Computers and Chemical Engineering 186 (2024) 108700

Contents lists available at ScienceDirect

Computers and Chemical Engineering

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

Production rescheduling via explorative reinforcement learning while considering nervousness

Sumin Hwangbo [a,b], J. Jay Liu [c], Jun-Hyung Ryu [d], Ho Jae Lee [e,*], Jonggeol Na [a,b,*]

[a] Department of Chemical Engineering and Materials Science, Ewha Womans University, Seoul 03760, Republic of Korea
[b] Graduate Program in System Health Science and Engineering, Ewha Womans University, Seoul 03760, Republic of Korea
[c] Department of Chemical Engineering, Pukyong National University, Busan 48513, Republic of Korea
[d] Department of Energy & Electricity Engineering, Dongguk University WISE Campus, Gyeongju, 38066, Republic of Korea
[e] Department of Chemical Engineering, Hongik University, Seoul 04066, Republic of Korea

ARTICLE INFO

Keywords:
Rescheduling
Reinforcement learning
Schedule nervousness
Machine learning
Production scheduling
Mathematical programming

ABSTRACT

Nervousness-aware rescheduling is essential in maximizing the profitability and stability of processes in manufacturing industries. It involves re-optimization to meet scheduling goals while minimizing deviations from the base schedule. However, conventional mathematical optimization becomes impractical due to high computational costs and the inability to handle real-time rescheduling. Here, we propose an online rescheduling agent trained by explorative reinforcement learning that autonomously optimizes schedules while considering schedule nervousness. In a static scheduling environment, our model consistently achieves over 90% of the cost objective with scalability and flexibility. A computational time comparison proves that the reinforcement learning methodology makes near-optimal decisions rapidly, irrespective of the complexity of the scheduling problem. Furthermore, we present several realistic rescheduling scenarios that demonstrate the capability of our methodology. Our study illustrates the significant potential of reinforcement learning methodology in expediting digital transformation and process automation within real-world manufacturing systems.

1. Introduction

Optimizing process operations stands as a critical factor for enterprises aiming to remain competitiveness in the global marketplace. The decision-making involved in process operations can be divided into three layers depending on their spatial and temporal scopes: production planning, scheduling, and control. Among them, production scheduling is the process wherein the optimal sequence and timing for various manufacturing tasks and activities are determined (Shahzad and Mebarki, 2012). It entails the coordination of resources – such as materials, equipment, and utilities – to ensure that production processes run efficiently, meet deadlines, and achieve the desired output with minimal costs and downtime (Pinedo, 2012). In spatial and temporal terms, these decisions typically span the entire plant and encompass spans of days to weeks. As a component of multiple-layer of decision-making, production scheduling is posed as an optimization problem within the decision space bounded by the planning layer and considering the constraints imposed by the control layer. Given the resource availability, facility layout, product recipes and production targets, in general, the major decisions to be made in production scheduling are (i) the selection of tasks (i.e., batch processes or product campaigns), (ii) the allocation of tasks to units, and (iii) the timing and sequencing of tasks in each unit.

However, given the diverse industrial sectors in which production scheduling is applied, ranging from discrete manufacturing to petrochemicals, some industrial sectors require sector-specific constraints and/or decisions (Pinedo and Chao, 1999). For example, in paper industries, cutting stock decisions are made simultaneously during scheduling to minimize the paper trim loss (Säynevirta and Luotojärvi, 2004) and in pharmaceutical manufacturing, batch integrity needs to be preserved for product back-trackability in cases of faulty products (Papavasilleiou et al., 2007). Moreover, sectors like the chemical industries must factor in supplementary aspects such as material-specific storage policies, grade loss stemming from campaign transitions, and product diversification (Harjunkoski et al., 2014). The considerable benefits that optimal production scheduling brings have prompted extensive research spanning various scientific communities which are well captured in several review papers (Floudas and Lin, 2004; Méndez et al., 2006; Li and Ierapetritou, 2008; Ribas et al., 2010; Maravelias, 2012; Harjunkoski et al., 2014; Castro et al., 2018).

* Corresponding authors.
E-mail addresses: hojaelee@hongik.ac.kr (H.J. Lee), jgna@ewha.ac.kr (J. Na).

https://doi.org/10.1016/j.compchemeng.2024.108700
Received 19 December 2023; Received in revised form 23 March 2024; Accepted 13 April 2024
Available online 17 April 2024
0098-1354/

Traditionally, production scheduling has been carried out by experienced individuals relying on rule-of-thumb techniques using spreadsheets. This method of scheduling is prone to human errors; leads to inconsistent/suboptimal schedules; and are volatile, having to rely on a select few individuals. Unfortunately, although advanced scheduling models and solution methods (Velez et al., 2015) have been developed in the research community, many industries still rely on the traditional approach. Nevertheless, there has been a recent shift towards developing proprietary scheduling tools or adopting commercially available software. These employ different techniques such as heuristics, metaheuristics, mathematical programming models, and data-driven models.

Among the simpler scheduling methods are those based on heuristics. Due to their simplicity, they are also some of the quickest methods. They often include simple dispatching rules, such as first-come–first-serve and earliest-due-date-first (Ruiz, 2016). Although scheduling heuristics are fast and simple to implement, the schedules produced often fall short of being optimal. In contrast, methods grounded in metaheuristics yield higher-quality schedules. These methods stochastically guide the solution to the near optimum by efficiently sampling the decision space. Durasević and Jakobović (2023) categorizes and consolidates heuristics and metaheuristics employed for solving the parallel unrelated machines scheduling problem. Additionally, Pellerin et al. (2020) delves into various hybrid forms of metaheuristic approaches, specifically tailored to solve the resource-constrained project scheduling problem. Genetic algorithms stand out as a prevalent choice among metaheuristic approaches, evolving into methodologies utilizing priority functions through genetic programming (Durasević et al., 2016). Integration of particle swarm optimization with genetic algorithms enhanced the speed of genetic algorithms and the solution quality of particle swarm optimization (Shang et al., 2018). Metaheuristics combining simulated annealing with genetic algorithms were demonstrated to be more efficient than existing algorithms, requiring smaller population sizes and iterations (Lin et al., 2010). Bewoor et al. (2018) showcased their application in foundry production scheduling with no-wait constraints, proving their effectiveness. Ahmadian et al. (2021) and Ahmadian and Salehipour (2021) introduced a variable neighborhood search method for solving job shop scheduling problems with specified due dates, which found application in cyclical multiple parallel machine scheduling within sugarcane unloading systems (Kusoncum et al., 2021). Using metaheuristic approaches in production scheduling offers the advantage of efficiency and broad applicability across a wide range of problems, owing to its non-problem-specific nature. This characteristic has spurred a rapid surge in advancements in new metaheuristic algorithms. However, these methods are susceptible to becoming trapped in local optima, and thus, do not guarantee optimal schedulings.

In recent decades, production scheduling based on mathematical programming has garnered substantial attention within the research community. While this approach guarantees optimal schedules and allows for the precise modeling of intricate constraints, it comes at a computational cost, particularly when applied to large-scale industrial problems. Over the years, significant advances have been made in model development and solution methods. The introduction of novel concepts has facilitated the creation of smaller scheduling models and models with mathematical structures amenable to decomposition and parallel computing strategies. Such concepts include models based on multiple unit-specific continuous-time grids (Giannelos and Georgiadis, 2002; Janak et al., 2005; Susarla et al., 2010; Seid and Majozi, 2012), and multiple nonuniform discrete-time grids (Velez and Maravelias, 2013b, 2015) and models using mixed time representation (Maravelias, 2005; Westerlund et al., 2007). Despite efforts to enhance model speed through as valid inequalities and tightening constraints (Velez et al., 2013; Merchan et al., 2013, 2016), reformulations (Sahinidis and Grossmann, 1991; Velez and Maravelias, 2013c), decomposition techniques (Wu and Ierapetritou, 2003; Calfa et al., 2013; Lee and Maravelias, 2018, 2019, 2020), and parallel solution algorithms (Subrahmanyam et al., 1996; Ferris et al., 2009; Velez and Maravelias, 2013a), the application of mathematical programming scheduling models to industrial problems remains a challenge.

As the trend of data-driven approaches gains momentum across research fields, data-driven scheduling models are gaining increased attention. One of the earliest of these studies utilized Q-learning to optimize the single machine dispatching rule (Wang and Usher, 2005). Genetic programming was harnessed by Nguyen et al. (2017) to model production scheduling, showcasing its adaptability in complex and dynamic production environments. Later, a production scheduling model utilizing the Deep Q-Network agent algorithm (Mnih et al., 2015), an algorithm developed by Google DeepMind, was developed and validated against scheduling problems at a semiconductor production facility (Waschneck et al., 2018) and EV charging strategies (Zhang et al., 2023). For chemical production scheduling, Hubbs et al. (2020) and Hubert et al. (2023) introduced deep reinforcement learning models to accommodate data uncertainty, comparing their performance against mathematical models. Kim and Maravelias (2022) devised techniques to estimate the feasibility and computational requirements required for solving mathematical programming scheduling models through supervised machine learning techniques. Nickel et al. (2020) and Li et al. (2020b) showcased the applicability of four distinct reinforcement learning techniques – Q-learning, Sarsa, Watkins's Q(λ), and Sarsa(λ) – was demonstrated for online single-machine scheduling aimed at minimizing the total earliness and tardiness of jobs. Li et al. (2020a) proposed a framework combining machine learning classification technique to identify rescheduling patterns with metaheuristic optimization. More recently, stochastic deep learning was used to develop models for integrated scheduling and control (Santander et al., 2023; Balasubramaniam et al., 2023; Cha et al., 2023). A key advantage of data-driven scheduling models lies in their computational efficiency post-training, allowing them to identify near-optimal solutions quickly.

Despite the recent developments in production scheduling methods, direct application to industrial problems remains a challenge. Among the many factors that contribute to this, applying the methods in an online scheduling setting gives rise to a couple of issues: (i) the need to generate alternative schedules quickly; and (ii) the need to reduce schedule nervousness to minimize disruptions to the shop-floor due to frequent schedule revisions (Lee et al., 2020a). Although research to overcome these issues has been conducted over the years, most studies are application-specific. Some have concentrated on discrete manufacturing contexts: van Donselaar and Gubbels (2002), Pujawan (2004) and Law and Gunasekaran (2010) addressed schedule nervousness reduction in truck manufacturing, shoe manufacturing, and high-tech electronics manufacturing, respectively. Hasachoo and Masuchun (2015, 2016) studied schedule nervousness in airline catering systems. Stability within materials requirements planning has been explored extensively (Kazan et al., 2000; Ho, 2002; Li and Disney, 2017; Koca et al., 2018). For chemical industries, Lee et al. (2020a) proposed systematic methods to generate alternative schedules while considering nervousness, and Ave et al. (2019) proposed a resource-job network-based model to avoid nervousness. More recently, state-space models for online scheduling have been proposed, and the methods for evaluating and improving the quality of implemented schedules have been investigated (Subramanian et al., 2012; Gupta et al., 2016; Gupta and Maravelias, 2019, 2020). Despite achieving advancements, these studies have limitations because they either (i) focused on developing policies and heuristics that often lead to far-from-optimal solutions or (ii) relied on computationally intensive methods such as mathematical programming models.

The objective of this paper is to address the challenges of applying scheduling methods to an online setting. Specifically, we propose a single-agent reinforcement learning (RL) model that can be applied to large-scale scheduling problems, with the computational time of

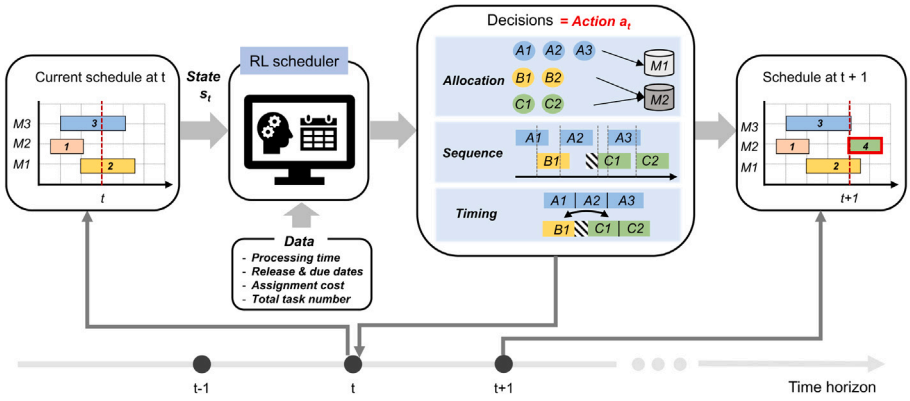

Fig. 1. Schematic illustration of the reinforcement learning scheduling system used in this study.

inference remaining consistent regardless of the problem scale and complexity. Furthermore, we develop a model capable of rapidly generating alternative schedules in fluctuating environments while accommodating varying degrees of schedule nervousness. In Section 2, we outline the problem statement and introduce the mathematical programming scheduling model utilized as a benchmark. Then, in Section 3, we present the specifics of the methods employed for training and evaluating the reinforcement learning model. Lastly, Section 4 showcases the computational studies for performance assessment, along with case studies illustrating the model's practical applicability.

2. Problem description

2.1. Single-stage scheduling problem

The single-stage scheduling problem investigated in this study entails a production environment where tasks undergo a single processing stage and are carried out in one of multiple parallel units (Georgiadis et al., 2019). Each job within the single-stage scheduling problem is composed of one operation that progresses through a sole unit for a specific duration, undergoing processing exclusively within that unit (Harjunkoski et al., 2014). Each job becomes available for scheduling after a predefined release time and must be completed before its due date. All operations can be completed on any machine, differing only in terms of cost and processing time (Fig. 1). Hence, the objective of this scheduling problem is to minimize the total assignment cost while ensuring that all jobs from a given set, I, are allocated to one among the parallel machines within set J. We addressed single-stage scheduling problems at five distinct scales using the data set provided in Harjunkoski and Grossmann (2002). The scales of the problems range from a simple case of allocating 3 jobs to 2 machines to a more complex scenario of allocating 20 jobs to 5 machines, which presents a substantial challenge. It encompasses 10 different instances with different data sets.

2.2. Mathematical formulation

To assess the validity of the optimization results achieved by the RL scheduler, mathematical programming, the conventional method for finding the global optimum under constraints, was applied to solve the scheduling problem. The scheduling problem was mathematically formulated as a mixed-integer linear programming (MILP) problem, based on objective function and seven constraints.

Eq. (1) defines the objective function of the scheduling problem as the minimization of the total assignment cost. The variable C_{ij} represents the cost associated with assigning job i to machine j, accounting for the variation in costs across different machines. X_{ij} is a binary variable that is equal to 1 when job i is allocated to machine j and 0 otherwise. The objective is to determine j_i, representing the machine where job i is scheduled, for all jobs in set I, aiming to minimize the total assignment cost.

$y_{ii'}$ is a binary variable that takes the value 1 when job i precedes job i' in the sequence. ts_i is a non-negative continuous variable that represents the starting time of job i, while R_i and D_i denote the release date and due date parameter of job i, respectively. Parameter P_{ij} represents the processing time of job i when it is assigned to machine j. M represents the large integer value used for the big M constraint method in the constraint. Based on the following parameters and variables, Eqs. (2) to (8) are the constraints of the scheduling problem. Specifically, Eq. (2) ensures that each job must be allocated to exactly one machine. Eqs. (5) and (6) ensure that each job is processed within the time interval bounded by its release and the due dates. Eq. (7) represents the constraint that the job i' scheduled after job i can begin no earlier than the elapsed time of job i, which is the starting time of job i plus the processing time (P_{ij}) using big M constraint. Lastly, Eq. (8) imposes a constraint on the job sequence, requiring that if job i and i' are allocated to the same machine j, they cannot be executed concurrently but in a specific sequential order.

$$\min_{j_i} \sum_{i\in I}\sum_{j\in J} C_{ij}X_{ij} \tag{1}$$

$$\text{s.t.} \sum_{j} X_{ij} = 1 \quad \forall i \in I \tag{2}$$

$$X_{ij} \in \{0,1\} \quad \forall i \in I, \forall j \in J \tag{3}$$

$$y_{ii'} \in \{0,1\} \quad i,i' \in I \tag{4}$$

$$ts_i \geq R_i \quad \forall i \in I \tag{5}$$

$$ts_i \leq D_i - \sum_{j} P_{ij}X_{ij} \quad \forall i \in I \tag{6}$$

$$ts_i + \sum_{j} P_{ij}X_{ij} - M(1-y_{ii'}) \leq ts'_i \quad \forall i,i' \in I, i \neq i' \tag{7}$$

$$y_{ii'} + y_{i'i} \geq X_{ij} + X_{i'j} - 1 \quad \forall i,i' \in I, \forall j \in J \tag{8}$$

3. Methods of reinforcement learning scheduler

3.1. RL architecture and network

We employed the Policy Proximal Optimization (PPO) algorithm, a model-free reinforcement learning technique based on policy gradients (Schulman et al., 2017). The PPO consists of an actor network and a critic network, as depicted in Fig. 2d. The actor network takes states as inputs and gives probability distributions over possible actions. Accordingly, within the actor network, policy updates are performed by comparing the ratio between the current policy ($\pi_\theta(a_t|s_t)$) and the previous policy ($\pi_{\theta_{\text{old}}}(a_t|s_t)$), as shown in Eq. (11). Consequently, the

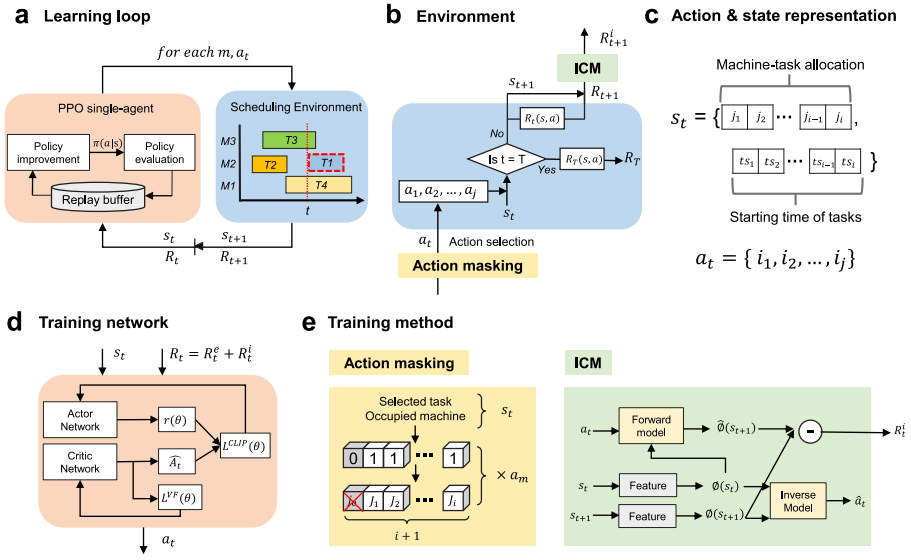

Fig. 2. Reinforcement learning training framework: (a) a representation of the single-agent reinforcement learning PPO algorithm training loop; (b) the scheduling environment structure; (c) the actor–critic based PPO algorithm training network; (d) a representation of the input state, including the machine-job allocation and starting time of each job, providing scheduling information required for the reinforcement learning model; and (e) the methods used in the training phase.

actor network is updated in the direction that maximizes the surrogate function $L^{CLIP}(\theta)$ while constraining the probability ratio through clipping, as shown in Eqs. (9) and (10). In these equations, ϵ denotes the clip parameter and, $\hat{A}_t$ represents the advantage, which captures the disparity between the reward in the current state and the predicted value. Here, clipping is implemented to ensure that the new policy does not deviate significantly from the previous one.

$$\max_{\theta} \quad L^{CLIP}(\theta) = E_t[\min(r(\theta)\hat{A}_t, \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t] \tag{9}$$

$$\text{s.t.} \quad \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon) = \begin{cases} 1-\epsilon, r(\theta) < 1-\epsilon \\ r(\theta), 1-\epsilon \le r(\theta) < 1+\epsilon \\ 1+\epsilon, r(\theta) \ge 1+\epsilon \end{cases} \tag{10}$$

$$r(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{11}$$

On the other hand, the critic network verifies the value associated with a given state. It utilizes a value function($V_\theta(s_t)$) to assess how favorable a state(s_t) is, and the update of the loss function revolves around minimizing the difference between the predicted state-value function and the target value (Eq. (12)). We compared several state-of-the-art algorithms, A2C (Advantage Actor–Critic), A3C (Asynchronous Advantage Actor–Critic), IMPALA (Importance Weighted Actor-Learner Architecture), and PPO, that can handle discrete action space and support action masking model simultaneously. While A2C leverages an actor–critic framework for efficient learning, A3C extends this approach with asynchronous learning for faster convergence. On the other hand, IMPALA excels in distributed settings, employing importance weights to maintain scalability and efficiency across multiple machines. As shown in Fig. 3a, PPO obtained the highest training episodic mean rewards, and it outperformed other algorithms in terms of reward convergence and stability. Ultimately, this signifies that PPO achieves a balance between simplicity and efficacy, offering stable and reliable performance in this scheduling environment. Additionally, considering the substantial action space inherent in the scheduling problem, we determined PPO to be the most appropriate algorithm to effectively address such a challenge. We set the discounting factor γ to 1, and employed HyperOptSearch for tuning hyperparameters, such as the learning rate and clipping parameter. Learning curves for 10 different sets of hyperparameters within a specific range are depicted in Fig. 3b. Among 10 different cases, case 3 set was able to obtain higher maximum episodic rewards and was selected as summarized in Table 1.

Table 1
Reinforcement learning PPO algorithm hyperparameters.

Hyperparameter	
Learning rate	2.314×10^{-6}
Discounting factor (γ)	1
Number of epochs to execute per train batch	25
Hidden layer	256, 256
Activation function	*tanh*
Clip parameter (ϵ)	0.3
ICM learning rate	1.243×10^{-5}
ICM weight for the forward loss	0.2

$$\min_{\theta} L^{VF}(\theta) = (V_\theta(s_t) - V_t^{\text{targ}})^2 \tag{12}$$

3.2. State, action and reward function

We formulated the single-stage scheduling problem as a sequential decision process, in which the decisions are made in time series to achieve a specific system objective. In reinforcement learning, an agent interacts with an environment by observing states, making actions based on these states, and receiving rewards, thus learning a policy. We adopted single-agent reinforcement learning to model the decision-making process of scheduling problems (Fig. 2a). The goal is to train an agent capable of scheduling, necessitating a particular setup. The state of the scheduling environment (Fig. 2b) consists primarily of two elements, as depicted in Eq. (13) - the allocation status of jobs on machines and the starting timing of each job (Fig. 2c). The variable j_i representing job allocation on a machine is -1 if job i has not been assigned to any machines yet, while it displays the index of the machine if it has been assigned already. Additionally, the variable ts_i denotes the starting time of each job as an integer. This setup allows the agent to discern the previous schedule configuration leading up to the decision-making point at time t merely by observing these states.

$$s_t = \{j_1, j_2, j_3, \ldots, j_i, ts_1, ts_2, ts_3, \ldots, ts_i\} \tag{13}$$

With this state representation, the agent can ascertain which machines are occupied and which jobs are already assigned. In this scheduling environment, at each time t, the agent chooses which job to allocate to each machine. Each machine was considered independent,

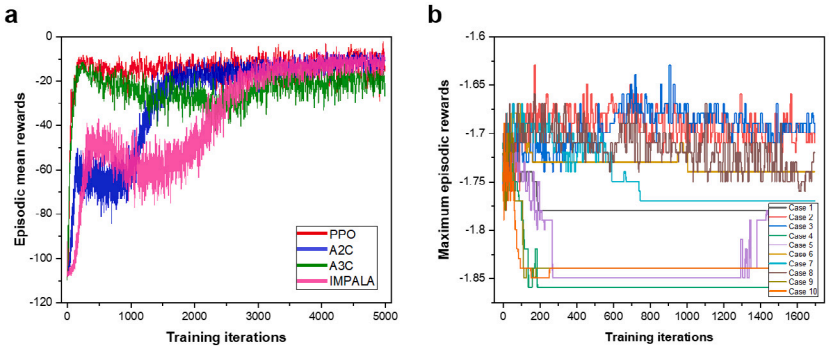

Fig. 3. Training curve comparison results for (a) reinforcement learning model algorithm selection and (b) hyperparameter tuning.

and the single agent takes actions a_t to select one of i jobs at every time step t for every machine j. Using the state represented by Eq. (14), the agent selects one job among those not yet chosen or an idle action, representing not selecting any job. Therefore, the action, as indicated in the equation, is represented by the variable i_j, denoting the index of the selected job. Given that idle action is represented as 0 and included in the action space, the action is defined as a multi-discrete action space to choose an integer among the total number of jobs $i+1$ for each of j machines. By iteratively selecting actions at each time step, this sequence of processes ultimately enables the determination of the complete schedule for the entire machine.

$$a_t = \{i_1, i_2, i_3, \ldots, i_j\} \tag{14}$$

Designing a proper reward function is a crucial element in enabling a reinforcement learning agent to learn the optimal policy efficiently. In our model, we enhanced the scheduling performance by distinguishing between the reward assigned at each time step and the one assigned at the end of the scheduling horizon. The reward functions are shown in Eqs. (15) and (16).

Eq. (15) is formulated to guide the direction of the agent's decision-making at each time step. Given that the objective of this scheduling problem is to minimize assignment costs, favoring the placement of jobs with lower costs is advantageous. Therefore, the reward assigned at each step is configured by taking the negative value of the selected job's cost. This setting prompts the agent to choose jobs with smaller costs to maximize cumulative rewards.

Additionally, when training the model, it is important to limit the selection of impractical actions to ensure the model's effectiveness in reducing costs and improving efficiency and reliability. In this environment, the reinforcement learning agent independently selects jobs for each machine, thereby rendering the decision-making for each machine independent and devoid of mutual influence. In scheduling problems, it is infeasible for the same job to be allocated to different machines. However, verifying whether the agent has placed the same job on different machines for each machine is impossible beforehand and only feasible after decisions are made. Hence, to train the agent against such actions, we applied penalties larger than the rewards to discourage these selections. Therefore, in cases where job i is assigned to distinct machines j and j' simultaneously – an impossible scenario – a penalty value PEN_{step} is imposed.

$$R_t(s,a) = \begin{cases} -C_{ij} \times 10^{-2} & \forall t: X_{ij} = 1 \quad \text{if } X_{ij} + X_{ij'} \neq 2 \\ PEN_{step} & \text{if } X_{ij} + X_{ij'} = 2 \end{cases} \tag{15}$$

$$R_T(s,a) = \begin{cases} \frac{PEN_{end}}{n_T} & \text{if } n_T \neq J \\ 0 & \text{if } n_T = J \end{cases} \tag{16}$$

In order to achieve the objective of minimizing the cost of scheduling, we assigned rewards $R_t(s,a)$ at each step and $R_T(s,a)$ at the end of the scheduling horizon. In Eq. (16), n_T denotes the total number of jobs that have been assigned up to the end of the time horizon T. In the reward function $R_T(s,a)$, we encouraged the agent to find feasible schedules by setting a reward of 0 at the end of an episode if a feasible schedule was found, and PEN_{end}/n_T with a large negative value of PEN_{end}, if an infeasible solution was generated. This was done to train the agent to prioritize finding feasible schedules. We balanced the trade-off between selecting idle actions and job assignment actions by scaling the rewards to ensure comprehensive training where all jobs are ultimately selected. Specifically, idle actions resulted in an immediate reward of 0 and a large negative R_T at the end, while job assignment actions yielded immediate negative rewards but a reward of 0 at the end.

3.3. Training methods

During the training phase of the model, learning was conducted under fixed problem size (e.g. the number of machines and jobs). In order to enhance the quality of solutions as the complexity escalates with the problem size, two different methods were employed during the training phase: (i) action masking and (ii) intrinsic rewards.

In process scheduling, conducting a viable schedule that can be completed within the scheduled time without resource over-utilization is important. Therefore, in this study, we imposed feasibility constraints by integrating action masking (Fig. 2e) into the learning environment to ensure that our reinforcement learning model generates executable schedules. The feasibility constraints can be encoded as a set of rules that can be controlled before taking actions to prevent the agent from selecting infeasible actions.

Our approach involved three types of constraints on the action space: constraints on the executable time of each job, constraints on available resources, and constraints on the duplicate selection of jobs. These constraints were applied before the start of training and at each step of the learning process to restrict the action space. By limiting the available actions in this way, we could significantly reduce the action space, which led to faster, more effective learning.

In the process of training a reinforcement learning agent, action policy improves and updates based on rewards. However, sparse rewards often lack sufficient information to accurately assess the quality of states (Bellemare et al., 2016). Moreover, manually crafted extrinsic rewards can greatly impact the model's performance in large and complex systems, as performance heavily depends on how the reward function is designed. Additionally, in the scheduling environment, it is difficult to capture the quality of the full schedule based on the individual state at each time point. This difficulty arises from the fact that learning the value of each state necessitates observing the effects of each state on the overall outcome (i.e., the final schedule), which can only be obtained at the end of time horizon T. To overcome these challenges, we leveraged an intrinsic curiosity module (ICM), as shown in Fig. 2e to enhance the agent's exploration during training.

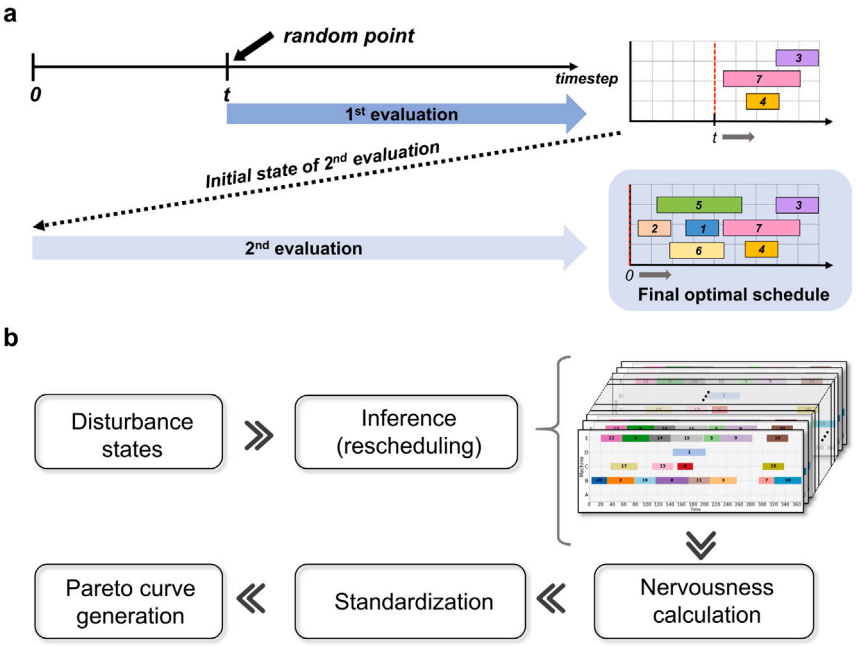

Fig. 4. Inference method: (a) the back-stepping method, which uses two sequential inference steps to provide an effective mechanism for schedule execution, and (b) the rescheduling process for generating the *nervousness versus cost* Pareto curve.

The forward model of the ICM predicts the next state, s_{t+1}, based on the current state, s_t, and action a_t (Pathak et al., 2017). By comparing the predicted state with the actual state, the intrinsic reward is assigned to reflect the difference between them as shown in Eq. (17). Here, η refers to the scaling factor and ϕ represents the feature vector of the state. The predicted feature vector of the next state ($\hat{\phi}(s_{t+1})$) is calculated from the forward model that takes the feature vector of the current state ($\hat{\phi}(s_t)$) and a_t as inputs with parameter θ_F (Eq. (18)). Higher intrinsic rewards are given when there is a larger difference between the predicted and actual states. In other words, by providing intrinsic rewards when the agent encounters outcomes that deviate from its expectations, we can reinforce exploration, encouraging the discovery of new possibilities.

$$R_t^i = \frac{\eta}{2}\|\hat{\phi}(s_{t+1}) - \phi(s_{t+1})\|_2^2 \tag{17}$$

$$\hat{\phi}(s_{t+1}) = f(\phi(s_t), a_t; \theta_F) \tag{18}$$

3.4. Inference methods

3.4.1. Back-stepping method

The conventional inference method involves generating the schedule sequentially from time step 0 to a given step using the trained model when extracting schedules in actual implementation. Traditionally, once the model is trained, inference is employed to generate schedules sequentially from 0 to a specified step. To enhance the quality and variety of the schedules obtained this way, we introduce a novel approach called the back-stepping method.

Fig. 4a illustrates the operating principle of the back-stepping method. For each machine, scheduling is first performed from a random point t to the end of the time horizon T. Next, based on the schedule generated in the first step, the state is updated in the environment to avoid overlapping. The second step then involves scheduling for all machines starting from the beginning of the horizon. With the utilization of this technique, it is expected that the agent will be able to explore the latter part of the horizon first, rather than forming and completing a schedule quickly. As a result, it is anticipated that the agent can potentially find a higher quality or closer-to-optimal solution.

3.4.2. Schedule nervousness calculation

Real-world industrial environments are dynamic and uncertain, leading to frequent rescheduling processes. Frequent rescheduling and repeated revisions resulting in significant deviations can lead to poor quality schedules and increased complexity in implementation. Therefore, a metric called "schedule nervousness" is used to evaluate the rescheduled results (Atadeniz and Sridharan, 2020). Schedule nervousness is quantified by comparing the revised schedule against the original.

$$SN = P^A + P^R + P^T + P^U \tag{19}$$

$$P^A = \sum_i \sum_{j\in J_i} \sum_{t\in T^2} g(t) Y_{ijt} \tag{20}$$

$$P^R = \sum_i \sum_{j\in J_i} \sum_{t\in T^2 : \Omega^X_{ijt}=1} g(t)(1 - Z_{ijt}) \tag{21}$$

$$P^T = \sum_i \sum_{j,j'\in J_i} \sum_{t,t'\in T\setminus T^1 : t\neq t'} g(t)|t - t'|U_{ijj'tt'} \tag{22}$$

$$P^U = \sum_i \sum_{j,j'\in J_i : J\neq j'} \sum_{t,t'\in T\setminus T^1} g(t) U_{ijj'tt'} \tag{23}$$

$$g(t) = \frac{\log(|T|)}{\log(|T|) - log(|T^1|)} - \frac{1}{\log(|T|) - log|T^1|} log(t) \tag{24}$$

In this study, schedule nervousness (SN) was quantified based on the method originally proposed by Lee et al. (2020b) with some revisions. The calculation was performed by considering four elements – addition, removal, time-shifting, and reassignment – which were utilized to compare the differences between the original and the revised schedules using Eq. (19). When a new job is added to the revised schedule, it is considered an addition, and when an existing job is removed, it is considered a removal. When a job is relocated from one unit to another, it is considered a reassignment. Lastly, an alteration to the start time of a job is considered time-shifting. The variables employed in the equation, Y, Z, and U, are binary variables. Specifically, at a given time t, Y takes the value of 1 if a new job i is assigned to a unit j. For a specific job i, Z is set to 1 if it is scheduled on unit j in both the base schedule and the revised schedule at time t. Additionally, U is assigned the value of 1 if job i undergoes a reassignment from its

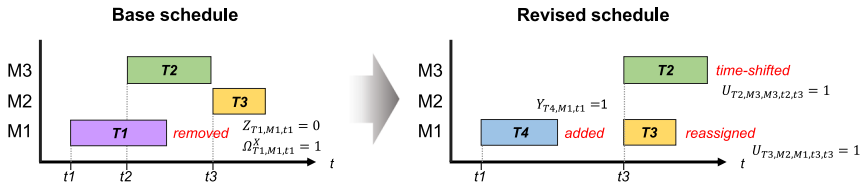

Fig. 5. Visualization of the behavior of the binary variables Y_{ijt}, Z_{ijt}, and $U_{ijj't't'}$ in response to job addition, removal, reassignment, and time-shift.

original placement on unit j to a new placement on unit j', along with a time shift from t to t' in the revised schedule. To illustrate, Fig. 5 depicts how Y, Z, and U respond to certain fluctuations. Variations closer to the rescheduling point (T^1) are believed to exert more significant negative impacts on nervousness, so a logarithmic decay function, $g(t)$, was utilized to assign higher weights to such close proximity variations during the calculation process.

Once the nervousness for the new schedule alternatives had been calculated, the calculated values for each element were standardized into a common distribution, effectively mitigating the influence of their respective magnitudes. Subsequently, the SN values and total assignment costs for each rescheduled alternative were plotted as points on a Pareto curve, empowering operators to swiftly identify the rescheduling options that best meet their system's requirements.

3.5. Implementation settings

During the training process of the model, we set the PEN_{step} value to -110 and the PEN_{end} value to -100. This deliberate choice reflects our intent to instruct the model to recognize situations where a single job is assigned to two machines concurrently as failures, as opposed to situations where all jobs are allocated successfully. We used an open-source library Ray (Moritz et al., 2018) to construct a reinforcement learning environment and communicate with the learning algorithms. The reinforcement learning model was trained using Ray 1.6.0 on a system featuring an Intel(R) Xeon Gold 6226R processor operating at 2.90 GHz and running on CentOS Linux 7.

4. Result & discussion

In this section, we assess the performance of the reinforcement learning scheduling model and verify the effectiveness of various methods employed during the training and inference stages, as described in Section 3. The benchmark problem used for model application is consistent with the one outlined in Section 2. Furthermore, to demonstrate the validity of the reinforcement learning model results, they were compared with the outcomes obtained from mathematical programming models. Finally, we conducted a case study on rescheduling to examine how the reinforcement learning model performs when sudden disruptions occur in the scheduling environment. We investigated the rescheduling process undertaken by the model in response to such changes. We performed mathematical optimization using the GAMS (General Algebraic Modeling System), a mathematical modeling and programming tool, and the CPLEX (Nickel et al., 2020) solver was used for solving the optimization problems. The mathematical programming solution was executed using solver the CPLEX 20.1.0.1 through GAMS 36.2.0 on a system with Intel(R) i7-11700K processors running at 3.60 GHz and 32 GB of RAM.

4.1. Fundamental scheduling capability demonstration

Starting from the simplest form of the reinforcement learning model, we progressively incorporated the methods described in Section 3 into the training and inference processes. The results, corresponding to the inference of problem instance 4-2, in which a total of 15 jobs are to be allocated to 5 machines—are presented in Fig. 6. The inference was conducted for 500 episodes and the minimum assignment cost among the results is displayed. The first three cases were obtained without the back-stepping inference method, using a conventional inference approach instead, and decisions were made sequentially from the beginning of the time horizon to the end.

The dashed red line in Fig. 6a corresponds to the optimal minimum assignment cost obtained through the application of the mathematical programming approach. The base model, i.e., the model without any modifications, had difficulty finding feasible solutions. This can be observed from the training curve of the model shown in Fig. 6c, where it is apparent that the episodic rewards become trapped in local optima, resulting in infeasibility penalties. This indicates that the base model finds it difficult to allocate all fifteen jobs feasibly. However, the introduction of action masking during the training phase brought about a notable reduction in the action space, thereby enhancing the efficiency of the learning process. As a result, the model was able to obtain feasible solutions more reliably and in a more effective manner. Examining the maximum episodic reward values before and after action masking was added, a distinct difference between the cases with and without action masking is evident (Fig. 6c). The maximum episodic reward steadily increases as the training progresses, indicating that the model was capable of finding feasible solutions during the training steps. However, as with previous observations, the model convergence still deviates significantly from the mathematical programming optimal values. To address these challenges, the integration of the ICM technique proved to be effective, increasing the model exploration during the training process (Fig. 6c). This allowed the model to explore a wider range of actions within the same training time, leading to observable improvements in policy, and ultimately an enhanced performance in which higher quality solutions were more readily found. Lastly, by implementing the back-stepping method, the model performed an initial inference from a random starting point. This allowed the later time horizons to be explored, and the model approximated minimum total assignment costs closer to the optimal value. Based on these findings, reinforcement learning models can enhance performance and improve training efficiency by integrating various techniques not only during the training phase but also during the inference phase. These results highlight the potential of reinforcement learning models to efficiently compute outcomes similar to those of existing mathematical programming optimization methods.

To thoroughly analyze the improvements brought about by the back-stepping method, we developed a case study comparing it with the conventional method using the same problem instances as above. Identical training models and checkpoints were used for the conventional inference and the back-stepping methods. As depicted on the right sides of the Gantt chart in Fig. 6b, the results obtained from the model trained and evaluated using the conventional method reveal a learning behavior tendency in which the agent did not consider waiting, instead making immediate job selections without adequately considering the possibility of superior alternatives. In other words, the model failed to properly identify cases in which it would be more optimal to wait for a slightly longer period before assigning jobs immediately. As a solution, we introduced the back-stepping inference to forcibly observe the entire time horizon during the inference process. Consequently, as illustrated in Fig. 6b, the back-stepping inference allowed the agent to explore a wider range of job-machine assignment possibilities by initially considering the jobs further into the future. This facilitated the discovery of higher quality, closer-to-optimal solutions.

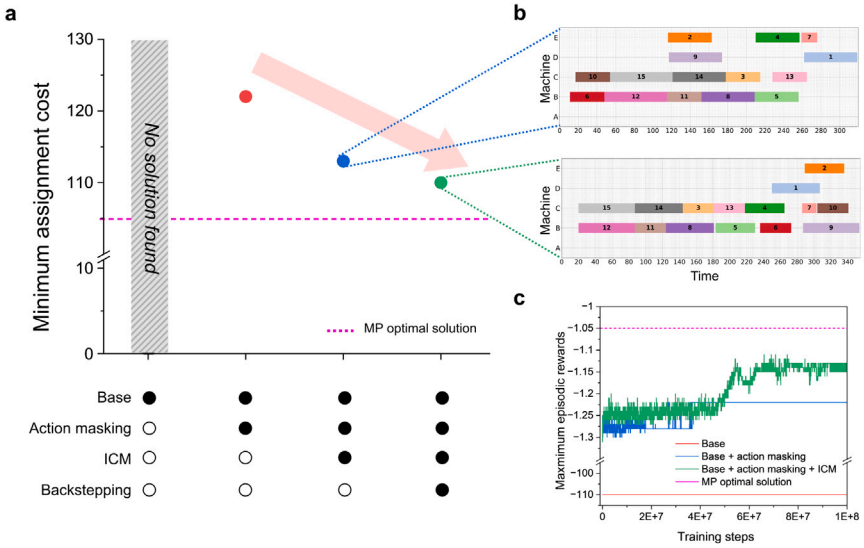

Fig. 6. The performance comparison between the mathematical programming solution (MP) and the RL models during the training and inference processes based on instance 4-2 with the same checkpoint (18820): (a) The minimum assignment cost computed by each model, showing the variations resulting from the addition of action masking and an intrinsic curiosity module (ICM); (b) a comparison between the back-stepping method and the conventional inference approach, illustrated using Gantt charts to showcase the resulting assignment schedules; and (c) the training curves of different methods, depicting the maximum episodic rewards attained during training.

Table 2
Overall performance of the RL model with various integrated methods as compared to the MP method.

Instance	MP CPU time (s)	MP equations	MP single variables	Objective value ($)	RL ($)	Solution quality (%)
1-1	0.196	999	873	26	26	100
1-2	0.204	954	1048	21	21	100
2-1	0.859	2798	3721	60	60	100
2-2	0.855	2708	4862	46	46	100
3-1	5.625	5713	11 521	104	104	100
3-2	2.901	5713	13 487	85	88	96.6
4-1	9.734	7416	18 896	116	123	94.3
4-2	5.907	7416	22 516	105	110	95.5
5-1	14.168	9521	26 911	159	166	95.8
5-2	9.568	9521	31 441	144	154	93.5

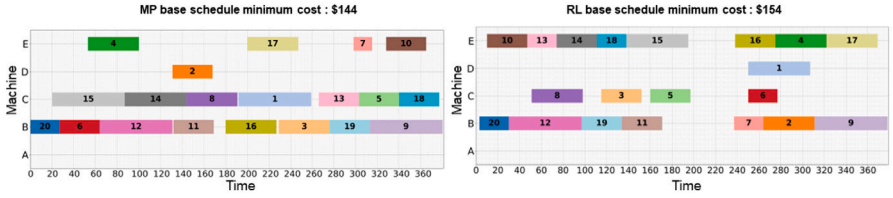

Fig. 7. Gantt chart showing solutions of problem 5-2 by using mathematical programming (MP) and reinforcement learning (RL) methods.

4.2. Model performance comparison on different problem scales

In Table 2, we compared the objective values obtained when solving the benchmark problems using the mathematical programming with the results of the RL model. The RL model employed the complete model, incorporating all the techniques used in the study, for training and inference on each of the 10 problems. The execution time for each method is also included in Table 2 to provide information about the computational costs associated with the mathematical programming approach. For each problem, the RL model underwent 500 episodes of rollout, employing a multiprocessing approach with 10 parallel processors.

As evidenced by the increasing CPU time used by the mathematical programming model as the scale of the instances grows, it can be inferred that the complexity of the problems also increases. For relatively small-scale problems (instances 1-1 to 3-1), the RL model found the optimal solutions. This indicates that RL models can swiftly and effectively attaining optimal outcomes aligned with specific goals. Importantly, this approach avoids the necessity of complex constraint formulations or extensive parameter adjustments. Instead, by appropriately configuring the environment to match the scheduling environment and undergoing a single training process, the RL model could extract the desired optimization results efficiently.

As the size of the problems increased (from instance 3-2 to 5-2), the RL model consistently maintained a solution quality above 90%, approaching values close to the optimal solutions. When considering the global optimal solution obtained through mathematical optimization methods as a benchmark at 100%, achieving a solution quality of approximately 90% is generally deemed sufficient (Kang et al., 2023; Liu et al., 2020). This acceptance of a slightly suboptimal outcome arises from the inherent intricacies of mathematical optimization, which necessitates the formulation of numerous complex equations and constraints. Among ten different problems, we compared the scheduling results of the most complex and large-scale problem, 5-2, using both mathematical programming and RL approaches. Through the comparison of Gantt charts as shown in Fig. 7, we were able

to demonstrate that RL autonomously learned an efficient scheduling strategy. For instance, the decision not to allocate tasks in machine A was consistent between mathematical programming and RL. This alignment can be justified by examining the cost data of problem 5-2, which indicates that allocating task in machine A typically results in higher costs on average, making it preferable not to schedule it. Similarly, in comparison to other machines, RL allocated fewer tasks to machine D, resembling an optimal solution. Additionally, some tasks were allocated almost identically in timing, and out of the 20 tasks, 10 exhibited machine-task allocations identical to those of MP. These diverse observations underscore RL capability to make reliable decisions autonomously, approaching near-optimality without the need for explicit mathematical equations or constraints, solely relying on a reward function composed of assignment cost terms.

While the RL model finds slightly suboptimal solutions, it exhibited a distinct advantage in computation time. Unlike in mathematical programming, where the computation time scales exponentially with problem size, the computation time of the RL remained relatively constant regardless of the problem size, as shown in Fig. 8a. The process of extracting a singular outcome from the RL model was accomplished within the temporal scope of one second across all problem sizes, which shows the scalability of the RL model. Interestingly, the scale of the problem and the accuracy of the RL model seem to be unrelated, indicating that the RL model performance is not hindered by the problem's complexity.

Another advantage of the RL model is that the inference process can be readily parallelized. To demonstrate this, we conducted an experiment where inference was done on 10 parallel cores. Fig. 8b shows a comparison of the execution times required to find a single feasible solution in the RL inference process with those of the mathematical programming method. The execution time of the RL model refers to the duration it takes for the model to perform only the inference phase. This is because when conducting iterative scheduling for the same process, the previously trained model can be reused without the need for a repeated learning phase. The total execution time of the RL model was divided by the number of feasible solutions obtained for each CPU core count. Each core count underwent 50 episodes of rollout, and we observed a reduction in CPU time per feasible solution resulting from parallelization. This is due to the fact that, with an increase in the number of CPU cores, the number of solutions obtained as results also increases. However, due to the parallel nature of the multiprocessing inference process, the overall execution time remained relatively constant. Consequently, the CPU time required for each individual data point decreased, owing to the greater efficiency afforded the model by the increased number of CPU cores. Moreover, we noticed that with a higher number of cores operating in parallel, the diversity of extracted solutions increased, leading to the discovery of solutions with lower total assignment costs. Ultimately, parallelizing the computations of the RL model amplified the synergy of its inherent scalability and the advantages derived from multiple cores. This confluence augmented the model capacity to discover high-quality solutions within a short temporal scope.

These findings highlight the efficacy of reinforcement learning models in addressing complex problems, showcasing their potential to overcome challenges associated with conventional optimization techniques.

4.3. Real-time rescheduling considering nervousness

In an industrial setting, disruption events such as rush orders, order cancellations, and unit breakdowns often occur. As a result, the implemented schedule can be rendered far from optimal or even infeasible. In such situations, the need for rapid and efficient rescheduling becomes paramount to ensure the optimal operation of the facilities. In this section, we aimed to demonstrate the flexibility of the RL model in handling such situations by manipulating the environment to reflect various scenarios of abrupt change. By subjecting the RL model to these dynamic environments, we assessed its ability to adapt and generate new schedules in response to unforeseen events. The scenarios included sudden changes in resource availability, job delays, order cancellations, and yield losses.

By employing an RL model in such scenarios, the rescheduling optimization problem can be decomposed into two steps: learning and inference. Through a lengthy training process in a static environment, the model can learn an optimal policy, which is then used in the subsequent high-speed inference stage to make optimal decisions even when the environment undergoes changes. This flexibility stems from the stochastic nature of the policy distribution of PPO algorithm, which the RL agent updates during the learning process. PPO algorithm updates policy function based on a stochastic distribution to explore a greater number of state–action pairs and to exhibit robustness in response to environmental changes or noise. This stochasticity allows the model to capture the inherent uncertainty in the dynamics of the scheduling problem, enabling it to generate a diverse range of solutions. Consequently, during inference, the trained model can effectively handle unexpected variations in the environment and make optimal decisions using the learned policy.

We conducted rescheduling to derive results concerning job delays. By simulating a delay in job number 12's completion, we emulated real-world scenarios in which unexpected delays can disrupt the original schedule. This scenario assumes a situation where job 12 completes 20 time steps later than its originally scheduled completion time, leading to a need for rescheduling at that particular point. We introduced a setup time of 10 time points to start from the rescheduling point, representing an unalterable duration necessary for rescheduling. Subsequently, the schedule is rearranged and adjusted from that point onward. The setup time can be dynamically adjusted to accommodate varying circumstances. The results for this specific scenario are illustrated in Fig. 9.

In the rescheduling scenarios, the altered state becomes the input, and the model effectively triggered the rescheduling mechanism. It considered the remaining jobs and the availability of resources, and generated an updated and optimized schedule. We conducted 500 episodes using a multiprocessing approach with 10 parallel processors, resulting in the generation of 384 schedule alternatives within 60 s through the inference process. This underscores the RL model's capacity to accommodate a broad range of scheduling scenarios under uncertainty. The model's ability to discover suboptimal values while concurrently proposing flexible schedules tailored to the environment serves as the foundation for extending its adaptability to nervousness. Additionally, during the inference phase, we assessed the RL model's ability to find high quality solutions (i.e., revised schedules with good objective function values), and solutions with low schedule nervousness by plotting a graph depicting nervousness versus total assignment cost for the obtained set of 384 schedules (Fig. 9).

Upon examining the bottom-left region of the graph, we observed that the points form a Pareto curve. This indicates a trade-off relationship between nervousness and scheduling objectives. Along the Pareto front, it is evident that as the schedule cost objective approaches the optimal value, nervousness increases, while moving farther from the optimal cost objective results in reduced nervousness. These results imply that, during rescheduling, achieving the optimal value often necessitates a substantial number of schedule variations. Conversely, obtaining solutions farther from the optimal value can help minimize schedule fluctuations. To further analyze the schedules along the Pareto curve, we presented them as Gantt charts on the right side of Fig. 9(charts a–d). The schedules towards the top have better economics, while schedules towards the bottom have lower schedule nervousness. Rescheduling in response to changing circumstances led to an increase in cost compared to the initial optimal cost, ranging from 1.3% to a maximum of 3.90%. However, considering the need for rapid schedule recalculations in real-world scenarios, these values can be

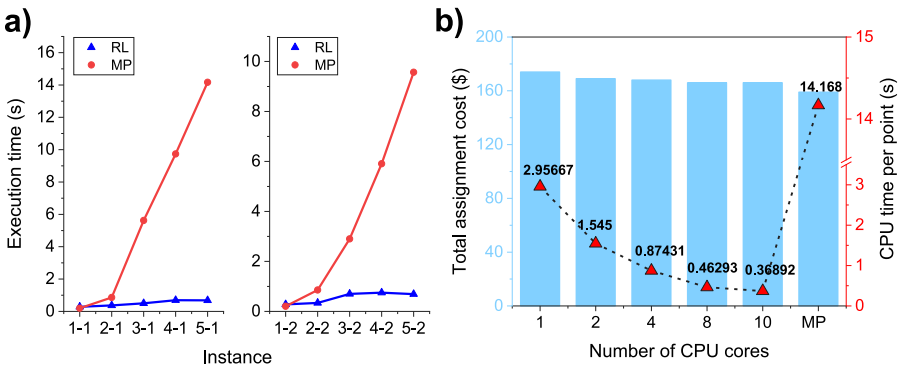

Fig. 8. (a) Plots showing the computational time requirements for extracting a single outcome using reinforcement learning (RL) and mathematical programming (MP) models. (b) Graph showing the CPU execution time(s) (red triangles) and total assignment cost (blue bars) for a single feasible solution point obtained in instance 5-1 using varying CPU core counts during inference.

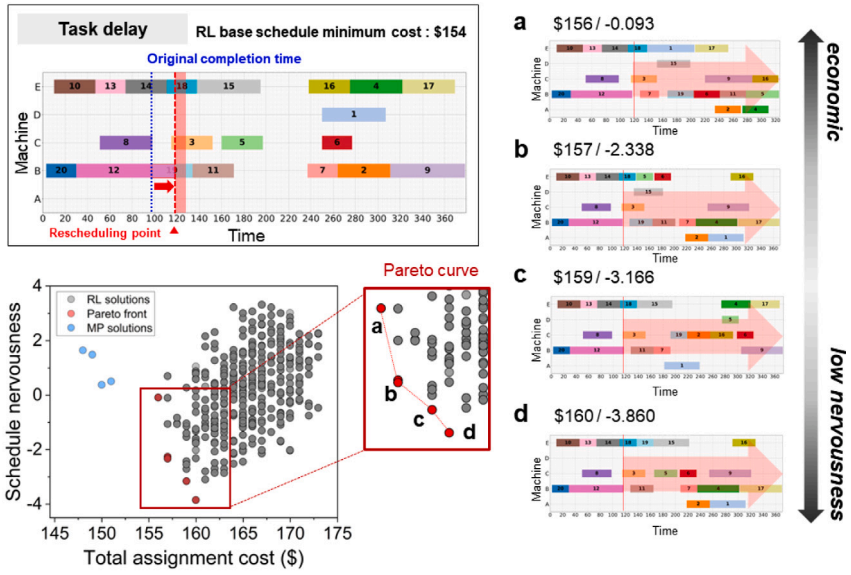

Fig. 9. Rescheduling results for the job delay scenario, showing the original schedule (above) and the imposed delay (blue dotted to red dashed line), the four alternative schedules along the Pareto front (schedules a–d; right) generated by the reinforcement learning (RL) model, and a scatterplot of all 384 alternative schedules (bottom) with the Pareto curve illustrating the trade-off between total assignment cost and schedule nervousness.

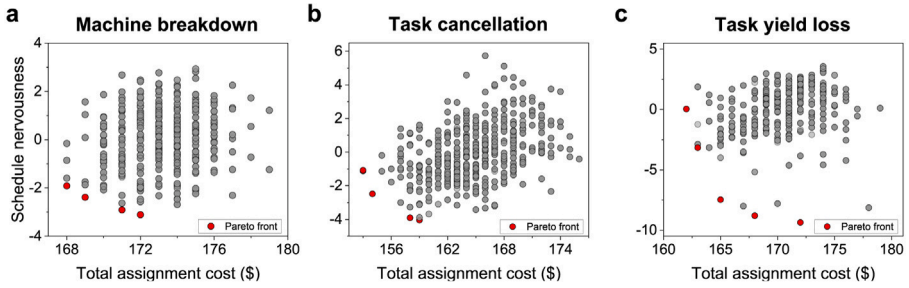

Fig. 10. Pareto curves representing the results of rescheduling under three different common disruption types that can occur in real-world environments including machine breakdown (a), job cancellation (b), and job yield loss scenarios (c).

regarded as promising. By exploiting the diverse schedule solutions obtainable through the RL model, operators will be able to make efficient decisions at the Pareto front aligning with the specific conditions and circumstances of the process, facilitating informed decision-making.

Reinforcement learning not only offered the advantage of providing diverse results through its stochastic policy but also enabled much faster execution compared to mathematical programming. For a direct comparison, we conducted rescheduling using the mathematical programming model, resulting in a total of four alternative schedules as depicted in Fig. 9. The findings indicate that the mathematical programming model falls short on two fronts. Firstly, while it can generally generate schedules with lower assignment costs, it struggles to systematically produce alternative schedules with varied schedule nervousness, an important factor in rescheduling scenarios. Secondly, although mathematical programming models can indeed yield diverse alternative schedules through iterative solution and cut generation, the overall computational burden to achieve this is substantial. Specifically, it took 92 s to compute the four alternative solutions shown in Fig. 9,

compared to 60 s required by the RL model to generate 384 solutions. As the speed of finding the alternative schedule is a crucial aspect in these scenarios, RL model holds a distinct advantage in this regard as well. By promptly presenting schedulers with a selection of alternatives with varying quality and nervousness, the RL model proves to be exceptionally valuable in guiding operations through disruption events. It showcases the potential of RL models in overcoming the limitations of traditional scheduling approaches and providing insights into the effective management of scheduling jobs in dynamic and unpredictable environments.

In addition to the job delay scenario, we extended our analysis to encompass various real-world industrial situations, including machine breakdowns, job cancellations, and job yield loss. By employing RL for rescheduling, we derived alternative schedules, and the resulting Pareto curves are depicted in Fig. 10. Machine breakdown refers to the sudden unavailability of a specific unit after a certain time point; job cancellation involves the abrupt cancellation of a particular job, necessitating rescheduling; and job yield loss represents instances where jobs fail to achieve the desired yield during execution, requiring its reassignment. Based on the plotted Pareto results, we can infer that the RL agent demonstrates flexibility in decision-making, even when confronted with unpredictable and previously unseen states, by leveraging its trained policy distribution.

5. Conclusions

Real-time rescheduling systems, which allow for avoiding rapid changes to existing production schedules, are a critical technology for advanced manufacturing systems in the Industry 4.0 era. We proposed an online rescheduling methodology in which the fully automated workflow integrates an RL methodology that considers cost and nervousness to generate Pareto curves, employing artificial intelligence in large-scale scheduling problems. The proposed RL scheduling system learns using job-assignment scenarios in a given scheduling environment through the PPO algorithm to make optimal decisions in the current state. During the inference step, the novel back-stepping method allows the scheduler to optimally allocate jobs to cover the entire time horizon, dramatically increasing the reward without additional training. We quantified the situations requiring rescheduling – job addition, removal, reassignment, and time-shift – and successfully performed Pareto curve generation, simultaneously optimizing assignment cost and schedule nervousness with significantly better computational efficiency than mathematical programming in the inference phase. We solved 10 instances known to be difficult in previous studies and have shown results ranging from 93.5%–100% of the global optimal assignment cost in these instances. For large-scale scheduling problems, it shows scalable execution time for inference compared to mathematical programming models, and based on this fast inferencing, it enables flexible online rescheduling with effective Pareto curve generation. Altogether, our results demonstrate that an RL rescheduling methodology that considers nervousness meets the demands of online scheduling systems in an uncertain real world and unlocks a new paradigm of multi-objective rescheduling.

Code availability

The codes that support this study's findings are available in the Github repository (https://github.com/tnals9983/RLScheduling).

CRediT authorship contribution statement

Sumin Hwangbo: Writing – review & editing, Writing – original draft, Visualization, Validation, Methodology, Conceptualization. **J. Jay Liu:** Supervision, Funding acquisition, Conceptualization. **Jun-Hyung Ryu:** Supervision, Methodology, Conceptualization. **Ho Jae Lee:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Conceptualization. **Jonggeol Na:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Funding acquisition, Conceptualization.

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.

Data availability

Github link was shared in the manuscript.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT, Republic of Korea (2021R1C1C1012031 and 2021R1A4A3025742).

Appendix. Supplementary data

The single-stage scheduling problem data (Harjunkoski and Grossmann, 2002) used in this study is given in Tables A.1 to A.5.

Table A.1
Problem 1 (J = 2, I = 3).

Job number	Instance 1-1				Instance 1-2			
	Dates		Processing time/cost		Dates		Processing time/cost	
	Release	Due	M1	M2	Release	Due	M1	M2
1	20	169	103/10	143/6	20	160	57/10	77/6
2	30	169	63/8	83/5	30	130	37/8	47/5
3	40	219	113/12	163/7	40	210	57/12	77/7

Table A.2
Problem 2 (J = 3, I = 7).

Job number	Instance 2-1					Instance 2-2				
	Dates		Processing time/cost			Dates		Processing time/cost		
	Release	Due	M1	M2	M3	Release	Due	M1	M2	M3
1	20	169	103/10	143/6	123/8	20	160	57/10	77/6	67/8
2	30	139	63/8	83/5	73/6	30	130	37/8	47/5	37/6
3	40	219	113/12	163/7	133/10	40	210	27/12	47/7	37/10
4	50	289	63/10	123/6	83/8	50	280	37/10	67/6	47/8

(*continued on next page*)

Table A.2 (*continued*).

Job number	Instance 2-1					Instance 2-2				
	Dates		Processing time/cost			Dates		Processing time/cost		
	Release	Due	M1	M2	M3	Release	Due	M1	M2	M3
5	100	249	103/8	163/5	123/7	100	240	27/8	47/5	37/7
6	10	289	73/12	123/7	103/10	10	280	17/12	37/7	27/10
7	20	239	103/12	83/7	103/10	20	230	17/12	27/7	17/10

Table A.3
Problem 3 (J = 3, I = 12).

Job number	Instance 3-1					Instance 3-2				
	Dates		Processing time/cost			Dates		Processing time/cost		
	Release	Due	M1	M2	M3	Release	Due	M1	M2	M3
1	20	360	103/10	143/6	123/8	20	360	57/10	77/6	67/8
2	30	330	63/8	83/5	73/6	30	330	37/8	47/5	37/6
3	40	310	113/12	163/7	133/10	40	310	27/12	47/7	37/10
4	50	380	63/10	123/6	83/8	50	380	37/10	67/6	47/8
5	100	340	103/8	163/5	123/7	100	340	27/8	47/5	37/7
6	10	380	73/12	123/7	103/10	10	380	17/12	37/7	27/10
7	20	330	103/12	133/10	103/11	20	330	17/12	27/10	17/11
8	40	250	43/9	103/5	83/7	40	250	27/9	57/5	47/7
9	100	380	23/10	43/6	33/8	100	380	47/10	67/6	67/8
10	10	390	73/8	143/5	113/7	10	390	37/8	57/5	27/7
11	50	300	83/15	163/9	123/12	50	300	27/15	37/9	27/12
12	20	200	33/13	63/7	53/10	20	200	27/13	67/7	47/10

Table A.4
Problem 4 (J = 5, I = 15).

Job number	Instance 4-1							Instance 4-2						
	Dates		Processing time/cost					Dates		Processing time/cost				
	Release	Due	M1	M2	M3	M4	M5	Release	Due	M1	M2	M3	M4	M5
1	20	330	103/10	143/6	123/8	113/7	133/9	20	330	57/10	77/6	67/8	57/9	67/9
2	30	340	63/8	83/5	73/6	63/7	73/7	30	340	37/8	47/5	37/6	37/7	47/7
3	40	310	113/12	163/7	133/10	113/11	123/10	40	310	27/12	47/7	37/10	27/11	37/10
4	50	330	63/10	123/6	83/8	73/9	83/8	50	330	37/10	67/6	47/8	37/9	47/8
5	100	340	103/8	163/5	123/6	123/7	133/7	100	340	27/8	47/5	37/6	27/7	27/7
6	10	340	73/12	123/7	103/10	83/11	93/10	10	340	17/12	37/7	27/10	27/11	27/10
7	20	330	103/12	133/10	103/11	113/12	123/11	20	330	17/12	27/10	17/11	17/12	17/11
8	40	250	43/9	103/5	83/7	53/9	63/8	40	250	27/9	57/5	47/7	37/9	37/8
9	100	380	23/10	43/6	33/8	23/9	33/8	100	380	47/10	67/6	67/8	57/9	57/8
10	10	370	73/8	143/5	113/6	83/7	103/6	10	370	27/8	57/5	37/6	27/7	37/6
11	50	300	83/15	163/9	123/12	103/14	113/13	50	300	27/15	37/9	27/12	27/14	27/13
12	20	200	33/13	63/7	53/10	43/12	53/11	20	200	27/13	67/7	47/10	37/12	37/11
13	40	320	43/9	103/5	73/6	53/8	63/7	40	320	17/9	37/5	37/6	27/8	27/7
14	60	200	23/10	43/6	43/8	33/10	33/9	60	200	27/10	57/6	57/8	27/10	37/9
15	20	250	73/8	143/5	133/6	103/7	113/7	20	250	47/8	77/5	67/6	47/7	57/7

Table A.5
Problem 5 (J = 5, I = 20).

Job number	Instance 5-1							Instance 5-2						
	Dates		Processing time/cost					Dates		Processing time/cost				
	Release	Due	M1	M2	M3	M4	M5	Release	Due	M1	M2	M3	M4	M5
1	20	330	103/10	143/6	123/8	113/9	133/9	20	330	57/10	77/6	67/8	57/9	67/9
2	30	340	63/8	83/5	73/6	63/7	73/7	30	340	37/8	47/5	37/6	37/7	47/7
3	40	310	113/12	163/7	133/10	113/11	123/10	40	310	27/12	47/7	37/10	27/11	37/10
4	50	330	63/10	123/6	83/8	73/9	83/8	50	330	37/10	67/6	47/8	37/9	47/8
5	100	340	103/8	163/5	123/6	123/7	133/7	100	340	27/8	47/5	37/6	27/7	27/7
6	10	340	73/12	123/7	103/10	83/11	93/10	10	340	17/12	37/7	27/10	27/11	27/10
7	20	330	103/12	133/10	103/11	113/12	123/11	20	330	17/12	27/10	17/11	17/12	17/11
8	40	250	43/9	103/5	83/7	53/9	63/8	40	250	27/9	57/5	47/7	37/9	123/8
9	100	380	23/10	43/6	33/8	23/9	33/8	100	380	47/10	67/6	67/8	57/9	57/8
10	10	370	73/8	143/5	113/6	83/7	103/6	10	370	27/8	57/5	37/6	27/7	37/6
11	50	300	83/15	163/9	123/12	103/14	113/13	50	300	27/15	37/9	27/12	27/14	27/13
12	20	200	33/13	63/7	53/30	43/12	53/11	20	200	27/13	67/7	47/10	37/12	37/11
13	40	320	43/9	103/5	73/6	53/8	63/7	40	320	17/9	37/5	37/6	27/8	27/7
14	60	200	23/10	43/6	43/8	33/10	33/9	60	200	27/10	57/6	57/8	27/10	37/9
15	20	250	73/8	143/5	133/6	103/7	113/7	20	250	47/8	77/5	67/6	47/7	57/7
16	20	340	33/9	83/5	73/7	53/9	63/8	20	340	27/9	47/5	37/7	27/9	37/8

(*continued on next page*)

Table A.5 (*continued*).

Job number	Instance 5-1							Instance 5-2						
	Dates		Processing time/cost					Dates		Processing time/cost				
	Release	Due	M1	M2	M3	M4	M5	Release	Due	M1	M2	M3	M4	M5
17	30	370	63/10	123/6	103/8	73/9	83/8	30	370	37/10	67/6	47/8	37/9	47/8
18	70	380	23/8	83/5	63/6	133/7	43/6	70	380	27/8	47/5	37/6	27/7	27/6
19	60	320	43/15	73/9	63/12	53/14	53/13	60	320	17/15	37/9	27/12	27/14	27/13
20	0	300	53/13	73/7	73/10	63/12	63/11	0	300	17/13	27/7	17/10	17/12	17/11

References

Ahmadian, M.M., Salehipour, A., 2021. The just-in-time job-shop scheduling problem with distinct due-dates for operations. J. Heuristics 27 (1), 175–204. http://dx.doi.org/10.1007/s10732-020-09458-6.

Ahmadian, M.M., Salehipour, A., Cheng, T.C.E., 2021. A meta-heuristic to solve the just-in-time job-shop scheduling problem. European J. Oper. Res. 288 (1), 14–29. http://dx.doi.org/10.1016/j.ejor.2020.04.017, URL: https://www.sciencedirect.com/science/article/pii/S0377221720303519.

Atadeniz, S.N., Sridharan, S.V., 2020. Effectiveness of nervousness reduction policies when capacity is constrained. Int. J. Prod. Res. 58 (13), 4121–4137.

Ave, G.D., Alici, M., Harjunkoski, I., Engell, S., An explicit online resource-task network scheduling formulation to avoid scheduling nervousness. 46, 61–66. http://dx.doi.org/10.1016/B978-0-12-818634-3.50011-4, 29th European Symposium on Computer-Aided Process Engineering (ESCAPE), Eindhoven, NETHERLANDS, JUN 16-19, 2019.

Balasubramaniam, S., Syed, M.H., More, N.S., Polepally, V., 2023. Deep learning-based power prediction aware charge scheduling approach in cloud based electric vehicular network. Eng. Appl. Artif. Intell. 121, 105869. http://dx.doi.org/10.1016/j.engappai.2023.105869, URL: https://www.sciencedirect.com/science/article/pii/S0952197623000532.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., Munos, R., 2016. Unifying count-based exploration and intrinsic motivation. In: Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., Garnett, R. (Eds.), Advances in Neural Information Processing Systems. Vol. 29, Curran Associates, Inc., URL: https://proceedings.neurips.cc/paper_files/paper/2016/file/afda332245e2af431fb7b672a68b659d-Paper.pdf.

Bewoor, L.A., Prakash, V.C., Sapkal, S.U., 2018. Production scheduling optimization in foundry using hybrid Particle Swarm Optimization algorithm. Procedia Manuf. 22, 57–64. http://dx.doi.org/10.1016/j.promfg.2018.03.010, URL: https://www.sciencedirect.com/science/article/pii/S2351978918303044.

Calfa, B.A., Agarwal, A., Grossmann, I.E., Wassick, J.M., 2013. Hybrid bilevel-Lagrangean decomposition scheme for the integration of planning and scheduling of a network of batch plants. Ind. Eng. Chem. Res. 52 (5), 2152–2167. http://dx.doi.org/10.1021/ie302788g.

Castro, P.M., Grossmann, I.E., Zhang, Q., 2018. Expanding scope and computational challenges in process scheduling. Comput. Chem. Eng. 114 (SI), 14–42. http://dx.doi.org/10.1016/j.compchemeng.2018.01.020.

Cha, Y.-J., Mostafavi, A., Benipal, S.S., 2023. DNoiseNet: Deep learning-based feedback active noise control in various noisy environments. Eng. Appl. Artif. Intell. 121, 105971. http://dx.doi.org/10.1016/j.engappai.2023.105971, URL: https://www.sciencedirect.com/science/article/pii/S0952197623001550.

Đurasević, M., Jakobović, D., 2023. Heuristic and metaheuristic methods for the parallel unrelated machines scheduling problem: a survey. Artif. Intell. Rev. 56 (4), 3181–3289. http://dx.doi.org/10.1007/s10462-022-10247-9.

Đurasević, M., Jakobović, D., Knežević, K., 2016. Adaptive scheduling on unrelated machines with genetic programming. Appl. Soft Comput. 48, 419–430. http://dx.doi.org/10.1016/j.asoc.2016.07.025, URL: https://www.sciencedirect.com/science/article/pii/S1568494616303519.

Ferris, M.C., Maravelias, C.T., Sundaramoorthy, A., 2009. Simultaneous batching and scheduling using dynamic decomposition on a grid. Informs J. Comput. 21 (3), 398–410. http://dx.doi.org/10.1287/ijoc.1090.0339, INFORMS Annual Meeting 2006, Pittsburgh, PA, 2006.

Floudas, C.A., Lin, X., 2004. Continuous-time versus discrete-time approaches for scheduling of chemical processes: a review. Comput. Chem. Eng. 28 (11), 2109–2129. http://dx.doi.org/10.1016/j.compchemeng.2004.05.002, URL: https://www.sciencedirect.com/science/article/pii/S0098135404001401.

Georgiadis, G.P., Elekidis, A.P., Georgiadis, M.C., 2019. Optimization-based scheduling for the process industries: from theory to real-life industrial applications. Processes 7 (7), 438.

Giannelos, N., Georgiadis, M., 2002. A simple new continuous-time formulation for short-term scheduling of multipurpose batch processes. Ind. Eng. Chem. Res. 41 (9), 2178–2184. http://dx.doi.org/10.1021/ie010399f.

Gupta, D., Maravelias, C.T., 2019. On the design of online production scheduling algorithms. Comput. Chem. Eng. 129, http://dx.doi.org/10.1016/j.compchemeng.2019.106517.

Gupta, D., Maravelias, C.T., 2020. Framework for studying online production scheduling under endogenous uncertainty. Comput. Chem. Eng. 135, http://dx.doi.org/10.1016/j.compchemeng.2019.106670.

Gupta, D., Maravelias, C.T., Wassick, J.M., 2016. From rescheduling to online scheduling. Chem. Eng. Res. Des. 116, 83–97. http://dx.doi.org/10.1016/j.cherd.2016.10.035.

Harjunkoski, I., Grossmann, I.E., 2002. Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods. Comput. Chem. Eng. 26 (11), 1533–1552.

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.

Hasachoo, N., Masuchun, R., 2015. Factors affecting schedule nervousness in the production operations of airline catering industry. pp. 499–503, IEEE International Conference on Industrial Engineering and Engineering Management IEEM, Singapore, SINGAPORE, DEC 06-09, 2015.

Hasachoo, N., Masuchun, R., 2016. Reducing schedule nervousness in production and operations under non stationary stochastic demand: The case of an airline catering company. pp. 941–945, IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), Bali, INDONESIA, DEC 04-07, 2016.

Ho, C.J., 2002. Evaluating dampening effects of alternative lot-sizing rules to reduce MRP system nervousness. Int. J. Prod. Res. 40, 2633–2652. http://dx.doi.org/10.1080/00207540210134489.

Hubbs, C.D., Li, C., Sahinidis, N.V., Grossmann, I.E., Wassick, J.M., 2020. A deep reinforcement learning approach for chemical production scheduling. Comput. Chem. Eng. 141, http://dx.doi.org/10.1016/j.compchemeng.2020.106982.

Hubert, S., Meintschel, J., Bleidorn, D., Ortmanns, Y., Wallrath, R., 2023. Production scheduling using deep reinforcement learning and discrete event simulation. http://dx.doi.org/10.1002/cite.202200242.

Janak, S., Lin, X., Floudas, C., 2005. Enhanced continuous-time unit-specific event-based formulation for short-term scheduling of multipurpose batch processes: Resource constraints and mixed storage policies (vol 43, pg 2529, 2002). Ind. Eng. Chem. Res. 44 (2), 426. http://dx.doi.org/10.1021/ie048866r.

Kang, D., Kang, D., Hwangbo, S., Niaz, H., Lee, W.B., Liu, J.J., Na, J., 2023. Optimal planning of hybrid energy storage systems using curtailed renewable energy through deep reinforcement learning. Energy 284, 128623. http://dx.doi.org/10.1016/j.energy.2023.128623, URL: https://www.sciencedirect.com/science/article/pii/S0360544223020170.

Kazan, O., Nagi, R., Rump, C.M., 2000. New lot-sizing formulations for less nervous production schedules. Comput. Oper. Res. 27, 1325–1345. http://dx.doi.org/10.1016/S0305-0548(99)00076-3.

Kim, B., Maravelias, C.T., 2022. Supervised machine learning for understanding and improving the computational performance of chemical production scheduling MIP models. Ind. Eng. Chem. Res. 61, 17124–17136. http://dx.doi.org/10.1021/acs.iecr.2c02734.

Koca, E., Yaman, H., Akturk, M.S., 2018. Stochastic lot sizing problem with nervousness considerations. Comput. Oper. Res. 94, 23–37. http://dx.doi.org/10.1016/j.cor.2018.01.021.

Kusoncum, C., Sethanan, K., Pitakaso, R., Hartl, R.F., 2021. Heuristics with novel approaches for cyclical multiple parallel machine scheduling in sugarcane unloading systems. Int. J. Prod. Res. 59 (8), 2479–2497. http://dx.doi.org/10.1080/00207543.2020.1734682.

Law, K.M.Y., Gunasekaran, A., 2010. A comparative study of schedule nervousness among high-tech manufacturers across the Straits. Int. J. Prod. Res. 48, 6015–6036. http://dx.doi.org/10.1080/00207540903246623.

Lee, H., Gupta, D., Maravelias, C.T., 2020a. Systematic generation of alternative production schedules. AIChE J. 66, http://dx.doi.org/10.1002/aic.16926.

Lee, H., Gupta, D., Maravelias, C.T., 2020b. Systematic generation of alternative production schedules. AIChE J. 66 (5), e16926.

Lee, H., Maravelias, C.T., 2018. Combining the advantages of discrete- and continuous-time scheduling models: Part 1. Framework and mathematical formulations. Comput. Chem. Eng. 116 (SI), 176–190. http://dx.doi.org/10.1016/j.compchemeng.2017.12.003.

Lee, H., Maravelias, C.T., 2019. Combining the advantages of discrete- and continuous-time scheduling models: Part 2. systematic methods for determining model parameters. Comput. Chem. Eng. 128, 557–573. http://dx.doi.org/10.1016/j.compchemeng.2018.10.020.

Lee, H., Maravelias, C.T., 2020. Combining the advantages of discrete-and continuous-time scheduling models: Part 3. General algorithm. Comput. Chem. Eng. 139, http://dx.doi.org/10.1016/j.compchemeng.2020.106848.

Li, Y., Carabelli, S., Fadda, E., Manerba, D., Tadei, R., Terzo, O., 2020a. Machine learning and optimization for production rescheduling in Industry 4.0. Int. J. Adv. Manuf. Technol. 110 (9), 2445–2463. http://dx.doi.org/10.1007/s00170-020-05850-5.

Li, Q., Disney, S.M., 2017. Revisiting rescheduling: MRP nervousness and the bullwhip effect. Int. J. Prod. Res. 55, 1992–2012. http://dx.doi.org/10.1080/00207543.2016.1261196.

Li, Y., Fadda, E., Manerba, D., Tadei, R., Terzo, O., 2020b. Reinforcement learning algorithms for online single-machine scheduling. In: 2020 15th Conference on Computer Science and Information Systems. FedCSIS, pp. 277–283. http://dx.doi.org/10.15439/2020F100.

Li, Z., Ierapetritou, M., 2008. Process scheduling under uncertainty: Review and challenges. Comput. Chem. Eng. 32 (4), 715–727. http://dx.doi.org/10.1016/j.compchemeng.2007.03.001, URL: https://www.sciencedirect.com/science/article/pii/S0098135407000580, Festschrift devoted to Rex Reklaitis on his 65th Birthday.

Lin, T.-L., Horng, S.-J., Kao, T.-W., Chen, Y.-H., Run, R.-S., Chen, R.-J., Lai, J.-L., Kuo, I.H., 2010. An efficient job-shop scheduling algorithm based on particle swarm optimization. Expert Syst. Appl. 37 (3), 2629–2636. http://dx.doi.org/10.1016/j.eswa.2009.08.015, URL: https://www.sciencedirect.com/science/article/pii/S0957417409007696.

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. http://dx.doi.org/10.1109/ACCESS.2020.2987820.

Maravelias, C., 2005. Mixed-time representation for state-task network models. Ind. Eng. Chem. Res. 44 (24), 9129–9145. http://dx.doi.org/10.1021/ie0500117.

Maravelias, C.T., 2012. General framework and modeling approach classification for chemical production scheduling. AIChE J. 58 (6), 1812–1828. http://dx.doi.org/10.1002/aic.13801.

Méndez, C.A., Cerdá, J., Grossmann, I.E., Harjunkoski, I., Fahl, M., 2006. State-of-the-art review of optimization methods for short-term scheduling of batch processes. Comput. Chem. Eng. 30 (6), 913–946. http://dx.doi.org/10.1016/j.compchemeng.2006.02.008, URL: https://www.sciencedirect.com/science/article/pii/S0098135406000287.

Merchan, A.F., Lee, H., Maravelias, C.T., 2016. Discrete-time mixed-integer programming models and solution methods for production scheduling in multistage facilities. Comput. Chem. Eng. 94, 387–410. http://dx.doi.org/10.1016/j.compchemeng.2016.04.034.

Merchan, A.F., Velez, S., Maravelias, C.T., 2013. Tightening methods for continuous-time mixed-integer programming models for chemical production scheduling. AIChE J. 59 (12), 4461–4467. http://dx.doi.org/10.1002/aic.14249.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D., 2015. Human-level control through deep reinforcement learning. Nature 518 (7540), 529–533. http://dx.doi.org/10.1038/nature14236.

Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M.I., et al., 2018. Ray: A distributed framework for emerging {AI} applications. In: 13th USENIX Symposium on Operating Systems Design and Implementation. OSDI 18, pp. 561–577.

Nguyen, S., Mei, Y., Zhang, M., 2017. Genetic programming for production scheduling: a survey with a unified framework. Complex Intell. Syst. 3, 41–66. http://dx.doi.org/10.1007/s40747-017-0036-x.

Nickel, S., Steinhardt, C., Schlenker, H., Burkart, W., Reuter-Oppermann, M., 2020. IBM ILOG CPLEX optimization studio. pp. 9–23. http://dx.doi.org/10.1007/978-3-662-54171-5_2.

Papavasilleiou, V., Koulouris, A., Silletti, C., Petrides, D., 2007. Optimize manufacturing of pharmaceutical products with process simulation and production scheduling tools. Chem. Eng. Res. Des. 85 (A7), 1086–1097. http://dx.doi.org/10.1205/cherd06240.

Pathak, D., Agrawal, P., Efros, A.A., Darrell, T., 2017. Curiosity-driven exploration by self-supervised prediction. In: International Conference on Machine Learning. PMLR, pp. 2778–2787.

Pellerin, R., Perrier, N., Berthaut, F., 2020. A survey of hybrid metaheuristics for the resource-constrained project scheduling problem. European J. Oper. Res. 280 (2), 395–416. http://dx.doi.org/10.1016/j.ejor.2019.01.063, URL: https://www.sciencedirect.com/science/article/pii/S0377221719300980.

Pinedo, M.L., 2012. Scheduling: Theory, algorithms, and systems. URL: https://link.springer.com/book/10.1007/978-1-4614-2361-4.

Pinedo, M., Chao, X., 1999. Operations scheduling with applications in manufacturing and services. URL: https://api.semanticscholar.org/CorpusID:62756100.

Pujawan, I.N., 2004. Schedule nervousness in a manufacturing system: a case study. Prod. Plan. Control 15, 515–524. http://dx.doi.org/10.1080/09537280410001726320.

Ribas, I., Leisten, R., Framiñan, J.M., 2010. Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. Comput. Oper. Res. 37 (8), 1439–1454. http://dx.doi.org/10.1016/j.cor.2009.11.001, URL: https://www.sciencedirect.com/science/article/pii/S0305054809002883, Operations Research and Data Mining in Biological Systems.

Ruiz, R., 2016. Scheduling Heuristics. Springer International Publishing, Cham, pp. 1–24. http://dx.doi.org/10.1007/978-3-319-07153-4_44-1.

Sahinidis, N., Grossmann, I., 1991. Reformulation of multiperiod MILP models for planning and scheduling of chemical processes. Comput. Chem. Eng. 15 (4), 255–272. http://dx.doi.org/10.1016/0098-1354(91)85012-J, URL: https://www.sciencedirect.com/science/article/pii/009813549185012J.

Santander, O., Giannikopoulos, I., Stadtherr, M.A., Baldea, M., 2023. An integrated stochastic deep learning-short-term production scheduling-optimal control framework for general batch processes. Ind. Eng. Chem. Res. 62, 2124–2137. http://dx.doi.org/10.1021/acs.iecr.2c02638.

Säynevirta, S., Luotojärvi, M., 2004. Integrated paper production and energy planning. In: Proceedings, PulPaper 2004 Conferences–Energy and Carbon Management.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

Seid, R., Majozi, T., 2012. A robust mathematical formulation for multipurpose batch plants. Chem. Eng. Sci. 68 (1), 36–53. http://dx.doi.org/10.1016/j.ces.2011.08.050.

Shahzad, A., Mebarki, N., 2012. Data mining based job dispatching using hybrid simulation-optimization approach for shop scheduling problem. Eng. Appl. Artif. Intell. 25 (6), 1173–1181. http://dx.doi.org/10.1016/j.engappai.2012.04.001, URL: https://www.sciencedirect.com/science/article/pii/S0952197612000899.

Shang, J., Tian, Y., Liu, Y., Liu, R., 2018. Production scheduling optimization method based on hybrid particle swarm optimization algorithm. J. Intell. Fuzzy Systems 34, 955–964. http://dx.doi.org/10.3233/JIFS-169389.

Subrahmanyam, S., Kudva, G.K., Bassett, M.H., Pekny, J.F., 1996. Application of distributed computing to batch plant design and scheduling. AIChE J. 42 (6), 1648–1661.

Subramanian, K., Maravelias, C.T., Rawlings, J.B., 2012. A state-space model for chemical production scheduling. Comput. Chem. Eng. 47, 97–110. http://dx.doi.org/10.1016/j.compchemeng.2012.06.025, URL: https://www.sciencedirect.com/science/article/pii/S0098135412002098, FOCAPO 2012.

Susarla, N., Li, J., Karimi, I.A., 2010. A novel approach to scheduling multipurpose batch plants using unit-slots. AIChE J. 56 (7), 1859–1879. http://dx.doi.org/10.1002/aic.12120.

van Donselaar, K.H., Gubbels, B.J., 2002. How to release orders in order to minimise system inventory and system nervousness? Int. J. Prod. Econ. 78, 335–343. http://dx.doi.org/10.1016/S0925-5273(01)00209-2.

Velez, S., Maravelias, C.T., 2013a. A branch-and-bound algorithm for the solution of chemical production scheduling MIP models using parallel computing. Comput. Chem. Eng. 55, 28–39. http://dx.doi.org/10.1016/j.compchemeng.2013.03.030.

Velez, S., Maravelias, C.T., 2013b. Multiple and nonuniform time grids in discrete-time MIP models for chemical production scheduling. Comput. Chem. Eng. 53, 70–85. http://dx.doi.org/10.1016/j.compchemeng.2013.01.014.

Velez, S., Maravelias, C.T., 2013c. Reformulations and branching methods for mixed-integer programming chemical production scheduling models. Ind. Eng. Chem. Res. 52 (10), 3832–3841.

Velez, S., Maravelias, C.T., 2015. Theoretical framework for formulating MIP scheduling models with multiple and non-uniform discrete-time grids. Comput. Chem. Eng. 72, 233–254. http://dx.doi.org/10.1016/j.compchemeng.2014.03.003.

Velez, S., Merchan, A.F., Maravelias, C.T., 2015. On the solution of large-scale mixed integer programming scheduling models. Chem. Eng. Sci. 136 (SI), 139–157. http://dx.doi.org/10.1016/j.ces.2015.05.021.

Velez, S., Sundaramoorthy, A., Maravelias, C.T., 2013. Valid inequalities based on demand propagation for chemical production scheduling MIP models. AIChE J. 59 (3), 872–887. http://dx.doi.org/10.1002/aic.14021.

Wang, Y.C., Usher, J.M., 2005. Application of reinforcement learning for agent-based production scheduling. Eng. Appl. Artif. Intell. 18, 73–82. http://dx.doi.org/10.1016/j.engappai.2004.08.018.

Waschneck, B., Reichstaller, A., Belzner, L., Altenmüller, T., Bauernhansl, T., Knapp, A., Kyek, A., 2018. Optimization of Global Production Scheduling with Deep Reinforcement Learning. Vol. 72, Elsevier B.V., pp. 1264–1269. http://dx.doi.org/10.1016/j.procir.2018.03.212.

Westerlund, J., Hastbacka, M., Forssell, S., Westerlund, T., 2007. Mixed-time mixed-integer linear programming scheduling model. Ind. Eng. Chem. Res. 46 (9), 2781–2796. http://dx.doi.org/10.1021/ie060991a.

Wu, D., Ierapetritou, M., 2003. Decomposition approaches for the efficient solution of short-term scheduling problems. Comput. Chem. Eng. 27 (8–9), 1261–1276. http://dx.doi.org/10.1016/S0098-1354(03)00051-6, 2nd Pan American Workshop on Process Systems Engineering, GUARUJA, BRAZIL, SEP 20-21, 2001.

Zhang, Y., Rao, X., Liu, C., Zhang, X., Zhou, Y., 2023. A cooperative EV charging scheduling strategy based on double deep Q-network and Prioritized experience replay. Eng. Appl. Artif. Intell. 118, 105642. http://dx.doi.org/10.1016/j.engappai.2022.105642, URL: https://www.sciencedirect.com/science/article/pii/S0952197622006327.