Swarm and Evolutionary Computation 96 (2025) 101973

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

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

# Deep reinforcement learning algorithm incorporating problem characteristics for dynamic multi-objective permutation flow-shop scheduling problem

Yuan-Yuan Yang [a,b], Bin Qian [a,b,*], Rong Hu [a,b], Zuocheng Li [a,b], Zi-Qi Zhang [a,b], Huai-Ping Jin [a], Jian-Bo Yang [c]

[a] School of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650500, China
[b] Higher Educational Key Laboratory for Industrial Intelligence and Systems of Yunnan Province, Kunming University of Science and Technology, Kunming 650500, China
[c] Alliance Manchester Business School, University of Manchester, Manchester M15 6PB, United Kingdom

ARTICLE INFO

*Keywords:*
Deep reinforcement learning
Dynamic permutation flow shop scheduling problem
Real-time scheduling
Multi-objective optimization

ABSTRACT

Dynamic permutation flow shop scheduling problem (DPFSP) plays a critical role in real-world production systems, characterized by complex uncertainties including machine breakdowns, variable processing times, and the unpredictable job arrivals. Developing real-time solution approaches for such complex dynamic environments represents both a significant industrial need and a substantial computational challenge. Deep reinforcement learning (DRL) has demonstrated excellent capability for rapid and adaptive decision-making in complex dynamic environments, making it particularly suitable for DPFSP applications. This paper proposes a novel DRL algorithm incorporating problem characteristics (NDRLA_IPC) that specifically addresses the dynamic multi-objective PFSP (DMPFSP), with the objectives of minimizing the weighted maximum completion time and the total tardiness. NDRLA_IPC leverages a key insight that DMPFSP can be decomposed into a series of static PFSPs requiring real-time solutions, and implements a double deep Q-network (DDQN) architecture with components specifically engineered for DMPFSP characteristics. The algorithm introduces three key innovations: (1) a state feature vector design with high discrimination and generalization capabilities; (2) an action space designed to minimize temporal gaps between operations on the Gantt chart by leveraging processing constraints dynamically derived from the evolving problem state; and (3) a theoretically-validated reward function that effectively evaluates the online execution impact of each action. Comprehensive experiments demonstrate that NDRLA_IPC, after training on small-scale instances, transfers effectively to larger-scale DMPFSPs, delivering high-quality real-time solutions that outperform existing approaches across multiple performance metrics.

## 1. Introduction

Permutation flow shop scheduling problem (PFSP) is one of the most extensively studied combinatorial optimization problems (COPs), which is typically NP-hard [1,2]. Existing literature on PFSP mainly focus on static and deterministic environments [3,4]. However, with the intensification of market competition and manufacturing complexity [5], unpredictable factors are increasingly prevalent, such as the arrival of new orders, machine breakdowns, and variations in job processing times due to process and workflow improvements, or component availability issues [6]. This makes predefined plans less ideal or even unfeasible. Therefore, dynamic PFSP (DPSP) has become an important research topic in the field of intelligent manufacturing systems [7].

The practical significance of DMPFSP is well illustrated through its application in electronics manufacturing, particularly in printed circuit board assembly lines. This complex manufacturing process typically comprises several sequential stages:

- Solder paste printing, applying solder paste to the board pads.

* Corresponding author.
*E-mail addresses:* yangyuanyuan0730@163.com (Y.-Y. Yang), bin.qian@vip.163.com (B. Qian), ronghu@vip.163.com (R. Hu), zuocheng_li@163.com (Z. Li), Albert.ziqi@hotmail.com (Z.-Q. Zhang), jinhuaiping@126.com (H.-P. Jin), jian-bo.yang@umist.ac.uk (J.-B. Yang).

https://doi.org/10.1016/j.swevo.2025.101973
Received 4 September 2024; Received in revised form 27 March 2025; Accepted 27 April 2025
Available online 14 May 2025

- Component placement using surface mount technology (SMT) machines.
- Reflow soldering, melting the solder paste to form permanent joints.
- Inspection (e.g., automated optical inspection) to detect assembly defects.
- Testing (e.g., in-circuit test or functional test) to ensure electrical functionality.

The dynamic nature of this system is evident in multiple aspects. At any given time, numerous boards, often belonging to different product batches (jobs), are simultaneously processed across these stages. New production orders arrive frequently, with an average interval of perhaps every few hours in high-mix environments. Equipment reliability introduces additional dynamics, with machine stoppages for maintenance or unexpected failures (e.g., component feeder issues on SMT machines, inspection system adjustments) occurring potentially multiple times per shift. Furthermore, job processing times can vary due to factors like component loading/unloading, minor setup adjustments between batches, or temporary component shortages requiring alternative actions. This operational environment requires the simultaneous optimization of multiple performance indicators, particularly production efficiency, on-time delivery performance, minimizing work-in-process, and maintaining high product yield/quality. Unlike the traditional static PFSP, DMPFSP more accurately captures these real-world manufacturing complexities [7]. Successful scheduling solutions in this context should possess three critical capabilities:

- Rapid responsiveness to both planned changes (e.g., new orders) and unplanned events (e.g., equipment failures).
- Real-time optimization capability to minimize idle time between operations and increase production efficiency.
- Effective management of multiple, often conflicting objectives, particularly balancing *makespan* minimization with due date adherence.

Existing research on DPFSP primarily involve two strategies: event-driven rescheduling algorithms and classical scheduling rules. Event-driven rescheduling algorithms decompose DPFSP into a series of static problems triggered by dynamic events, and then apply heuristic or intelligent optimization algorithms to solve these static problems. While this approach can achieve high-quality solutions, it is often computationally intensive and time-consuming, as discussed in Section 2, limiting their applicability in real-time production environments. Classical scheduling rules, on the other hand, operate by assigning priorities to jobs based on predetermined criteria (such as first-come-first-served (FIFO), shortest processing time (SPT) priority, longest processing time (LPT) priority) and processing them accordingly. These rules offer significant advantages in terms of computational efficiency, typically with a time complexity of O(*nlogn*) or less, enabling rapid response times and effective "online" scheduling capabilities. However, their simple dependence on job priorities can lead to lower resource utilization and certain limitations [8]. For instance, in diverse production environments, the SPT rule may result in early-completing machines remaining idle while slow-completing machines cause jobs to queue, creating inefficiencies in the overall manufacturing process. Given these limitations, there is a critical need for a method that combines solution quality computational efficiency—providing both outstanding performance and fast responsiveness in the complex dynamic environment of DPFSP. This approach aims to adapt to the complex dynamic environment of DPFSP and further enhance productivity.

Deep reinforcement learning (DRL) has recently emerged as a powerful approach for solving various production scheduling problems (PSPs) [9]. A key advantage over deep learning is its learning paradigm: DRL agents interact with the environment to accumulate experience and develop optimal strategies, rather than relying on extensive labeled data or predefined knowledge [10]. This interaction-based learning inherently equips DRL with the ability to dynamically adapt to changing conditions without significant reconfiguration. Therefore, when faced with dynamic events inherent in dynamic PSP (DPSP), including DPFSP, DRL offers a distinct advantage over traditional rescheduling methods that typically require problem decomposition and algorithm restarts. A pre-trained DRL agent can instantly utilize its learned policy to evaluate the current system state and select effective actions, enabling efficient real-time scheduling decisions. This responsiveness and adaptive capability derived from learned experience are central to DRL's effectiveness and speed in tackling DPSP. Section 2 will provide a detailed review exploring how DRL are specifically designed in existing literature components to address DPSP.

Recent studies have identified significant challenges in applying DRL to DMPFSP. These challenges primarily focus on the following three aspects.

- State design: Current state design approaches face the issue of highly correlated state sequences [11] that attempt to capture extensive machine, job, and buffer information, as discussed in Section 2. However, these information-dense vectors not only increase computational load but, more critically, often obscure the key features needed for effective scheduling [12]. This issue becomes particularly critical during dynamic events like machine breakdowns. Existing state representations struggle to capture how disruptions propagate through subsequent operations. For instance, the downstream effects of machine idle time and maintenance on job queuing times and operation dependencies are poorly represented. Consequently, the extracted state features may lack relevance or consistency, severely degrading the decision quality of the DRL agent.
- Action space: Most existing methods define the action space using a limited set of generic scheduling rules (e.g., heuristics like SPT, FIFO). However, the effectiveness of these rules varies significantly under different processing constraints and optimization objectives [13]. This simplistic approach often lacks the adaptability needed for specific PFSP constraints and operational dependencies, leading to suboptimal schedules with unnecessary idle time. This problem is intensified during dynamic events such as new job arrivals, processing time variations, or machine unavailability. In such scenarios, the cumulative effect of repeatedly applying these generic rules becomes unpredictable, potentially reducing the stability of the scheduling process. For example, while prioritizing a long-waiting job might address a local concern, this localized decision rarely optimizes the overall production process or minimizes *makespan* effectively.
- State-action correlation: As highlighted by fundamental RL theory [14], the interaction between state and action is critical. Yet, existing DRL approaches for PFSP typically design state representations and action spaces independently, often overlooking their inherent connection and potential synergies. This disconnect hinders the agent's stable decisions, especially when responding to dynamic events like new jobs arrivals or machine breakdowns. In these environments, a coherent and integrated state-action design is essential for achieving robust scheduling performance.

These three interconnected challenges—inefficient state representations, simplistic action definitions, and the disconnect between state and action—create significant barriers to effective DRL implementation in dynamic manufacturing scheduling environments. To address these fundamental challenges, we propose a novel DRL algorithm incorporating problem characteristics (NDRLA_IPC) for solving DMPFSP with the objectives of minimizing the weighted maximum completion time and the total tardiness. The key contributions of this work are summarized as follows.

First, NDRLA_IPC leverages DMPFSP production constraints to extract processing completion times, generating a compact yet

**Table 1**
Summary of DRL methods in DPSPs.

| Reference | Year | Problem | Dynamic situation | Methodology | State | Action |
|---|---|---|---|---|---|---|
| Luo [23] | 2020 | Dynamic flexible JSP (DFJSP) | New job insertions | Deep Q network (DQN) | 7 | 6 |
| Luo [24] | 2021 | DFJSP | New job insertions and machine breakdowns | PPO | 20 | 11 |
| Zhao [25] | 2021 | DJSP | Urgent orders, machine failures | DQN | 5 | 10 |
| Zhang [26] | 2023 | DFJSP | Machine processing time uncertainty | PPO | 6 | 8 |
| Wang [27] | 2023 | DJSP | Random job arrivals | PPO | 9 | 11 |
| Wang [28] | 2022 | DFJSP | Job insertion, job cancellation, job operation modification, machine addition, machine tool replacement and machine breakdown | Double DQN (DDQN) | 6 | 9 |
| Gui [29] | 2023 | DFJSP | Changes in due date, order size, and arrival interval of new jobs | Deep deterministic policy gradient | 20 | 7 |
| Xu [30] | 2024 | DFJSP | New jobs arrive | Niching genetic programming (GP) +RL | 15 | Heuristic rules selection by Niching GP |
| Hu [31] | 2020 | Dynamic AGV scheduling | Real-time task | DDQN | 5 | 5 |
| Yang [32] | 2021 | DFSP | New jobs arrive | Advantage actor-critic (A2C) | 17 | 5 |
| Yang [33] | 2022 | Dynamic distributed FSP | New jobs arrive | SA-NET | 8 | 8 |
| Liu [7] | 2023 | Dynamic hybrid FSP (DHFSP) | Urgent demands and unexpected interruptions | Multi-agent DRL | 14 | 8 |
| Wang [34] | 2022 | DHFSP | New jobs arrive | Multi-agent DRL | 27 | 23 |
| Grumbach [35] | 2024 | Robust-stable scheduling for DFSP | Uncertain processing times, stochastic machine failures and uncertain repair times | PPO and A2C | 12 | 3 |
| Kim [36] | 2023 | Robotic FSP | - | DQN | 6 | Unloading and loading actions |
| Ren [37] | 2021 | FSP | - | State and action network | 9 | 10 |
| Gil [38] | 2022 | FSP | - | DQN | $m/2$ | $m/2$ |
| Wang [39] | 2022 | Non-permutation FSP | - | LSTM-based approach | 15 | 14 |

comprehensive feature vector that represents the processing environment state. Crucially, we design five novel actions specifically tailored to address different scheduling scenarios identified through this state representation, establishing an effective and targeted action space. Using DDQN as the agent, we develop a weighted composite reward function to handle multi-objective optimization, guiding the agent to discover superior scheduling policies through iterative environmental interaction.

Second, we address the significant computational challenge of continuous state feature extraction in highly dynamic scheduling environments. While traditional approaches rely on high-dimensional state representations that impede training efficiency and convergence, NDRLA_IPC prioritizes schedule compactness—visualized as minimizing idle times between operations on the Gantt chart. Our approach captures essential inter-operation dependencies through a significantly lower-dimensional state representation derived directly from DMPFSP timing constraints. This strategic simplification substantially reduces learning complexity, reducing training difficulty and improving model accuracy robustness in different scheduling scenarios.

Third, NDRLA_IPC establishes a highly generalizable framework for real-time DPFSP scheduling through a direct and interpretable mapping between environmental states and appropriate scheduling actions. The framework triggers specific actions based on precisely assessed system states, explicitly targeting the elimination between processing time gaps on the Gantt chart. This targeted approach facilitates more compact job arrangements with minimal idle time, substantially improving solution quality across diverse problem scales. A crucial advantage emerges from the dimensional efficiency of our state-action design: agents trained on small-scale problems demonstrate exceptional generalization capabilities when applied to larger-scale instances during online scheduling. This framework provides both theoretical foundations and practical implementation guidelines for developing efficient real-time DRL algorithms across diverse FSP environments.

## 2. Literature review

Rescheduling algorithms serve as a core methodology in addressing DPSPs, with extensive research dedicated to their development. These traditional approaches address dynamic events by iteratively revising existing schedules, primarily through reactive strategies that require complete re-executing of algorithms when dynamic events occur. For instance, Adibi [15] proposed a greedy iterative rescheduling algorithm for dynamic job shop scheduling problems (DJSP), with reported runtimes ranging from 76.41 to 437.4 seconds for small-scale problems. Similarly, Long [16] developed a hybrid rescheduling algorithm for dynamic steelmaking continuous casting, where comparative experiments were subject to a 250-second termination criterion. The application of metaheuristics is evident in Kundakcı's [17] hybrid genetic algorithm (HGA) for DJSP, achieving completion times between 0.22 and 114 seconds. Li [18]'s GSH algorithm for FSP with new job arrivals outperformed Nawaz-Enscore-Ham (NEH) algorithm, but exhibited execution times between 0.12 and 664.91 seconds for problem scales of 10 to 100 jobs. Similarly, Rahman [19] explored a hybrid approach combining a genetic algorithm and particle swarm optimization for dynamic FSP (DFSP), reporting runtimes between 11.57 and 183.21 seconds for problem scales of 20 to 200 jobs. A critical limitation of these algorithms is their dependence on restarting the entire optimization process when dynamic events occur, resulting in computationally expensive operations that become prohibitive as problem complexity increases. Specifically, Valledor [20]'s rescheduling method for DFSP, where new schedules are generated upon dynamic changes, and the runtime was experimentally determined as: runtime=number of machines$\times$ (number of jobs)$^2\times$100 milliseconds.

As previously discussed, rescheduling algorithms have proven effective for static scheduling problems. However, they encounter substantial computational barriers when scaled to medium and large-scale industrial problems, limiting their practical applicability in real-time environments. Each time a dynamic event occurs, the algorithm must be restarted to generate a new strategy. In our study of DPFSP, we simulate real-world production scenarios by incorporating three types of

dynamic events: new job arrivals following a Poisson process, stochastic machine breakdowns, and probabilistic variations in processing times. This high degree of dynamism makes it particularly challenges. Therefore, current rescheduling approaches face a fundamental trade-off between solution quality and computational efficiency, making real-time response to frequent dynamic events virtually unattainable for complex industrial scenarios.

DRL has emerged as a promising alternative paradigm for dynamic scheduling problems, drawing substantial research attention for its effectiveness in uncertain environments. This strength stems from its ability to learn directly from environmental interaction and adapt rapidly to dynamic changes [21,22]. However, despite the promising potential of DRL, a key research challenge remains: designing state representations and action spaces that capture the complexity of dynamic scheduling environments while ensuring computational efficiency and scalability. The ongoing evolution of these design approaches, as summarized in Table 1, highlights systematic progress in addressing these fundamental challenges across different DPSPs.

Recent applications of DRL in dynamic scheduling problems reveal a persistent limitation in state space design and action representation that constrains their full potential. In the domain of DJSP, Luo [23] designed a deep Q network (DQN) algorithm for the dynamic flexible JSP (DFJSP), which uses seven features and six heuristic rules as actions. Building on this work, the authors [24] advanced to a hierarchical multi-agent proximal policy optimization (PPO) method for the dynamic partial-no-wait multi-objective FJSP. This method employs 20 state features to comprehensively capture production states, along with 5 job selection rules and 6 machine assignment rules as actions. The evolution of DRL applications in this domain reveals a consistent pattern of increasing complexity in state representation while maintaining reliance on heuristic rules for action spaces. Notable implementations include Zhao[25]'s DQN method with five state features and ten heuristic rules. Zhang [26] and Wang[27]'s PPO framework for DJSP. Wang [28] improves solution quality using double DQN (DDQN) for DFJSP. Gui's deep deterministic policy gradient (DDPG) algorithm [29] that utilizes 20 production environment features and seven rules as actions. To select higher-quality scheduling rules as actions, Xu proposed a novel method [30] that integrates Niching genetic programming (GP) with RL. Instead of relying on directly selecting heuristic rules, this approach leverages the evolutionary capabilities of GP to automatically select scheduling heuristics as actions for the RL agent. Besides, DRL is also used to solve dynamic intelligent vehicle scheduling problem. Hu [31] designed a DDQN algorithm with a variable state dimension of 5 features to address the dynamic AGV scheduling problem.

In the domain of DFSP, recent DRL applications exhibit similar patterns in state-action design. Yang [32] implemented an advantage actor-critic algorithm with the 17 state features and five heuristic rules as actions. In their subsequent work [33], they introduced the state-action network (SA-NET) algorithm for dynamic distributed PFSP with eight state features and eight heuristic rules. Liu [7] addressed dynamic hybrid FSP (DHFSP) through multi-agent DRL approach. Similarly, Wang [34] proposed a MADRL algorithm for DHFSP with 27 state features and 23 heuristic rules. Addressing the need for robust solutions in dynamic environments, Grumbach [35] developed a DRL approach for DFSP specifically focusing on robust-stable scheduling. They designed 12 state features and 3 actions to handle operation time to achieve stable and robust schedules against disruptions. In static FSP applications, which can be viewed as a special case of dynamic scheduling, Kim [36] developed a DQN algorithm with six state features and feasible operations as actions. Similar approaches were adopted by Ren [37] (nine features, ten heuristic rules) and Gil [38] (m/2 features, m/2 heuristic rules) for various FSP variants. Notably, Wang [39] proposed a more sophisticated approach for non-permutation FSP (NPFSP), incorporating 15 state features and 14 heuristic rules as actions, while utilizing long short-term memory networks to capture temporal dependencies in state sequences.

A systematic review of existing literature reveals that current research focuses on two directions: designing complex state representations to comprehensively capture machine status, job dependencies, and environmental variations, while still limiting action spaces to predefined simple heuristic rules. There exists a fundamental mismatch between these complex state representations and constrained action spaces, making it difficult for agents to effectively utilize excessive environmental information and overcome the limitations of simplified action mechanisms. Inspired by these insights, there is a compelling need to explore novel architectural designs, including more effective state extraction methods, flexible action space definitions, and tighter state-action integration mechanisms, to achieve significant performance improvements in highly dynamic scheduling environments

## 3. Problem description and analysis

This section provides a detailed introduction to the DMPFSP process. A set of jobs $J = \{J_1, ..., J_n\}$ is to be processed sequentially on a set of machines $M = \{M_1, ..., M_m\}$. At time 0, jobs are waiting in the buffer for processing, and new jobs will arrive dynamically. During processing, there exists a probability that the processing time of jobs in the buffer may change, and machines may encounter breakdowns. The notations used in the DMPFSP model are summarized in Table 2.

For the static component of DMPFSP, we formulate a mixed integer linear programming (MILP) model as the basis for further analysis. The objective function of DMPFSP, consistent with [40], is to minimize the weighted total tardiness and maximum completion time, as follows:

$$\text{Minimize } \alpha * C_{\max}(\boldsymbol{\pi}) + (1-\alpha)\sum_{J_j \in \boldsymbol{J}} T_j \tag{1}$$

Where $C_{\max}(\boldsymbol{\pi}) = \max\{C_{mj}\}, j \in \{1,2,...n\}, T_j = \max\{0, C_{mj} - d_j\}, j \in \{1, 2, ...n\}$.

Subject to:

$$\sum\nolimits_{k\in\{1,2,...n\}} x_{jk} = 1, \forall j \in \{1, 2, ..., n\}, \tag{2}$$

$$\sum\nolimits_{j\in\{1,2,...n\}} x_{jk} = 1, \forall k \in \{1, 2, ..., n\}, \tag{3}$$

**Table 2**
Notations applied in the model of the DMPFSP.

| Parameters | Description |
|---|---|
| $\boldsymbol{\pi}$ | The total sequence of jobs, i.e., $\boldsymbol{\pi} = [\pi_1, \pi_2, ..., \pi_n]$. |
| $\pi_j$ | The $j$-th job of $\boldsymbol{\pi}$. |
| $C_{max}$ | The maximum completion time (*makespan*). |
| $\alpha$ | The weighting factors between objectives ($0 \le \alpha \le 1$) |
| $J_j$ | The index for jobs where $j = 1, 2, ..., n$. |
| $M_i$ | The index for machines where $i = 1, 2, ..., m$. |
| $\boldsymbol{O}$ | The set of operations, i.e., $\boldsymbol{O} = \{O_{i1}, O_{i2}, ..., O_{in}\}$. |
| $p_{i,j}$ | The processing time of operation $O_{ij}$ on $M_i$. |
| $S_{i,j}$ | The start time of $J_j$ on $M_i$ where $j = 1, 2, ..., n, i = 1, 2, ..., m$. |
| $C_{i,j}$ | The completion time of $J_j$ on $M_i$ where $j = 1, 2, ..., n, i = 1, 2, ..., m$. |
| $C_j$ | The completion time of $J_j$ on the last machine where $j = 1, 2, ..., n$. |
| $d_j$ | The due date of $J_j$ where $j = 1, 2, ..., n$. |
| $r_j$ | The release time of $J_j$ |
| $T_j$ | The tardiness of job $J_j$ |
| $\Lambda$ | The average arrival rate of new jobs follows a Poisson process |
| $A_{i,t}$ | The availability status of $M_i$ at time $t$ |
| Decision Variables | |
| $x_{jk}$ | Binary variable, $x_{jk} = 1$ if job $J_j$ is processed at the $k$-th position and $x_{jk} = 0$ otherwise. |
| $y_{ijt}$ | Binary variable, $y_{ijt} = 1$ if job $J_j$ starts processing on $M_i$ at the time $t$ and $y_{ijt} = 0$ otherwise. |
| $A_{it}$ | Binary variable, equals 1 if $M_i$ is available at the time $t$. |
| Sets | |
| $\boldsymbol{J}$ | Set of jobs |
| $\boldsymbol{M}$ | Set of machines |
| $\boldsymbol{T}$ | Set of discrete time periods |

$$\sum_t y_{ijt} \le 1, \forall i \in \{1,2,...,m\}, t \in \boldsymbol{T}, \tag{4}$$

$$S_{i,j} = \sum_{t \in \boldsymbol{T}} t \cdot y_{ijt}, \forall i \in \{1,2,...,m\}, j \in \{1,2,...n\}, \tag{5}$$

$$S_{i,j} \ge r_j, \forall i \in \{1,2,...m\}, j \in \{1,2,...n\}, \tag{6}$$

$$S_{i+1,j} \ge S_{i,j} + p_{i,j}, \forall i \in \{1,2,...,m\}, j \in \{1,2,...n\}, \tag{7}$$

$$C_{i,j} = S_{i,j} + p_{i,j} \tag{8}$$

$$\begin{aligned} &S_{i+1,j} \ge S_{i,j} + p_{i,j} - G\left(2 - x_{jk} - x_{j'(k+1)}\right), \forall i \in \{1,2,...,m\}, \\ &j, j' \in \{1,2,...n\}, j \ne j', k \in \{1,2,...n-1\} \end{aligned}, \tag{9}$$

$$C_{\max} \ge \sum_{t \in \boldsymbol{T}} \left(t \cdot y_{mjt}\right) + p_{m,j}, \forall j \in \{1,2,...n\}, C_{\max} \ge 0, C_{\max} \ge C_{m,j}, \tag{10}$$

$$T_j \ge \sum_{t \in \boldsymbol{T}} \left(t \cdot y_{mjt}\right) + p_{m,j} - d_j, \forall j \in \{1,2,...n\}, T_j \ge 0. \tag{11}$$

Eqs. (2) and (3) ensure that each job is assigned to exactly one position and each position is filled by exactly one job. Eq. (4) is a machine capacity constraint. It ensures that on each machine, at most one job can be processed at any given time $t$. Eq. (5) defines the constraint that the start time for $J_j$ on $M_i$. Eq. (6) indicates the constraint that the start time of a job on each machine must be after its release time. Eq. (7) imposes the constraint that a job's start time on the next machine must be after the completion time on the previous machine. Eq. (8) defines that the completion time of a job on $M_i$ is the sum of its start time and processing time on $M_i$. Eq. (9) represents the job order constraint. On each machine, if $J_j$ is in position $k$ and $J_{j'}$ is in position $k$+1 in the processing sequence, this constraint ensures $J_{j'}$ begins processing only after $J_j$ completes on $M_i$. $G$ is a sufficiently large positive number. Eq. (10) defines the *makespan*, which is the maximum completion time among all jobs on the last machine. Eq. (11) defines the tardiness. It is the positive difference between the job's completion time and due date.

It should be pointed out that DMPFSP studied in this paper incorporates three types of dynamic events: new job arrivals, machine breakdowns, and processing time variations. These dynamic characteristics can be mathematically expressed as follows:

1) *New job arrivals*: For each new job, it has

$$r_j = AT_j, \forall J_j \in \boldsymbol{J}_{new} \tag{12}$$

where $AT_j$ is the arrival time of new job $J_j$, and $J_{new}$ is the set of newly arrived jobs. Then we can get the start time constraint of the $J_j$:

$$S_{1,j} \ge \max\left(r_j, C_{1,j-1}\right), \left(C_{1,0} = 0\right). \tag{13}$$

In Eq. (13), the arrival time $AT_j$ follows a Poisson process with rate $\lambda$, where the probability of $s$ new jobs arriving in a time interval $\Delta t$ is given by

$$P(s;\lambda\Delta t) = \frac{(\lambda\Delta t)^s e^{-\lambda\Delta t}}{s!}, s = 0,1,2,.... \tag{14}$$

Here, $\lambda$ is the average arrival rate (jobs per unit time), and $\Delta t$ is the time interval.

2) *Machine breakdowns*: For each machine $M_i \in M$:

$$P(breakdown_i(t)) = \theta_i, \forall t \in \boldsymbol{T} \tag{15}$$

$$RepairTime_i \sim R(\mu_i, \sigma_i), \tag{16}$$

where $\theta_i$ is the breakdown probability of $M_i$ and $R(\mu_i, \sigma_i)$ represents the repair time distribution.

$$A_{i,t} = \begin{cases} 0, \forall i \in \{1,2,...,m\}, t \in [t_{breakdown}, t_{breakdown} + RepairTime_i] \\ 1, \text{otherwise} \end{cases}, \tag{17}$$

$$y_{ijt} \le A_{i,t}, \forall i \in \{1,2,...,m\}, j \in \{1,2,...n\}, t \in \boldsymbol{T}, \tag{18}$$

Eqs. (17) and (18) describe the update mechanism for machine availability status and enforce constraints to prevent jobs from being scheduled during periods when machines are under breakdown or repair.

3) *Processing time variations*: The actual processing time $p'_{ij}$ can be expressed as:

$$p'_{i,j} \ge p_{i,j}\left(1 \pm \delta_{i,j}\right), \forall \boldsymbol{M}_i \in \boldsymbol{M}, J_j \in \boldsymbol{J}. \tag{19}$$

where $\delta_{i,j}$ represents the variation factor.

Obviously, DMPFSP consists of a series of scheduling decision problems (i.e., a series of static PFSPs with gradually increasing scale) triggered by dynamic events. The above MILP formulation can be used to describe each PFSP in DMPFSP. The main characteristic of DMPFSP lies in its dynamics.

In many rapid production scenarios of DMPFSP in the modern manufacturing industry, high-performance algorithm needs to be utilized to solve each DMPFSP's inherent scheduling decision problem (i.e., each PFSP) in real time to ensure stable and effective production execution. Unfortunately, the following Theorem 1 proves the NP-hardness of DMPFSP.

**Theorem 1**. DMPFSP incorporating dynamic job arrivals, machine breakdowns, and job processing time variations is strongly NP-hard.

**Proof**. Obviously, when no new jobs arrive (i.e., $\lambda$=0), no machine breakdowns (i.e., $\theta_i$=0, $\forall M_i \in \boldsymbol{M}$), and no processing time variations (i.e., $\delta_{ij}$=0, $\forall M_i \in \boldsymbol{M}, J_j \in \boldsymbol{J}$), DMPFSP is transformed into the static PFSP. That is, PFSP is a special case of DMPFSP.

Based on the reduction concept of complexity theory, it can be concluded that PFSP reduces to DMPFSP. Since the static PFSP is known to be NP-hard in the strong sense, DMPFSP is also strongly NP-hard.

Theorem 1 indicates that mathematical programming methods, approximate methods and commercial solvers (e.g., Gurobi, Cplex) cannot ensure to obtain high-quality solution of each PFSP in DMPFSP within a real-time computing time [2]. Thus, it is a challenge to devise an efficient algorithm to address DMPFSP, i.e., a series of PFSPs with different scales. Inspired by the good performance of DRL-based algorithm in solving some DPSPs in recent years (see Table 1), a novel DRL-based algorithm, namely NDRLA_IPC (see Section 4), is designed to efficiently solve the considered DMPFSP.

In addition, to better illustrate the DMPFSP mathematical formulation, we present an example with five initial jobs ($J_1$-$J_5$) on four machines ($M_1$-$M_4$), where two additional jobs ($J_6$, $J_7$) arrive during production. Table 3 presents the corresponding processing times.

**Table 3**
An example of processing information for each job.

| Job | $M_1$ | $M_2$ | $M_3$ | $M_4$ | Arrival time | Due date |
|---|---|---|---|---|---|---|
| $J_1$ | 3 | 4 | 2 | 5 | 0 | 20 |
| $J_2$ | 4 | 3 | 5 | 2 | 0 | 22 |
| $J_3$ | 2 | 5 | 3 | 4 | 0 | 25 |
| $J_4$ | 5 | 2 | 4 | 3 | 0 | 23 |
| $J_5$ | 3 | 4 | 2 | 5 | 0 | 21 |
| $J_6$ | 4 | 3 | 5 | 2 | 5 | 30 |
| $J_7$ | 2 | 5 | 3 | 4 | 8 | 35 |

Fig. 1(a) shows the Gantt chart of a static FSP with the processing sequence {$J_1$, $J_2$, $J_3$, $J_4$, $J_5$} when no new jobs arrive. It can be seen that after each job is processed, it is removed from the buffer while the next job continues to be processed. This process continues until all jobs are processed the buffer is empty. Subsequently, Fig. 1(b) demonstrates how the DMPFSP handles the arrival of new jobs. Specifically, when $J_6$ arrives at time $t$=5, it is added to the buffer to wait for processing. According to the constraint relationship between job release time and its processing start time defined in Eq. (13), the job being processed will not be interrupted by the arrival of new jobs. Instead, after $M_1$ completes processing the current job, the next job to be processed is selected from the buffer. Similarly, $J_7$ arrives at $t$=8 to wait for processing. The final Gantt chart incorporating all 7 jobs is shown in Fig. 1(b).

## 4. Novel deep reinforcement learning algorithm incorporating problem characteristics

This section introduces the key components of NDRLA_IPC in detail, including the state feature, the action space, the weighted composite reward function and its feasibility proof, the update process of DDQN in NDRLA_IPC, the proposed framework of the NDRLA_IPC, and the implementation example of NDRLA_IPC.

### 4.1. State feature

The state features in NDRLA_IPC are extracted based on the relative completion time interval of the current job across all machines. In other words, it describes the "completion time shape" of the current job on the Gantt chart. This is done to better match the processing time of the next job with the current state (the completion time shape of the current job) when selecting the next job through action.

Based on the DMPFSP model characteristics, the jobs are arranged in the same order on each machine, and the new operation starts after the current operation is completed. Therefore, once the next job to be processed is determined, its completion time on each machine can be calculated. Assuming that the $j$-th job $\pi_j$ has been selected at the current time, let $(C_{1,j},\ C_{2,j},\ \ldots,\ C_{m,j})$ denote the vector consisting of its completion time on each machine. The completion time interval on the $i$-th machine is defined as follows.

$$\triangle I_{i,j} = C_{i+1,j} - C_{i,j}, j = 1, 2, \ldots, n; i = 1, 2, \ldots, m-1. \tag{20}$$

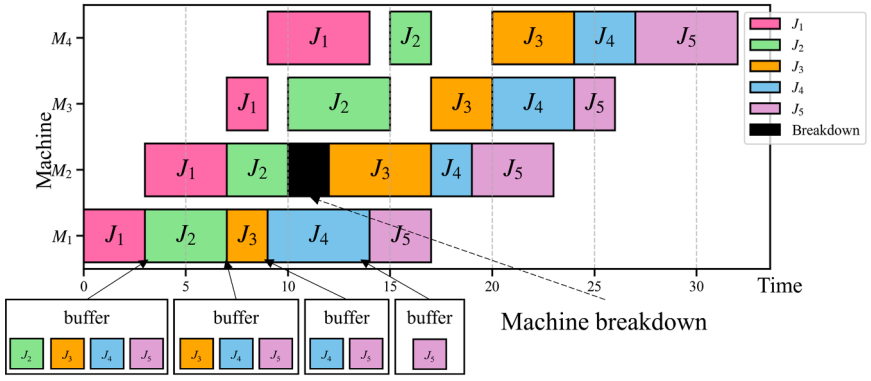
Then the completion time interval vector $(\triangle I_{1,j},\ \triangle I_{2,j},\ \ldots,\ \triangle I_{m,j})$ of $\pi_j$ is normalized to reduce the complexity of the state space by constraining the size of each element to the range [0, 1], as follows:

$$\triangle I'_{i,j} = \frac{\triangle I_{i,j}}{\sum_{i=1}^{m-1} \triangle I_{i,j}}. \tag{21}$$

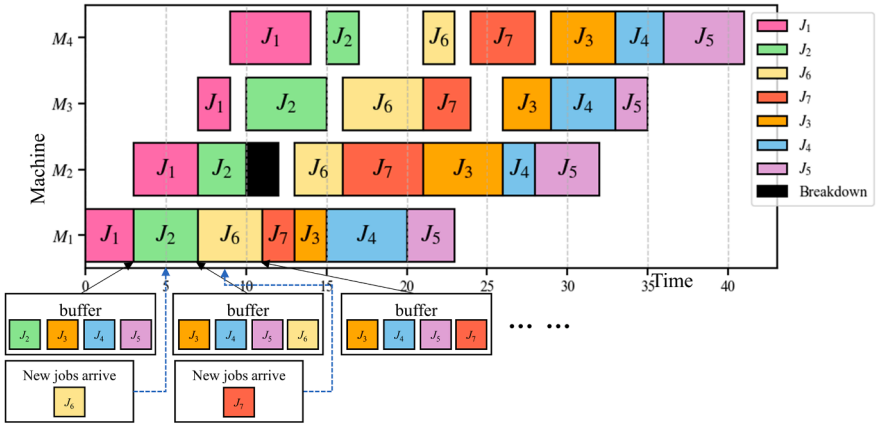
Thus, the state feature $s_j$ can be obtained when $\pi_j$ is determined as follows:

$$s_j = \left(\triangle I'_{1,j}, \triangle I'_{2,j}, \ldots, \triangle I'_{m-1,j}\right). \tag{22}$$

For example, consider job $J_1$'s with completion times: $M_1$: $C_{1,1}$=5, $M_2$: $C_{1,2}$=9, $M_3$: $C_{1,3}$=14. The completion time intervals are calculated as follows: $\triangle I_{1,1}$=$C_{1,1}$=5 (first machine uses absolute completion time), $\triangle I_{2,1}$=$C_{1,2}$-$C_{1,1}$=9-5=4 (interval between $M_2$ and $M_1$), $\triangle I_{3,1}$=$C_{1,3}$-$C_{1,2}$=14-9=5 (interval between $M_3$ and $M_2$). These intervals reflect the distribution of processing times across machines. Next, Eq. (21) normalizes all values to the range [0,1] to make the state representation scale-invariant, which is beneficial for DRL network training. In our




(a) Gantt chart of a static FSP without considering the arrival of new jobs




(b) Gantt chart of the processing process of the DMPFSP with new jobs arriving

**Fig. 1.** An example explanation of DMPFSP.

example: $\triangle I'_{1,1}$=5/14≈0.357, $\triangle I'_{2,1}$=4/14≈0.286, $\triangle I'_{3,1}$=5/14≈0.357. The state vector $s$ = [0.357, 0.286, 0.357], obtained using Eq. (22), captures the normalized time distribution pattern, machine completion information, and serves as the basis for action selection.

In DMPFSP, machine breakdowns are considered. Jobs can only resume processing after the broken machine is repaired. Therefore, the repair time of the machine can be regarded as additional processing time, which may result in $C_{i,j} > C_{i+1,j}$. In this case, $\triangle I_{ij} = 0$.

### 4.2. Action space

This section introduces the proposed action space, including an action to select the first job and four actions to select non-first jobs.

1) ***Action for selecting the first job***

When the first job $\pi_1$ is to be selected, the weights $w_{BT}$ and $w_{due\_j}$ are assigned to the jobs in the buffer (*BF*) according to Eqs. (23) and (24), and the job with the largest $v_j^1$ is as the $\pi_1$.

$$w_{BT} = \frac{\left(1 - \frac{BT_j - \min BT_{j\in BF}}{\max BT_{j\in BF} - \min BT_{j\in BF}}\right)}{\sum_{j\in BF}\left(1 - \frac{BT_j - \min BT_{j\in BF}}{\max BT_{j\in BF} - \min BT_{j\in BF}}\right)}. \tag{23}$$

$$w_{due\_j} = \frac{\left(1 - \frac{d_j - \min d_{j\in CR}}{\max d_{j\in CR} - \min d_{j\in CR}}\right)}{\sum_{j\in CR}\left(1 - \frac{d_j - \min d_{j\in CR}}{\max d_{j\in CR} - \min d_{j\in CR}}\right)}. \tag{24}$$

$$v_j^1 = \alpha * w_{BT} + (1-\alpha) * w_{due\_j}, j \in CR. \tag{25}$$

Here, $BT_j$ represents the total accumulated waiting time of $J_j$ on each machine throughout the entire processing period, starting from the time $J_j$ begins processing.

The selection of the first job is crucial, as it establishes the foundation for subsequent scheduling decisions. Eq. (23) normalizes the $BT_j$ of all jobs. The lower the $w_{BT}$ for a job, the shorter the waiting time on each machine when the job is processed first. Eq. (24) normalizes the $d_j$ of all job, where $d_j$ is the due date of $J_j$ (as defined in Table 2). The lower the $w_{due\_j}$ for a job, the higher its due date priority. Then the Eq. (25) combines the two factors from Eqs. (23) and (24) to calculate the composite priority of the jobs, balancing processing time and due date considerations.

2) ***Actions for selecting non-first jobs***

In NDRLA_IPC, four scheduling rules ($a_1$ to $a_4$) as actions are proposed for selecting non-first jobs. Assuming the first $j$-1 jobs have been identified, select the job from the *BF* as the $\pi_j$.

$a_1$: ① Determine the key machine $M_{i*}$ based on the biggest $\Delta I_{i*\hat{j}}$ in $(\Delta I_{1\hat{j}}, \Delta I_{2\hat{j}}, \ldots, \Delta I_{m-1\hat{j}})$. ② Sort the jobs in non-decreasing order based on their $di_{i*j}$, where $di_{i*j} = \left|p_{i*j} - \Delta I_{i*\hat{j}}\right|$, $j \in BF$. Add the first $\max\left\{2, \left[\log_2^{|BF|} - 1\right]\right\}$ jobs with the smallest $di_{i*j}$ and the earliest $d_j$ job in the *BF* into the candidate region (CR). ③ Assign the weights $w_{1j}$ and $w_{2j}$ for the jobs in the CR as follows, and the job with the biggest $v_j$ is selected as $\pi_j$.

$$w_{1j} = \frac{\left|1 - \frac{di_{i*j} - \min di_{i*}}{\max di_{i*} - \min di_{i*}}\right|}{\sum_{j\in CR}\left|1 - \frac{di_{i*j} - \min di_{i*}}{\max di_{i*} - \min di_{i*}}\right|}. \tag{26}$$

$$w_{2j} = \frac{\left(1 - D_j^{norm}\right)}{\sum_{j\in CR}\left(1 - D_j^{norm}\right)}. \tag{27}$$

$$D_j^{norm} = \frac{\left|C_{i*-1,j} - S_{i*j}\right| - \min\left|C_{i*-1,j} - S_{i*j}\right|_{j\in CR}}{\max\left|C_{i*-1,j} - S_{i*j}\right|_{j\in CR} - \min\left|C_{i*-1,j} - S_{i*j}\right|_{j\in CR}}. \tag{28}$$

$$v_j = \frac{1}{2}\alpha * \left(w_{1j} + w_{2j}\right) + (1-\alpha) * w_{due\_j}, j \in CR. \tag{29}$$

The action $a_1$ is designed to consider both the current machine load distribution and job processing characteristics. We provide a detailed explanation of its components and working mechanism using Fig. 2.

*Step* 1: Key machine identification. As shown in Fig. 2(a), action $a_1$ first identifies the key machine $M_{i*}$ based on the maximum completion time interval ($\Delta I_{i*\hat{j}}$) among all machines. This step aims to identify the potential bottleneck in the current schedule. For example, in Fig. 2(a), $M_2$ is identified as the key machine ($i^*$=2) due to its largest interval $\Delta I_{2*\hat{j}}$.

*Step* 2: Job evaluation and candidate region formation. The action then evaluates jobs in the *BF* using the difference measure $di_{i*j}$. Here, $di_{i*j}$ represents the absolute difference between the job's processing time ($p_{i*j}$) and the current interval ($\Delta I_{i*\hat{j}}$). Smaller $di_{i*j}$ values indicate better matching between job processing time and machine interval. The CR is formed by selecting $\max\left\{2, \left[\log_2^{|BF|} - 1\right]\right\}$ jobs with smallest $di_{i*j}$. As illustrated in Fig. 2(a), four jobs ($J_1$-$J_4$) are evaluated with their processing times and due dates. Jobs with smaller differences are prioritized for the CR.

*Step* 3: Weight assignment and final selection. Fig. 2(b) illustrates final selection process through weight assignment:

$w_{1j}$: Eq. (26) considers the normalized processing time difference.

$w_{2j}$: Eq. (27) accounts for the normalized completion time difference.

$D_j^{norm}$: Eq. (28) normalizes the completion time differences across machines.

$v_j$: Eq. (29) combines all factors using the importance coefficient $\alpha$.

$a_2$: ① Determine the key machine $M_{i*}$ based on the smallest $\Delta I_{i*\hat{j}}$ in $(\Delta I_{1\hat{j}}, \Delta I_{2\hat{j}}, \ldots, \Delta I_{m-1\hat{j}})$.The remaining steps are the same as those for $a_1$.

$a_3$: ① Let $\boldsymbol{I}_j = (\Delta I_{1\hat{j}}, \Delta I_{2\hat{j}}, \ldots, \Delta I_{m-1\hat{j}})$, and the average value of $\boldsymbol{I}_j$ is denoted as $Avg_j$. ② Sort the jobs in non-decreasing order based on the variance $Var_j$ of their processing time across all machines, where $j \in BF$. Add the first $\max\left\{2, \left[\log_2^{|BF|} - 1\right]\right\}$ jobs with the smallest $Var_j$ and the earliest $d_j$ job in the *BF* into the CR. ③ Assign the weights $w'_{1j}$ and $w'_{2j}$ for the jobs in the CR as follows, and the job with the biggest $v'_j$ is selected as $\pi_j$.

$$w'_{1j} = \frac{\left|1 - \frac{Var_j - \min Var_{j\in CR}}{\max Var_{j\in CR} - \min Var_{j\in CR}}\right|}{\sum_{j\in CR}\left|1 - \frac{Var_j - \min Var_{j\in CR}}{\max Var_{j\in CR} - \min Var_{j\in CR}}\right|}. \tag{30}$$

$$w'_{2j} = \frac{1 - \frac{Avg_j - \min Avg_{j,j\in CR}}{\max Avg_{j,j\in CR} - \min Avg_{j,j\in CR}}}{\sum\left|1 - \frac{Avg_j - \min Avg_{j,j\in CR}}{\max Avg_{j,j\in CR} - \min Avg_{j,j\in CR}}\right|}. \tag{31}$$

$$v'_j = \frac{1}{2}\alpha * \left(w'_{1j} + w'_{2j}\right) + (1-\alpha) * w_{due\_j}, j \in CR. \tag{32}$$

The purpose of action $a_3$ is as follows: when the completion time intervals across machines are similar, select the jobs with relatively uniform processing time on each machine and whose processing time is

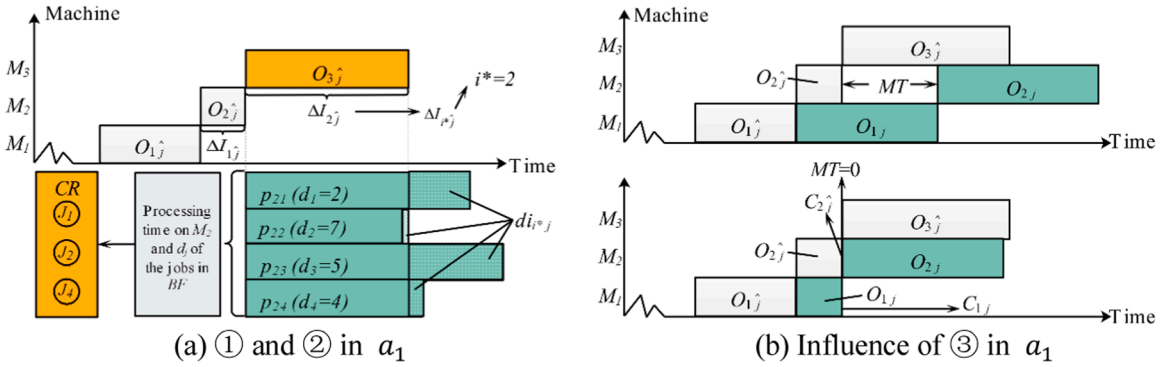



**Fig. 2.** Explanation of the legend for $a_1$.

close to the current interval vector $\boldsymbol{I}_j$. This approach aims to better align new jobs with the current completion time intervals. Fig. 3 demonstrates the implementation process and effects of this action using a detailed example.

*Step* 1: Time interval vector construction. For each job $J_j$ in the *BF*, an interval vector $\boldsymbol{I}_j = (\Delta I_{1\hat{j}}, \Delta I_{2\hat{j}}, \ldots, \Delta I_{m-1\hat{j}})$ is constructed. The average value $Avg_{\boldsymbol{I}_j}$ of these intervals provides a reference for evaluating schedule consistency. As shown in Fig. 3(a), the intervals $\Delta I_{1\hat{j}}$ and $\Delta I_{2\hat{j}}$ represent the completion time intervals that need to be considered in scheduling.

*Step* 2: Job evaluation based on variance. The action calculates and sorts jobs based on their processing time variance ($Var_j$). Processing times are represented as columns for each job ($p_{1,1}$, $p_{1,2}$, $p_{1,3}$ for $J_1$, etc.). More uniform the processing time across machines lead to smaller variance values. The CR is formed by selecting $\max\left\{2, \left\lfloor \log_2^{|BF|} - 1 \right\rfloor\right\}$ jobs with the smallest $Var_j$ value and the earliest due dates.

*Step* 3: Weight assignment and final selection. The final selection process, illustrated in Fig. 3(b), involves three weighted calculations.

$w'_{1j}$: Eq. (30) normalizes the variance of candidate jobs' processing time across all machines, where jobs with smaller variance have higher weights.

$w'_{2j}$: Eq. (31) evaluates the difference between the average interval of jobs in the CR and the average completion time interval across all machines. This difference is normalized for all jobs to enable comparison.

$v'_j$: Eq. (32) integrates variance and interval considerations using a coefficient $\alpha$ to incorporate the due date priority $w_{due\_j}$. This achieving a balance between the selecting jobs with appropriate processing time and the prioritizing jobs based on their due date, optimizing both efficiency (minimizing idle time) and timeliness (meeting due dates) in the scheduling process.

$a_4$: ① Determine the key machine $M_{i*}$ based on the smallest $\Delta I_{i*\hat{j}}$ in $(\Delta I_{1\hat{j}}, \Delta I_{2\hat{j}}, \ldots, \Delta I_{m-1\hat{j}})$. ② Sort the jobs in non-decreasing order based on their $di_{i*j}$, where $di_{i*j} = \left| p_{i*j} - \Delta I_{i*\hat{j}} \right|$, $j \in BF$. Add the first $\max\left\{2, \left\lfloor \log_2^{|BF|} - 1 \right\rfloor\right\}$ jobs with the smallest $di_{i*j}$ and the earliest $d_j$ job in the *BF* into the CR. ③ Assign the weight $w''_{2j}$ for the jobs in the CR as follows, and the job with the biggest $v''_j$ is selected as $\pi_j$.

$$w''_{2j} = \frac{1 - \frac{p_{i*+1,j} - \min p_{i*+1,j \in CR}}{\max p_{i*+1,j \in CR} - \min p_{i*+1,j \in CR}}}{\sum_{j \in CR} \left| 1 - \frac{p_{i*+1,j} - \min p_{i*+1,j \in CR}}{\max p_{i*+1,j \in CR} - \min p_{i*+1,j \in CR}} \right|}. \tag{33}$$

$$v''_j = \frac{1}{2}\alpha * \left(w_{1j} + w''_{2j}\right) + (1-\alpha) * w_{due\_j}, j \in CR. \tag{34}$$

The action $a_4$ focuses on selecting jobs whose processing times closely match the time interval on the key machine, incorporating them into the CR. Subsequently, the job with the shortest processing time on the next process of the key machine is selected to minimize the machine idle time. Fig. 4 illustrates the implementation process and effects of this action using a detailed example.

*Step* 1: Key machine identification. As shown in Fig. 4 (a), $a_4$ first identifies the key machine $M_{i*}$ by selecting the machine with the minimum interval $\Delta I_{i\hat{j}}$ in the interval vector $(\Delta I_{1\hat{j}}, \Delta I_{2\hat{j}}, \ldots, \Delta I_{m-1\hat{j}})$. In the example, $M_1$ is identified as the key machine ($i^*$=1).

*Step* 2: Job evaluation based on variance. This step evaluates the difference between the processing time of the jobs in the *BF* and the completion time interval of the key machine using the $di_{i*j}$. A lower $di_{i*j}$ value indicates a closer alignment between the job's processing

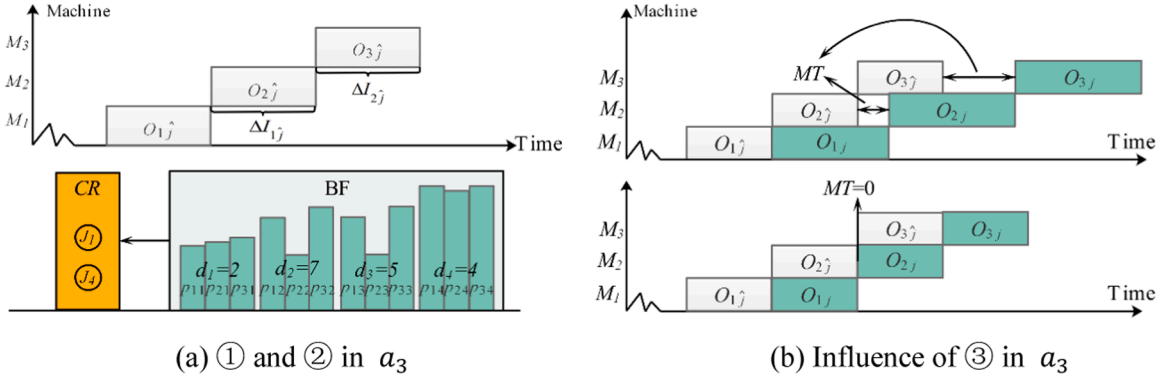



**Fig. 3.** Explanation of the legend for $a_3$.

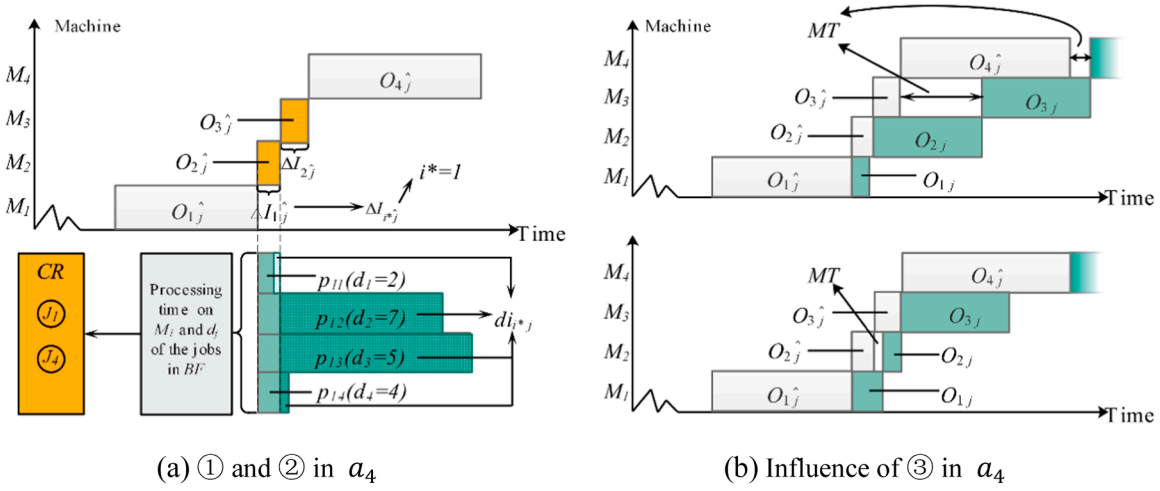



**Fig. 4.** Explanation of the legend for $a_4$.

time and the current processing state. As shown in Fig. 4(a), four jobs are evaluated: $J_1(d_1=2)$, $J_2(d_2=7)$, $J_3(d_3=5)$, $J_4(d_4=4)$. The difference $di_{i*j}$ is computed for comparison, and select the $\max\left\{2, \left\lfloor \log_2^{|BF|} - 1 \right\rfloor\right\}$ jobs with the smallest $di_{i*j}$ to enter the CR.

*Step* 3: Weight assignment and final selection process. Fig. 4(b) demonstrates the effect of weight assignment process using Eqs. (33) and (34):

$w''_{2j}$: Eq. (33) normalizes the processing times of jobs in the CR on the subsequent machine ($i^*+1$) of jobs in the CR, accounting for the impact of a continuous short time interval on the job selection.

$v''_j$: Eq. (34) integrates the alignment between the job processing times and the current completion state with the due date factor to determine the final job priority.

### 4.3. Weighted composite reward function and its feasibility proof

In this section, a weighted composite reward function is designed to evaluate the performance of the agent in executing actions, and its feasibility in optimizing the objective function of the problem is demonstrated.

Assuming the $\pi_j$ has been selected for processing, then the reward $r_j$ is as follows:

$$r_j = -\left[\alpha \sum_{i=1}^{m} \int_{S_{1j}}^{C_j} \delta_i(t)dt + (1-\alpha) \int_{S_{1j}}^{C_j} \psi_j(t)dt\right]. \tag{35}$$

Where $\delta_i(t)$ and $\psi_j(t)$ are indicator functions for $M_i$ and $J_j$ at time $t$, respectively, as follows.

$$\delta_i(t) = \begin{cases} 0, M_i \text{ is busy at time } t \\ -1, M_i \text{ is idle at time } t \end{cases}. \tag{36}$$

$$\psi_j(t) = \begin{cases} 0, S_{1j} \le t < \min\{C_j, d_j\} \\ -1, \min\{C_j, d_j\} \le t \le C_j \end{cases}. \tag{37}$$

The composite reward function Eq. (35) serves two primary purposes through its weighted components:

The first term, weighted by $\alpha$, evaluates machine utilization efficiency. It measures idle time across all machines ($i$=1 to $m$) during the job processing interval [$S_{1j}$, $C_j$]. The indicator function $\delta_i(t)$ captures the binary state of each machine, assuming a value of -1 during idle time and 0 when the machine is operational.

The second term, weighted by (1-$\alpha$), quantifies job completion performance based on due dates. Using the indicator function $\psi_j(t)$, it measures the cumulative tardiness effect. The function transitions from 0 to -1 when the processing time exceeds the minimum of the completion time ($C_j$) and the due date ($d_j$).

The weighting coefficient $\alpha$ enables a balanced consideration of operational efficiency and timely completion objectives. It allows flexible prioritization between these competing performance metrics.

**Theorem 1.** Minimizing the objective function is equivalent to maximizing the total reward $R$ obtained by running a single experiment.

**Proof:**

Given that the total reward $R$ obtained by running a single experiment is the sum of the reward from $k$ decisions, that is

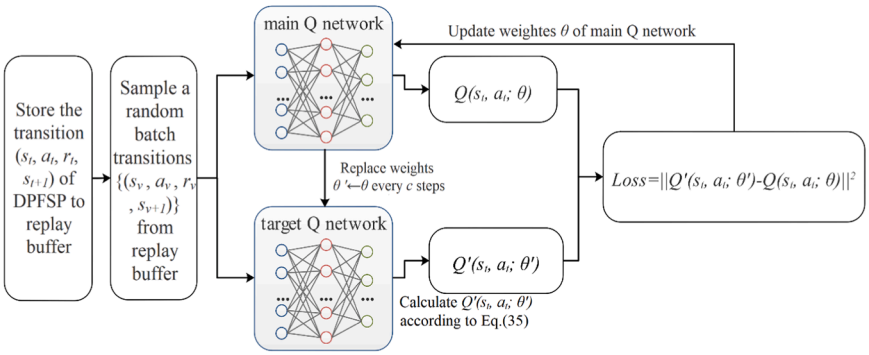



**Fig. 5.** Update process of DDQN in NDRLA_IPC.

$$\begin{aligned} R=\sum_{j=1}^{n} r_j &= -\sum_{j=1}^{n}\left[\alpha\sum_{i=1}^{m}\int_{S_{1j}}^{C_j}\delta_i(t)dt+(1-\alpha)\int_{S_{1j}}^{C_j}\psi_j(t)dt\right] \\ &= -\left[\alpha\sum_{j=1}^{n}\sum_{i=1}^{m}\int_{S_{1j}}^{C_j}\delta_i(t)dt+(1-\alpha)\sum_{j=1}^{n}\int_{S_{1j}}^{C_j}\psi_j(t)dt\right] \\ &= -\left[\alpha\sum_{i=1}^{m}\int_{0}^{C_{\max}}\delta_i(t)dt+(1-\alpha)\sum_{j=1}^{n}\int_{S_{1j}}^{C_j}\psi_j(t)dt\right]. \end{aligned} \tag{38}$$

The total reward $R$ is expressed as the sum of rewards from individual decisions (Eq. (38)), combining both machine utilization and tardiness components. The first term, weighted by $\alpha$, accounts for machine idle time. The second term, weighted by 1-$\alpha$, captures job tardiness.

Let $\varphi_1 = \{j | C_j \geq d_j, 1 \leq j \leq n\}$, $\varphi_2 = \{j | C_j < d_j, 1 \leq j \leq n\}$. Where $\varphi_1$ represents jobs completed after their due dates, $\varphi_2$ represents jobs completed before their due dates. We have

$$\begin{aligned} R &= -\left\{\alpha\sum_{i=1}^{m} Id_i+(1-\alpha)\left\{\sum_{j\in\varphi_1}\left[\sum_{j=1}^{n}\int_{S_{1j}}^{d_j}\psi_j(t)dt+\int_{d_j}^{C_j}\psi_j(t)dt\right]+\sum_{j\in\varphi_2}\int_{S_{1j}}^{C_j}\psi_j(t)dt\right\}\right\} \\ &= -\left\{\alpha\sum_{i=1}^{m} Id_i+(1-\alpha)\sum_{j\in\varphi_1}\left[0+\int_{d_j}^{C_j}\psi_j(t)dt\right]+0\right\} \\ &= -\left[\alpha\sum_{i=1}^{m} Id_i+(1-\alpha)\sum_{j\in\varphi_1}\int_{d_j}^{C_j}\psi_j(t)dt\right] = -\left[\alpha\sum_{i=1}^{m} Id_i+(1-\alpha)\sum_{j\in\varphi_1}(C_j-d_j)\right], \end{aligned} \tag{39}$$

Through Eq. (39), the reward function is decomposed into the machine idle time component $\alpha\sum_{i=1}^{m} Id_i$ and the tardiness component $(1-\alpha)\sum_{j\in\varphi_1}(C_j-d_j)$, where $Id_i$ is the total idle time of $M_i$.

Given $Id_i = C\sum_{j=1}^{n} p_{ij_{max}}$, and

$$TDT(\boldsymbol{\pi})=\sum_{j=1}^{n}(C_j-d_j)=\sum_{j\in\varphi_1}(C_j-d_j)+\sum_{j\in\varphi_2}(C_j-d_j)=\sum_{j\in\varphi_1}(C_j-d_j). \tag{40}$$

Therefore

$$\begin{aligned} R &= -[\alpha\sum_{i=1}^{m}\left(C_{\max}(\boldsymbol{\pi})-\sum_{j=1}^{n}p_{ij}\right)+(1-\alpha)TDT(\boldsymbol{\pi}) \\ &= -\left[\alpha mC_{\max}(\boldsymbol{\pi})-\alpha\sum_{i=1}^{m}\sum_{j=1}^{n}p_{ij}+(1-\alpha)TDT(\boldsymbol{\pi})\right]. \end{aligned} \tag{41}$$

Due to $\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}$ is a constant, maximizing the total reward $R$ is equivalent to minimizing both $C_{max}$ and $TDT(\boldsymbol{\pi})$, which corresponds to the objective function.

### 4.4. Update process of DDQN in NDRLA_IPC

This section details of the update process of DDQN in NDRLA_IPC, as shown in Fig. 5. Its network structure is shown in Table 4.

DDQN addresses the overestimation issue common in DQN by employing two Q-networks. One Q-network, the main network, selects actions and estimates their values. The other, the target Q-network, is a copy of the main network and provides fixed Q-values for updating the main network. By decoupling action selection and target calculation, DDQN reduces the risk of overestimating action values during updates. During the weight update process, the DDQN algorithm first stores the transition $(s_t, a_t, r_t, s_{t+1})$ by interacting with the environment. It then updates the weights of the main Q network by sampling a random batch of transitions from the replay buffer. The target Q-value $Q'(s_t, a_t; \theta')$ is calculated using Eq. (42). The loss function is derived from the temporal difference error between the main Q-value and the target Q-value, guiding the network towards more accurate value estimation. Specifically, the loss function $L(\theta) = ||(Q'(s_t, a_t; \theta') - Q(s_t, a_t; \theta))^2||$, where $Q'(s_t, a_t; \theta')$ is computed as follows:

$$Q'(st, at; \theta') = \begin{cases} r_t, \ s_t \text{ is terminal} \\ r_t + \gamma Q'(s_{t+1}, argmax_a Q(st+1, a; \theta); \theta'), \text{otherwise} \end{cases} \tag{42}$$

The weights of the target Q network are replaced with the weights of the main Q network every $c$ steps.

**Table 4**
Structure parameters of two Q-networks.

| Layer | Node number | Activation function | Description |
|---|---|---|---|
| Input layer | Machine number-1 | None | Complete states information $s_t$ |
| Hidden layer 1 | 32 | Tanh | None |
| Hidden layer 2 | 32 | Tanh | None |
| Hidden layer 3 | 32 | Tanh | None |
| Output layer | Action number | None | Q value of each action |

### 4.5. Proposed framework of the NDRLA_IPC

This section details the main framework of the NDRLA_IPC, as shown in Algorithm 1, and its flowchart is illustrated in Fig. 6.

As shown in Algorithm 1, the NDRLA_IPC schedules all jobs and generates a processing sequence $\boldsymbol{\pi}$ that optimizes the objective function. Specifically, the NDRLA_IPC extracts the relative interval vector of job completion time across all machines to construct state features. It then selects and executes actions using an $\varepsilon$ -greedy exploration strategy (Lines 3 to 12), balancing exploration of new actions and exploitation of known high-reward actions. The reward for the executed action is computed using the weighted composite reward function, and the transition $(s_t, a_t, r_t, s_{t+1})$ is stored for training. Thereafter, the DDQN is trained by sampling transitions from the replay buffer (Lines 13 and 15), which breaks temporal correlations and stabilizes training.

### 4.6. Implementation example of NDRLA_IPC

To better illustrate how NDRLA_IPC solves DMPFSP, we provide an example of solving a small-scale DMPFSP instance. This example demonstrates the process of state extraction, action execution, operation selection, reward calculation, and Q-table updates. For simplicity, we use Q-learning instead of the original DDQN in NDRLA_IPC. Consider a DMPFSP with three machines and five jobs. In the initial system state (0,0), representing all machines being idle and the *BF* in its initial configuration, the Q-table is empty, with no Q-values assigned. The

**Algorithm 1**
Main framework of the NDRLA_IPC.

**Initialize**: weights of DDQN, empty replay buffer $D$, and the state is $s_0$

**Input**: all the jobs that need to be processed

**Output**: sorted job processing sequence $\boldsymbol{\pi}$

**For** *episode*=1 to MAX_EPISODE **do**
  **While** termination condition is not satisfied **do**
    **If** the current state is $s_0$
      Select the first job $\pi_1$ according to the action settings; // Subsection 4.2 (1)
    **End If**
    Obtain the current state $s_t$ at time $t$; // Section 4.1
    **If** the *rand*$\in$ [0, 1] and *rand*$\leq \varepsilon$
      Select a random action $a_t$;
    **Else**
      Select $a_t = argmax_a\, Q(s_t, a; \theta)$;
    **End If**
    Execute action $a_t$ and add the selected job $\pi_j$ to $\boldsymbol{\pi}$; // Subsection 4.2 (2)
    Calculate the reward $r_t$; and transition to the new state $s_{t+1}$; // Section 4.3
    Store the transition $(s_t, a_t, r_t, s_{t+1})$ into the replay buffer $D$ and update the weights of DDQN; // Section 4.4
    $s_t = s_{t+1}$
    **End If**
  **End While**
**End For**
**Return** $\boldsymbol{\pi}$

scheduling environment begins with three jobs in the *BF*, and one additional job arrives at a scheduled time.

Initial *BF* configuration: The system starts with three jobs in the *BF*, each characterized by their processing times across three operations, due dates, and total processing times:

$J_1$: Processing time [5,4,8], due date 20, total processing time 17.
$J_2$: Processing time [3,6,4], due date 16, total processing time 13.
$J_3$: Processing time [8,3,6], due date 23, total processing time 17.

Future job arrivals: One additional job is scheduled to arrive during the processing period: $J_4$: Processing time [4,7,5], arrival time 8, due date 31, total processing time 16.

Processing time changes: During the scheduling process, the processing time of $J_3$ changes from [8,3,6] to [8,5,7] at time 7, and its total processing time increases from 17 to 19.

Machine breakdowns: Machine $M_2$ breaks down after processing two consecutive operations. A repair period of 1 time unit is required, after which $M_2$ resumes normal operation and can process subsequent operations.

Initially, the first job is selected based on the method described in Subsection 4.2 (1). The *BT* values of each job are [14,12,19]. Using Eq. (23), the corresponding weight $w_{BT}$ is calculated to be [0.417,0.583,0]. Next, the due dates of each job in the *BF* are considered, which are [20,16,23]. According to Eq. (24), the corresponding final weight $w_{due\_j}$ is computed to be [0.3,0.7,0]. The final weight $v_j^1$ for each job is then determined by combining weights $w_{BT}$ and $w_{due\_j}$ using the Eq. (25). Applying Eq. (26), the final weights for each job are calculated as follows: $v_1^1$=0.8*0.417+0.2*0.3=0.394, $v_2^1$=0.8*0.583+0.2*0.7=0.606, $v_3^1$=0.8*0+0.2*0=0. According to the calculated weights $v_j^1$, $J_2$ is selected as the first job to be processed.

Following the initial job selection, the system state evolved as follows. The current completion times of the three machines are [3,9,13], and the machine completion time interval vector is [6,4], calculated using Eq. (20). This vector is then normalized using Eq. (21) to obtain the current state [0.6, 0.4] as described in Eq. (22). At this stage, $J_1$ and $J_3$ are retained in the *BF*.

To select the next job, an action is randomly executed ($a_1$) due to the empty Q-table, as detailed in Subsection 4.2(2). The first machine has

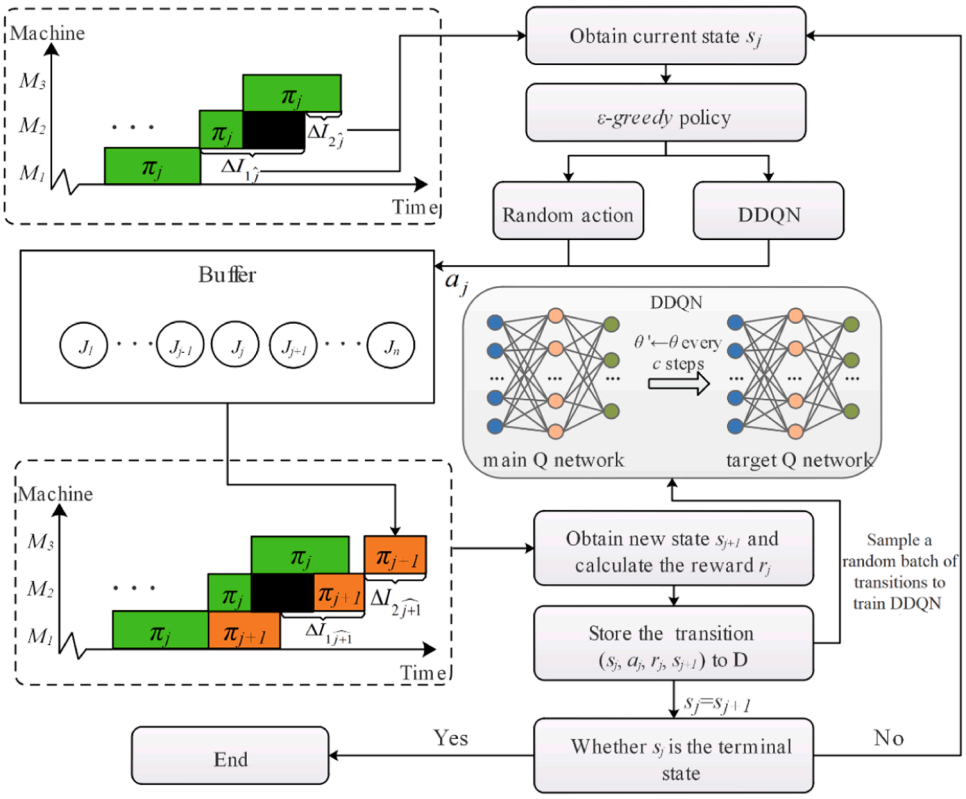



**Fig. 6.** Flow chart of NDRLA_IPC.

the largest completion time interval, and jobs whose processing time are most similar to this interval, $J_1$ and $J_3$, are added to the CR. Here $i*=1$, $di_{i*,1}=1$, $di_{i*,2}=2$. Based on Eqs. (26), (27), and (24), the weights are calculated as $w_{11}=1$, $w_{13}=0$, $w_{21}=1$, $w_{23}=0$, $w_{due\_1}=0.5$, and $w_{due\_3}=0.5$. Finally, the $v_j$ for each job in the CR is obtained using Eq. (29), resulting in $v_1=0.9$ and $v_3=0.1$. Therefore, $J_1$, which has the largest $v_j$, is selected for processing.

After completing the processing of $J_1$, the completion time intervals across all machines are updated to [5,8], and the corresponding state vector becomes [0.385, 0.615]. However, after accounting for a breakdown of machine $M_2$ following the completion of two operations and its subsequent one-time-unit repair period, the completion time interval is adjusted to [6,7]. Consequently, the state vector is updated to [0.462, 0.538]. For the purpose of demonstration, the proposed algorithm replaces DDQN in the NDRLA_IPC framework with Q-learning. The Q-value update follows the standard Q-learning procedure. At this stage, the Q-value for the current state is initially 0.0, and the maximum Q-value in next state is also 0.0. The reward is calculated based on the weighted sum of machine idle time and tardiness following the Eq. (35). Specifically, the idle time between the current job $J_1$ and last job $J_2$ is 0 ($\alpha$ is 0.8), and the tardiness increases by 4 (1-$\alpha$ is 0.2). Thus, the reward is computed as $r=-(0\times0.8+4\times0.2)=-0.8$. Assuming a learning rate ($lr$) of 0.001 and a discount factor ($\gamma$) of 0.99, Eq. (42) is modified to update the Q value according to the following formula: $Q(s,a)=Q(s,a)+lr*[r+\gamma\times\max(Q(s',a'))-Q(s,a)]=0+0.001[-0.8+0.99\times0-0]=-0.0008$. Thus, the current Q-table is updated as shown in Table 5.

When $M_1$ completes processing $J_1$ at time 8, the *BF* contains the remaining jobs $J_3$ and $J_4$, as $J_4$'s arrival time coincides with this completion time. The subsequent action is selected based on the Q-table values, which will determine the next job to be processed. Since the processing time of $J_3$ has changed, Eqs. (26) and (34) are used to calculate the job weight based on the updated processing time of $J_3$. Once the next job is selected and processed, the reward is calculated, and the Q-table is updated accordingly. This process is repeated until all jobs have been successfully scheduled.

**Table 5**
Q table in a concrete example demonstrating the algorithm flow.

| Current state | $a_1$ | $a_2$ | $a_3$ | $a_4$ |
|---|---|---|---|---|
| [0.6, 0.4] | -0.0008 | 0 | 0 | 0 |
| [0.462, 0.538] | 0 | 0 | 0 | 0 |

## 5. Computational tests

This section evaluates the effectiveness of NDRLA_IPC in solving DMPFSP. It includes details on the experimental setup and parameter settings, the reward convergence process, the comparative evaluation of action variants, the comparisons of NDRLA_IPC and classical scheduling rules, the comparisons of NDRLA_IPC and state-of-the-art methods, and the comparison with Gurobi and state-of-the-art methods on small-scale instances.

### 5.1. Experimental setup and parameter settings

The test instances used in this study are sourced from well-known benchmark datasets commonly used in scheduling research, as referenced in [41]. The machine scale considered are {3, 5, 10}, and the job scale are {10, 20, 30, 40, 50}. All training processes and comparative experiments are conducted on a personal computer running the Windows 10 operating system and using Python 3.7. The computer is configured with an Intel(R) Core(TM) i7-12700K 3.6-GHz CPU, a GeForce RTX 3080Ti GPU, and 32-GB RAM. Carefully selected execution tool ensures the accuracy and reliability of the experiments.

In all test results, the average relative percentage deviation (*ARPD*) and standard deviation (*SD*) obtained from repeated experiments with $L=10$ runs are used as response variables to evaluate algorithm performance, as shown below.

$$ARPD = \frac{1}{L}\sum_{l=1}^{L}\left(\frac{C^l - C^{best}}{C^{best}}\right) \times 100\%, \tag{43}$$

$$SD = \sqrt{\frac{\sum_{l=1}^{L}\left(C^l - \overline{C}\right)^2}{L}}. \tag{44}$$

Here $C^l$ represents the *makespan* achieved by the algorithm in the $l$-th experiment, and $C^{best}$ represents the best *makespan* achieved by all algorithms in a given test instance. *ARPD* measures the relative deviation between the algorithm's results and the best results. It is a widely used indicator for evaluating algorithm performance. A smaller *ARPD* value indicates better performance of the algorithm across all test results. *SD* measures the stability of the algorithm's results in repeated experiments. Similarly, a smaller *SD* value indicates a stronger stability of the algorithm's results.

In DMPFSP, three constraints are considered, namely, machine breakdowns, variations in processing time of jobs, and the arrival of new jobs. For machine failures, we propose a simplified discrete-time model based on the exponential failure distribution model in [42]. While the original model uses failure rate $\lambda$ to determine the mean time to failure (MTTF=$1/\lambda$), our study adopts a simplified discrete-time model to maintain the random failure characteristics. Specifically, we set the machine failure probability to 0.1 per time unit (equivalent to expecting one failure every 10 time units) with a random breakdown duration within [0, 20] time units. This simplification preserves the essential stochastic characteristics while facilitating implementation. For processing time variations of jobs, reference [43] defines four dynamic levels that adjust processing times by adding varying percentages to the estimated duration, requiring recalculations and rescheduling for each dynamic level. To simplify the computation, this study adopts a unified probability and change range approach, preserving the dynamic characteristics while reducing experimental complexity. We assume that job processing times on each machine change randomly within the interval [0, 50] with a probability of 0.1. Following the guidelines specified in reference [33], the parameters TF and RDD, which affect the job due date $d_j$, are set to 0.5. For each test instance, half of the jobs are randomly selected into the buffer, while the other half arrive at the *BF* during processing as new jobs following a Poisson process with arrival rate parameters $\lambda$=0.05, 0.08, and 0.1, as described in [29]. For example, for 20 jobs, their arrival times under these rate parameters are as follows. Note that NDRLA_IPC and the comparison methods run in the same environment, ensuring consistent variations in job processing time, machine breakdown times, and the arrival time and sequence of new jobs. Besides, $\gamma$ is usually set to 0.99.

Fig. 7

### 5.2. Reward convergence process

Due to the ability of NDRLA_IPC to extract the shape features of job completion times across all machines for decision making, it exhibits strong generalization capability. By learning from the decisions based on different completion time shapes generated in moderate-sized instances (e.g., instances with 30 jobs), NDRLA_IPC can effectively solve problems of varying job scales. For different numbers of machines, we train DDQN using instances with 30 jobs for 10,000 episodes. For example, for $m$=3, we train DDQN with instance $m$=3 and $n$=30 to solve problems with $m$=3 and $n$={10, 20, 30, 40, 50}. Similarly, for $m$=5 and $m$=10, the training and solving approaches remain consistent.

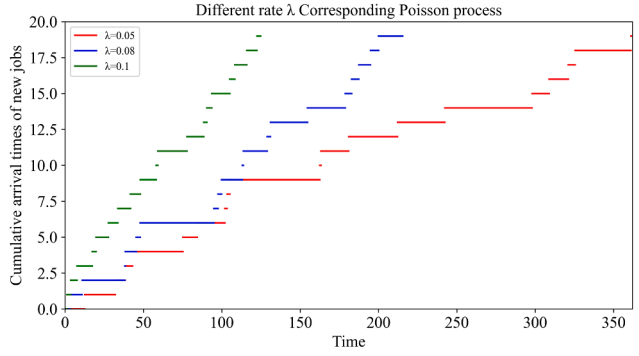



**Fig. 7.** Arrival times of 20 jobs following Poisson processes at different $\lambda$.

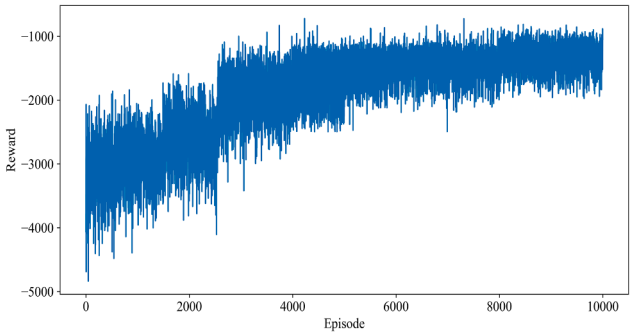



**Fig. 8.** Reward-Episode curve.

The curve of reward variation across episodes is shown below. It can be observed that the reward gradually increases from episode 0 to episode 4000 and converges within the range of [-2000, -1000] after 5000 episodes. This demonstrates that the weighted composite reward function effectively evaluates actions to minimize the objective function.

Fig. 8

### 5.3. Comparative evaluation of action variants

To evaluate the ability of NDRLA_IPC to generate suitable strategies, we designed five variants: NDRLA_IPC_$a_1$ to NDRLA_IPC_$a_4$ and NDRLA_IPC_Rand. (1) The NDRLA_IPC_$a_1$ to NDRLA_IPC_$a_4$ are four variants where NDRLA_IPC_$a_x$ is restricted to a single action $a_x$ in each state. (2) NDRLA_IPC_Rand randomly selects an action from $a_1$ to $a_4$ in each state. The comparison of NDRLA_IPC and its variants verifies that the effectiveness of NDRLA_IPC stems from its ability to generate appropriate strategies, rather than relying solely on the superior performance of individual actions. To further verify the statistical significance of the performance advantage of NDRLA_IPC, we performed Friedman and Wilcoxon signed-rank tests according to the statistical analysis framework described in [44], as shown in Tables 9 and 10. Additionally, the comparative results of five variants and NDRLA_IPC regarding *ARPD* at $\lambda$=0.05, 0.08, and 0.1 are provided in Tables 6 to 8. We summarize the results of all comparative algorithms for $\lambda$ values of 0.05, 0.08, 0.1, as well as for all instances at different $\lambda$. The corresponding box plots are shown in Fig. 9.

The Friedman test results show that NDRLA_IPC achieves the best ranking of 1.0067 among all variants, followed by NDRLA_IPC_$a_4$ (3.7200) and NDRLA_IPC_$a_2$ (3.7767), while NDRLA_IPC_rand performs the worst with a ranking of 4.4133. Additionally, the Wilcoxon signed-rank test results demonstrate that NDRLA_IPC significantly outperforms all its variants at both $\alpha$=0.1 and $\alpha$=0.05 significance levels, with extremely small $p$-values (ranging from 1.0427E-25 to 3.3539E-26). The consistent R+ values around 11320 and R- values of 0 across all pairwise comparisons provide strong statistical evidence that NDRLA_IPC's superior performance is not due to chance but rather stems from its effective integration of state representation and action selection mechanisms.

As shown in Tables 6-8, NDRLA_IPC demonstrates superior performance compared to the other five algorithm variants in almost all instances in terms of *ARPD* and *SD*. Additionally, Fig. 9 shows that NDRLA_IPC exhibits a more competitive statistical distribution than five

**Table 6**
Comparisons between five variants and NDRLA_IPC (New jobs arrival rate $\lambda$=0.05).

| ($\|M\|$, $\|J\|$) | NDRLA_IPC_$a_1$ | | NDRLA_IPC_$a_2$ | | NDRLA_IPC_$a_3$ | | NDRLA_IPC_$a_4$ | | NDRLA_IPC_Rand | | NDRLA_IPC | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* |
| 3_10 | 10.65 | 56.38 | 13.46 | 32.91 | 10.74 | 37.19 | 7.23 | 40.30 | 19.17 | 46.31 | **4.21** | **23.41** |
| 3_20 | 8.79 | 58.56 | 11.20 | 40.33 | 11.69 | 46.16 | 5.61 | **15.88** | 10.18 | 53.29 | **3.12** | 17.55 |
| 3_30 | 13.83 | 36.12 | 10.01 | 69.02 | 7.47 | **21.73** | 10.55 | 77.50 | 12.84 | 78.08 | **3.55** | 32.77 |
| 3_40 | 9.02 | 83.20 | 9.46 | 71.79 | 8.38 | 77.51 | 6.85 | 59.92 | 7.87 | 47.47 | **3.42** | **38.35** |
| 3_50 | 13.56 | 43.48 | 12.07 | 103.75 | 9.75 | 142.11 | 8.95 | **38.96** | 8.38 | 94.21 | **3.98** | 58.52 |
| 5_10 | 15.10 | 34.94 | 12.12 | 48.08 | 13.98 | 36.10 | 16.79 | 50.67 | 18.21 | 41.31 | **5.31** | **31.27** |
| 5_20 | 7.95 | 33.13 | 7.80 | 40.64 | 9.88 | 57.84 | 11.75 | 80.32 | 12.19 | 66.99 | **2.89** | **19.29** |
| 5_30 | 12.05 | 73.13 | 11.43 | 75.77 | 14.37 | **49.76** | 10.26 | 50.75 | 16.11 | 79.41 | **6.35** | 63.85 |
| 5_40 | 12.99 | 100.07 | 11.39 | 47.69 | 10.48 | **25.43** | 9.33 | 137.85 | 12.62 | 51.10 | **2.41** | 46.41 |
| 5_50 | 13.79 | 100.37 | 15.52 | 80.93 | 11.82 | **53.87** | 10.95 | 132.44 | 13.83 | 114.19 | **5.23** | 93.21 |
| 10_10 | 14.82 | 96.19 | 19.97 | 68.50 | 19.95 | 71.96 | 21.26 | 48.36 | 17.04 | 51.59 | **5.93** | **35.97** |
| 10_20 | 15.39 | **31.64** | 15.11 | 50.30 | 16.84 | 53.63 | 17.34 | 73.46 | 18.52 | 65.21 | **9.68** | 57.71 |
| 10_30 | 9.05 | 64.32 | 9.87 | 74.52 | 10.92 | 60.63 | 12.52 | 96.53 | 15.26 | 107.47 | **4.02** | **53.02** |
| 10_40 | 8.64 | 50.83 | 8.23 | 65.23 | 8.38 | **46.42** | 7.80 | 82.28 | 10.12 | 65.99 | **4.17** | 59.58 |
| 10_50 | 8.65 | 122.93 | 9.15 | 89.46 | 9.82 | **52.20** | 9.77 | 93.72 | 11.93 | 69.65 | **4.05** | 80.97 |

**Table 7**
Comparisons between five variants and NDRLA_IPC (New jobs arrival rate $\lambda$=0.08).

| ($\|M\|$, $\|J\|$) | NDRLA_IPC_$a_1$ | | NDRLA_IPC_$a_2$ | | NDRLA_IPC_$a_3$ | | NDRLA_IPC_$a_4$ | | NDRLA_IPC_Rand | | NDRLA_IPC | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* |
| 3_10 | 13.52 | 44.24 | 16.21 | 41.36 | 13.19 | **36.66** | 14.63 | 51.08 | 24.58 | 40.81 | **8.24** | 37.60 |
| 3_20 | 8.65 | 35.04 | 11.95 | 55.11 | 11.17 | 43.29 | 5.87 | 38.02 | 10.81 | 49.85 | **2.80** | **23.99** |
| 3_30 | 20.97 | 32.04 | 19.81 | 60.50 | 17.57 | **14.38** | 17.92 | 102.99 | 20.09 | 59.39 | **11.08** | 60.76 |
| 3_40 | 29.85 | 53.24 | 30.29 | 56.19 | 26.40 | 91.98 | 26.70 | **38.43** | 33.32 | 116.67 | **20.50** | 122.95 |
| 3_50 | 6.95 | 38.64 | 6.64 | 53.77 | 6.49 | 100.9 | 4.00 | 50.01 | 5.73 | 91.22 | **1.32** | **20.00** |
| 5_10 | 22.08 | 44.13 | 16.43 | 50.28 | 22.93 | 37.80 | 20.29 | 33.47 | 23.79 | **33.24** | **11.07** | 46.89 |
| 5_20 | 8.94 | 64.51 | 9.04 | 39.37 | 10.82 | 40.56 | 12.64 | 59.29 | 15.91 | 102.77 | **3.76** | **23.27** |
| 5_30 | 10.85 | 62.44 | 9.32 | 70.21 | 12.11 | 67.92 | 10.87 | 82.78 | 15.99 | 77.64 | **3.45** | **44.72** |
| 5_40 | 11.33 | 66.06 | 8.53 | 69.25 | 8.39 | 46.24 | 6.04 | 73.04 | 13.50 | 118.04 | **2.68** | **44.90** |
| 5_50 | 13.74 | 110.38 | 12.40 | 104.04 | 8.18 | **54.14** | 10.39 | 150.47 | 10.67 | 142.05 | **2.53** | 55.98 |
| 10_10 | 19.84 | 78.58 | 24.00 | 55.33 | 19.39 | 96.34 | 26.94 | 72.26 | 22.94 | 61.70 | **6.73** | **42.06** |
| 10_20 | 9.27 | 76.55 | 12.06 | 70.91 | 11.46 | **35.54** | 10.57 | 55.03 | 12.64 | 85.56 | **4.76** | 46.52 |
| 10_30 | 13.11 | 98.49 | 10.74 | **51.15** | 13.95 | 82.18 | 15.50 | 112.58 | 15.58 | 119.73 | **6.15** | 63.32 |
| 10_40 | 8.40 | 70.71 | 8.08 | 58.40 | 7.84 | 61.05 | 7.95 | 65.60 | 24.58 | 40.81 | **8.24** | **37.60** |
| 10_50 | 6.29 | 65.71 | 8.72 | 116.87 | 7.87 | 51.88 | 14.63 | 51.08 | 10.81 | 49.85 | **2.80** | **23.99** |

**Table 8**
Comparisons between five variants and NDRLA_IPC (New jobs arrival rate $\lambda$=0.1).

| ($\|M\|$, $\|J\|$) | NDRLA_IPC_$a_1$ | | NDRLA_IPC_$a_2$ | | NDRLA_IPC_$a_3$ | | NDRLA_IPC_$a_4$ | | NDRLA_IPC_Rand | | NDRLA_IPC | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* |
| 3_10 | 17.80 | **35.37** | 21.88 | 47.67 | 20.67 | 39.68 | 18.93 | 42.40 | 25.71 | 39.41 | **12.77** | 36.46 |
| 3_20 | 11.42 | 42.37 | 11.74 | 43.15 | 9.48 | 33.28 | 5.84 | 31.56 | 9.31 | 46.31 | **2.08** | **22.26** |
| 3_30 | 23.57 | 72.57 | 22.96 | 62.12 | 18.59 | **30.84** | 23.90 | 129.97 | 27.89 | 90.90 | **13.22** | 68.71 |
| 3_40 | 13.79 | **37.67** | 15.86 | 58.69 | 14.03 | 90.39 | 16.46 | 99.73 | 13.93 | 49.01 | **8.38** | 78.36 |
| 3_50 | 22.33 | **38.38** | 22.46 | 61.92 | 22.57 | 178.07 | 18.98 | 54.59 | 18.95 | 50.56 | **11.84** | 124.93 |
| 5_10 | 17.96 | 68.43 | 16.84 | **48.37** | 19.01 | 66.62 | 18.63 | 60.11 | 16.11 | 61.22 | **7.36** | 50.00 |
| 5_20 | 12.73 | 51.88 | 12.01 | 62.63 | 13.11 | 40.01 | 17.50 | 55.80 | 18.16 | 58.85 | **6.03** | **39.52** |
| 5_30 | 14.20 | 85.42 | 12.71 | 71.09 | 15.41 | **44.26** | 13.80 | 47.21 | 17.05 | 98.18 | **7.56** | 63.24 |
| 5_40 | 15.00 | 82.16 | 17.31 | 94.95 | 14.79 | **69.11** | 11.99 | 130.73 | 17.54 | 109.55 | **7.03** | 77.29 |
| 5_50 | 12.26 | **65.51** | 10.40 | 83.81 | 9.51 | 91.52 | 7.84 | 99.09 | 10.59 | 101.40 | **3.94** | 74.35 |
| 10_10 | 16.87 | 101.86 | 21.77 | 53.60 | 22.13 | 94.41 | 20.65 | 85.99 | 16.26 | 61.90 | **4.77** | **30.80** |
| 10_20 | 15.36 | **47.44** | 18.41 | 99.37 | 15.90 | 61.75 | 18.71 | 96.08 | 16.99 | 72.11 | **10.56** | 84.10 |
| 10_30 | 7.73 | 72.45 | 9.66 | 92.17 | 10.66 | 51.64 | 9.98 | 82.87 | 10.65 | 71.07 | **3.44** | **50.31** |
| 10_40 | 6.36 | 76.04 | 6.20 | 88.88 | 5.46 | 59.81 | 5.99 | 68.07 | 6.54 | 83.13 | **1.79** | **39.04** |
| 10_50 | 7.14 | 80.08 | 8.87 | 95.98 | 7.93 | **35.55** | 7.87 | 139.21 | 10.60 | 103.91 | **3.64** | 56.32 |

algorithm variants. This is due to NDRLA_IPC's consideration of multiple states when designing actions, and establishes correlations between states and actions. Furthermore, the proposed actions are designed to optimize the temporal alignment of jobs from different perspectives. This shared objective contributes to a synergistic effect among the actions, enhancing their collective impact on scheduling performance. Therefore, NDRLA_IPC can select appropriate actions based on the current state, producing stable and efficient scheduling strategies in dynamic environments across various test instances.

In contrast, the other five algorithm variants either perform a single action fixedly or select actions randomly. They not only ignore the job completion time "shape" across all machines but also lead to poor results due to the accumulation of suboptimal actions. Besides, Fig. 9 (d) provides the box plots for the summary of the comparison between the five algorithm variants and NDRLA_IPC on all machines, using different values of $\lambda$ (0.05, 0.08 and 0.1, respectively). This further confirms the

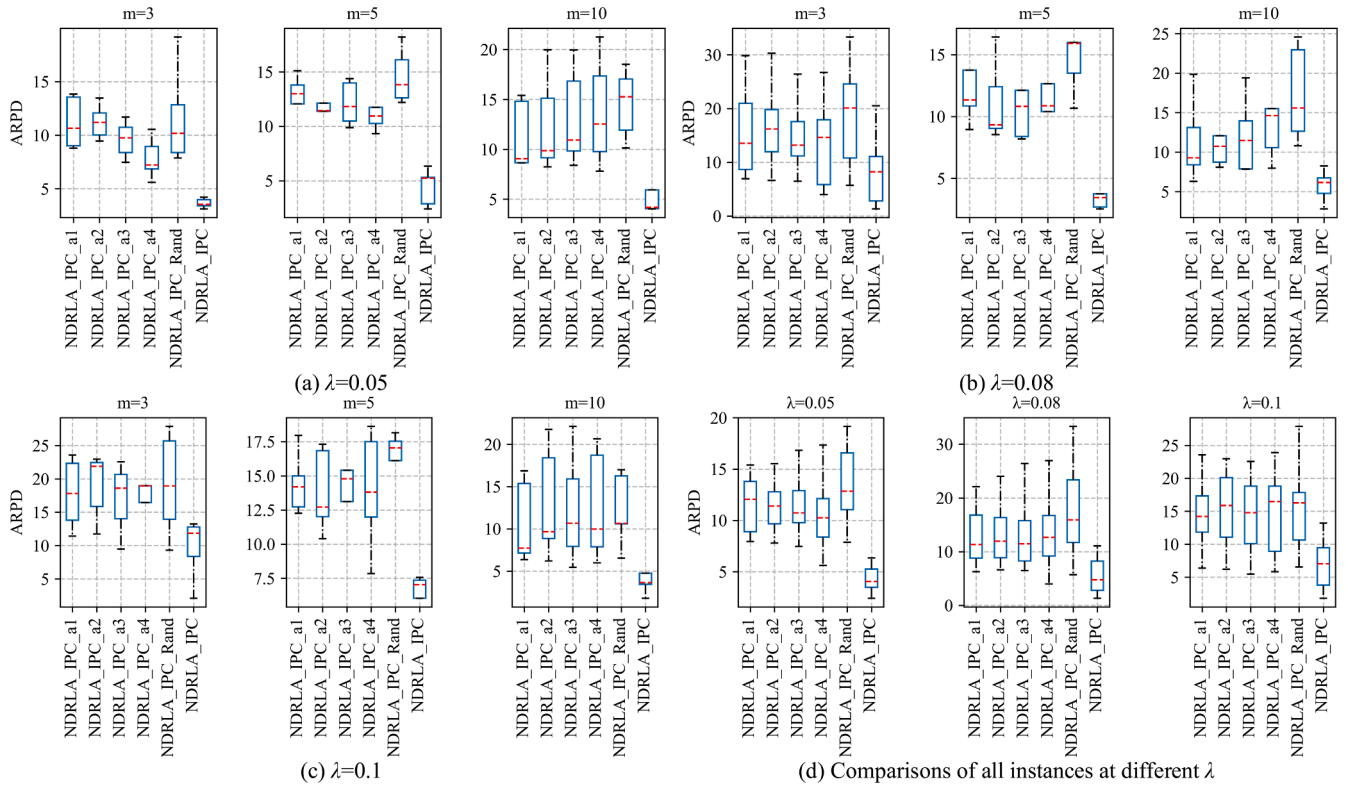



**Fig. 9.** Box plots for the comparisons of five variants and NDRLA_IPC.

competitiveness of NDRLA_IPC.

### 5.4. Comparisons of NDRLA_IPC and classical scheduling rules

To evaluate the effectiveness of NDRLA_IPC, we compare it with classical scheduling rules, including FIFO, LPT, SPT [45], EDD, CR rule [46], and GUPTA [47]. These six scheduling rules are well-known in the field of production scheduling, characterized by fast response, flexible application, and excellent performance. Thus, they have been widely used to provide satisfactory solutions for various production environments and serve as common benchmarks for evaluating the performance of DRL methods. To rigorously verify the statistical significance of performance differences between NDRLA_IPC and classical scheduling rules, Friedman and Wilcoxon signed-rank tests were conducted, with results presented in Tables 14 and 15. The comparative results of the six scheduling rules and NDRLA_IPC regarding *ARPD* at $\lambda$=0.05, 0.08, and 0.1 are provided inTables 11 to 13. Additionally, the corresponding box plots for all comparative algorithms with regard to $\lambda = 0.05, 0.08, 0.1$, and all instances at different $\lambda$ values are shown in Fig. 10. The heat maps in terms of *ARPD* for NDRLA_IPC and the six scheduling rules over all instances are shown in Fig. 11. Furthermore, the heat maps in terms of the weighted sum of *ARPD* and *SD* are shown in Fig. 12, with darker red colors indicating better performance.

The Friedman test rankings reveal NDRLA_IPC's clear superiority with a ranking of 1.0000, substantially outperforming all traditional scheduling rules. The GUPTA ranks second with 2.9667, while other rules show considerably higher rankings ranging from 3.7900 to 4.3733, with SPT achieving the least favorable ranking. The Wilcoxon signed-rank test results reinforce these findings, showing that NDRLA_IPC consistently outperforms all classical scheduling rules with statistically significant differences at both $\alpha$=0.1 and $\alpha$=0.05 confidence levels. The uniformly high R+ values of 11325.0 and R- values of 0.0, coupled with extremely small *p*-values (ranging from 2.9236E-25 to 2.2977E-26), provide robust statistical evidence that NDRLA_IPC's performance advantages are systematic and reliable across all test instances. These statistical results quantitatively support the superior capability of NDRLA_IPC in handling dynamic scheduling scenarios compared to traditional scheduling rules.

**Table 9**
Friedman's test ranking of NDRLA_IPC and variants.

| Algorithm | Ranking |
|---|---|
| NDRLA_IPC | 1.0067 |
| NDRLA_IPC_$a_4$ | 3.7200 |
| NDRLA_IPC_$a_2$ | 3.7767 |
| NDRLA_IPC_$a_3$ | 3.9567 |
| NDRLA_IPC_$a_1$ | 4.1067 |
| NDRLA_IPC_Rand | 4.4133 |

**Table 10**
Wilcoxon signed-rank test results for NDRLA_IPC and variants.

| NDRLA_IPC VS | R+ | R- | *p*-value | $\alpha$=0.1 | $\alpha$=0.05 |
|---|---|---|---|---|---|
| NDRLA_IPC_$a_1$ | 11324.0 | 0.0 | 3.3539E-26 | YES | YES |
| NDRLA_IPC_$a_2$ | 11322.0 | 0.0 | 4.8958E-26 | YES | YES |
| NDRLA_IPC_$a_3$ | 11324.0 | 0.0 | 3.3543E-26 | YES | YES |
| NDRLA_IPC_$a_4$ | 11315.0 | 0.0 | 1.0427E-25 | YES | YES |
| NDRLA_IPC_Rand | 11325.0 | 0.0 | 2.2975E-26 | YES | YES |

In terms of *ARPD*, it can be observed that NDRLA_IPC outperforms six scheduling rules from Tables 11-13. Additionally, Figs. 10 and 11 illustrate the competitive results of NDRLA_IPC under statistical distributions. This is because NDRLA_IPC, based on the processing constraints of DMPFSP, can select the jobs whose processing times align with the "shape" of the current job's completion times across all machines. This results in a more compact and efficient job arrangement on the Gantt chart, enhancing the overall scheduling. Compared to traditional heuristic rules, NDRLA_IPC exhibits greater flexibility and generalization ability, enabling it to output appropriate strategies based on the characteristics of the processing environment and jobs.

However, in terms of *SD*, NDRLA_IPC does not achieve the optimal

**Table 11**
Comparisons between six scheduling rules and NDRLA_IPC (New jobs arrival rate $\lambda$=0.05).

| ($\|M\|$, $\|J\|$) | FIFO | | LPT | | SPT | | EDD | | CR rule | | GUPTA | | NDRLA_IPC | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* |
| 3_10 | 17.14 | 40.28 | 14.39 | 19.56 | 18.67 | **9.24** | 15.23 | 29.04 | 17.31 | 24.55 | 8.75 | 22.95 | **5.73** | 27.54 |
| 3_20 | 9.56 | 42.57 | 12.08 | 19.90 | 13.46 | **0.00** | 9.94 | 46.17 | 10.53 | 30.99 | 10.07 | 19.74 | **3.75** | 21.09 |
| 3_30 | 10.95 | 64.70 | 13.22 | 22.47 | 9.33 | **0.00** | 12.76 | 53.93 | 12.19 | 44.76 | 6.47 | 48.24 | **3.83** | 35.91 |
| 3_40 | 5.90 | 53.01 | 6.10 | 2.45 | 6.76 | **0.00** | 5.69 | 44.69 | 6.11 | 22.47 | 7.70 | **0.00** | **1.81** | 28.83 |
| 3_50 | 3.55 | 36.35 | 7.40 | 11.08 | 7.31 | **0.00** | 5.10 | 97.58 | 6.04 | 48.70 | 3.34 | **0.00** | **1.58** | 30.34 |
| 5_10 | 21.18 | 51.66 | 16.27 | 23.99 | 20.51 | **17.15** | 21.34 | 37.41 | 18.86 | 31.05 | 12.71 | 27.50 | **6.72** | 37.80 |
| 5_20 | 13.83 | 58.51 | 20.15 | 33.90 | 18.44 | **30.48** | 13.22 | 47.02 | 12.72 | 56.29 | 21.78 | 29.40 | **6.09** | 44.61 |
| 5_30 | 16.64 | 82.29 | 19.13 | **24.20** | 9.00 | 28.65 | 17.31 | 100.52 | 16.70 | 67.43 | 16.86 | 34.74 | **3.33** | 49.59 |
| 5_40 | 14.35 | 95.26 | 15.06 | 28.23 | 8.41 | **23.73** | 12.37 | 84.94 | 14.79 | 106.77 | 10.72 | 46.54 | **2.38** | 55.33 |
| 5_50 | 9.20 | 90.38 | 11.21 | 38.76 | 7.19 | 75.04 | 9.86 | 79.21 | 8.82 | 43.96 | 7.46 | 58.20 | **1.48** | **27.20** |
| 10_10 | 10.34 | 54.93 | 12.75 | 72.58 | 16.79 | 22.34 | 9.43 | 49.41 | 6.26 | 51.93 | 8.17 | 25.29 | **1.18** | **16.81** |
| 10_20 | 16.10 | 72.26 | 10.50 | 53.13 | 19.20 | **5.61** | 14.29 | 76.37 | 11.74 | 74.20 | 19.98 | 53.72 | **4.71** | 33.87 |
| 10_30 | 11.57 | 85.25 | 5.92 | 70.14 | 13.57 | **6.80** | 11.62 | 90.21 | 8.63 | 58.57 | 11.79 | 39.83 | **2.75** | 32.62 |
| 10_40 | 9.24 | 77.86 | 4.48 | 44.66 | 15.20 | **29.63** | 8.64 | 116.19 | 7.38 | 77.55 | 8.62 | 37.05 | **2.57** | 48.24 |
| 10_50 | 10.60 | 144.35 | 5.90 | 26.85 | 13.13 | **23.05** | 8.81 | 107.09 | 6.98 | 85.48 | 7.39 | 43.61 | **3.03** | 57.56 |

**Table 12**
Comparisons between six scheduling rules and NDRLA_IPC (New jobs arrival rate $\lambda$=0.08).

| ($\|M\|$, $\|J\|$) | FIFO | | LPT | | SPT | | EDD | | CR rule | | GUPTA | | NDRLA_IPC | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* |
| 3_10 | 12.77 | 24.70 | 17.27 | 48.40 | 18.12 | **0.00** | 13.46 | 37.81 | 17.35 | 46.08 | 4.01 | 29.30 | **1.88** | 12.14 |
| 3_20 | 6.86 | 33.13 | 9.20 | 18.62 | 10.42 | **0.00** | 13.66 | 56.18 | 10.62 | 64.28 | 6.15 | 17.67 | **2.62** | 16.76 |
| 3_30 | 11.33 | 51.12 | 13.21 | 22.63 | 9.68 | **0.00** | 10.38 | 55.08 | 11.75 | 52.31 | 7.10 | 52.15 | **3.08** | 25.17 |
| 3_40 | 6.83 | 72.03 | 6.81 | 2.00 | 7.53 | **0.00** | 7.08 | 66.44 | 6.90 | 47.34 | 6.67 | 28.01 | **2.02** | 33.01 |
| 3_50 | 4.40 | 58.29 | 7.69 | 15.21 | 6.95 | **0.00** | 4.06 | 57.17 | 5.92 | 51.28 | 4.76 | 35.74 | **1.07** | 32.31 |
| 5_10 | 24.64 | 47.53 | 20.97 | 27.06 | 24.93 | **14.32** | 24.19 | 42.70 | 20.40 | 53.15 | 14.54 | 34.53 | **12.19** | 39.70 |
| 5_20 | 15.80 | 75.10 | 18.77 | **27.99** | 16.75 | 32.77 | 16.86 | 65.46 | 14.94 | 73.28 | 20.66 | 19.15 | **4.58** | 47.42 |
| 5_30 | 16.55 | 109.20 | 22.22 | **21.90** | 9.59 | 23.18 | 17.32 | 93.36 | 20.61 | 60.40 | 14.91 | 34.71 | **5.09** | 42.39 |
| 5_40 | 15.29 | 84.21 | 15.85 | 21.99 | 8.73 | **14.89** | 15.70 | 94.58 | 16.39 | 73.29 | 8.45 | 46.95 | **2.65** | 67.73 |
| 5_50 | 11.08 | 118.30 | 11.12 | 19.68 | 7.39 | **18.75** | 10.88 | 96.05 | 9.92 | 109.34 | 4.76 | 35.74 | **2.77** | 60.38 |
| 10_10 | 13.00 | 53.44 | 15.11 | 78.63 | 19.66 | **4.77** | 16.29 | 74.67 | 11.65 | 58.82 | 7.28 | 12.88 | **3.15** | 15.58 |
| 10_20 | 10.90 | 84.84 | 6.59 | 40.37 | 15.15 | **22.77** | 12.04 | 76.78 | 8.21 | 51.00 | 19.28 | 49.37 | **2.26** | 22.33 |
| 10_30 | 7.38 | 84.20 | 2.88 | 68.13 | 12.12 | **25.36** | 9.23 | 134.63 | 5.58 | 79.72 | 12.50 | 53.15 | **0.95** | 13.96 |
| 10_40 | 14.19 | 101.95 | 9.73 | 97.75 | 17.76 | **17.37** | 13.98 | 121.82 | 11.56 | 101.33 | 8.08 | 22.38 | **6.09** | 63.36 |
| 10_50 | 12.94 | 127.53 | 8.01 | 7.29 | 15.00 | **23.95** | 11.10 | 77.23 | 8.40 | 51.27 | 9.83 | 44.12 | **3.95** | 82.03 |

**Table 13**
Comparisons between six scheduling rules and NDRLA_IPC (New jobs arrival rate $\lambda$=0.1).

| ($\|M\|$, $\|J\|$) | FIFO | | LPT | | SPT | | EDD | | CR rule | | GUPTA | | NDRLA_IPC | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* |
| 3_10 | 18.87 | 38.88 | 14.79 | 17.00 | 18.12 | **0.00** | 12.94 | 29.18 | 18.53 | 41.42 | 4.47 | 27.16 | **3.95** | 21.01 |
| 3_20 | 10.94 | 59.30 | 9.20 | 18.62 | 10.42 | **0.00** | 11.36 | 51.46 | 9.37 | 58.18 | 6.05 | 17.50 | **2.07** | 16.69 |
| 3_30 | 10.28 | 25.91 | 11.98 | 26.72 | 8.78 | **0.00** | 12.09 | 47.53 | 13.65 | 44.63 | 6.55 | 45.18 | **3.42** | 40.19 |
| 3_40 | 7.84 | 59.69 | 6.10 | 2.45 | 6.76 | **0.00** | 6.29 | 76.49 | 6.37 | 26.86 | 6.16 | 13.31 | **2.16** | 19.92 |
| 3_50 | 2.75 | 34.90 | 7.49 | 12.00 | 7.15 | **0.00** | 6.59 | 65.74 | 7.84 | 48.51 | 3.31 | 37.22 | **1.69** | 30.42 |
| 5_10 | 22.46 | 52.56 | 18.19 | 18.60 | 24.06 | **15.57** | 20.83 | 47.55 | 20.32 | 49.06 | 11.80 | 35.29 | **7.04** | 36.85 |
| 5_20 | 11.89 | 77.80 | 17.24 | 27.90 | 15.76 | 31.20 | 20.56 | 57.68 | 15.62 | 84.62 | 22.01 | 30.37 | **3.40** | **23.68** |
| 5_30 | 14.57 | 91.19 | 23.60 | 32.35 | 10.72 | **13.22** | 16.53 | 116.46 | 17.27 | 65.09 | 15.91 | 16.00 | **5.82** | 45.15 |
| 5_40 | 14.18 | 119.85 | 16.25 | 24.80 | 9.28 | 30.75 | 16.36 | 80.35 | 16.78 | 53.83 | 8.19 | 48.67 | **1.07** | **17.58** |
| 5_50 | 9.55 | 44.34 | 10.07 | **22.45** | 5.93 | 36.07 | 11.95 | 94.35 | 12.44 | 81.25 | 6.73 | 37.38 | **1.99** | 40.11 |
| 10_10 | 17.87 | 51.74 | 20.09 | 79.07 | 25.50 | **14.99** | 19.67 | 71.52 | 18.22 | 51.16 | 8.63 | 19.08 | **7.21** | 38.38 |
| 10_20 | 14.40 | 69.71 | 8.70 | 63.87 | 16.66 | **5.17** | 9.11 | 69.09 | 7.87 | 51.01 | 19.29 | 45.31 | **3.33** | 21.42 |
| 10_30 | 12.24 | 91.92 | 5.03 | **30.28** | 13.92 | 36.46 | 9.68 | 93.12 | 7.05 | 59.49 | 10.74 | 38.02 | **3.01** | 32.78 |
| 10_40 | 10.80 | 110.75 | 4.55 | 59.33 | 12.53 | **20.85** | 9.13 | 111.21 | 6.61 | 104.56 | 7.83 | 23.68 | **1.58** | 27.61 |
| 10_50 | 8.67 | 98.31 | 5.81 | **7.72** | 12.45 | 13.62 | 9.60 | 94.78 | 7.50 | 71.98 | 8.47 | **39.52** | **2.54** | 75.19 |

results in all instances, as SPT obtained smaller *SD* values in 77.8% of instances. This is due to the fixed and limited nature of SPT, which selects the job with the shortest processing time at each decision point. In this scenario, each set of experiments randomly pre-selects half of the job scale of the given instance in the buffer, leading to overlapping job selections in repeated experiments and resulting in more similar results. Consequently, SPT performs better performance in terms of *SD* but poorer performance in *ARPD*.

In terms of the combined aspects of *ARPD* and *SD*, Fig. 12 shows that NDRLA_IPC consistently exhibits a redder color than the other methods, indicating its superior overall performance. Additionally, the CPU times for all methods remain within 0.5 seconds. Therefore, NDRLA_IPC demonstrates superior comprehensive performance in the DMPFSP environment while maintaining the same fast response speed as scheduling rules.

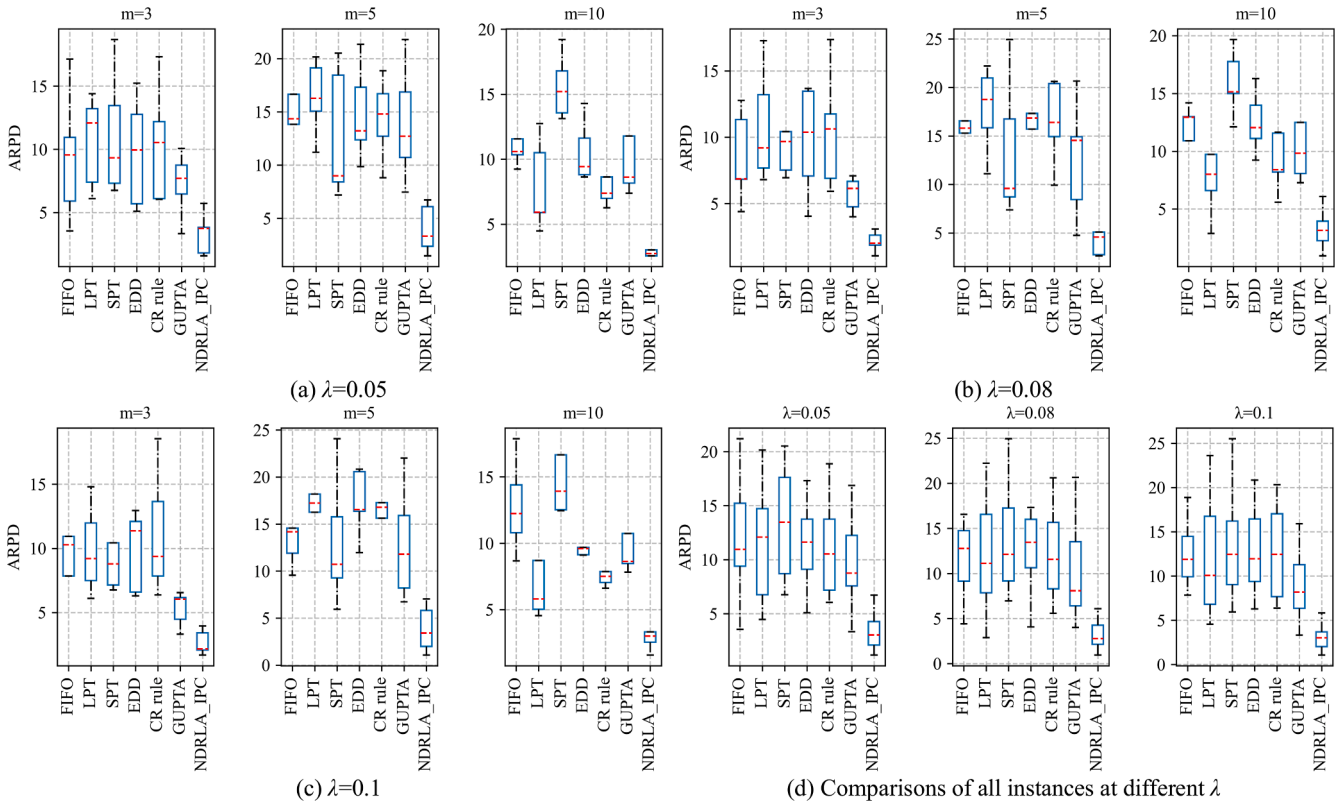



**Fig. 10.** Box plots for the comparisons of six scheduling rules and NDRLA_IPC.

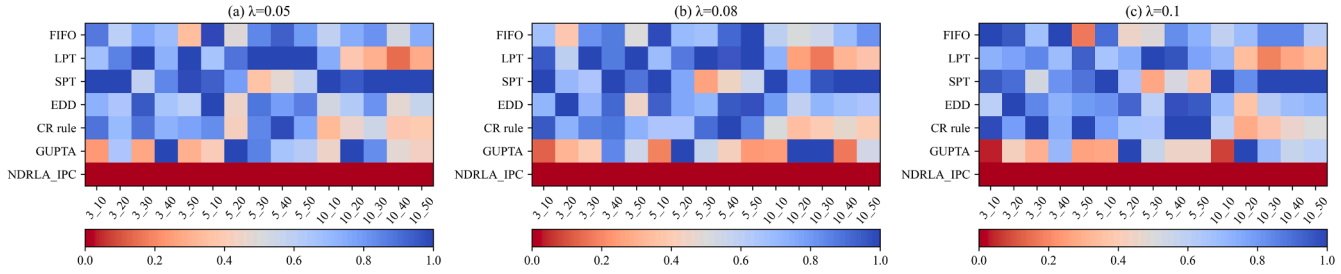



**Fig. 11.** Heat maps of *ARPD* for all comparisons between six scheduling rules and NDRLA_IPC at different $\lambda$.

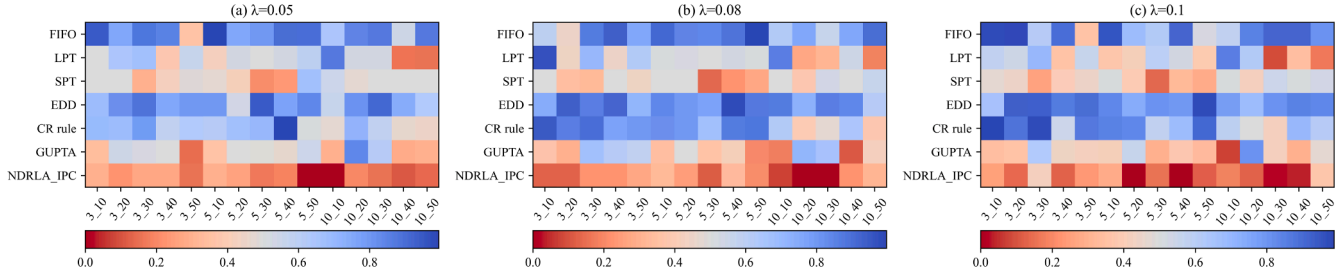



**Fig. 12.** Heat maps of the composite values (*ARPD:SD* = 0.5:0.5) for all comparisons between six scheduling rules and NDRLA_IPC at different $\lambda$.

### 5.5. Comparisons of NDRLA_IPC and state-of-the-art methods

To further assess the performance of NDRLA_IPC, the most recent state-of-the-art DMPFSP methods, including IG [48], GSH [18], and SA-NET [33], are used for comparison with the proposed algorithm. To the best of our knowledge, these three algorithms are among the best performing DMPFSP algorithms for testing RL-based solution methods currently available in the literature. IG is an iterative optimization algorithm that is commonly used. It obtains sequence results through iterative use of deconstruction and insertion strategies. Similarly, GSH uses insertion strategies to explore better sequence structures to obtain satisfactory solutions for DMPFSP. Additionally, SA-NET is a representative DRL algorithm that takes the characteristics of jobs or machines as states and utilizes various heuristic rules as actions to optimize scheduling plans.

To establish the statistical significance of the comparative results

**Table 14**
Friedman's test ranking of NDRLA_IPC and scheduling rules.

| Algorithm | Ranking |
|---|---|
| NDRLA_IPC | 1.0000 |
| GUPTA | 2.9667 |
| LPT | 3.7900 |
| CR rule | 3.9833 |
| FIFO | 3.9864 |
| EDD | 4.0600 |
| SPT | 4.3733 |

**Table 15**
Wilcoxon signed-rank test results for NDRLA_IPC and scheduling rules.

| NDRLA_IPC VS | R+ | R- | *p*-value | $\alpha$=0.1 | $\alpha$=0.05 |
|---|---|---|---|---|---|
| GUPTA | 11325.0 | 0.0 | 2.9236E-25 | YES | YES |
| LPT | 11325.0 | 0.0 | 2.2956E-26 | YES | YES |
| CR rule | 11325.0 | 0.0 | 2.2966E-26 | YES | YES |
| FIFO | 11325.0 | 0.0 | 2.2976E-26 | YES | YES |
| EDD | 11325.0 | 0.0 | 2.2977E-26 | YES | YES |
| SPT | 11325.0 | 0.0 | 2.2962E-26 | YES | YES |

**Table 16**
Friedman's test ranking of NDRLA_IPC and efficient methods.

| Algorithm | Ranking |
|---|---|
| NDRLA_IPC | 1.3567 |
| IG(100) | 2.5767 |
| IG(30) | 2.6067 |
| SA-NET | 3.9000 |
| GSH | 4.5600 |

between NDRLA_IPC and state-of-the-art methods, both Friedman and Wilcoxon signed-rank tests were performed, as shown inTables 16 and 17. The Friedman test results demonstrate that NDRLA_IPC achieves the best ranking of 1.3567, followed by IG(100) and IG(30) with rankings of 2.5767 and 2.6067, respectively. SA-NET ranks fourth with 3.9000, while GSH shows the least competitive performance with a ranking of 4.5600. The Wilcoxon signed-rank test results further confirm NDRLA_IPC's advantages with statistical significance at both $\alpha$=0.1 and $\alpha$=0.05 levels. The consistently positive R+ values (ranging from 9180.0 to 11325.0) and R- values of 0.0, along with extremely small p-values (from 1.1672E-15 to 2.2486E-26), provide strong statistical evidence for NDRLA_IPC's superior performance across all test instances. These statistical findings quantitatively validate that NDRLA_IPC's improvements over existing state-of-the-art methods are both substantial and reliable.

For the three reference algorithms, we adopt the parameter settings found in the literature. Among them, the termination iterations for IG are set to 30 and 100. The average CPU time for all algorithms across different test instances is presented in Table 18. Additionally, the comparative results concerning $\lambda$ = 0.05, 0.08, 0.1, and all instances at different $\lambda$ values are reported in Tables 19 to 21, with the corresponding box plots for different machine numbers given in Fig. 13. The heat maps in terms of *ARPD* for NDRLA_IPC and state-of-the-art methods over all instances are shown in Fig. 14. Furthermore, the heat maps in terms of the weighted sum of *ARPD* and *SD* are shown in Fig. 15. And The heat maps in terms of the weighted sum of *ARPD, SD*, and CPU time are shown in Fig. 16, with darker red colors indicating better performance.

From Tables 19-21, it can be observed that NDRLA_IPC achieved the best results in terms of *ARPD* for all instances. Figs. 13 and 14 demonstrate the competitive performance of NDRLA_IPC in *ARPD* statistical distribution. Regarding *SD*, NDRLA_IPC achieves the optimal results in only 24.4% of instances. However, the proposed algorithm achieves

**Table 17**
Wilcoxon signed-rank test results for NDRLA_IPC and efficient methods.

| NDRLA_IPC VS | R+ | R- | *p*-value | $\alpha$=0.1 | $\alpha$=0.05 |
|---|---|---|---|---|---|
| IG(100) | 10545.0 | 0.0 | 1.1672E-15 | YES | YES |
| IG(30) | 9180.0 | 0.0 | 5.9328E-20 | YES | YES |
| GSH | 11319.0 | 0.0 | 7.1439E-26 | YES | YES |
| SA-NET | 11325.0 | 0.0 | 2.2486E-26 | YES | YES |

optimal average results in terms of both overall *ARPD* and *SD*. Furthermore, considering the combined results of *ARPD* and *SD*, Fig. 15 indicates that NDRLA_IPC still exhibits higher overall efficiency compared to other algorithms. Regarding CPU time, as shown in Table 18, both SA-NET and NDRLA_IPC are real-time scheduling algorithms with short and similar CPU times, completing within 0.1 seconds. On the other hand, IG and GSH, as iterative optimization algorithms, are highly sensitive to problem size, requiring longer times to solve larger-scale problems. This results in relatively slower response times.

From the comprehensive results of *ARPD, SD*, and CPU time, Fig. 16 illustrates that NDRLA_IPC achieves the best results in 93.33% of the instances. Although IG, and GSH, based on the insertion operation, are considered excellent iterative algorithms in many COPs, they do not consistently outperform others in this study. This can be attributed to four factors:

1. Dynamic problems like DMPFSP involve frequent dynamic events, leading to a continuously changing and expanding solution search space, which creates challenges for the effectiveness of iterative algorithms.
2. Iterative algorithms make decisions based solely on static problems that are fixed at the current moment, limiting their ability to promptly adapt to dynamic environmental changes.
3. The complexity and uncertainty of the dynamic problem increase the likelihood of iterative algorithms getting trapped in local optima and struggling to escape.
4. Rescheduling with every dynamic event occurrence may result in excessively high scheduling frequencies, increasing computational costs and making it difficult for iterative algorithms to find satisfactory solutions within a reasonable time.

In this context, SA-NET, as a real-time DRL scheduling algorithm, demonstrates greater adaptability and global optimization capabilities, outperforming IG and GSH in instances of $m$=10. Similarly, NDRLA_IPC adopts a more direct state representation approach by extracting the "shape" of job completion times to obtain global processing information. This approach avoids the need for high-dimensional feature extraction in complex dynamic scheduling environments, effectively improving the

**Table 18**
The CPU time for comparisons between state-of-the-art algorithms and NDRLA_IPC.

| ($\|M\|$, $\|J\|$) | IG(30) | IG(100) | GSH | SA-NET | NDRLA_IPC |
|---|---|---|---|---|---|
| 3_10 | 0.023 | 0.065 | 0.033 | 0.048 | 0.001 |
| 3_20 | 0.367 | 1.178 | 0.553 | 0.051 | 0.010 |
| 3_30 | 1.533 | 5.003 | 2.821 | 0.053 | 0.012 |
| 3_40 | 4.624 | 15.081 | 5.482 | 0.057 | 0.022 |
| 3_50 | 13.147 | 43.074 | 15.396 | 0.061 | 0.030 |
| 5_10 | 0.043 | 0.137 | 0.052 | 0.050 | 0.003 |
| 5_20 | 0.429 | 1.374 | 0.634 | 0.053 | 0.010 |
| 5_30 | 2.192 | 7.098 | 4.146 | 0.055 | 0.011 |
| 5_40 | 6.606 | 21.374 | 8.248 | 0.060 | 0.019 |
| 5_50 | 16.765 | 55.967 | 19.215 | 0.065 | 0.026 |
| 10_10 | 0.070 | 0.230 | 0.013 | 0.049 | 0.002 |
| 10_20 | 0.916 | 2.992 | 1.067 | 0.051 | 0.007 |
| 10_30 | 4.275 | 13.934 | 9.463 | 0.055 | 0.024 |
| 10_40 | 14.288 | 47.592 | 18.126 | 0.061 | 0.026 |
| 10_50 | 32.675 | 104.645 | 41.253 | 0.067 | 0.031 |

**Table 19**
Comparisons between state-of-the-art algorithms and NDRLA_IPC (New jobs arrival rate $\lambda$=0.05).

| ($\|M\|$, $\|J\|$) | IG(30) | | IG(100) | | GSH | | SA-NET | | NDRLA_IPC | |
|---|---|---|---|---|---|---|---|---|---|---|
| | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* |
| 3_10 | 9.29 | 25.55 | 10.06 | 29.97 | 10.75 | **19.16** | 11.38 | 25.33 | **6.30** | 19.60 |
| 3_20 | 5.97 | 22.93 | 6.66 | 15.57 | 12.67 | **2.00** | 9.40 | 59.50 | **5.05** | 12.21 |
| 3_30 | 4.71 | 40.38 | 3.90 | 31.12 | 7.80 | **5.40** | 8.23 | 29.95 | **1.20** | 19.62 |
| 3_40 | 4.33 | 41.44 | 4.35 | 33.22 | 8.76 | **0.00** | 4.89 | 74.02 | **2.10** | 27.40 |
| 3_50 | 1.39 | 27.47 | 2.13 | 21.14 | 5.10 | **0.00** | 6.01 | 73.11 | **1.70** | 26.20 |
| 5_10 | 11.45 | 32.81 | 12.21 | 29.32 | 11.10 | 31.75 | 10.71 | 35.49 | **6.03** | **22.37** |
| 5_20 | 6.57 | 38.68 | 7.88 | 41.53 | 11.82 | 26.86 | 13.51 | 46.56 | **3.64** | **26.23** |
| 5_30 | 6.69 | **32.39** | 6.27 | 33.09 | 7.93 | 58.15 | 10.20 | 78.35 | **4.12** | 39.47 |
| 5_40 | 8.33 | 32.84 | 8.71 | 54.34 | 7.64 | **28.19** | 11.15 | 61.78 | **4.18** | 51.81 |
| 5_50 | 3.73 | 99.73 | 3.09 | 45.35 | 4.52 | 56.53 | 4.55 | 86.89 | **1.74** | **29.78** |
| 10_10 | 22.80 | 35.49 | 4.25 | **12.55** | 20.27 | 27.60 | 7.67 | 49.24 | **4.24** | **12.55** |
| 10_20 | 12.85 | 95.04 | 10.66 | 137.33 | 19.75 | 48.88 | 6.15 | 63.64 | **3.83** | **37.95** |
| 10_30 | 10.36 | 120.87 | 9.16 | 64.73 | 14.70 | 60.85 | 7.76 | 102.34 | **3.58** | 56.88 |
| 10_40 | 5.99 | 60.91 | 5.70 | 75.49 | 7.45 | 60.00 | 6.09 | 81.52 | **2.81** | 43.51 |
| 10_50 | 5.07 | 71.95 | 4.05 | 71.56 | 4.82 | 73.64 | 3.30 | **28.14** | **2.21** | 32.53 |
| Average | 7.97 | 51.90 | 6.61 | 46.42 | 10.34 | 33.27 | 8.07 | 59.72 | **3.52** | **30.54** |

**Table 20**
Comparisons between state-of-the-art algorithms and NDRLA_IPC (New jobs arrival rate $\lambda$=0.08).

| ($\|M\|$, $\|J\|$) | IG(30) | | IG(100) | | GSH | | SA-NET | | NDRLA_IPC | |
|---|---|---|---|---|---|---|---|---|---|---|
| | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* |
| 3_10 | 3.27 | 23.15 | 3.27 | 23.15 | 3.69 | **19.58** | 11.04 | 28.63 | **3.01** | 20.46 |
| 3_20 | 2.82 | 23.68 | 1.17 | 14.95 | 8.28 | **2.50** | 12.25 | 56.35 | **0.59** | 9.17 |
| 3_30 | 4.84 | 49.99 | 3.51 | 40.94 | 7.72 | **1.50** | 9.60 | 68.47 | **1.15** | 8.78 |
| 3_40 | 2.32 | 39.87 | 1.20 | 36.06 | 6.52 | **0.00** | 11.23 | 90.67 | **1.13** | 23.69 |
| 3_50 | 1.59 | 27.29 | 1.24 | 24.60 | 5.10 | **0.00** | 7.35 | 79.90 | **0.58** | 15.65 |
| 5_10 | 13.82 | 28.60 | 12.59 | 45.48 | 12.31 | **17.80** | 15.12 | 45.41 | **7.69** | 32.17 |
| 5_20 | 7.31 | 33.40 | 6.10 | 27.74 | 10.18 | **14.09** | 9.79 | 47.11 | **2.96** | 32.16 |
| 5_30 | 5.33 | **32.08** | 6.56 | 43.11 | 6.30 | 32.45 | 9.65 | 60.96 | **2.74** | 45.40 |
| 5_40 | 5.83 | 47.98 | 4.91 | 58.59 | 7.01 | **35.75** | 9.02 | 50.62 | **2.49** | 54.21 |
| 5_50 | 1.59 | 27.29 | 1.24 | 24.60 | 5.10 | **0.00** | 7.35 | 79.90 | **0.58** | 15.65 |
| 10_10 | 18.83 | 62.48 | 5.38 | 40.87 | 18.91 | 23.03 | 6.97 | 34.94 | **3.98** | **8.73** |
| 10_20 | 12.33 | 90.84 | 7.21 | 97.78 | 16.46 | 36.59 | 2.77 | **28.00** | **2.34** | 31.10 |
| 10_30 | 9.90 | 95.85 | 11.90 | 76.81 | 14.70 | 33.90 | 4.11 | **29.83** | **3.71** | 54.20 |
| 10_40 | 5.27 | 58.27 | 6.26 | 55.88 | 7.46 | 41.58 | 4.78 | 55.53 | **2.17** | 29.03 |
| 10_50 | 8.22 | 92.03 | 6.07 | 83.50 | 6.28 | 85.17 | 3.17 | 59.09 | **3.01** | **58.34** |
| Average | 6.88 | 48.85 | 5.24 | 46.27 | 9.07 | **22.93** | 8.28 | 54.36 | **2.54** | 29.25 |

**Table 21**
Comparisons between state-of-the-art algorithms and NDRLA_IPC (New jobs arrival rate $\lambda$=0.1).

| ($\|M\|$, $\|J\|$) | IG(30) | | IG(100) | | GSH | | SA-NET | | NDRLA_IPC | |
|---|---|---|---|---|---|---|---|---|---|---|
| | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* | *ARPD* | *SD* |
| 3_10 | 4.68 | 30.96 | 4.68 | 30.96 | 2.67 | **18.78** | 11.04 | 28.63 | **2.33** | 19.33 |
| 3_20 | 2.81 | 26.97 | 2.58 | 26.81 | 8.18 | **2.29** | 12.25 | 56.35 | **1.85** | 6.36 |
| 3_30 | 3.44 | 45.02 | 1.62 | 23.36 | 7.68 | **0.00** | 9.60 | 68.47 | **1.31** | 13.42 |
| 3_40 | 2.10 | 25.99 | 1.87 | 40.56 | 6.52 | **0.00** | 11.23 | 90.67 | **1.03** | 17.00 |
| 3_50 | 2.20 | 27.53 | 0.89 | 15.64 | 5.10 | **0.00** | 7.35 | 79.90 | **0.61** | 11.14 |
| 5_10 | 13.50 | 45.69 | 10.69 | 58.62 | 22.68 | **14.87** | 12.88 | 45.41 | **4.69** | 23.82 |
| 5_20 | 7.65 | 41.20 | 6.92 | 36.61 | 11.44 | 27.58 | 10.41 | 47.11 | **2.36** | **20.94** |
| 5_30 | 8.78 | 57.23 | 7.33 | 47.65 | 9.35 | 42.13 | 10.95 | 60.96 | **5.00** | 41.96 |
| 5_40 | 4.68 | 47.44 | 4.01 | **25.73** | 6.17 | 50.26 | 8.41 | 50.62 | **2.07** | 31.39 |
| 5_50 | 4.71 | 46.37 | 3.81 | 40.97 | 6.07 | 60.55 | 11.22 | 111.74 | **1.97** | **25.41** |
| 10_10 | 19.06 | 73.59 | 10.04 | 78.81 | 21.79 | 19.70 | 7.51 | 43.88 | **5.97** | **10.24** |
| 10_20 | 7.02 | 63.91 | 11.09 | 99.75 | 17.83 | 50.40 | 4.13 | 57.46 | **1.66** | **27.26** |
| 10_30 | 9.55 | 59.58 | 10.46 | 127.66 | 12.89 | 55.22 | 4.10 | 68.93 | **3.39** | 53.79 |
| 10_40 | 7.06 | 65.96 | 5.72 | 43.55 | 7.84 | 96.03 | 5.12 | 51.56 | **2.44** | 44.89 |
| 10_50 | 5.22 | 75.86 | 6.06 | 69.08 | 8.51 | 94.94 | 3.62 | **57.35** | **3.01** | 64.32 |
| Average | 6.83 | 48.89 | 5.85 | 51.05 | 10.31 | 35.52 | 8.65 | 61.27 | **2.65** | **27.42** |

alignment of neighboring jobs on the Gantt chart and generating stable scheduling strategies.

Based on the provided analysis, NDRLA_IPC can be considered as an effective real-time scheduling method for DMPFSP.

### *5.6. Comparison with Gurobi and state-of-the-art methods on small-scale instances*

To rigorously validate NDRLA_IPC's capabilities under controlled conditions, this section applies it to small-scale DMPFSP instances where optimal solutions can be obtained for comparison. Therefore, three

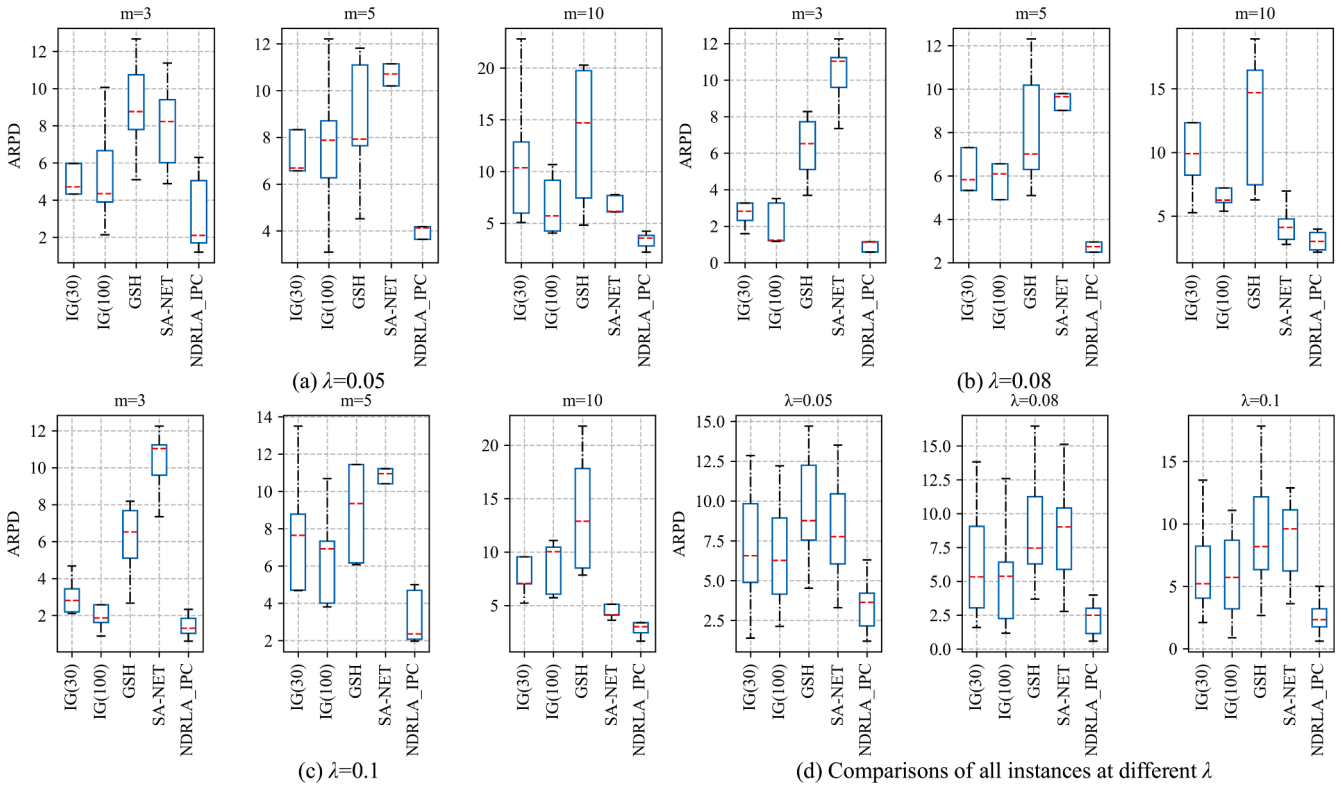



**Fig. 13.** Box plots for the comparisons between state-of-the-art algorithms and NDRLA_IPC.

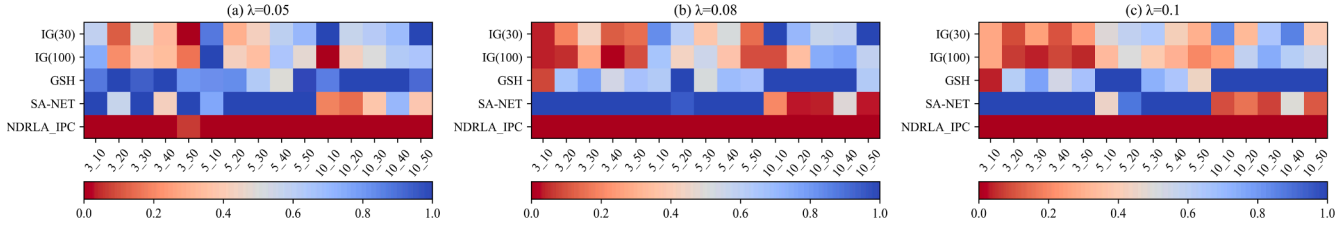



**Fig. 14.** Heat maps of *ARPD* for all comparisons between state-of-the-art algorithms and NDRLA_IPC at different $\lambda$.

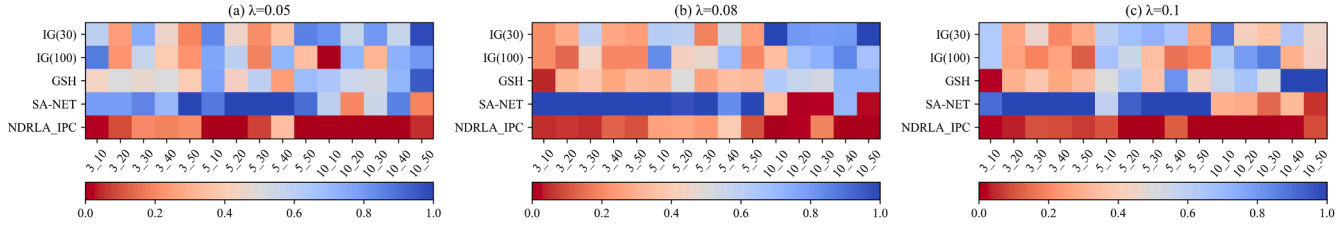



**Fig. 15.** Heat maps of the composite values (*ARPD:SD* = 0.5:0.5) for all comparisons between state-of-the-art algorithms and NDRLA_IPC at different $\lambda$.

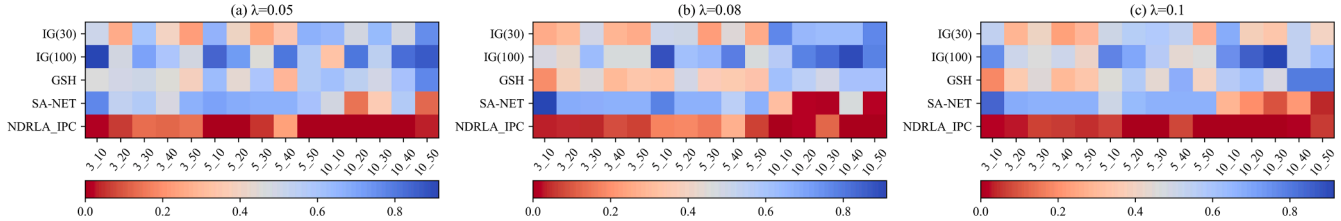



**Fig. 16.** Heat maps of the composite values (*ARPD: SD*: CPU time = 1/3: 1/3: 1/3) for all comparisons between state-of-the-art algorithms and NDRLA_IPC at different $\lambda$.

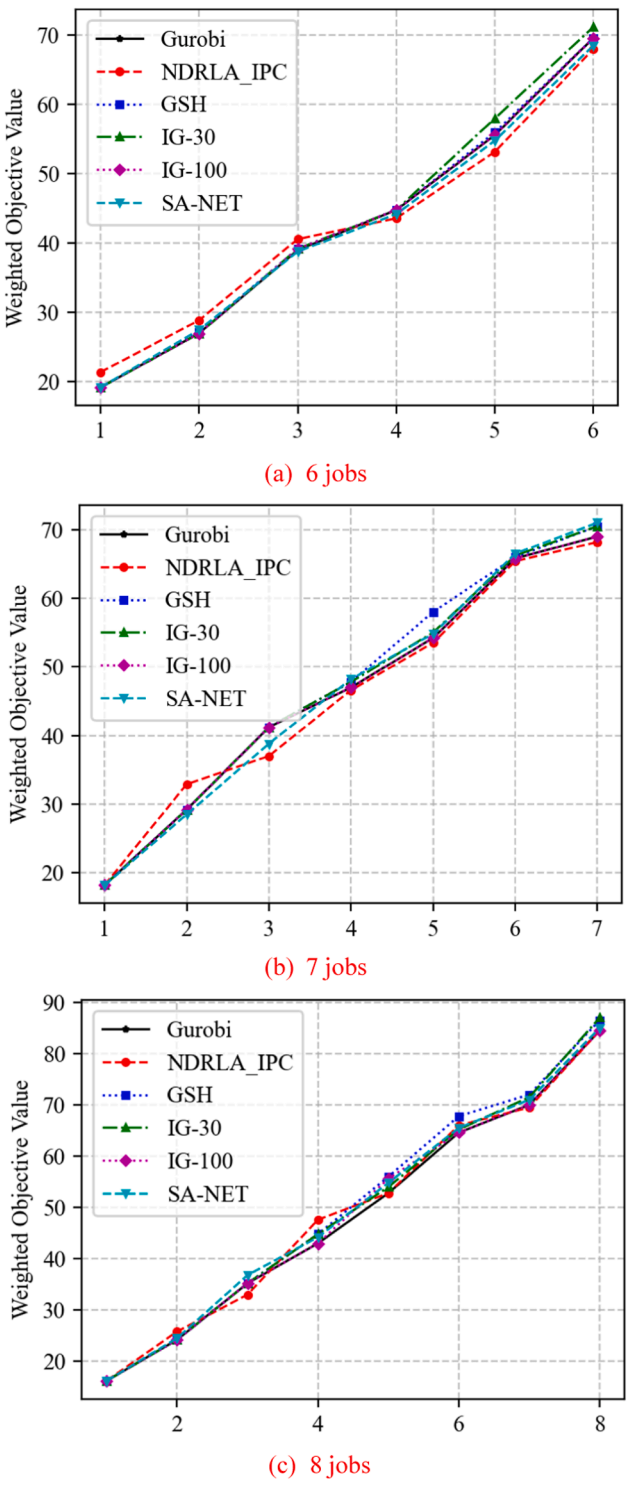



**Fig. 17.** Comparison of the evolution of objective value for sequentially scheduling jobs between approximate and baseline methods in a small-scale dynamic example.

small-scale instances with 6, 7, and 8 jobs on three machines respectively are constructed. For these comparative experiments, Gurobi serves as a theoretical performance benchmark by exhaustively evaluating all possible sequences at each decision point. Meanwhile, NDRLA_IPC is compared with state-of-the-art methods (i.e., GSH, IG variants, and SA-NET) and Gurobi in solving these small-scale instances.

Initially, $\lfloor n/2 \rfloor$ jobs are available in the *BF* at $t$=0, with processing times randomly generated from [5,15] time units. Their due dates are set to 1.2 times their total processing times. The remaining $\lfloor n/2 \rfloor$ jobs arrive dynamically during the scheduling process. For consistency across different problem scales, we designed a scalable arrival pattern where the first dynamic job arrives at 10 time units, followed by subsequent arrivals at 15 time unit intervals. The due dates of these jobs are set to 1.3 times the sum of their arrival time and total processing time. Additionally, random processing time variations occur within the [10,20] time unit window. Fig. 17 (a), (b), and (c) illustrate the objective value progression as jobs are sequentially processed for instances with 6, 7, and 8 jobs, respectively.

The results demonstrate that NDRLA_IPC shows remarkable performance, closely tracking the solutions provided by Gurobi. Compared to other heuristic methods, NDRLA_IPC exhibits superior adaptability to dynamic changes. While GSH and IG variants occasionally match the optimal solution at certain points, they demonstrate greater volatility in solution quality, particularly after dynamic events occur. SA-NET demonstrates reasonable performance as a DRL approach. The observed performance differences among these methods can be attributed to several key factors:

Given the NP-hard property and highly dynamic nature of DMPFSP, a fundamental optimization challenge emerges: even state-of-the-art mathematical programming approaches must decompose the problem into a series of static snapshots, optimizing each independently as the production environment evolves. This fragmented optimization paradigm inherently sacrifices global optimality across the complete production horizon for local optimality at discrete time points, creating a significant methodological gap in handling truly dynamic scheduling environments. Consequently, methods like Gurobi tend to optimize based on currently available information but lack comprehensive consideration of future uncertainties in dynamic scheduling environments. When dynamic events such as new job arrivals or machine breakdowns occur, the originally optimal solution may quickly become suboptimal or even infeasible.

In contrast, NDRLA_IPC can capture the “completion time shape” on the Gantt chart through carefully designed state features, thereby achieving a more accurate alignment between the current job and subsequent jobs. This helps minimize idle times between adjacent jobs and reduce potential delays. Its four scheduling rules focus on reducing the operation interval, enabling the algorithm to maintain high adaptability when facing machine failures or processing time changes. Traditional methods such as GSH and IG mainly rely on predefined rules and local search mechanisms, which are relatively rigid when dealing with complex dynamic environments. Regarding SA-NET, this DRL method introduces more flexible strategies to a certain extent, but its network architecture still has limitations and struggles to fully capture the complex dependencies between scheduling decisions and environmental states.

The test results demonstrate that NDRLA_IPC can obtain high-quality solutions for small-scale instances in real time. This indicates that NDRLA_IPC has a powerful and robust search mechanism, which can efficiently address DMPFSP under different scales.

## 6. Conclusion

This research introduces novel deep reinforcement learning algorithm incorporating problem characteristics (NDRLA_IPC) that systematically integrates problem-specific characteristics to address the complex challenges of dynamic multi-objective permutation flow-shop scheduling problem. Specifically, the algorithm captures the “completion time shape” on the Gantt chart to extract state features, thereby deriving global processing information across machines. This innovative representation mechanism provides NDRLA_IPC with robust environmental perception capabilities for addressing the aforementioned methodological gaps in dynamic scheduling environments. Additionally, five new scheduling rules are designed to constitute the action space, aiming to optimize the alignment of neighboring jobs from

multiple perspectives based on different states. The architecture further employs a weighted composite reward function that evaluates action effectiveness, guiding the decision-making process of the double deep Q-network agent. By establishing robust correlations among state representation, action selection, and reward evaluation, NDRLA_IPC maintains objective consistency throughout the learning and decision-making processes, addressing the fragmented optimization paradigm inherent in traditional approaches. Comprehensive experimental results confirm NDRLA_IPC's superior performance across diverse dynamic scenarios, from small-scale instances with theoretical benchmarks to large-scale industrial problems. The algorithm consistently demonstrates remarkable adaptability to dynamic changes and maintains solution quality compared to state-of-the-art alternatives.

Future research will focus on two principal directions. First, we plan to enhance the computational efficiency and cognitive decision-making capabilities of DRL agents through the systematic extraction of promising heuristics and patterns inherent in dynamic production environments. Second, we will develop more robust frameworks for addressing the real dynamical scenarios of tin chemical production, tin New Material production, and tobacco leaf redrying process.

## CRediT authorship contribution statement

**Yuan-Yuan Yang:** Writing – review & editing, Writing – original draft, Software, Methodology, Investigation. **Bin Qian:** Writing – review & editing, Supervision, Methodology, Funding acquisition, Conceptualization. **Rong Hu:** Writing – review & editing, Supervision, Funding acquisition, Conceptualization. **Zuocheng Li:** Writing – review & editing, Supervision, Funding acquisition, Conceptualization. **Zi-Qi Zhang:** Supervision, Funding acquisition. **Huai-Ping Jin:** Validation, Supervision. **Jian-Bo Yang:** Validation, Supervision, 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 statement*

The data that support the findings of this study are openly available in reference [41].

### *Acknowledgment*

This research was supported by the National Natural Science Foundation of China (Grant Number: U24A20273, 62173169, 72362026, 7220011091), the Basic Research Key Project of Yunnan Province (Grant Number: 202201AS070030), the Major Science and Technology Project of China National Tobacco Corporation Yunnan Company (2024530000241029), and the Construction Project of Higher Educational Key Laboratory for Industrial Intelligence and Systems of Yunnan Province (Grant Number: KKPH202403003).

## Data availability

No data was used for the research described in the article.

## References

[1] Z.Q. Zhang, B. Qian, R. Hu, et al., A matrix-cube-based estimation of distribution algorithm for the distributed assembly permutation flow-shop scheduling problem, Swarm. Evol. Comput. 60 (2021).
[2] B. Qian, Z.Q. Zhang, R. Hu, et al., A Matrix-Cube-Based Estimation of Distribution Algorithm for No-Wait Flow-Shop Scheduling With Sequence-Dependent Setup Times and Release Times, IEEE Transactions on Systems, Man, and Cybernetics: Systems 53 (3) (2023) 1492–1503.
[3] M.M. Yenisey, B. Yagmahan, Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends, Omega (Westport) 45 (2014) 119–135.
[4] Q.K. Pan, R. Ruiz, A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime, Comput. Oper. Res. 40 (1) (2013) 117–128.
[5] K. Tliba, T.M.L. Diallo, O. Penas, et al., Digital twin-driven dynamic scheduling of a hybrid flow shop, J. Intell. Manuf. 34 (5) (2022) 2281–2306.
[6] D. Rahmani, R. Ramezanian, A stable reactive approach in dynamic flexible flow shop scheduling with unexpected disruptions: A case study, Comput. Ind. Eng. 98 (2016) 360–372.
[7] Y. Liu, J. Fan, L. Zhao, et al., Integration of deep reinforcement learning and multi-agent system for dynamic scheduling of re-entrant hybrid flow shop considering worker fatigue and skill levels, Robot. Comput. Integr. Manuf. 84 (2023).
[8] A. El-Bouri, S. Balakrishnan, N. Popplewell, Cooperative dispatching for minimizing mean flowtime in a dynamic flowshop, Int. J. Prod. Econ. 113 (2) (2008) 819–833.
[9] N. Mazyavkina, S. Sviridov, S. Ivanov, et al., Reinforcement learning for combinatorial optimization: A survey, Comput. Oper. Res. 134 (2021).
[10] S. Li, F. Wang, Q. He, et al., Deep reinforcement learning for multi-objective combinatorial optimization: A case study on multi-objective traveling salesman problem, Swarm. Evol. Comput. 83 (2023).
[11] V. Mnih, Playing atari with deep reinforcement learning, arXiv preprint (2013) arXiv:1312.5602.
[12] Y. Lu, Y. Yuan, A. Sitahong, et al., An Optimization Method for Green Permutation Flow Shop Scheduling Based on Deep Reinforcement Learning and MOEA/D, Machines 12 (10) (2024) 721.
[13] S. Luo, Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning, Appl. Soft. Comput. 91 (2020) 106208.
[14] R. S. Sutton, Open theoretical questions in reinforcement learning. 11-17.
[15] M.A. Adibi, M. Zandieh, M. Amiri, Multi-objective scheduling of dynamic job shop using variable neighborhood search, Expert. Syst. Appl. 37 (1) (2010) 282–287.
[16] J. Long, Z. Zheng, X. Gao, Dynamic scheduling in steelmaking-continuous casting production for continuous caster breakdown, Int. J. Prod. Res. 55 (11) (2016) 3197–3216.
[17] N. Kundakcı, O. Kulak, Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem, Comput. Ind. Eng. 96 (2016) 31–51.
[18] G. Li, N. Li, N. Sambandam, et al., Flow shop scheduling with jobs arriving at different times, Int. J. Prod. Econ. 206 (2018) 250–260.
[19] H.F. Rahman, M.N. Janardhanan, I.E. Nielsen, Real-time order acceptance and scheduling problems in a flow shop environment using hybrid GA-PSO algorithm, IEEe Access. 7 (2019) 112742–112755.
[20] P. Valledor, A. Gomez, P. Priore, et al., Modelling and solving rescheduling problems in dynamic permutation flow shop environments, Complexity. (2020) 1–17, 2020.
[21] Z.Jalali Khalil Abadi, N. Mansouri, M.M. Javidi, Deep reinforcement learning-based scheduling in distributed systems: a critical review, Knowl. Inf. Syst. (2024) 1–74.
[22] J. Wu, Y. Liu, A modified multi-agent proximal policy optimization algorithm for multi-objective dynamic partial-re-entrant hybrid flow shop scheduling problem, Eng. Appl. Artif. Intell. 140 (2025) 109688.
[23] S. Luo, Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning, Appl. Soft. Comput. 91 (2020).
[24] S. Luo, L. Zhang, Y. Fan, Real-time scheduling for dynamic partial-no-wait multiobjective flexible job shop by deep reinforcement learning, IEEE Transactions on Automation Science and Engineering 19 (4) (2021) 3020–3038.
[25] Y. Zhao, Y. Wang, Y. Tan, et al., Dynamic jobshop scheduling algorithm based on deep Q network, IEEe Access. 9 (2021) 122995–123011.
[26] L. Zhang, Y. Feng, Q. Xiao, et al., Deep reinforcement learning for dynamic flexible job shop scheduling problem considering variable processing times, J. Manuf. Syst. 71 (2023) 257–273.
[27] Z. Wang, W. Liao, Smart scheduling of dynamic job shop based on discrete event simulation and deep reinforcement learning, J. Intell. Manuf. (2023).
[28] H. Wang, J. Cheng, C. Liu, et al., Multi-objective reinforcement learning framework for dynamic flexible job shop scheduling problem with uncertain events, Appl. Soft. Comput. 131 (2022).
[29] Y. Gui, D. Tang, H. Zhu, et al., Dynamic scheduling for flexible job shop using a deep reinforcement learning approach, Comput. Ind. Eng. 180 (2023) 109255.
[30] M. Xu, Y. Mei, F. Zhang, et al., Niching Genetic Programming to Learn Actions for Deep Reinforcement Learning in Dynamic Flexible Scheduling, IEEE Transactions on Evolutionary Computation (2024).
[31] H. Hu, X. Jia, Q. He, et al., Deep reinforcement learning based AGVs real-time scheduling with mixed rule for flexible shop floor in industry 4.0, Comput. Ind. Eng. 149 (2020).
[32] S. Yang, Z. Xu, J. Wang, Intelligent Decision-Making of Scheduling for Dynamic Permutation Flowshop via Deep Reinforcement Learning, Sensors. (Basel) 21 (3) (2021).
[33] S. Yang, J. Wang, Z. Xu, Real-time scheduling for distributed permutation flowshops with dynamic job arrivals using deep reinforcement learning, Advanced Engineering Informatics 54 (2022).
[34] M. Wang, J. Zhang, P. Zhang, et al., Independent double DQN-based multi-agent reinforcement learning approach for online two-stage hybrid flow shop scheduling with batch machines, J. Manuf. Syst. 65 (2022) 694–708.
[35] F. Grumbach, A. Müller, P. Reusch, et al., Robust-stable scheduling in dynamic flow shops based on deep reinforcement learning, J. Intell. Manuf. 35 (2) (2024) 667–686.

[36] H.J. Kim, J.H. Lee, Look-ahead based reinforcement learning for robotic flow shop scheduling, J. Manuf. Syst. 68 (2023) 160–175.
[37] J. Ren, C. Ye, F. Yang, Solving flow-shop scheduling problem with a reinforcement learning algorithm that generalizes the value function with neural network, Alexandria Engineering Journal 60 (3) (2021) 2787–2800.
[38] C.B. Gil, J.H. Lee, Deep Reinforcement Learning Approach for Material Scheduling Considering High-Dimensional Environment of Hybrid Flow-Shop Problem, Applied Sciences 12 (18) (2022).
[39] Z. Wang, B. Cai, J. Li, et al., Solving non-permutation flow-shop scheduling problem via a novel deep reinforcement learning approach, Comput. Oper. Res. 151 (2023) 106095.
[40] A. Allahverdi, T. Aldowaisan, No-wait flowshops with bicriteria of makespan and maximum lateness, European Journal of Operational Research 152 (1) (2004) 132–147.
[41] E. Vallada, R. Ruiz, J.M. Framinan, New hard benchmark for flowshop scheduling problems minimising makespan, European Journal of Operational Research 240 (3) (2015) 666–677.
[42] M. Ghaleb, H. Zolfagharinia, S. Taghipour, Real-time production scheduling in the Industry-4.0 context: Addressing uncertainties in job arrivals and machine breakdowns, Comput. Oper. Res. 123 (2020) 105031.
[43] M.Shahgholi Zadeh, Y. Katebi, A. Doniavi, A heuristic model for dynamic flexible job shop scheduling problem considering variable processing times, Int. J. Prod. Res. 57 (10) (2019) 3020–3035.
[44] Z. Hu, W. Gong, W. Pedrycz, et al., Deep reinforcement learning assisted co-evolutionary differential evolution for constrained optimization, Swarm. Evol. Comput. 83 (2023) 101387.
[45] Z. Zhang, W. Wang, S. Zhong, et al., Flow Shop Scheduling with Reinforcement Learning, Asia-Pacific Journal of Operational Research 30 (05) (2013).
[46] M.L. Pinedo, Scheduling : theory, algorithms, and systems, A I I E Transactions 28 (8) (2016) 695–697.
[47] J.N.D. Gupta, A Functional Heuristic Algorithm for the Flowshop Scheduling Problem, Journal of the Operational Research Society 22 (1) (1971) 39–47.
[48] R. Ruiz, T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, European Journal of Operational Research 177 (3) (2007) 2033–2049.