \alpha_L \cdot \left(r + \gamma \cdot \max_{a' \in \mathcal{A}(s')} Q(s',a') \right) \quad (1)$$

where $Q(s,a)$ is the previous Q-value for the (s,a) pair, the term $\max Q(s',a')$ denotes the maximum Q-value amongst all the possible actions in the subsequent state s', and r represents the immediate reward the agent receives by performing the particular state–action pair. The discount factor γ balances immediate rewards against future rewards, influencing the agent's preference for short-term gains versus

long-term benefits. Through the iterative refinement in the MDP, the agent refines its policy to estimate the optimal Q-values.

Deep neural networks are frequently utilised as function approximators for the $Q(s,a)$ function, which facilitates the generalisation of Q-values across states that share common characteristics and therefore may consequently yield similar future rewards. Function approximation is particularly advantageous for problems with large state spaces, which contrasts with the traditional Q-table approach: Unlike Q-tables that explicitly record Q-values for every state–action combination, DQN provides a more scalable and efficient means of approximating the Q-values by bypassing the memory challenges associated with maintaining extensive records.

We employ the DQN algorithm (Mnih et al., 2015) for agent training, incorporating key techniques including the replay buffer and target network. The replay buffer is an embedded memory replay mechanism that stores the agent's past interactions with the environment as (s,a,r,s',done) tuples inside a memory tank. During training, the agent samples mini-batches from the memory tank to update the main network parameters, enabling the utilisation of transition tuples drawn from various historical time points stored inside the buffer, thereby enhancing training stability compared to learning solely from the most recent transition tuples in an online manner. The main and target networks are function approximators, with the former responsible for estimating the predicted Q-values and the latter for providing the target Q-values. The target network is a duplicate of the main network that is updated periodically by copying the main network's parameters. Common optimisation methods, such as stochastic gradient descent, are employed to minimise the mean squared error between the predicted and target Q-values.

During training, the DQN algorithm samples iterative batches of previously taken actions from the replay buffer to train the main network. In this step, the algorithm takes the current state of the environment as input and outputs a vector of Q-values corresponding to the list of available actions. The main network parameters are periodically transferred to the target network at a certain number of episodes to stabilise the training process. The target network remains consistent over this certain interval and is used to predict the future Q-values for state–action pairs.

2.3.2. Hybrid decomposition framework

The proposed hybrid decomposition framework, depicted in Fig. 5, begins with training the production sequencing agent within the DQN framework marked in the blue dotted box. The production sequencing process is cast into a MDP for the DQN agent to iteratively learn to select products and form a production sequence until reaching a terminal state for each period. The agent needs to determine the sequence based on randomly generated historical product demands that optimise the overall production profit while minimising the penalty costs.

The proposed framework adopts an open-loop optimisation approach. During training, the DQN agent's policy is re-evaluated periodically for each sample batch, where environmental feedback is applied to adjust future actions. Once the training is finished, the neural network parameters are fixed and utilised for evaluation.

During the DQN training, the agent forms experience replay and stores the agent–environment interaction as tuples of (s,a,r,s',done) in the replay buffer. Next, a sample batch of transition tuples is utilised to estimate the predicted Q-value $Q_\theta(s,a)$ as the output of the main network, given the specific state–action pair and under the current main network parameter set θ. The target Q-value is computed using $y = r + \gamma \cdot \max_{a'} Q_\theta(s',a') \cdot (1 - \text{done})$, which the main network learns to predict. Subsequently, the main network parameters θ are updated using the Adam optimiser, which aims to minimise the discrepancy between the two networks, guided by the loss function computed according to $(y - Q_\theta(s,a))^2$. Finally, the target network is periodically updated by copying from the main network over the MDP dynamics.

After the DQN agent is trained, it is embedded into the hybrid framework and determines the production sequence in an offline manner whenever new information, such as product demands, becomes available and deteriorates the current decision. When a new production sequence is formed, the sequence order is converted into the corresponding set of discrete decisions, specifically the product indices $O_{i,p}$, the first and last products $F_{i,p}$ and $L_{i,p}$ inside the sequences, and the procession relationships between products $Z_{i,j,p}$ and $\text{ZF}_{i,j,p}$. Those sequencing-related discrete decision variables are then treated as known inputs and passed onto the MILP model, which is simplified to be an LP formulation presented below as (2). The simplified LP model resolves the remaining continuous decisions that include the production amount, time, sales, inventory, and backlog decisions, which are addressed with the chosen optimisation solver.

$$\max \sum_{i\in I}\sum_{p\in P} \text{PS}_i \cdot S_{i,p} - \sum_{i\in I}\sum_{p\in P} \text{CB}_i \cdot B_{i,p} - \sum_{i\in I}\sum_{p\in P} \text{CI}_i \cdot V_{i,p} \tag{2a}$$

subject to

(3m), (3n), (3o), (3p), (3q), (3r)

Since all discrete decision variables are predetermined based on the production sequences generated by the DQN agent, the MILP formulation is simplified by treating all binary $(E_{i,p}, F_{i,p}, L_{i,p}, Z_{i,j,p}, ZF_{i,j,p})$ and integer $(O_{i,p})$ decision variables as parameters. This reduction transforms the original MILP model into an LP model that can be operated efficiently and solved significantly faster online compared to the original combinatorial problem.

2.3.3. Bayesian optimisation

We employ Bayesian optimisation (BO) (Frazier, 2018) for hyperparameter tuning for the DQN framework to optimise algorithm performance. BO builds a probability model of the objective function, using it to guide the hyperparameter selection process to evaluate the true objective function of the underlying problem. The advantage of BO lies in its ability to efficiently converge to near-optimal solutions within a relatively small set of samples. The algorithm is outlined in the following Algorithm 1.

Algorithm 1 Bayesian Optimisation for DQN Hyperparameters

1: **Input:** Objective function $f(x)$, random sample domain $D = \{(x_i, f(x_i)\}_{i\in N}$
2: Define surrogate model S and acquisition function A
3: **for** $t \in \{1, ...T\}$ or until convergence: **do**
4: Find $x^* = \max_x A(x, S_{t-1})$
5: Evaluate current objective $f(x^*)$
6: Update sample domain $D \leftarrow D \cup (x^*, f(x^*))$
7: Fit new surrogate model S_t to updated sample domain D
8: **end for**
9: **Output:** return $x^{best} = \max_{x\in D} f(x)$ as best hyperparameter set

We select Gaussian processes as the surrogate model for its ease of optimisation and wide application to functions, and expected improvement as the acquisition function. The two functions are commonly chosen for their advantageous trade-off between exploration and exploitation (Bergstra et al., 2011).

We apply BO to optimise on 5 DQN model hyperparameters that include the learning rate α_L and the discount factor γ governing the trade-off between short-term and long-term rewards from (1), the initial exploration rate ϵ_0 and the decay rate ϵ in the epsilon-greedy policy, and the maximum batch size $batch$ which controls the sample size for the replay buffer. We establish a boundary for each hyperparameter, wherein BO identifies its optimal value during the search. Once the best set of hyperparameters is determined by BO, the set is subsequently treated as inputs for the training and evaluation of the DQN agent within the hybrid decomposition framework.

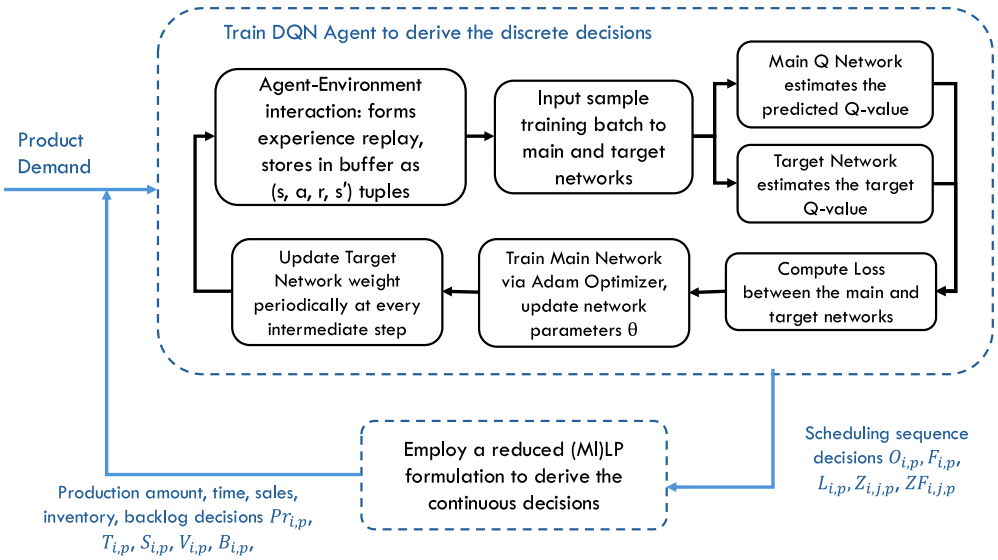

Fig. 5. Proposed hybrid decomposition framework to tackle the planning and scheduling problem as a sequential problem by DQN agent and an LP problem by linear programming solvers.

3. Computational experiments

This section showcases several evaluation case studies conducted using the trained DQN agents under single-period and multi-period scenarios. The solution quality and efficiency of the proposed DQN-LP decomposition framework are analysed by comparison to other baselines and the optimal MILP solutions. Section 3.1 discusses the experimental design and parameter setup. Section 3.2 and Section 3.3 present case studies to analyse the DQN-LP framework performance in single-period and multi-period scenarios, respectively. Section 3.4 examines the profit and a range of penalty costs across various instance sizes, and investigates the number of constraints and variables encountered by the original MILP model and the proposed hybrid framework.

3.1. Experiment configurations

3.1.1. Planning and scheduling problem setup

For the single-period scenarios, we evaluate the performance of the hybrid decomposition framework over a predetermined instance size (fixed number of products) and one week that encompasses 168 h. For the multi-period scenarios, in which each episode encompasses multiple periods (weeks), the DQN agent iteratively predicts the production sequence of each period before proceeding to the subsequent period of the same episode. Each episode begins with a reset of parameter values that remain static throughout the duration of the episode and are updated only when the current episode completes. Consequently, the parameter updates for multi-period are consistent with those in the single-period scenario.

The parameters are generated from the given distributions below. Within each period of a particular episode, the unit of demand generated for each product follows a uniform distribution within the range $[0, 20]$. The sales quantity is constrained by the sum of each product's total demand and any backlog from the previous period. The production level is not restricted by the demand constraints, but once a product is selected and sequenced, it must have a minimum production of at least 1 unit to allow sufficient warm-up and cool-down time for the machine.

At the beginning of each episode, the sale price for all products (PS_i) is generated according to a uniform distribution within the range of $[5, 20]$ per unit. The inventory and backlog costs in each episode are adhered to 10% and 20% of the generated sale price for each unit of product, respectively. The inventory and backlog levels at the initial state that signify the start of the first period in each episode are both initialised to 0. For the multi-period scenario in which an episode contains multiple periods, the inventory and backlog levels are calculated based on information from the preceding period via constraints (3p) and (3q), respectively. The production rate, following a conversion metric of 0.7 units per hour, dictates the conversion scale between the quantity produced and the time required for production. Lastly, the inter-product changeover times are generated from a range of $[30, 100]$ minutes for each pair of different products within each episode. For the single-period setting, only inter-product changeover times are considered and referred to as "inner transitions". In the multi-period setting, both inter-product and inter-period changeovers are accounted for, where the latter is referred to as "outer transitions". The outer transitions in a multi-period setting can be treated as the changeover time from the last product of the previous period to the first product in the subsequent period, which is generated using the same range of changeover time parameters.

3.1.2. DQN agent setup and evaluation

During the model training phase, a random seed is utilised for each episode to determine the parameter generation. The DQN agent undergoes 10,000 episodes, which is equivalent to approximately 5 h of training time. Throughout the training, validation of the agent's performance occurs every 25 steps. At each validation step, the current network is used to predict outcomes for a designated set of 64 validation scenarios, each associated with an individual random seed not utilised during training. The total objective value is calculated to assess whether the current network demonstrates sufficiently strong prediction accuracy to replace the previous network as the best model with its corresponding parameters stored in the system.

We define the domain of learning rate to be $\alpha_L \in [0.00001, 0.001]$ and employ the stochastic gradient method with the Adam optimiser to iteratively update the neural network parameters using sampled batches. We set the maximum batch size of the replay buffer to be between $[128, 1024]$ and a dynamical sample batch size equal to 20% of the maximum size. The agent employs an epsilon-greedy policy, generating a random number between 0 and 1 at each time step. Any number smaller than epsilon results in a random selection of action from the action set. Otherwise, the agent greedily selects the action associated with the highest Q-value as estimated by the network. The exploration rate governs the balance between exploration and exploitation: towards

Table 1
Features used by DQN agent to represent state.

Feature	Type	Rescale	Description
decision step	global	onehot	indicate whether $\varphi_t = 0$ or 1
sequence length	global	maxabs	sequence length at current step
product index	indep	maxabs	all product indices in current episode
product demand	indep	maxabs	all product demands in current episode
inventory level	indep	maxabs	all product inventory levels in current episode
backlog level	indep	maxabs	all product backlog levels in current episode
unit price	indep	maxabs	all product prices in current episode
in-or-not	dep	onehot	if a product is included or not in sequence
position index	dep	maxabs	the position of a product in sequence
pairwise distance	dep	maxabs	changeover time between current and previous products
cumul distance	dep	maxabs	total changeover times between all products in sequence
is-last-action	dep	onehot	whether current product is the last chosen

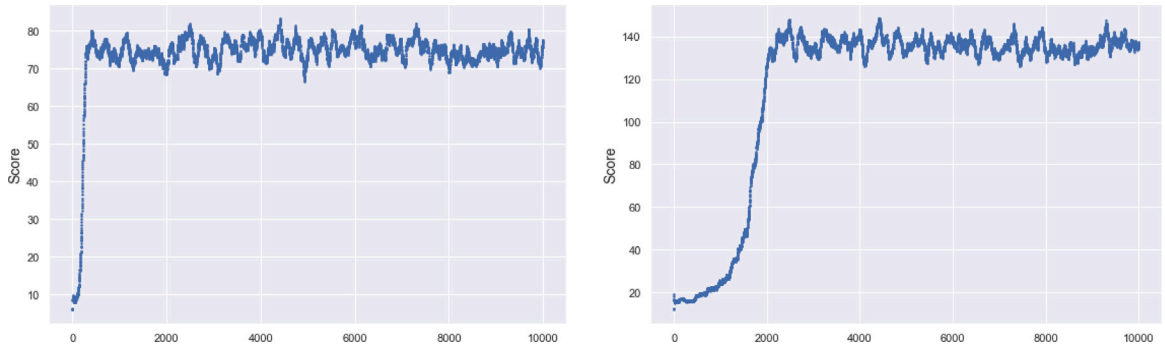

Fig. 6. Training progression graphs for problems with 10 products (left) and 20 products (right).

the beginning of training, the agent is given a much larger epsilon value to allow the exploration of more actions with similar probability, whereas a decrease in the rate leads towards the exploitation of actions with the highest Q-value. We set the initial exploration rate ϵ_0 to be between $[0.1, 1.0]$ with a decay rate between $[0.0001, 0.01]$. Lastly, the discount factor γ is set within the boundary $[0.9, 0.999]$ to be optimised via BO.

During the model implementation phase, the trained DQN agent is embedded into the hybrid framework and its offline performance is assessed based on 50 distinct evaluation rounds. To benchmark the performance of the proposed hybrid DQN-LP framework, it is compared against three baseline approaches: a random baseline agent (RAN) with a uniform random strategy that selects all possible actions at each state with equal probability, a travelling salesman algorithm (TSP) that utilises a local search-based heuristic to iteratively construct sequence while ensuring the total time capacity is satisfied and simultaneously tracking the best sequence achieved so far, and a full MILP model solved using the CPLEX optimizer (MIP). This comparison aims to highlight the effectiveness of the DQN-LP framework in optimising process scheduling with stochasticity in instance parameters.

The neural network model is implemented and executed using Python 3.8.5 and PyTorch 2.3 (Paszke et al., 2019), while the MILP model was coded using Pyomo 6.7 (Hart et al., 2011) and solved using IBM ILOG CPLEX 20.1 optimizer with Python interfaces. All models are developed and run on a workstation with a 2 GHz Quad-Core Intel Core i5 processor and 32 GB memory.

3.1.3. State representation features

Now we introduce the features for representing the DQN states. The feature matrix includes the following information given in Table 1, in which the node-based features can be categorised into global features and local features, where the latter can be further grouped into sequence-dependent (“dep”) and sequence-independent (“indep”) features. The sequence-dependent features are only included when the corresponding product has been selected and placed inside the sequence, otherwise the corresponding position has a value of 0. On the contrary, the sequence-independent features will be included despite the selection status of the product.

We adapt different encoding and normalisation techniques to different features based on their types and scales. One-hot encoding (“onehot”) is applied to convert categorical information to binary 0 and 1 before feeding into the network. Maximum absolute scaling (“maxabs”) is applied to numerical data to scale the data using its global maximum value.

3.2. Case study 1: Single-period planning and scheduling

For the single-machine planning and scheduling problem with single-period scenarios, Fig. 6 showcases the training progressions of the DQN agent. Fig. 7 and Fig. 8 illustrate the performance of the proposed hybrid DQN-LP framework from the evaluation set for 10 and 20 product instances, respectively.

From the top left figure from Fig. 7, we observe comparable performance levels in terms of objective values amongst the TSP, MIP and the hybrid DQN-LP framework. We also observe that these methods consistently outperform the RAN approach. Specifically, the optimality gap between the MIP solution and the solution derived by the proposed hybrid DQN-LP framework is less than 4% on average. This difference is primarily influenced by the inner-transition costs associated with product changeovers, as illustrated in the lower left figure. The reason is that in most of the 10 product scenarios, the accumulated demands fit within the single period without exceeding time capacity, ensuring that regardless of the production sequence, all products are ultimately produced. For this small-scale instance, the TSP heuristic identifies a production sequence with near-optimal inner-transition time by exhaustively enumerating an extensive set of possible sequences in a brute-force manner. This is reflected in the highly similar profit levels shown in the upper middle figure in Fig. 7. Such scenarios favour the use of TSP and its embedded complete enumeration mechanism, allowing for filtering optimal sequences with minimal penalty costs,

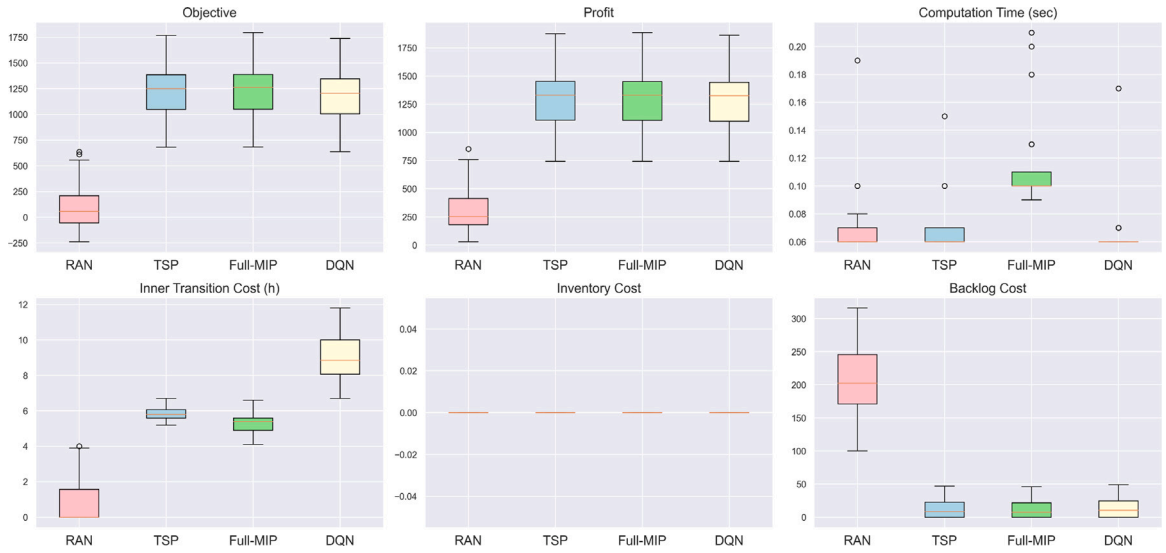

Fig. 7. Evaluation summary for a single-period instance with 10 products.

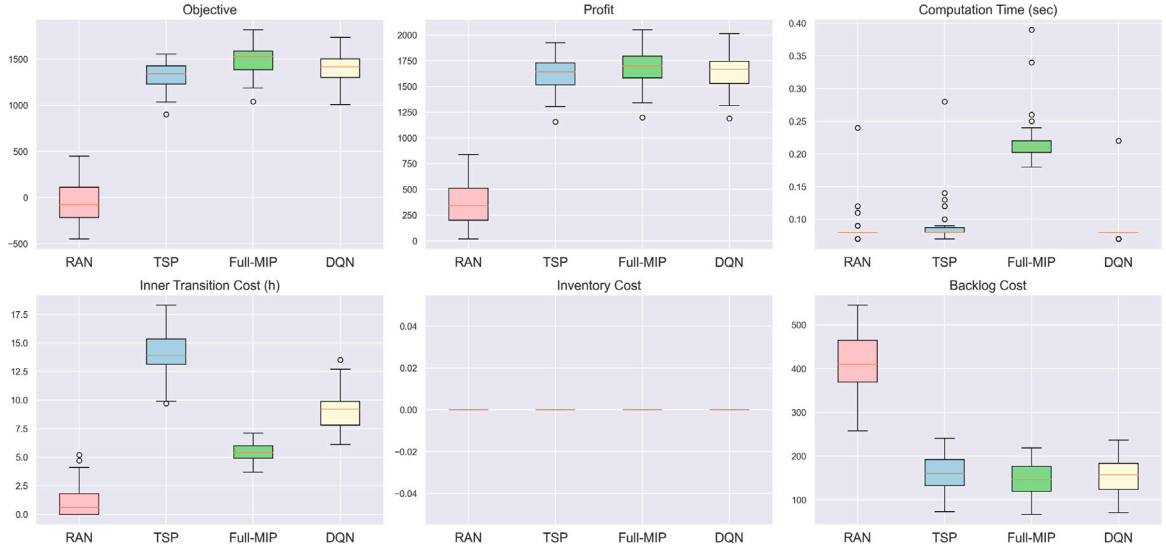

Fig. 8. Evaluation summary for 20 products single period instance.

provided there is sufficient computational power. The top right figure demonstrates the significant computational efficiency of our hybrid DQN-LP framework, as evidenced by a notable reduction of over 66% in average computational time compared to the MIP method.

The summary of the 20 product instances in Fig. 8 showcases a similar trend, where the DQN agent achieves performance levels comparable to the MIP but surpasses both the TSP and RAN. Notably, an optimality gap of 5.8% is observed compared to the exact solution provided by MIP. Additionally, the DQN agent demonstrates more than 50% reduction in average computational time compared to MIP. The transition costs associated with inter-product changeovers, as illustrated in the lower left figure, demonstrate that the DQN-LP framework captures a more efficient production sequence with smaller inner-transition costs and higher computational efficiency compared to the TSP heuristic. While the exhaustive enumeration approach employed by the TSP heuristic ensures competitive performance for small-scale instances, it tends to exhibit limitations as the computing time increases with the instance size.

Since nearly all scenarios fail to accommodate the accumulated demands in a single period without exceeding the total time capacity, we conduct additional analysis focusing on the actual solution corresponding to the median-performance scenario from the 50 evaluation. Fig. 9 illustrates the decision levels associated with total sales, production and backlog, where inventory is not shown, as both plots for all products have no inventory planned. Particularly, we observe a high resemblance amongst the production decisions made by the DQN agent and MIP, with the primary differences arising from an additional 1 unit of production on product 10 and 2 units on product 16 performed by the hybrid framework. Due to time capacity, the DQN agent is required to reduce its production on product 15. Overall, the total objective value difference for this specific scenario is 3.9% between the DQN and MIP.

3.3. Case study 2: Multi-period planning and scheduling

In this second case study, we extend the scope of the hybrid DQN-LP framework to encompass multiple time periods. Within each episode, the agent utilises observed information from the outset of each period to generate predictions for subsequent periods. For the single-machine planning and scheduling problem with multi-period scenarios, we include another random baseline "RAN-b" as a benchmark approach that shuffles the order of all products with demand within a period and

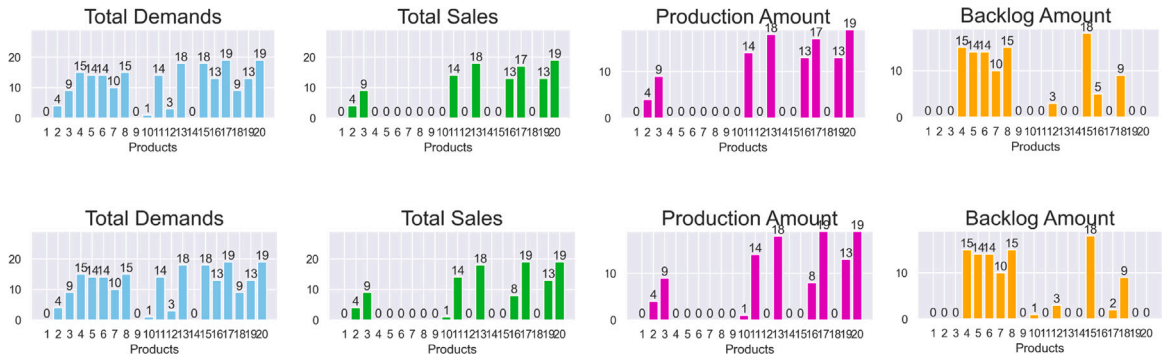

Fig. 9. Evaluation performance for the DQN agent (above) versus MILP optimal solution (below) with median performance scenario for 20 product single period instance. Note that inventory levels are not displayed, as all inventory values are zero.

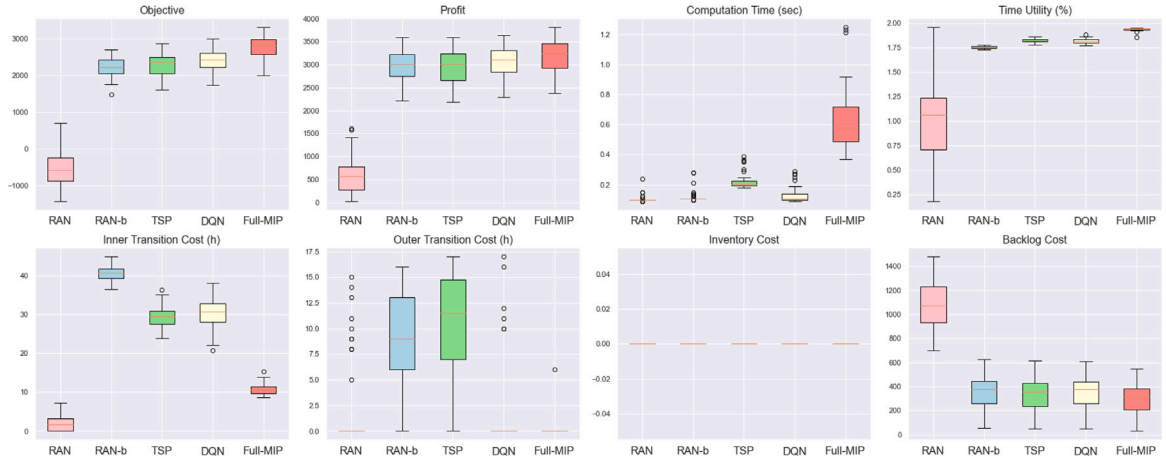

Fig. 10. Evaluation summary for 20 products and 2 periods within each episode.

forms a random sequence. This approach aims to evaluate whether the inclusion of more products in the production sequence, despite incurring higher inter-product transition costs, can help reduce the backlog costs and enhance the solution quality in a multi-period scenario.

Fig. 10 presents the evaluation summary for instances with 20 products over a two-week period for each episode. The top left graph of objective value shows that both TSP and the hybrid framework achieves competitive performance compared to MIP, with optimality gaps of 13.2% and 15.3%, respectively. Both methods consistently outperform the RAN and RAN-b benchmarks in delivering higher average objective values. Additionally, our framework exhibits superior computational efficiency compared to the exact method, reducing the time required to generate solutions by over 92%. The DQN-LP framework generates production sequences that yield competitive performance in terms of inner-transition costs, as shown in the bottom left graph in Fig. 10. Due to the presence of multiple periods within each episode, the outer transition cost associated with product changeovers between two consecutive periods is also computed, as depicted in the second graph in the bottom row. This indicates that the proposed hybrid framework outperforms the TSP heuristics and RAN-b benchmark. We also measure the time utility across all periods and rescale the percentage between 0 and the number of periods. As shown in the right-most graph in the top row, the hybrid DQN-LP framework achieves a higher utilisation of production time during each period than the TSP and both random baselines. Similar tendencies are observed inside the 20 product instances with 3 periods as shown in Fig. 11.

We select the evaluation scenario with the median set of performance and visualise the specific production schedules using Gantt charts for each period generated by the DQN agent, TSP and MIP in Fig. 12 for the 20-product 2-period instances, and in Fig. 13 for the 20-product 3-period instances. Analysis of both charts reveals that the DQN agent effectively identifies and prioritises products that conclude at the end of the previous period. This strategy minimises changeover times between the two periods, demonstrating the agent's capability in optimising scheduling decisions.

3.4. Study on larger-scale instances

In this section, we investigate the scalability of the proposed learning-based approach by applying it to larger-scale instances. Specifically, we evaluate its performance in scenarios involving 40, 50, 60, 70 and 80 products and an extended planning horizon with number of periods up to 8 weeks. Each value inside the graph is averaged from 5 evaluation rounds.

In the backlog level plots (left column) presented in Fig. 15 and Fig. 17, we observe that the DQN agent demonstrates better performance in terms of producing lower backlog than the TSP heuristic, RAN and RAN-b methods across the evaluated scenarios. The only exceptions occur in the 40-product instance with 2, 3, 5 and 6 periods and 80-product instance with 3 periods, where the backlog created by the DQN agent is slightly higher than any of the 3 benchmark methods. The average difference between the DQN backlog levels and the optimal solution is 15% on average across all periods. In the inventory plots (middle column) presented in Fig. 15 and Fig. 17, we observe a relatively stable trend for DQN across instances with 40 and 50 products involving different periods. In the scenarios with 60, 70 and 80 products, the DQN-generated inventory levels appear slightly elevated, which is due to one seed giving poor performance and reducing the profit. Both the TSP heuristic and RAN-b methods result in less inventory levels,

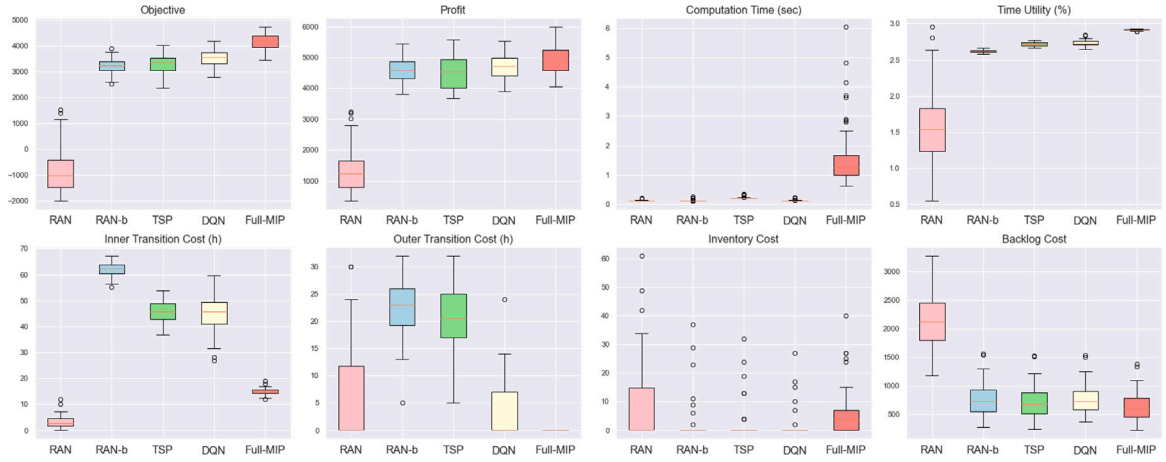

Fig. 11. Evaluation summary for 20 products and 3 periods within each episode.

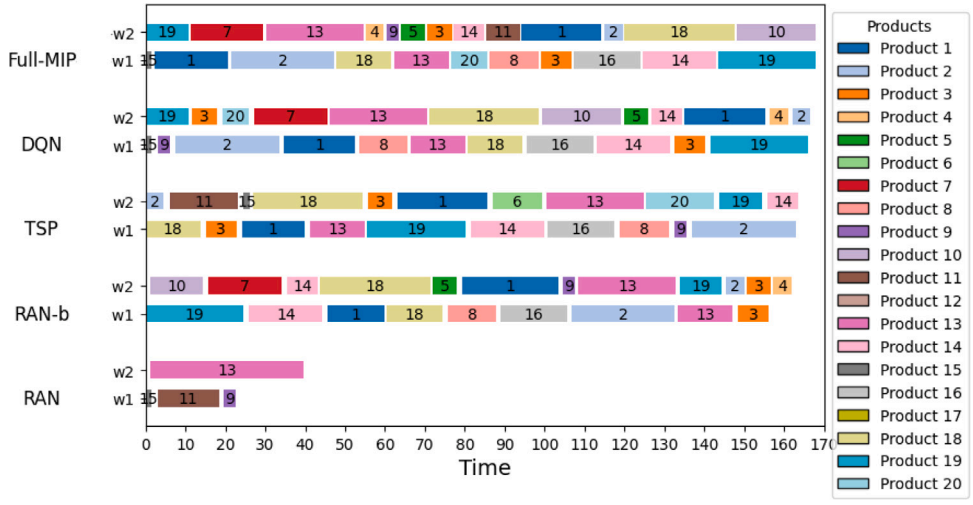

Fig. 12. Gantt chart for the median performance scenario with 20 products and 2 periods.

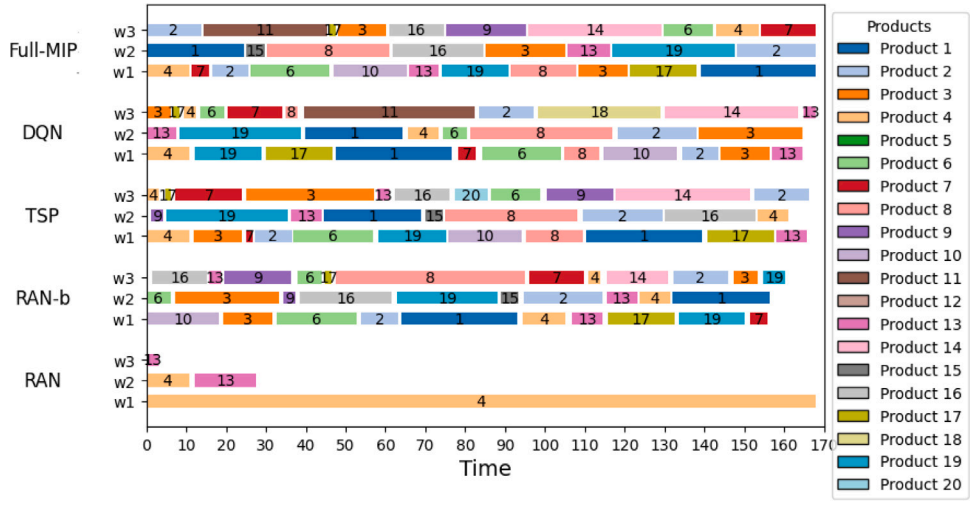

Fig. 13. Gantt chart for the median performance scenario with 20 products and 3 periods.

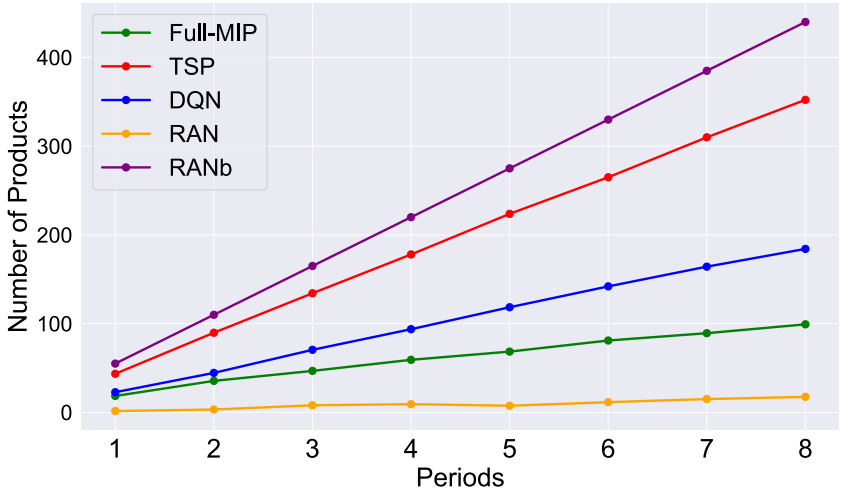

Fig. 14. Averaged total production sequence length summed across up to 8 periods generated by different methods, averaged over 40, 50, 60, 70 and 80 products instances.

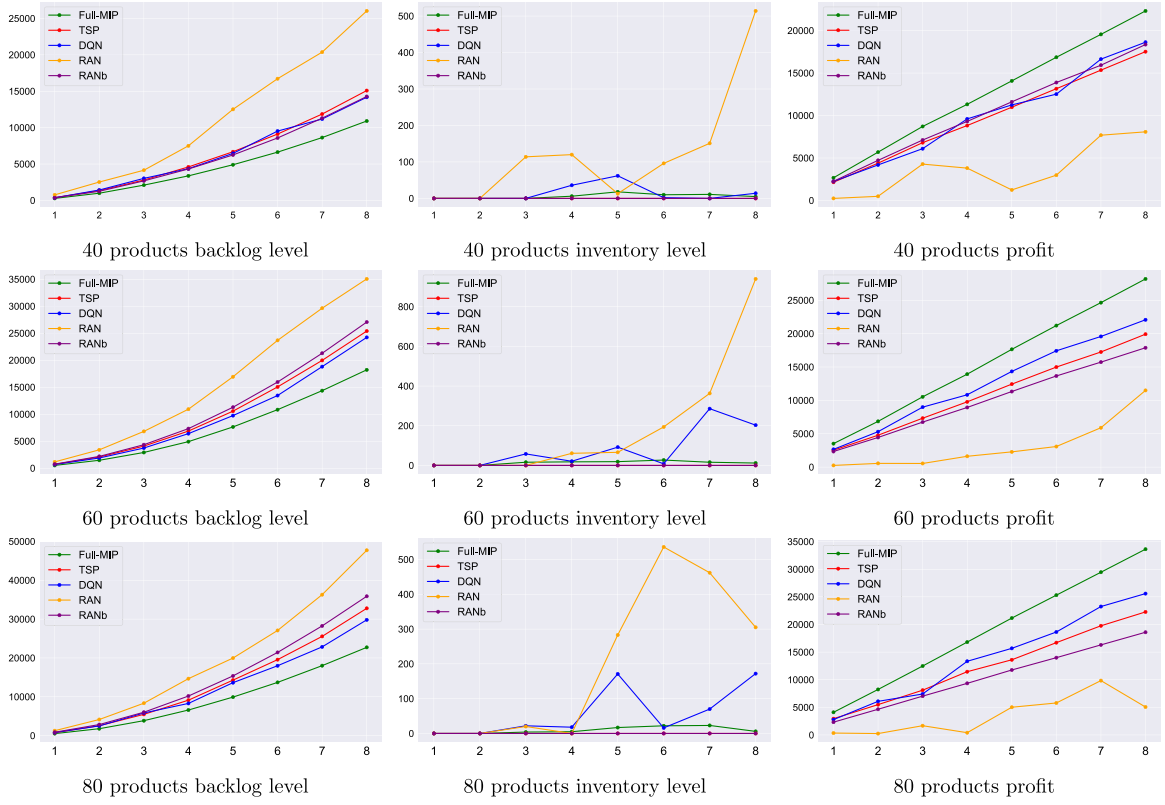

Fig. 15. Backlog level (left column) inventory level (middle column) and profit (right column) for instance sizes 40, 60 and 80 products (from top to bottom row) with 1 to 8 periods in each subplot. Each subplot compares performance level between MIP, TSP, DQN, RAN and RAN-b methods averaged from 5 evaluation rounds.

primarily due to longer sequence lengths they generate that lead to higher changeover times and demand backlog. The RAN method creates higher inventory level as it creates shorter sequence lengths on average.

In the inner transition plots (middle column) presented in Fig. 16 and Fig. 18, we observe that the DQN agent consistently gives better performance compared to the TSP heuristic and RAN-b method in generating lower inner transition cost across various instance sizes. Moreover, from the production sequence average length analysis depicted in Fig. 14, it shows that the TSP heuristic and RAN-b method generate longer production sequences than the DQN agent, leading to higher changeovers and inner transition costs. On the contrary, the RAN method creates a relatively shorter sequence length, which causes significant backlog amount but lower inner transition costs between product changeovers. Similar trend can be observed in the outer transition plots (middle column) in Fig. 16 and Fig. 18, where the DQN agent produces lower outer changeover costs between consecutive periods compared to the TSP heuristic.

The right-most column of subplots in Fig. 15 and Fig. 17 showcase the overall objective in profit. Given the computational complexity with exact solvers for larger size instances, a 1-hour CPU limit was imposed on the solution of each instance. The computational time for different instance sizes is shown in the right-most column in Fig. 16 and Fig.

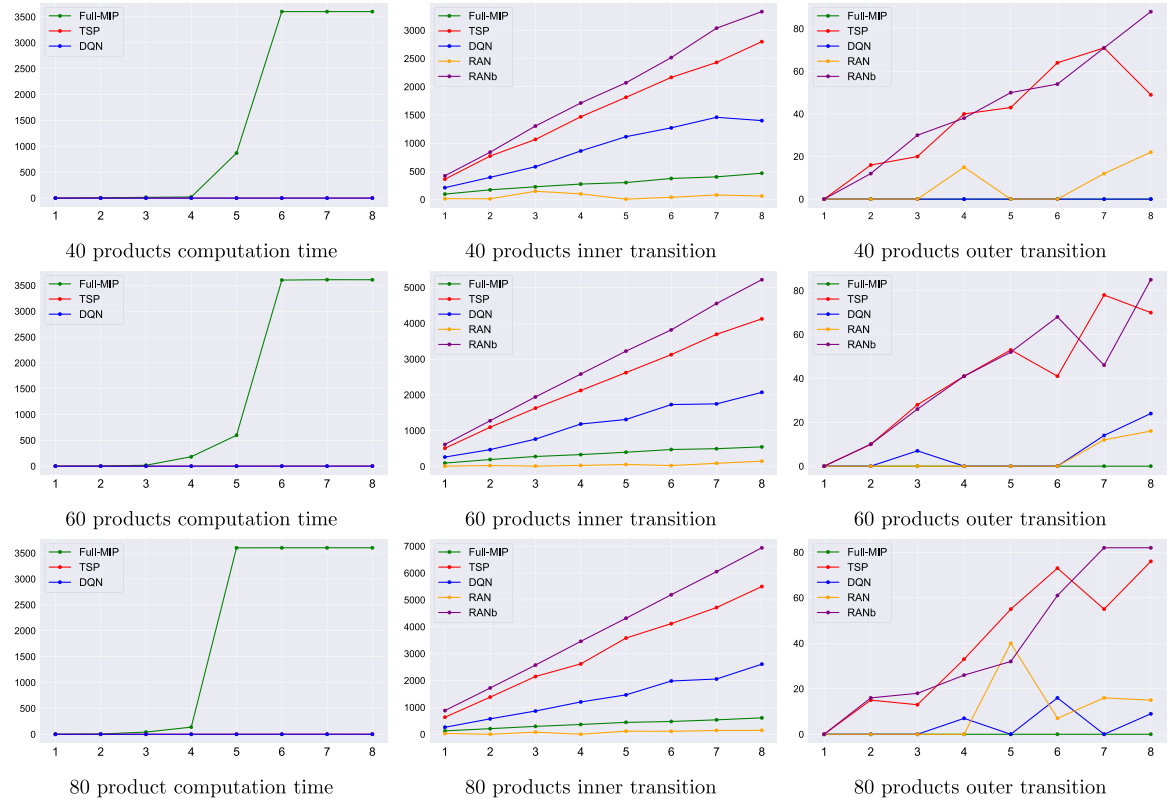

Fig. 16. Computational time [s] (left column), inner transition cost (middle column) and outer transition cost (right column) for instance sizes 40, 60 and 80 products (from top to bottom row) with 1 to 8 periods in each subplot. Each subplot compares performance level between MIP, TSP, DQN, RAN and RAN-b methods averaged from 5 evaluation rounds.

18. The plots present a comparative efficiency analysis that reveals a balance point at which the computational complexity is significant for the exact solver. We observe that the computating time begins to rise significantly when 5 or more periods are involved in the scenario. Overall, the general trend illustrates that the DQN agent generates production sequences of good quality associated with better profit level, contributing to a strong overall performance. As a result, the DQN agent outperforms the TSP heuristic, RAN and RAN-b methods for large instance sizes up to 80 products, compared to the optimal solution provided by the Full-MIP model.

Lastly, we conduct a comparative analysis of the number of constraints and variables generated by the full MILP model in contrast to the reduced LP model utilised within the decomposition framework. We also compare the number of decisions made by the DQN and RAN agents. As illustrated in Fig. 3, we observe that an increase in the length of decision process contributes to a more extensive set of agent decisions, resulting in the formation of a longer production sequence. This trend is reflected in Table 4, where the DQN's formed sequence is longer than RAN, and that the total number of decisions increase as the number of period increases. Moreover, Table 2 and Table 3 show that both the number of constraints and variables from the MILP model are substantially higher than those from the reduced LP model. Consequently, our proposed hybrid framework operates on a smaller formulation size compared to the original MILP model, thereby enhancing computational efficiency.

4. Conclusions and future work

In this work, we introduced a hybrid DQN-LP decomposition framework that recurrently optimises the production decisions for continuous manufacturing processes. The proposed framework decomposes the problem into two parts, containing an online rescheduling of production sequences tackled by the DQN agent after converting the problem into a MDP, and the remaining decisions representable by a much smaller simplified LP model that can be efficiently tackled using state-of-the-art solvers. For the model training, we employed DQN to determine the optimal production sequences for a single-unit, multi-product planning and scheduling problem, considering both the single-period and multi-period scenarios. By integrating DQN decisions based on the identified sequences into the MILP formulation, we have significantly enhanced computational efficiency to achieve competitive solution quality and enabled fast calculation of production quantities and scheduling durations. This advancement underscores the effectiveness of bridging the gap between RL and traditional optimisation techniques to streamline and accelerate complex decision-making processes for real-time dynamic manufacturing problems.

A key direction for future expansions will focus on extending the current framework to a multi-machine, multi-line setting, enabling inter-line operations and collaboration among machines to collectively produce toward a unified manufacturing objective. This effectively forms a multi-agent learning system, with the aim of further enhancing the accuracy and flexibility of the production scheduling decision-making processes in addressing more complex and dynamic industrial scenarios. Another future work direction is to incorporate a more generalised convolution neural network architecture, such as Graph Neural Network, to better generalise the training process on graph-structured data for different instance sizes.

CRediT authorship contribution statement

Syu-Ning Johnn: Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Vassilis M. Charitopoulos:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization.

Table 2
Number of constraints for instance sizes 40, 50, 60, 70 and 80 products with 1 to 8 periods in each subplot. Comparison between MILP and reduced LP formulation.

Products	Method	1	2	3	4	5	6	7	8
40	Full-MIP	3733	5938	8143	10 348	12 553	14 758	16 963	19 168
	Reduced-LP	328	610	892	1174	1456	1738	2020	2302
50	Full-MIP	5663	8918	12 173	15 428	18 683	21 938	25 193	28 448
	Reduced-LP	408	760	1112	1464	1816	2168	2520	2872
60	Full-MIP	7993	12 498	17 003	21 508	26 013	30 518	35 023	39 528
	Reduced-LP	488	910	1332	1754	2176	2598	3020	3442
70	Full-MIP	10 723	16 678	22 633	28 588	34 543	40 498	46 453	52 408
	Reduced-LP	568	1060	1552	2044	2536	3028	3520	4012
80	Full-MIP	13 853	21 458	29 063	36 668	44 273	51 878	59 483	67 088
	Reduced-LP	648	1210	1772	2334	2896	3458	4020	4582

Table 3
Number of variables for instance sizes 40, 50, 60, 70 and 80 products with 1 to 8 periods in each subplot. Comparison between MILP and reduced LP formulation.

Products	Method	1	2	3	4	5	6	7	8
40	Full-MIP	3566	7126	10 686	14 246	17 806	21 366	24 926	28 486
	Reduced-LP	1806	3606	5406	7206	9006	10 806	12 606	14 406
50	Full-MIP	5456	10 906	16 356	21 806	27 256	32 706	38 156	43 606
	Reduced-LP	2756	5506	8256	11 006	13 756	16 506	19 256	22 006
60	Full-MIP	7746	15 486	23 226	30 966	38 706	46 446	54 186	61 926
	Reduced-LP	3906	7806	11 706	15 606	19 506	23 406	27 306	31 206
70	Full-MIP	10 436	20 866	31 296	41 726	52 156	62 586	73 016	83 446
	Reduced-LP	5256	10 506	15 756	21 006	26 256	31 506	36 756	42 006
80	Full-MIP	13 526	27 046	40 566	54 086	67 606	81 126	94 646	108 166
	Reduced-LP	6806	13 606	20 406	27 206	34 006	40 806	47 606	54 406

Table 4
Number of trained DQN agent decisions for instance sizes 40, 50, 60, 70 and 80 products with 1 to 8 periods in each subplot. Comparison between DQN and RAN agents.

Products	Method	1	2	3	4	5	6	7	8
40	DQN	40	76	112	160	206	248	270	284
	RAN	4	6	36	26	12	20	28	28
50	DQN	46	90	146	166	232	262	328	366
	RAN	2	4	8	18	10	30	22	36
60	DQN	48	92	146	218	250	318	342	394
	RAN	4	8	8	14	20	18	28	42
70	DQN	48	98	160	206	260	308	374	430
	RAN	2	8	12	16	18	24	42	34
80	DQN	48	116	162	238	288	380	400	498
	RAN	8	4	18	8	32	32	40	42

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

Financial support from the Engineering & Physical Sciences Research Council, UK grants EP/V051008/1, EP/W003317/1 is gratefully acknowledged.

Appendix A. TSP-based MILP formulation

Let $i \in I$ be the set of products on the processing unit and $p \in P$ as the set of planning periods. Following the notation given in Charitopoulos et al. (2017), we define the list of model parameters in Table 5 and the list of decision variables in Table 6 for the complete MILP formulation given in (3).

Table 5
MILP parameters summary.

Symbol	Category	Description
D_i	real	demand of product i at the end of period p
$\tau_{i,j}$	real	changeover time from product i to j
PS_i	real	sale price of product i
CI_i	real	unit inventory cost of product i
CB_i	real	unit backlog cost of product i
$\mathrm{CC}_{i,j}$	real	changeover cost from product i to j
r_i	real	processing rate of product i
θ_p^{low}	real	lower bound on processing rate in period p
θ_p^{up}	real	upper bound on processing rate in period p
V_i^{min}	real	lower bound on inventory level for product i
V_i^{max}	real	upper bound on inventory level for product i
M	integer	large number, takes the value of the cardinality of set I

The complete MILP formulation for the planning and scheduling problem is presented below.

$$\max \sum_{i\in I}\sum_{p\in P} \mathrm{PS}_i \cdot S_{i,p} - \sum_{i\in I}\sum_{\substack{j\in I\\ i\neq j}}\sum_{p\in P} \mathrm{CC}_{i,j}\cdot Z_{i,j,p} - \sum_{i\in I}\sum_{\substack{j\in I\\ i\neq j}}\sum_{\substack{p\in P\\ p\geq 2}} \mathrm{CC}_{i,j}\cdot \mathrm{ZF}_{i,j,p} - \sum_{i\in I}\sum_{p\in P} \mathrm{CB}_i \cdot B_{i,p} - \sum_{i\in I}\sum_{p\in P} \mathrm{CI}_i \cdot V_{i,p} \tag{3a}$$

subject to

$$\sum_{i\in I} F_{i,p} = 1 \qquad p\in P \tag{3b}$$

$$\sum_{i\in I} L_{i,p} = 1 \qquad p\in P \tag{3c}$$

$$F_{i,p} \leq E_{i,p} \qquad i\in I, p\in P \tag{3d}$$

$$L_{i,p} \leq E_{i,p} \qquad i\in I, p\in P \tag{3e}$$

$$\sum_{i\in I} Z_{i,j,p} = E_{i,p} - F_{i,p} \qquad j\in I, i\neq j, p\in P \tag{3f}$$

Table 6
MILP decision variables summary.

Symbol	Category	Description
$E_{i,p}$	binary	whether product i is scheduled for production in period p
$F_{i,p}$	binary	whether product i is the first product in the sequence in period p
$L_{i,p}$	binary	whether product i is the last product in the sequence in period p
$Z_{i,j,p}$	binary	whether product i precedes product j in period p
$\mathrm{ZF}_{i,j,p}$	binary	whether product i precedes product j in periods p and $p+1$
$O_{i,p}$	integer	order index of product i in the sequence in period p
$\mathrm{Pr}_{i,p}$	real	amount of product i produced in period p
$S_{i,p}$	real	sales of amount of product i in period p
$T_{i,p}$	real	processing time of product i in period p
$V_{i,p}$	real	inventory level of product i in period p
$B_{i,p}$	real	backlog level of product i in period p

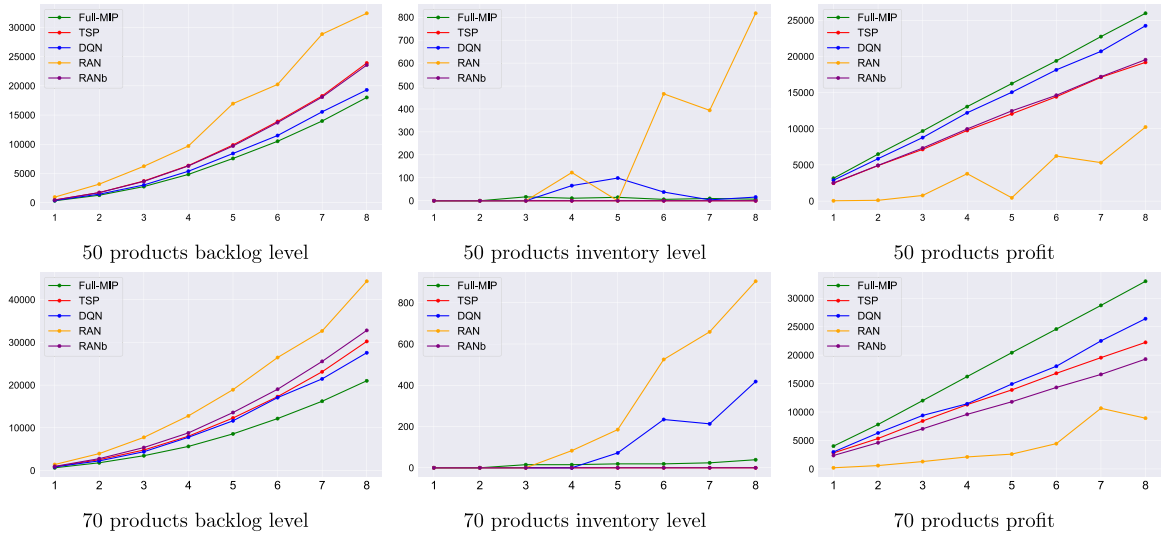

Fig. 17. Backlog level (left column) inventory level (middle column) and profit (right column) for instance sizes 50 and 70 products (from top to bottom row) with 1 to 8 periods in each subplot. Each subplot compares performance level between MIP, TSP, DQN, RAN and RAN-b methods averaged from 5 evaluation rounds.

$$\sum_{j\in I} Z_{i,j,p} = E_{i,p} - L_{i,p} \qquad i \in I, i \neq j, p \in P \quad (3g)$$

$$\sum_{i\in I} \mathrm{ZF}_{i,j,p} = F_{j,p} \qquad j \in I, p \in P, p \geq 2 \quad (3h)$$

$$\sum_{j\in I} \mathrm{ZF}_{i,j,p} = L_{j,p-1} \qquad j \in I, p \in P, p \geq 2 \quad (3i)$$

$$O_{j,p} - (O_{i,p} + 1) \leq -M(1 - Z_{i,j,p}) \qquad i, j \in I, i \neq j, p \in P \quad (3j)$$

$$O_{i,p} \leq M \cdot E_{i,p} \qquad i \in I, p \in P \quad (3k)$$

$$F_{i,p} \leq O_{i,p} \leq \sum_{i\in I} E_{i,p} \qquad i \in I, p \in P \quad (3l)$$

$$\theta_p^{low} E_{i,p} \leq T_{i,p} \leq \theta_p^{up} E_{i,p} \qquad i \in I, p \in P \quad (3m)$$

$$\sum_{i\in I} T_{i,p} + \sum_{i\in I} \sum_{j\in I: i\neq j} \tau_{i,j}(Z_{i,j,p} + ZF_{i,j,p}) \leq \theta^{up} \qquad p \in P \quad (3n)$$

$$\mathrm{Pr}_{i,p} = r_i T_{i,p} \qquad i \in I, p \in P \quad (3o)$$

$$B_{i,p} = B_{i,p-1} + D_{i,p} - S_{i,p} \qquad i \in I, p \in P \quad (3p)$$

$$V_{i,p} = V_{i,p-1} + \mathrm{Pr}_{i,p} - S_{i,p} \qquad i \in I, p \in P \quad (3q)$$

$$V_i^{\min} \leq V_{i,p} \leq V_i^{\max} \qquad i \in I, p \in P \quad (3r)$$

where the objective function (3a) represents an optimisation that maximises the total production revenue while simultaneously minimising the penalties that include the internal products changeover costs, the external products changeover costs between two consecutive periods, the demand backlog costs, and the additional production inventory costs.

Constraints (3b) and (3c) specify that in any given planning period, only one product can be produced first and only one product can be produced last, respectively. Constraints (3d) and (3e) ensure that a product cannot be scheduled as the first or last in a period unless it is assigned to that period, thereby maintaining the feasibility of the numerical index for each product. Constraints (3f) and (3g) specify that if a product is assigned to a planning period, it will result in a changeover with another assigned product unless it is either the first or the last to be processed in that period. Similarly, constraints (3h) and (3i) model the changeovers across adjacent periods.

The following set of symmetric-breaking constraints (3j)–(3l) exclude infeasible production sub-cycles via the introduction of procession order index $O_{i,p}$, thereby avoiding the enumeration of symmetric solutions. Specifically, constraints (3j) indicate the order index of any product j that is processed after i must be bigger than 1. Constraints (3k) suggest that if a product i is not assigned to a planning period, then its order index should be set to 0. Constraints (3l) specify the upper and lower bounds of the order index.

To track timing within each period, we use constraints (3m) to set the minimum and maximum processing time thresholds that can be dedicated to any product scheduled inside the sequence. Constraints (3n) provide an upper bound for each period on the total allowed time, consisting of both the total production time, the internal changeover time, and the changeover time between two periods. For the remaining continuous variables, constraints (3o) specify the production rate and link the production amount to the production time variables. Constraints (3p) ensure the conservation of backlog balance for any product in any period. Similarly, constraints (3q) ensure the inventory balance for any product in any period. Lastly, constraints (3r) limit the inventory level of any product within a specified range of upper and lower bounds.

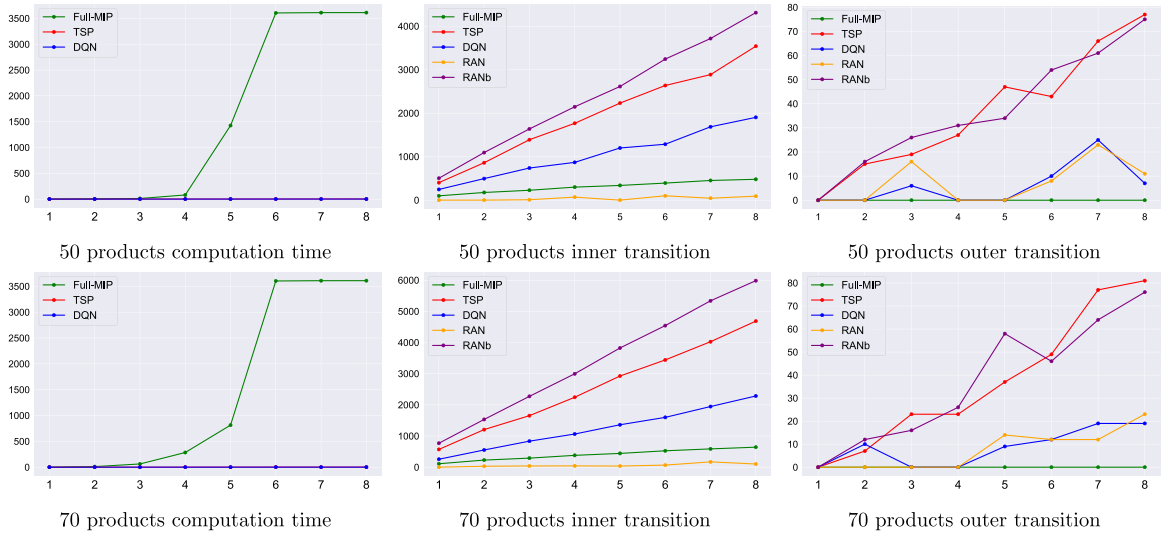

Fig. 18. Computational time in [s] (left column), inner transition cost (middle column) and outer transition cost (right column) for instance sizes 50 and 70 products (from top to bottom row) with 1 to 8 periods in each subplot. Each subplot compares performance level between MIP, TSP, DQN, RAN and RAN-b methods averaged from 5 evaluation rounds..

Appendix B. Additional plots

See Figs. 17 and 18.

Data availability

Data will be made available upon request by the corresponding author.

References

Badejo, O., Ierapetritou, M., 2022. Integrating tactical planning, operational planning and scheduling using data-driven feasibility analysis. Comput. Chem. Eng. 161, 107759.

Baker, K.R., 1977. An experimental study of the effectiveness of rolling schedules in production planning. Decis. Sci. 8 (1), 19–27.

Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B., 2011. Algorithms for hyper-parameter optimization. NeurIPS 24.

Castro, P.M., Grossmann, I.E., Zhang, Q., 2018. Expanding scope and computational challenges in process scheduling. Comput. Chem. Eng. 114, 14–42.

Charitopoulos, V.M., Dua, V., Papageorgiou, L.G., 2017. Traveling salesman problem-based integration of planning, scheduling, and optimal control for continuous processes. Ind. Eng. Chem. Res. 56 (39), 11186–11205.

Charitopoulos, V.M., Papageorgiou, L.G., Dua, V., 2019. Closed-loop integration of planning, scheduling and multi-parametric nonlinear control. Comput. Chem. Eng. 122, 172–192.

Chiang, L.H., Braun, B., Wang, Z., Castillo, I., 2022. Towards artificial intelligence at scale in the chemical industry. AIChE J. 68 (6), e17644.

Erdirik-Dogan, M., Grossmann, I.E., 2006. A decomposition method for the simultaneous planning and scheduling of single-stage continuous multiproduct plants. Ind. Eng. Chem. Res. 45 (1), 299–315.

Erdirik-Dogan, M., Grossmann, I.E., 2008. Simultaneous planning and scheduling of single-stage multi-product continuous plants with parallel lines. Comput. Chem. Eng. 32 (11), 2664–2683.

Framinan, J.M., Fernandez-Viagas, V., Perez-Gonzalez, P., 2019. Using real-time information to reschedule jobs in a flowshop with variable processing times. Comput. Ind. Eng. 129, 113–125.

Frazier, P.I., 2018. A tutorial on Bayesian optimization. arXiv preprint arXiv:1807.02811.

Fuentes-Cortés, L.F., Flores-Tlacuahuac, A., Nigam, K.D., 2022. Machine learning algorithms used in PSE environments: A didactic approach and critical perspective. Ind. Eng. Chem. Res. 61 (25), 8932–8962.

Gu, S., Lillicrap, T., Ghahramani, Z., Turner, R.E., Levine, S., 2016. Q-prop: Sample-efficient policy gradient with an off-policy critic. arXiv preprint arXiv:1611.02247.

Gupta, D., Maravelias, C.T., 2019. On the design of online production scheduling algorithms. Comput. Chem. Eng. 129, 106517.

Gupta, D., Maravelias, C.T., Wassick, J.M., 2016. From rescheduling to online scheduling. Chem. Eng. Res. Des. 116, 83–97.

Harjunkoski, I., Maravelias, C.T., Bongers, P., Castro, P.M., Engell, S., Grossmann, I.E., Hooker, J., Méndez, C., Sand, G., Wassick, J., 2014. Scope for industrial applications of production scheduling models and solution methods. Comput. Chem. Eng. 62, 161–193.

Hart, W.E., Watson, J.-P., Woodruff, D.L., 2011. Pyomo: modeling and solving mathematical programs in python. Math. Program. Comput. 3, 219–260.

Hubbs, C.D., Li, C., Sahinidis, N.V., Grossmann, I.E., Wassick, J.M., 2020a. A deep reinforcement learning approach for chemical production scheduling. Comput. Chem. Eng. 141, 106982.

Hubbs, C.D., Perez, H.D., Sarwar, O., Sahinidis, N.V., Grossmann, I.E., Wassick, J.M., 2020b. Or-gym: A reinforcement learning library for operations research problems. arXiv preprint arXiv:2008.06319.

Karimi-Mamaghan, M., Mohammadi, M., Pasdeloup, B., Meyer, P., 2022. Learning to select operators in meta-heuristics: An integration of Q-learning into the iterated greedy algorithm for the permutation flowshop scheduling problem. European J. Oper. Res..

Kopanos, G.M., Pistikopoulos, E.N., 2014. Reactive scheduling by a multiparametric programming rolling horizon framework: a case of a network of combined heat and power units. Ind. Eng. Chem. Res. 53 (11), 4366–4386.

Kopanos, G.M., Puigjaner, L., 2019. Solving Large-Scale Production Scheduling and Planning in the Process Industries. Springer.

Lee, J.H., Shin, J., Realff, M.J., 2018. Machine learning: Overview of the recent progresses and implications for the process systems engineering field. Comput. Chem. Eng. 114, 111–121.

Li, Z., Ierapetritou, M.G., 2008. Reactive scheduling using parametric programming. AIChE J. 54 (10), 2610–2623.

Liu, C.-L., Chang, C.-C., Tseng, C.-J., 2020. Actor-critic deep reinforcement learning for solving job shop scheduling problems. Ieee Access 8, 71752–71762.

Liu, S., Pinto, J.M., Papageorgiou, L.G., 2008. A TSP-based MILP model for medium-term planning of single-stage continuous multiproduct plants. Ind. Eng. Chem. Res. 47 (20), 7733–7743.

Liu, S., Pinto, J.M., Papageorgiou, L.G., 2010. MILP-based approaches for medium-term planning of single-stage continuous multiproduct plants with parallel units. Comput. Manag. Sci. 7 (4), 407–435.

Ma, Y., Zhu, W., Benton, M.G., Romagnoli, J., 2019. Continuous control of a polymerization system with deep reinforcement learning. J. Process Control 75, 40–47.

McAllister, R.D., Rawlings, J.B., Maravelias, C.T., 2019. Rescheduling penalties for economic model predictive control and closed-loop scheduling. Ind. Eng. Chem. Res. 59 (6), 2214–2228.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., et al., 2015. Human-level control through deep reinforcement learning. Nature 518 (7540), 529–533.

Nian, R., Liu, J., Huang, B., 2020. A review on reinforcement learning: Introduction and applications in industrial process control. Comput. Chem. Eng. 139, 106886.

Pan, E., Petsagkourakis, P., Mowbray, M., Zhang, D., del Rio-Chanona, A., 2021. Constrained Q-learning for batch process optimization. IFAC-PapersOnLine 54 (3), 492–497.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al., 2019. Pytorch: An imperative style, high-performance deep learning library. NeurIPS 32.
Perez, H.D., Amaran, S., Erisen, E., Wassick, J.M., Grossmann, I.E., 2021. Optimization of extended business processes in digital supply chains using mathematical programming. Comput. Chem. Eng. 152, 107323.
Petsagkourakis, P., Sandoval, I.O., Bradford, E., Zhang, D., del Rio-Chanona, E.A., 2020. Reinforcement learning for batch bioprocess optimization. Comput. Chem. Eng. 133, 106649.
Sung, C., Maravelias, C.T., 2007. An attainable region approach for production planning of multiproduct processes. AIChE J. 53 (5), 1298–1315.
Sutton, R.S., Barto, A.G., 2018. Reinforcement Learning: An Introduction. MIT Press.
Watkins, C.J., Dayan, P., 1992. Q-learning. Mach. Learn. 8, 279–292.
Zheng, X., Chen, Z., 2024. An improved deep Q-learning algorithm for a trade-off between energy consumption and productivity in batch scheduling. Comput. Ind. Eng. 188, 109925.