Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

A Hybrid Model of Reinforcement Learning and Swarm Intelligence for Dynamic Flexible Scheduling in Robotic-Arm Production Lines

Jia-Xuan Khoo
Yi-Yun Ni
Yu-Ting Lai
Kang-Ting Ma
ktma@gms.ndhu.edu.tw

National Dong Hwa University https://orcid.org/0000-0003-2735-7264

Research Article

Keywords: Smart Manufacturing, Dynamic Scheduling, Reinforcement Learning, Simplified Swarm Optimization, Robotic-Arm Production Scheduling

Posted Date: May 14th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9647321/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

A Hybrid Model of Reinforcement Learning and Swarm Intelligence for Dynamic Flexible Scheduling in Robotic-Arm Production Lines

Jia-Xuan Khoo [1], Yi-Yun Ni[1], Yu-Ting Lai[1], Kang-Ting Ma [1*]

[1]Graduate Institute of Logistics Management, College of Management, National Dong Hwa University, Hualien, Taiwan

*Corresponding author e-mail address:
ktma@gms.ndhu.edu.tw (K.-T. Ma).

ABSTRACT

Industry 4.0 is accelerating robotic-assisted manufacturing, where a single robotic-arm supports multi-machine material transport and cross-station operations. Real-time scheduling in such lines is challenging because orders arrive dynamically, processing times are heterogeneous, and decisions must satisfy both machine and robot resource constraints. Traditional dispatching rules and swarm-based heuristics often rely on fixed strategies, making them prone to local optima and weakening the balance between efficiency and due-date performance. We propose a hybrid Reinforcement Learning–based Simplified Swarm Optimization (RLSSO) approach for dynamic flexible scheduling with one robotic-arm serving multiple machines. RLSSO preserves the population-based structure and memory mechanisms of simplified swarm optimization, while reinforcement learning adaptively selects search operators during optimization. The key idea is learning when to apply operators rather than modifying the swarm structure; thus, improvements are not attributable to parameter tuning alone. A weighted composite fitness function integrates makespan, total tardiness, and the number of tardy jobs. Experiments on a multi-machine production line with dynamic order arrivals show that, compared with NSGA-II, RLSSO reduces makespan by 3.67%, total tardiness by 26.92%, the number of tardy jobs by 18.75%, and the integrated objective value by 24.22% in the dynamic job insertion scenario. An ablation study that disables learning-guided operator selection while keeping the same swarm structure and parameter settings yields consistently inferior results, supporting the importance of adaptive operator selection in improving real-time robotic-assisted scheduling performance.

Keywords: *Smart Manufacturing, Dynamic Scheduling*, *Reinforcement Learning, Simplified Swarm Optimization, Robotic-Arm Production Scheduling*

1. Introduction

In past metal-machining shop floors, production and material-handling decisions relied heavily on operators' experience. Automated production lines increasingly involve multi-machine coordination and robotic-arm-assisted material handling [1]. High-mix low-volume production and customization have made product combinations and operating routes increasingly variable. As a result, traditional scheduling concepts based on mass production no longer satisfy both due-date flexibility and production stability [2]. In this context, scheduling is no longer limited to arranging processing sequences, but must also consider operational linkages and the cascading effects of waiting and material-handling under dynamic production conditions.

Based on this background, this study focuses on a metal-machining production line in which the core processing resources include Wire Electrical Discharge Machining (EW), Computer Numerical Control (CNC) milling machines, and Electrical Discharge Machining (EDM), whose job routes and resource requirements frequently change. The processing-time characteristics of these machine types are heterogeneous, leading to uneven waiting and shifting bottlenecks across stations. If scheduling decisions do not allocate resources appropriately, waiting and work-in-process accumulation may occur at specific stations, and bottleneck locations may shift with production states [1]. Consequently, scheduling decisions must continually adapt to shifting bottlenecks and heterogeneous processing times, rather than relying on static resource allocation.

Material-handling resource constraints are a key factor influencing scheduling performance [3]. The production line in this study is equipped with a single-rail robotic-arm as a shared resource responsible for inter-station transportation and loading/unloading. Since the system contains only one robotic-arm and the path is fixed, material-handling operations cannot be executed in parallel, and robotic-arm availability directly affects job transfer times and machine start times. If scheduling decisions focus only on processing sequencing and ignore robotic-arm availability, the material-handling subsystem may become a bottleneck [4], resulting in machine idleness and job waiting, which increases efficiency loss and due-date risk. Under these constraints, this study addresses an integrated scheduling problem in which processing, material handling, and resource conflicts must be resolved simultaneously.

From a methodological perspective, shop floors commonly rely on dispatching rules for real-time decision-making, such as Earliest Due Date (EDD), Shortest Processing Time (SPT), and First In First Out (FIFO). These rules are computationally efficient but rely mainly on local information such as due-dates and processing times. Consequently, they are unable to explicitly represent material-handling requirements induced by changes in processing sequences [5], nor can they capture the impact of material-handling constraints on machine start times and job waiting behavior. Under dynamic job arrivals or shifting workload distributions, rule-based decision-making often leads to increased variability in due-date compliance and performance. Maintaining acceptable performance therefore requires frequent rule switching or manual intervention, reducing reproducibility and robustness in complex manufacturing environments [6].

Another category of methods is population-based search, such as the Non-dominated Sorting Genetic Algorithm II (NSGA-II). In integrated and multi-objective scheduling problems, the performance of NSGA-II strongly depends on problem-specific encoding and decoding procedures, which must jointly represent machine assignments, operation precedence, and resource interaction constraints to ensure solution feasibility and coverage of the search space [7]. As the scheduling environment becomes more complex or dynamic, standard NSGA-II with fixed operators and static parameter settings often requires additional modification or re-optimization to maintain solution quality, limiting its direct applicability in real-time or disturbance-driven decision-making scenarios [8]. For multi-objective scheduling, NSGA-II remains a widely used evolutionary baseline, although recent studies often extend its framework to improve performance in more complex environments [9,10].

Given these limitations, this study adopts an RLSSO hybrid scheduling framework for integrated processing and material-handling decision-making. Simplified Swarm Optimization (SSO) generates and improves candidate

schedules in the solution space using operators, and Reinforcement Learning (RL) adaptively selects search operators based on the system state. Under this framework, the search process becomes adaptive and state-driven in an integrated scheduling environment. Compared with dispatching rules, RLSSO does not rely on a single fixed decision logic. Compared with GA-based evolutionary methods that control the search process through pre-designed procedures [9], RLSSO treats search control as a learnable process, reducing dependence on fixed strategies and parameter settings. This study uses makespan, total tardiness, and the number of tardy jobs as performance indicators, and evaluates the method in a discrete-event simulation environment with dynamic job arrivals and insertion mechanisms. Benchmark methods, EDD, SSO, and NSGA-II, are compared to evaluate the proposed method's solution quality, stability, and reproducibility under dynamic job insertion.

The remainder of this paper is organized as follows. Section 2 reviews the related literature on integrated scheduling, RL, and swarm intelligence methods. Section 3 presents the system description, assumptions, objective functions, the RLSSO framework, and the online job insertion mechanism. Section 4 reports the experimental scenarios and results. Section 5 concludes the paper and discusses future directions.

2. Literature Review

2.1 Multi-operation scheduling and robotic-arm transportation systems

Multi-operation production systems are governed by precedence constraints, under which each operation can start only after its predecessor has been completed [11]. This requires scheduling decisions to maintain temporal consistency and dependency integrity across stages, thereby increasing decision complexity [12]. When automated material-handling resources are introduced, scheduling must further coordinate processing and transportation to reduce waiting and prevent delay propagation across operations [13,14]. In such systems, material-handling devices act as shared resources, and their task durations depend on routing and real-time system states [15]. This issue is especially critical in single-rail robotic-arm systems, where robotic-arm position and machine processing status jointly affect scheduling feasibility [1]. From a solvability perspective, Xu et al. [15] and Feng et al. [16] indicated that both modeling and solution approaches require trade-offs between computational efficiency and the representation of detailed system interactions. Fully capturing real-time interactions and state-dependent behaviors substantially increases computational cost, making real-time decision-making infeasible. To address this issue, recent studies have explored learning-based approaches to real-time decision-making in scenarios where the robotic-arm serves as a shared, dynamic resource. For example, Kim and Lee [17] employed Deep Q-learning (DQN) to study the dual-gripper robot problem, using Petri nets to describe the internal transportation and processing flow and construct the RL state space. Zisgen et al. [3] also adopted DQN for robotic-arm scheduling involving parallel machines, while Wang et al. [4] applied a Deep Reinforcement Learning (DRL) framework with Graph Neural Network for state representation and Proximal Policy Optimization for training in flexible robotic cell scheduling. Overall, research on learning-based coordination of dispatching, movement, and processing sequences for robotic-arms, treated as shared, path-dependent material-handling resources, remains in its early stages, suggesting significant potential for further study and practical application. However, studies that jointly consider robotic-arm movement, multi-operation processing, and dynamic scheduling decisions remain limited.

2.2 Reinforcement Learning and Heuristic Scheduling Approaches

In relatively simple scheduling environments, traditional dispatching rules such as EDD are widely used because of their low computational cost and fast decision-making. Their effectiveness declines when practical constraints such as buffer capacity, transportation, precedence, and due-date tightness are introduced. Under dynamic conditions, single-rule dispatching often struggles to balance efficiency and tardiness-related performance [5,18]. To overcome the limitations of single dispatching rules, research has introduced search-based optimization methods that handle multiple decisions and constraints within a larger solution space. SSO is one representative method. It uses discrete encoding to represent candidate solutions and updates them according to current, personal-best(pbest), and global-best(gbest) information, thereby balancing search diversity and solution quality [19]. However, SSO still relies on fixed update mechanisms. In complex or dynamic environments, the lack of adaptive operator selection or parameter adjustment may reduce convergence efficiency and lead to unstable solution quality.

Multi-objective scheduling studies often adopt evolutionary methods as benchmark approaches. NSGA-II is frequently used because its non-dominated sorting and elitism mechanisms support the simultaneous treatment of multiple conflicting objectives [9]. However, in more complex scheduling environments, its performance depends not only on the search mechanism but also on hyperparameter configuration, and recent studies often report parameter calibration or additional search designs to improve convergence and solution quality [10]. Even so, such methods are mainly designed for offline optimization and become less suitable when dynamic events occur. Under job insertion or rescheduling conditions, the solution process may need to be restarted, which weakens real-time applicability and increases parameter-setting cost [4]. As dynamic events become more frequent, scheduling research has increasingly shifted toward learning-based approaches that update policies through interaction with the environment. RL is well suited to such settings because it can revise decision policies in response to uncertainty and system variation. Q-learning, in particular, learns the value of state–action pairs through trial and error and has been applied to dynamic scheduling problems [20]. Prior studies have used Q-learning to treat dispatching rules or composite dispatching rules as selectable actions, allowing the agent to adjust decisions when job insertions or other environmental changes occur [21,22]. Despite this advantage, RL-based scheduling becomes difficult when the state and action spaces are high-dimensional, especially in problems such as FJSP that require simultaneous sequencing and machine-assignment decisions [23,24]. Existing studies have therefore reduced decision complexity by simplifying action design or improving state representation. Some restrict the set of selectable jobs at each decision point or embed sequencing decisions into the dispatching process [25,26]. Others combine DRL with graph-based models to better capture complex shop-floor states under stochastic job arrivals [27].

In robotic-arm-oriented and dynamic scheduling

problems, RL has also been used to improve adaptability under changing system conditions. Related studies show that environment-aware action design can improve convergence and reduce uncertainty in dynamic settings [28]. Still, relying on RL alone often leads to high training cost and slow convergence. This limitation has motivated hybrid frameworks that combine RL with evolutionary or heuristic search. Some studies use RL to adjust evolutionary parameters dynamically [29], while others formulate operator selection as a sequential decision problem and use RL to guide the choice of low-level search operators during optimization [30–32]. These approaches improve adaptability during the search process and help alleviate premature convergence. However, RL-guided operator selection has rarely been extended to integrated scheduling problems that simultaneously consider multi-operation processing, material-handling resources, and dynamic job insertions. Research on such integrated designs remains limited.

2.3 Summary of research gaps

Based on the above review, existing studies have not yet fully addressed integrated scheduling in robotic-arm-assisted production systems under dynamic conditions. Prior research has often examined machine scheduling, robotic-arm coordination, or dynamic job arrivals separately, while the joint consideration of multi-operation processing, robotic-arm transportation, and dynamic job insertion remains limited [3,17]. In addition, although RL has been introduced into scheduling research, many studies focus on dispatching-rule selection or problem-specific action and state design [21,22,25–28], whereas learning-guided search control within an optimization framework has received comparatively less attention [29–32]. These limitations indicate the need for an integrated and adaptive scheduling approach. Therefore, this study proposes RLSSO, in which Q-learning dynamically adjusts particle update selection during the search process. The proposed method is evaluated against NSGA-II, SSO, and EDD to examine its search efficiency and solution quality for the integrated scheduling problem.

3. Problem Description and System Modeling

This study proposes an RLSSO hybrid scheduling framework for a production system with multiple machines and multiple operations, served by a single rail-mounted robotic-arm for material handling. The framework combines RL and simplified swarm optimization to update scheduling decisions at event-triggered decision points. Inputs include job information and a transport-time matrix. Decision information is constructed from machine states, robotic-arm states, and job states.

As shown in Fig. 1, the RLSSO framework operates in a closed loop. At each decision point, the system collects and updates states. The RL agent selects an update action based on the current state to guide SSO in searching candidate solutions. Each candidate solution is decoded into executable processing and transport timings. The system computes performance metrics from the schedule results and generates feedback, which is used to update the states and policy for the next decision point. The agent design, SSO update procedure, and schedule decoding rules are introduced in the following subsections.

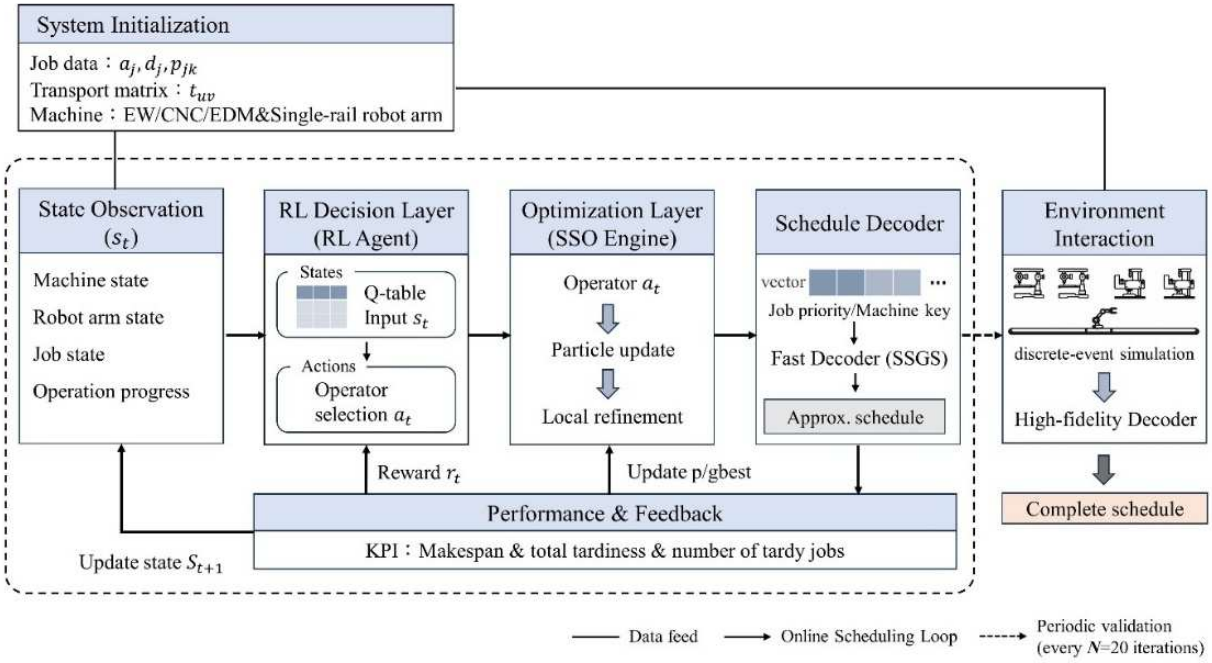


Fig. 1 Framework of the RLSSO scheduling system.

3.1 System description and assumptions

This study examines a multi-stage machining system composed of three process types, EW, CNC, and EDM. Although each job may require only a subset of these operations, the processing sequence is fixed in the order of EW–CNC–EDM. As shown in Fig. 2, the workstations are connected by a single rail-mounted robotic-arm responsible for inter-station transportation as well as loading and unloading operations. Therefore, a subsequent operation can begin only after the preceding one has been completed and the job has been transported to the designated workstation. Jobs arrive dynamically over time. Each workstation contains multiple identical parallel machines, and each machine can process at most one job at a time. Machine breakdowns and setup or changeover times are not considered. Because only one robotic-arm is available, transportation tasks are executed sequentially. If the robotic-arm is occupied when a job

completes processing, the job waits in the upstream output buffer until transportation becomes available. After transportation, the job may further wait if no downstream machine is ready for processing. The performance measures considered in this study are makespan, total tardiness, and the number of tardy jobs.

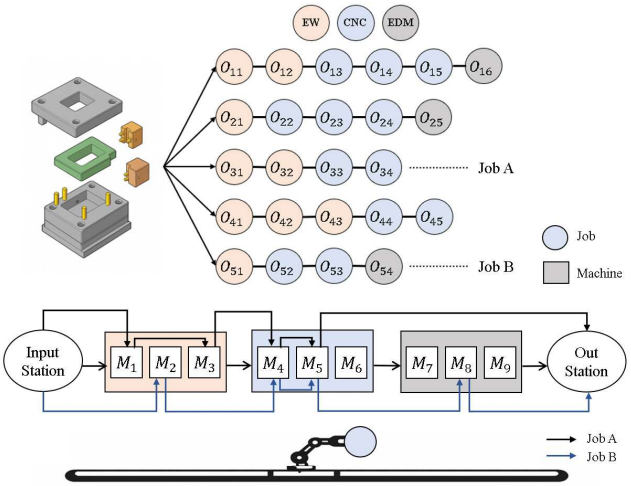


Fig. 2 Schematic diagram of the multi-stage production system with a single-rail robotic-arm

3.2 Notations and parameters

This section summarizes the notations used throughout the system model, the objective functions, the constraints, and the decoding procedures. All symbols appearing in the subsequent sections follow the definitions below.

Sets and indices:

$J = \{1,2,\dots,n\}$: Set of jobs, indexed by j.

$K = \{1,2,3\}$: Set of processing stages, corresponding to EW, CNC, and EDM

$O_j \subseteq K$: required stage set of job j; If multiple stages are required, the precedence follows EW–CNC–EDM.

$\bar{k}_j = max(O_j)$: Last required stage of job j.

M_k: Set of parallel machines at stage k, machine index $m \in M_k$.

w: Set of workstation nodes served by the robotic-arm, including the input buffer, the output buffer, and the load/unload points of all machines; indices $u, v \in w$

$s \in \{1,\dots,q_j\}$: unit index within job j.

Input parameters:

a_j: Arrival time or release time of job j.

d_j: Due-date of job j.

q_j: Quantity demand of job j.

p_{jk}: Processing time of job j at stage k, defined only when $k \in O_j$

t_{uv}: Travel time of the robotic-arm from workstation u to workstation v, where $u, v \in W$.

h: Handling time for load/unload or place action.

Decision Variables:

S^R_{jks}: Start time of the robotic-arm transportation for item s of job j at stage k.

C^R_{jks}: Completion time of the robotic-arm transportation for item s of job j at stage k.

S_{jks}: Processing start time of item s of job j at stage k.

C_{jks}: Completion time of item s of job j at stage k.

$u_{jks}, v_{jks} \in W$: Origin and destination workstation nodes of the robotic-arm task associated with item s of job j before stage k.

y_{jkm}: Machine assignment indicator for job j at stage k on machine m.

Performance Measures:

C_j: Completion time of job j.

C_{max}: Makespan of the schedule.

TD_j: Tardiness of job j.

N_j: Tardy-job indicator of job j.

Z: Aggregated objective value.

$C_{ref}, T_{ref}, N_{ref}$: Normalization constants used in Z.

W_1, W_2, W_3 : Weights of the makespan, total tardiness, and the number of tardy jobs in Z.

3.3 Objective Function and Constraints

Time is measured in seconds. The system operates under a fixed day-shift schedule from 08:00 to 18:00. Processing operations may be interrupted during non-working hours and resumed on the next workday, whereas robotic-arm transportation and placement tasks must be completed within the same workday. If the remaining workable time for the current day is insufficient for a robotic-arm task, the task's start time is postponed to 08:00 on the next workday.

To align the mathematical formulation with the schedule decoding procedure, let $\Phi(t)$ denote a time-alignment function that maps any time t to the nearest feasible processing start time. If t falls within working hours, it remains unchanged; if t <08:00, it is aligned to 08:00 of the same day; if $t \geq$18:00, it is aligned to 08:00 of the next workday. Let $\Gamma(t,\delta)$ denote the completion time obtained after accumulating δ seconds of effective working time starting from $\Phi(t)$. This function is used for processing operations, which are allowed to span multiple workdays. In addition, let $\Psi(t,\delta)$ denote the start time for a non-preemptive task of duration δ beginning from $\Phi(t)$. If the remaining working time from $\Phi(t)$ to 18:00 is insufficient to accommodate δ, then $\Psi(t,\delta)$ is set to 08:00 of the next workday; otherwise,$\Psi(t,\delta) = \Phi(t)$.

Each job $j \in J$ has a quantity demand q_j and follows an item-by-item processing and transportation policy. For each required operation $k \in O_j$ and each item index $s = 1,\dots,q_j$. Let S_{jks} and C_{jks} denote the processing start and completion times, respectively, and let S^R_{jks} and C^R_{jks} denote the corresponding robotic-arm start and completion times. Let $\bar{k}_j = max(O_j)$ denote the last required operation of job j. Then, the completion time of job j is defined as the finishing time of the last unit in the final operation, as shown in Eq. (1).

$$C_j = C_{j\bar{k}_j q_j}, \forall j \in J \tag{1}$$

The makespan is defined in Eq. (2), and the tardiness and tardy-job indicator of each job are defined in Eqs. (3) and (4), respectively.

$$C_{max} = max_{j\epsilon J} C_j \tag{2}$$

$$TD_j = max\left\{0, C_{j\bar{k}_j q_j} - d_j\right\}, \forall j \in J \tag{3}$$

$$N_j = \begin{cases} 1, & if\ C_{j\bar{k}_j q_j} > d_j \\ 0, & otherwise \end{cases}, \quad \forall j \in J \tag{4}$$

Using these three components, the composite objective value Z is formulated as Eq. (5),

$$min\ Z = W_1 \frac{C_{max}}{C_{ref}} + W_2 \frac{\sum_{j\in J} TD_j}{T_{ref}} + W_3 \frac{\sum_{j\in J} N_j}{N_{ref}} \tag{5}$$

In Eq. (5), C_{ref}, T_{ref}, N_{ref} are normalization constants for makespan, total tardiness, and the number of tardy jobs, respectively, while W_1, W_2, W_3 are weighting coefficients.

For processing feasibility, the processing time of item s of job j at operation k is denoted by p_{jks} defined for $k \in O_j$. Since processing may span multiple workdays, its completion

time is computed by accumulating effective workable time, as shown in Eq. (6).

$$C_{jks} = \Gamma\left(S_{jks}, p_{jks}\right), \forall\, j \in J, \forall k \in O_j, \forall s = 1, \ldots, q_j \tag{6}$$

Before processing starts, the robotic-arm must complete transport and placement. Let u_{jks} and v_{jks} denote the origin and destination nodes of the corresponding robotic-arm task. The travel time is given by the matrix t_{uv}, and the handling time is a constant $h = 15$. Since robotic-arm tasks are non-preemptive within a workday, their start and completion times are defined in Eq. (7).

$$S^{R}_{jks} = \Psi\left(\hat{S}^{R}_{jks}, t_{u_{jks}v_{jks}} + h\right), C^{R}_{jks} = S^{R}_{jks} + t_{u_{jks}v_{jks}} + h, \forall\, j, k \in O_j, \forall s \tag{7}$$

Here $\widehat{S^{R}}jks$ denotes the earliest feasible robotic-arm start time determined by resource availability before applying the day-shift feasibility adjustment. Each processing task cannot start earlier than the completion of its corresponding robotic-arm placement, as shown in Eq. (8).

$$S_{jks} \geq C^{R}_{jks}\ , \forall\, j \in J, k \in O_j, \forall s \tag{8}$$

For dynamic arrivals, let a_j be the release time of job *j*. Then, the first robotic-arm task of the first required operation cannot start before the job arrives, as shown in Eq. (9).

$$S^{R}_{jk1} \geq a_j, \forall\, j \in J, k = min\left(O_j\right) \tag{9}$$

Within the same operation, items are processed sequentially. Therefore, the robotic-arm task of item $s+1$ cannot start before item *s* finishes processing, as shown in Eq. (10).

$$S^{R}_{jk(s+1)} \geq C_{jks}\ , \forall\, j \in J, \forall k \in O_j, \forall s = 1, \ldots, q_j - 1 \tag{10}$$

If job *j* requires a downstream operation, the first robotic-arm task of that downstream operation cannot start before the last item of the preceding required operation has been completed, as shown in Eq. (11).

$$S^{R}_{jk1} \geq C_{j(k-1)q_j}, \forall\, j \in J, \forall\, k \in O_j \cap \{2,3\} \tag{11}$$

Resource feasibility is further enforced by mutual-exclusion constraints. Each machine can process at most one item at a time, as described in Eq. (12).

$$y_{jkm} = y_{ikm} = 1 \Rightarrow \left[S_{jks}, C_{jks}\right) \cap \left[S_{ikr}, C_{ikr}\right) = \emptyset, \forall\, m \in M_k, \forall\, k, \forall\, i < j \tag{12}$$

Since the system is equipped with the only one robotic-arm, the execution intervals of any two robotic-arm tasks must not overlap, as shown in Eq. (13).

$$\left[S^{R}_{jks}, C^{R}_{jks}\right) \cap \left[S^{R}_{ilr}, C^{R}_{ilr}\right) = \emptyset, \forall\ (j,k,s) \neq (i,l,r) \tag{13}$$

Together, these constraints describe job completion, day-shift feasibility, processing-transport linkage, dynamic release-time feasibility, operation precedence, and the mutual exclusion of machine and robotic-arm resources in the event-driven scheduling model.

3.4 Solution representation and decoders

3.4.1 Encoding

In this study, each solution is represented as a particle using a continuous random-key encoding scheme. Let $J = \{1,2,\ldots,n\}$ denote the set of jobs, where *n* is the number of jobs. Each job *j* is associated with a sequencing key r_j, and the particle is therefore represented as follows:

$$X = [r_1, r_2, \ldots, r_n] \tag{14}$$

where r_j is the sequencing key of job *j*. During decoding, the keys in *X* are sorted in ascending order to determine the job priority sequence. Jobs with smaller key values are given higher priority. Since the processing route and workstation information of each job are predefined by the input data, machine assignment is not encoded in the particle. The search therefore focuses on job sequencing, which affects workstation queues, robotic-arm transportation timing, completion times, and due-date delays.

Algorithm 1 Particle Encoding

```
begin
    set random seed;
    generate sequencing keys X_seq;
    sort X to obtain job order;
    return X;
end
```

3.4.2 Fast decoding

The fast decoder adopts a serial schedule generation scheme (SSGS), in which jobs are dispatched one by one according to the priority order $\pi \leftarrow argsort(X)$ given by each particle. For each job, its operations are scheduled sequentially along the predefined route. At each operation, the decoder determines the feasible start time by jointly considering job release time, workstation availability, robotic-arm availability, inter-station transportation time, loading/unloading time, and shift-calendar constraints. After an operation is assigned, the next-available times of the workstation and the robotic-arm are updated immediately. Once all jobs are decoded, the fast evaluator outputs approximate makespan, total tardiness, and the number of tardy jobs for RLSSO fitness evaluation.

Algorithm 2 Fast decoding for RLSSO fitness evaluation

```
begin
    obtain job priority order from X;
    initialize workstation and robotic-arm status;
    for each job j in the priority order do
        set current time to the release time of job j;
        for each operation in the route of job j do
            determine assigned machine;
            determine transport-feasible start time;
            compute completion time;
            update workstation and robotic-arm status;
        end
        record completion time of job j;
    end
    compute makespan, tardiness, and the tardy jobs;
    compute fitness;
    return fitness and performance indicators;
end
```

3.4.3 High-fidelity DES Decoder

To reduce the estimation bias of the fast decoder and provide the final reported KPIs, a high-fidelity discrete-event simulation (DES) decoder is adopted. The DES model explicitly tracks machine availability, robotic-arm availability and position, job readiness, and operation progress under precedence, transport, and shift constraints. During simulation, feasible operations are generated dynamically and dispatched by the rule RobotPick (·), while machine and robotic-arm event logs are recorded. These logs are further used to compute the final makespan, total tardiness, and the number of tardy jobs. During RLSSO training, the DES decoder is invoked every G generations ($G = 20$) to periodically validate the current best solution and reduce the bias introduced by fast decoding.

Algorithm 3 High-fidelity DES decoder

```
begin
    decode job priority and machine assignment;
    initialize machine, robotic-arm, and job states;
    while unfinished jobs exist do
        construct the feasible candidate set;
        select the next operation by RobotPick(·);
        record machine and robotic-arm events;
        update machine, robotic-arm, and job states;
        advance simulation time;
    end
    compute final KPIs and objective value Z;
    return event logs, KPIs, and Z;
end
```

3.5 Reinforcement Learning Module

3.5.1 State definition

To enable the RL agent to learn decision logic in a multi-machine, multi-operation scheduling environment, this study compresses the real-time shop-floor state into a 12-dimensional continuous vector that summarizes resource availability and due-date risk. At each decision time t_{dec}, the state vector is defined in Eq. (15):

$$\phi(t) = \left[\tilde{A}_1(t), \dots, \tilde{A}_9(t), \tilde{S}_{avg}(t), \tilde{S}_{min}(t), \tilde{I}_R(t)\right], \quad (15)$$

where the first nine components correspond to the normalized availability features of the nine workstations. The components are defined as follows:

1. Machine state

Let $A_w^{ws}(t)$ denote the next available time of workstation w. Its relative workload level is normalized by min-max scaling over all workstation ready times, as shown in Eq. (16).

$$\tilde{A}_w(t) = \frac{A_w^{ws}(t) - min_{1\le q\le 9} A_q^{ws}(t)}{max_{1\le q\le 9} A_q^{ws}(t) - min_{1\le q\le 9} A_q^{ws}(t) + 10^{-9}}, \quad (16)$$

$$w = 1, \dots, 9$$

Here, $\min_{1\le q\le 9} A_q^{ws}(t)$ and $\max_{1\le q\le 9} A_q^{ws}(t)$ are the earliest and latest machine-ready times among all workstations.

2. Job state

Let d_j be the due-date of job j, and let $\hat{F}_j(t)$ be the completion time estimated by the fast evaluator (in seconds). The slack is defined as Eq. (17).

$$s_j(t) = d_j - \hat{F}_j(t) \quad (17)$$

For the set of unfinished jobs J_t, the average and minimum slack are used as system-level due-date risk indicators, as shown in Eqs. (18) and (19).

$$S_{avg}(t) = \frac{1}{|J_t|} \sum_{j\in J_t} s_j(t) \quad (18)$$

$$S_{min}(t) = min_{j\in J_t} s_j(t) \quad (19)$$

To maintain a consistent scale, both indicators are normalized by the daily workable time $H = 36000$ seconds and clipped to $[0, 1]$, as shown in Eqs. (20) and (21).

$$\tilde{S}_{avg}(t) = clip\left(\frac{S_{avg}(t)}{H}, 0,1\right) \quad (20)$$

$$\tilde{S}_{min}(t) = clip\left(\frac{S_{min}(t)}{H}, 0,1\right) \quad (21)$$

3. Robotic-arm state

Let $A^R(t)$ be the next available time of the robotic-arm. If the robotic-arm is executing a task, $A^R(t) > t$; otherwise, $A^R(t) = t$. The robotic-arm availability pressure indicator is normalized by the shift length $H = 36000$ seconds, as shown in Eq (22).

$$\tilde{I}_R(t) = clip\left(\frac{(A^R(t) - t)_+}{H}, 0,1\right) \quad (22)$$

Here, $(x)_+ = \max(x, 0)$.

Table 1 Definition of state features used in the RL agent.

Index	State feature	Category	Definition
1-9	$\tilde{A}(t) = [\tilde{A}_1(t), \dots, \tilde{A}_9(t)]$	Machine	min-max normalized workstation next-available times
10	$\tilde{S}_{avg}(t)$	Job	average slack across jobs
11	$\tilde{S}_{min}(t)$	Job	minimum slack among jobs
12	$\tilde{I}_R(t)$	Robotic-arm	Robotic-arm availability pressure indicator

The state vector represents the current scheduling context and guides operator selection at each decision step. Before Q-table lookup, the 12-dimensional continuous state is discretized into finite intervals. The state does not directly specify the exact job pair or subsequence boundaries; instead, these detailed modifications are determined internally by the selected operator according to the current sequence configuration and local urgency and slack conditions.

3.5.2 Action Definitions

At each decision epoch, the RL agent selects one operator from a finite action set. Accordingly, the proposed RLSSO framework formulates decision making as an operator-selection problem rather than direct job-level manipulation. The selected action determines which update mechanism is activated, while the detailed sequence modification is executed internally by the corresponding operator. As shown in Fig. 3, the discrete action space consists of five sequence update operators. These five actions are defined as follows:

- Action 0: Particle reinitialization.
  A portion of the current job sequence is randomly reinitialized to enhance exploration and reduce the risk of premature convergence.
- Action 1: Probabilistic selection based on pbest or gbest.
  The operator selects either the particle's personal best solution or the current global best solution as the reference sequence, and then partially aligns the current particle with the selected reference.
- Action 2: Dynamic event handling.
  An insertion-based adjustment is performed for urgent or unfinished jobs to improve scheduling responsiveness under dynamic disturbances.
- Action 3: Urgency-aware swap operator.
  The relative positions of selected jobs are locally revised through swap-based modification according to urgency-related information.
- Action 4: Pressure-aware shuffle.
  A controlled subsequence reordering is conducted under high due-date pressure or workload imbalance to promote search diversification beyond local optima.

These five actions define the candidate update behaviors available to RLSSO. The RL agent does not directly specify the exact job pair or subsequence boundaries. Instead, once an action is selected, the corresponding operator performs the detailed sequence modification according to the current

solution structure and urgency-related information.

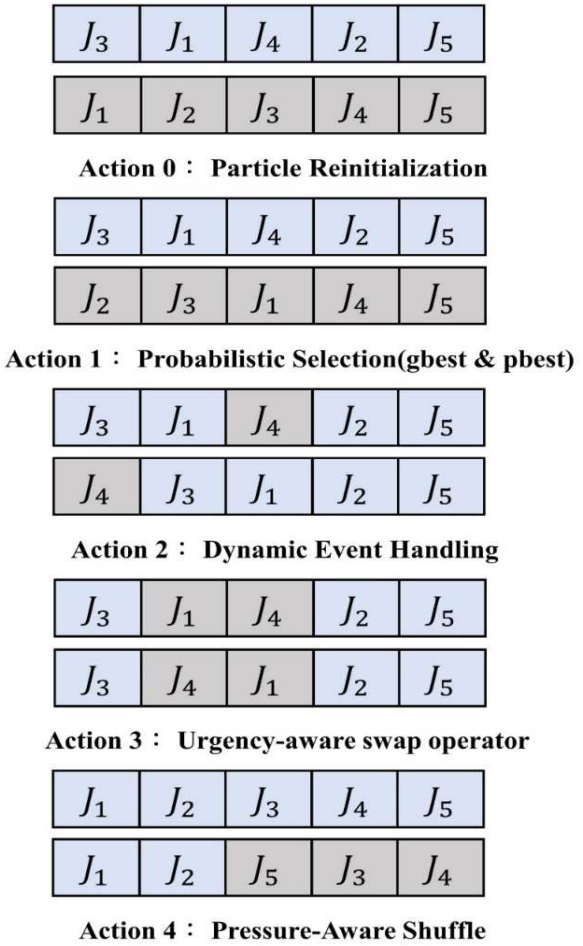


Fig. 3 Discrete action space and sequence update operators

3.5.3 Reward function and Q-learning update

In order to keep the RL process consistent with the overall scheduling evaluation criteria, this study does not introduce additional heuristic indicators for reward design. Instead, the reward function directly uses the fitness value corresponding to the scheduling objective function as the learning signal. Specifically, after each selected operator is applied, the RL agent computes a reward based on the change in fitness before and after the solution update, thereby guiding the agent to learn search strategies that improve overall scheduling performance.

The fitness function used in this study is a weighted aggregation of multiple scheduling performance metrics. Furthermore, scale normalization is incorporated to prevent indicators with different magnitudes from biasing the search direction. The fitness function is defined as shown in Eq. (23).

$$f = W_1 \frac{C_{max}}{C_{ref}} + W_2 \frac{\sum_{j \in J} TD_j}{T_{ref}} + W_3 \frac{\sum_{j \in J} N_j}{N_{ref}} \quad (23)$$

C_{max} denotes the final completion time (makespan), $\sum TD_j$ represents the total tardiness, and $\sum N_j$ is the number of tardy jobs. C_{ref}, T_{ref}, and N_{ref} are the normalization references for these three metrics, which are used to map different performance indicators into a comparable numerical range. W_1, W_2, and W_3 are the weighting parameters corresponding to the three performance metrics, reflecting the decision maker's relative emphasis on production efficiency and due-date performance. The reward is defined as improvement in fitness:

$$r_t = f(x_t) - f(x_{t+1}) \quad (24)$$

When a sub-action update reduces the fitness value, the schedule quality is improved and the agent receives a positive reward. Conversely, if the update worsens the fitness value, a negative reward is assigned. This design ensures full consistency between the RL objective and the final scheduling evaluation criteria, preventing misalignment between the reward function and the actual performance metrics, and guaranteeing that the learned policy directly translates into overall improvements in scheduling quality.

This study adopts the standard Q-learning update rule, as shown in Eq. (25).

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta \left[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \quad (25)$$

η is the learning rate, and γ is the discount factor. Actions are selected using an ε-greedy policy, and the exploration rate is reduced through a multiplicative decay strategy. This setting maintains sufficient exploration in the early stage and gradually shifts toward exploitation in the later stage, thereby stably driving structured updates of the particle sequence.

3.6 RLSSO search procedure

3.6.1 Local search framework

In order to improve candidate solution quality and stabilize convergence, a local search step is applied after each operator update using two neighborhoods: (1) segment reversal (2-opt) and (2) adjacent swaps around the most urgent job. It should be noted that the local search procedure is separate from Action 3. Action 3 is one of the RL-selectable update operators, whereas the local search step is a post-update refinement procedure applied after the selected operator has generated a candidate solution. The local search follows a limited-try, improvement-accept rule:

1. 2-opt: Select a segment in the candidate sequence and reverse it to form a neighbor. Decode and evaluate the fitness. Accept the move if fitness improves; otherwise reject it.
2. Adjacent swap: Identify the position of the most urgent job and test adjacent swaps (left/right). Apply the same improvement-accept rule. This focuses computation on positions related to delay risk instead of scanning the full sequence.

The acceptance rule is elitist: only moves that reduce fitness are accepted. If no improvement is found within the preset number of trials, local search stops and outputs the final candidate solution. This prevents immediate fitness degradation and keeps the RL reward directionally consistent with performance improvement.

3.6.2 Overall RLSSO Procedure

Combining the action definition in Section 3.5.2, the reward and Q-learning update in Section 3.5.3, and the local search framework in Section 3.6.1, RLSSO executes a closed-loop update for each particle in each generation, including state observation, action selection, operator update, local improvement, fitness evaluation, memory update, and policy update.

1. State observation: For the current particle, construct the state vector from the current machine, job, and robotic-arm conditions.
2. Action selection and candidate generation: Select one action by the ε-greedy policy and apply the corresponding operator to generate a candidate solution.
3. Local refinement: Apply the local search procedure to the candidate solution using the improvement-accept rule.
4. Evaluation and memory update: Decode the refined candidate using the fast decoder and compute its fitness.

Update the particle's current solution with the refined candidate, and update the pbest and gbest solutions if improvement is obtained.

5. Policy learning and periodic validation: Compute the immediate reward from the fitness improvement, update the Q-table using the Q-learning rule, and periodically validate the current global-best solution by the high-fidelity DES decoder.

For reproducibility, the pseudocode of the overall procedure is summarized below (the RL update equation, reward definition, and exploration-rate setting are referenced from the previous section).

Algorithm 4 RLSSO

```
begin
    initialize population, fitness, pbest, gbest, and Q-table;
    for τ = 1 to T do
        for p = 1 to P do
            observe current state s_t;
            select action a_t by ε-greedy policy;
            store current fitness;
            apply operator to generate candidate;
            refine the candidate by local search;
            decode and evaluate the candidate;
            observe next state s_{t+1};
            update particle, pbest, and gbest if improved;
            compute reward and update Q-table;
        end
        decay exploration rate;
        if mod(τ, G) = 0 then
            validate best solution by high-fidelity;
        end
    end
    return the best validated solution;
end
```

4 Experimental Setup and Results

4.1 Experimental Environment and Dataset

This study evaluates RLSSO in a discrete-event simulation of a multi-operation production system with a single rail-mounted robotic-arm. The system contains 9 machines distributed across three process groups, namely EW, CNC, and EDM. Each job follows its required process route in sequence, while all inter-station transportation and loading/unloading tasks are executed by the same robotic-arm. Scheduling decisions, therefore, depend on machine availability, robotic-arm availability and position, and the inter-station transportation-time matrix.

The experimental data are calibrated to the layout and processing-time scale of the Research and Development Center for Smart Manufacturing at Chung Yuan Christian University. Station positions, equipment configurations, and representative processing-time ranges were obtained from site interviews and records and used to generate the test instances.

1. Robotic-arm travel matrix: To avoid arbitrary assumptions on transportation time, this study first constructs station coordinates according to the on-site layout. The geometric distance between stations u and v is computed by the Euclidean distance in Eq. (26).
$$\rho_{uv} = \sqrt{(x_u - x_v)^2 + (y_u - y_v)^2}, u \neq v \quad (26)$$
The distance is then linearly mapped to travel time, with a range of 5 to 20 seconds. A small random perturbation is added to reflect on-site variability, and the final values are clipped to the same range. Because the single-rail movement system is bidirectional, the transportation-time matrix is assumed to be symmetric, i.e., $t_{uv} = t_{vu}$.
2. Processing-time ranges for EW, CNC, and EDM are calibrated from site interviews and on-site measurements. Based on the collected information, job-specific processing times are sampled from the corresponding calibrated ranges. To control the bottleneck location across scenarios, process-group weights are introduced to determine the probability that each stage appears in a job route. In the baseline setting, the workload weights are EW:CNC:EDM = 0.3:0.5:0.2, so CNC appears more frequently. Alternative scenarios adjust these weights to shift the bottleneck among the three process groups.
3. Job data generation: The dataset contains 100 jobs. Each job includes a release time, due-date, order quantity, required processing stages, and stage-specific processing times. Release times are randomly sampled over a 7-day range from a common baseline starting at 08:00. Due-dates are generated by adding a due-date window of 14 to 28 days to the release time. Order quantities are sampled from 50 to 80 to preserve variability in job size.

For each job, 1 to 3 processing stages are selected according to the workload weights and then arranged in the fixed process order. Processing times are sampled from the calibrated range of each selected process type. During schedule construction, once an operation is assigned to a specific machine, the robotic-arm travel and handling times are computed according to its current position and the transportation-time matrix, thereby integrating transportation and machine-processing constraints into the scheduling results.

4.2 Compared Methods

To evaluate the effectiveness of the proposed method for the multi-operation integrated scheduling problem with a single-rail robotic-arm, this study considers representative benchmarks from different decision paradigms, including rule-based dispatching, swarm-based heuristic search, and multi-objective evolutionary optimization. The compared methods are summarized as follows.

1. EDD: Jobs are prioritized by due-date. At each decision epoch, an assignable job with the earliest due-date is selected first. Because of its low computational cost and suitability for real-time decision making, EDD is adopted as the rule-based benchmark.
2. SSO: A swarm-based search method explores the solution space using a single performance indicator. Candidate sequences are iteratively updated to improve scheduling quality. This method is included as a representative swarm-intelligence heuristic.
3. NSGA-II: A multi-objective evolutionary framework is applied to generate a Pareto solution set and is included as the representative multi-objective optimization method. To enable comparison with other methods under a single quantitative criterion, this study selects a representative solution from the Pareto set using the same weighted and normalized performance indicator Z as in the main experiments.
4. RLSSO (proposed): RL is integrated with swarm-based search. A learning-based policy adjusts update operations during the search process. Under an event-driven rescheduling procedure, the method modifies decisions that have not yet started within a limited computational budget, balancing solution quality and real-time

adjustment capability.

To ensure fair comparison, all methods are tested under the same system environment and the same input dataset. A consistent, feasible decoding procedure is used to transform sequencing decisions into executable schedules, and the same performance indicators are applied to evaluate scheduling performance. For methods requiring iterative search (SSO, NSGA-II, and RLSSO), the stopping conditions and key parameter settings are aligned (Table 2). Repeated experiments follow the same statistical aggregation procedure to support cross-method comparative analysis.

Table 2 Parameter settings of compared methods

Method	Parameter	Value
SSO	generations	200
	particles	500
	Cg	0.35
	Cp	0.65
	Cw	0.9
NSGA-II	population size	10
	generations	20
	pc	0.9
	pm	0.03
	Eta(SBX/PM)	15/20
	seed	42

Based on the preliminary calibration results in Tables 3 and 4, the final NSGA-II settings used in the main experiments were selected as $pm = 0.03, pc = 0.9$, and 20 generations, since this combination achieved the best overall fitness and the lowest tardiness level among the tested settings.

Table 3 Preliminary NSGA-II parameter configurations

ID	pm	pc	generations
0	0.03	0.8	10
1	0.05	0.8	10
2	0.03	0.9	10
3	0.05	0.9	10
4	0.03	0.8	20
5	0.05	0.8	20
6	0.03	0.9	20
7	0.05	0.9	20

Table 4 Scheduling results of preliminary NSGA-II calibration

ID	C_{max}	TD_j	N_j	Z	Cpu time
0	722.02	2880.26	18	17.65	1458.29
1	721.60	2877.64	17	17.53	1480.25
2	703.54	2912.43	18	17.77	1481.26
3	698.46	3107.01	18	18.73	1474.64
4	722.37	2823.02	17	17.26	2761.93
5	703.19	2900.50	17	17.61	2806.45
6	704.99	2815.60	16	17.09	2819.36
7	680.96	3185.25	19	19.19	2804.90

Note: NSGA-II uses population size 10 in all calibration setting.

4.3 Sensitivity analysis of fitness-scaling parameters

All experimental results in this study are evaluated using the integrated performance indicator *Z* defined in Section 3.3 To examine the influence of the normalization references and weight coefficients in *Z* on the performance of RLSSO, this study conducts a one-factor sensitivity analysis on $C_{ref}, T_{ref}, N_{ref}$ and the weights W_1, W_2, W_3 . The experiments follow a one-factor-at-a-time (OFAT) design: only one parameter is adjusted in each run while all other parameters remain fixed, and the mean *Z* value computed under the same test set is used for comparison. A smaller *Z* value indicates better overall scheduling quality.

Tables 5 and 6 summarize the calibration results under different parameter levels. Table 5 reports the results for the reference parameters, whereas Table 6 presents the results for the weight parameters. For the reference parameters, the lowest Z values were obtained at $C_{ref} = 1000$, $T_{ref} = 1000$ and $N_{ref} = 5$ while other levels resulted in inferior performance, indicating that inappropriate normalization references may reduce the effectiveness of the integrated indicator *Z*. Regarding the weight coefficients, W_1 and W_2 show a more pronounced impact on *Z*, suggesting that the weights assigned to C_{max} and TD_j directly influence the preference structure of the evaluation. When these weights deviate from appropriate levels, overall performance deteriorates. In contrast, W_3 exhibits a milder effect, although excessively large values still lead to performance degradation. Based on these results, the parameter combination $C_{ref} = 1000$, $T_{ref} = 1000$, $N_{ref} = 5$, $W_1 = 2$, $W_2 = 5$, and $W_3 = 0.5$. is adopted in the subsequent experiments.

Table 5 Calibration results of reference parameters

Parameter	Level	Min *Z*
C_{ref}	800	67.64
	1000	24
	1200	49.79
T_{ref}	500	52.41
	1000	28.68
	1500	60.23
N_{ref}	8	54.5
	5	23.49
	10	33.97

Table 6 Calibration results of weight parameters

Parameter	Level	Min *Z*
W_1	1	44.11
	2	26.44
	3	74.54
W_2	2	47.61
	5	31.22
	10	54.44
W_3	0.1	43.34
	0.5	31.94
	0.9	57.95

4.4 RLSSO parameter calibration and learning diagnostics

This subsection reports the calibration of key RLSSO-related parameters and provides additional diagnostic analyses of the learning process. First, a Taguchi orthogonal design was used to determine representative settings for the main learning hyperparameters. Next, the effect of the high-fidelity evaluation interval *G* was examined to balance solution quality and computational cost. Finally, an action–state diagnostic analysis was conducted to better understand the operator preferences learned by the RL agent under the selected setting.

4.4.1 Hyperparameter design and orthogonal experiment

This study adopted a Taguchi orthogonal design to calibrate the main learning-related parameters of RLSSO in a systematic yet computationally efficient manner. Four hyperparameters were selected as control factors: the learning rate η, the discount factor γ, the initial exploration rate ε_0, and the exploration decay coefficient ε_{decay}. These factors directly affect policy updating and the exploration–exploitation balance during search. Each factor was assigned three levels representing relatively conservative, moderate, and aggressive settings, as summarized in Table 7.

Table 7 RLSSO hyperparameter factors and levels

Factor	Parameter	Level 1	Level 2	Level 3
A	η	0.1	0.3	0.5
B	γ	0.5	0.7	0.9
C	ε_0	0.5	0.7	0.9
D	ε_{decay}	0.99	0.995	0.999

Because a full-factorial design would require $3^4 = 81$ parameter combinations, an L9 orthogonal array was used to reduce the experimental workload while retaining representative parameter patterns. For each configuration, RLSSO was executed under the same dataset, resource setting, and evaluation procedure, with consistent initial conditions and fixed random seeds to improve fairness and reproducibility. Since the scheduling objective jointly considers makespan, total tardiness, and the number of tardy jobs, performance was evaluated using the integrated indicator *Z*, where a smaller value indicates better overall scheduling quality. The L9 settings and corresponding results are reported in Tables 8 and 9. Among the tested configurations, Experiment 1 achieved the best overall result; therefore, the setting $\eta = 0.1$, $\gamma = 0.5$, $\varepsilon_0 = 0.5$ and $\varepsilon_{\text{decay}} = 0.99$ was adopted in the subsequent experiments.

Table 8 L9 orthogonal array settings

ID	η	γ	ε_0	ε_{decay}
1	0.1	0.5	0.5	0.99
2	0.1	0.7	0.7	0.995
3	0.1	0.9	0.9	0.999
4	0.3	0.5	0.7	0.999
5	0.3	0.7	0.9	0.990
6	0.3	0.9	0.5	0.995
7	0.5	0.5	0.9	0.995
8	0.5	0.7	0.5	0.999
9	0.5	0.9	0.7	0.990

Table 9 Performance results of L9 orthogonal experiments

ID	C_{max}	TD_j	N_j	Z
1	679.09	2057.50	13	12.95
2	679.00	2458.80	14	15.05
3	681.84	2606.36	15	15.90
4	700.45	2886.34	17	17.53
5	681.84	2655.45	15	16.14
6	676.23	2590.36	14	15.70
7	697.84	2657.02	15	16.18
8	726.95	2746.78	15	16.69
9	681.70	2544.25	15	15.58

4.4.2 Effect of the high-fidelity evaluation interval G

Table 10 reports the sensitivity analysis of the high-fidelity evaluation interval G. Smaller G leads to more frequent high-fidelity evaluations and better solution quality, but at a substantially higher computational cost. Although $G = 1$ achieved the best Z, it required 3995.56 s of CPU time. In contrast, $G = 20$ reduced CPU time to 1281.88 s while maintaining comparable solution quality. Therefore, $G = 20$ was adopted in subsequent experiments as a practical compromise between solution quality and computational efficiency.

Table 10 Effect of different high-fidelity evaluation intervals G on solution quality and CPU time

G	C_{max}	TD_j	N_j	Z	Cpu(sec)
1	723.59	2,864.69	14	17.17	3995.56
5	724.07	2923.25	14	17.46	1734.81
10	724.07	2923.25	14	17.46	1435.81
20	726.41	2988.45	12	17.59	1281.88

Note: generations = 200 and particles = 500 for all settings.

4.4.3 Action-State Diagnostic of RLSSO

An action–state diagnostic run was conducted to further examine the learning behavior of RLSSO under setting *G*=20. Table 11 summarizes the selection count, mean reward, reward standard deviation, and average slack associated with each operator. Action 1 is the dominant operator, with both the highest selection count and the highest mean reward, indicating that RLSSO mainly relies on guided exploitation around promising solutions. In contrast, Action 0 has the most negative mean reward and the largest variability, suggesting that it is exploratory but generally disruptive. Actions 2 and 4 provide smaller positive contributions, whereas Action 3 shows a slightly negative mean reward, indicating that its effectiveness is more context-dependent.

From the state perspective, actions with higher average slack are generally associated with more stable improvements, while tighter slack conditions tend to coincide with more conservative or locally corrective updates. The remaining state indicators, including minimum slack, robot idle condition, and average workstation utilization, vary within a narrower range across actions, suggesting that operator preference is influenced more strongly by due-date pressure than by large shifts in overall resource loading.

Table 11 Action usage and average reward of the RLSSO operators under the *G*=20 diagnostic run.

Action	Selection count	Mean reward	Reward Std.	Avg.Slack (hr)
0	4391	-70.20	41.59	103.09
1	66487	5.32	14.26	146.07
2	9427	0.45	3.91	113.07
3	6823	-0.32	5.32	92.32
4	12872	0.81	5.29	92.5

4.5 Overall Performance Comparison under Dynamic Job Arrival

This section adopts dynamic job arrivals as the baseline scenario and establishes a reference point for subsequent comparisons under different workload structures and parameter settings. The experiment first generates an initial schedule using 100 existing jobs, and then inserts 5 newly arriving jobs during execution to represent rescheduling pressure from real-time arrivals in practice. In this baseline setting, the five inserted jobs follow the early-arrival distribution, meaning that their arrival times are randomly sampled within the first 30% of the planning horizon. Here, the planning horizon is defined from the earliest arrival time to the latest due time in the original job set.

All methods are executed under the same dataset and the same resource configuration. Performance is measured by makespan, total tardiness, and the number of tardy jobs, and the integrated objective value Z defined in Section 3.3. Since NSGA-II produces a Pareto solution set, each Pareto solution is converted into Z, and the solution with the smallest Z is selected as the representative result for comparison. In addition, RLSSO and SSO share the same swarm-search budget and decoding procedure, while RLSSO additionally incorporates RL-guided operator selection and feedback updating. Therefore, the performance gap between RLSSO and SSO can be mainly attributed to the learning-guided update mechanism rather than differences in swarm settings. This comparison also serves as an ablation setting, in which SSO represents the counterpart without learning-guided operator selection but with the same swarm-search backbone and aligned parameter settings.

As shown in Table 12, RLSSO achieves the best result with Z = 12.95, outperforming NSGA-II (17.09), SSO (19.49), and EDD (22.72). Compared with EDD, RLSSO reduces makespan from 748.19 hr to 679.09 hr (9.24%), total tardiness from 3905.26 hr to 2057.50 hr (47.31%), and the number of tardy jobs from 17 to 13 (23.53%), leading to a 43.00% reduction in Z. These results indicate that RLSSO improves both production efficiency and due-date performance under dynamic job insertion.

Compared with SSO, RLSSO achieves clear improvements in all four indicators. Specifically, it reduces makespan from 729.96 hr to 679.09 hr (6.97%), total tardiness from 3267.00 hr to 2057.50 hr (37.02%), and the number of tardy jobs from 17 to 13 (23.53%). As a result, Z decreases from 19.49 to 12.95, corresponding to a 33.56% reduction. These results show that RL-guided operator selection provides a more effective balance between search intensification and due-date risk control than pure swarm search.

Compared with the representative NSGA-II solution, RLSSO also achieves better overall schedule quality. It reduces makespan from 704.99 hr to 679.09 hr (3.67%), total tardiness from 2815.60 hr to 2057.50 hr (26.92%), and the number of tardy jobs from 16 to 13 (18.75%), resulting in a Z reduction from 17.09 to 12.95 (24.22%). This suggests that although NSGA-II remains competitive, RLSSO provides stronger overall control under dynamic insertion disturbances through state-driven policy adjustment.

The results show limited variation across runs, indicating that RLSSO maintains stable performance under dynamic job insertions. Overall, the baseline comparison suggests that RLSSO provides advantages not only in due-date control but also in overall schedule efficiency under insertion disturbances.

Table 12 Comparison of the integrated performance indicator Z among four methods under the dynamic job insertion scenario.

Method	C_{max}	TD_j	N_j	Z
EDD	748.19	3905.26	17	22.72
SSO	729.96	3267	17	19.49
NSGA-II	704.99	2815.6	16	17.09
RLSSO	679.09	2057.50	13	12.95

4.5.1 Repeated tests under job insertion scenarios

To reduce the possibility that the result of a single job insertion case is affected by a specific insertion combination, this study further conducts repeated tests across 10 job insertion scenarios, to examine the robustness of RLSSO and quantify its average improvement. The machine configuration, robotic-arm configuration, and the parameter settings and dispatching rules of each method are kept identical across tests. Only the composition of the inserted jobs changes, reflecting the uncertainty in the combination of unexpected job arrivals in practical environments.

As shown in Table 13, RLSSO does not outperform EDD in makespan across all scenarios, with a win rate of 8/10. However, RLSSO consistently improves EDD in the two due-date related indicators, total tardiness and the number of tardy jobs, in all 10 scenarios (10/10). This indicates that under dynamic job insertion conditions, the RLSSO policy tends to prioritize the suppression of tardiness propagation. When inserted jobs increase due-date pressure or aggravate queueing at bottleneck stations, RLSSO reduces tardiness accumulation and the number of tardy jobs through due-date-oriented dispatching and local adjustments. In contrast, such adjustments may prevent makespan from decreasing in some scenarios, reflecting the trade-off between efficiency objectives and due-date objectives in dynamic environments.

Based on the average values calculated from the 10 scenarios in Table 13, RLSSO reduces makespan by approximately 4.94% relative to EDD, reduces total tardiness by approximately 34.33%, and reduces the number of tardy jobs by approximately 30.49%. In addition, the standard deviation of total tardiness under RLSSO is smaller than that under EDD, indicating that under variations in insertion combinations, RLSSO achieves not only better average performance but also lower performance variability. These results further suggest that RLSSO is particularly effective in stabilizing due-date-related performance when dynamic insertion patterns vary across scenarios.

Table 13 Repeated-test results under the job insertion scenarios.

	C_{max}		TD_j		N_j	
ID	EDD	RLSSO	EDD	RLSSO	EDD	RLSSO
1	818.03	777.75	4020.18	3798.02	18	17
2	801.42	765.75	6565.09	3923.55	30	15
3	843.28	725.79	6616.42	2870.66	26	12
4	800.92	797.87	5298.75	3634.83	21	16
5	840.71	797.69	6195.56	4155.78	22	19
6	794.53	729.95	4528.82	2934.19	17	13
7	825.73	748.05	6925.83	3407.88	26	15
8	777.31	748.01	5560.60	3550.05	26	15
9	792.24	796.76	4410.39	3441.69	18	15
10	764.46	773.11	4213.73	3970.19	19	18

4.5.2 Statistical significance test (paired-sample ANOVA / paired test)

To confirm that the performance improvement of RLSSO over EDD under job insertion scenarios is not caused by chance due to specific insertion combinations, this study conducts paired comparisons based on the 10 insertion scenarios reported in Table 13. Since each insertion scenario generates a matched pair of performance values for EDD and RLSSO, a paired-sample *t*-test is applied. Under the two-method comparison setting, this test is equivalent to a two-level repeated-measures ANOVA, and is used to examine whether the mean difference between methods is statistically significant for each performance indicator. The tested indicators include makespan, total tardiness, and the number of tardy jobs. The significance level is set to 0.05.

Table 14 summarizes the test results. The method effect is statistically significant for all three indicators, indicating that across the 10 insertion scenarios, RLSSO achieves significant improvements over EDD in makespan, total tardiness, and the number of tardy jobs. The differences are more pronounced for the due-date related indicators, consistent with the stability analysis results showing sustained improvements in due-date performance.

Table 14 Paired-sample ANOVA test results for EDD and RLSSO under job insertion scenarios ($N = 10$).

	C_{max}	TD_j	N_j
SSA	14,432.40	34,776,767	462.40
SST	30,352.65	47,945,959	724.00
MSA	14,432.40	34,776,767	462.40
MSE	1,768.92	1,463,243	29.07
F value	8.16	23.77	15.91
P-value	0.0189	0.00088	0.00316

4.6 Performance under Different Load Distributions

To evaluate the adaptability of the proposed RLSSO under bottleneck-shift scenarios, this study constructs three load distribution settings (Table 15) under the same system configuration and the same dynamic job insertion rules by adjusting the relative processing-time proportions of the three process types (EW, CNC, and EDM). The three settings correspond to CNC-heavy, EDM-heavy, and EW-heavy, representing typical metal machining environments in which the bottleneck is dominated by the middle stage, the final stage, and the initial stage, respectively. To avoid confounding effects caused by dataset scale and insertion ratio, all settings adopt the same dynamic arrival mechanism: 100 baseline jobs are scheduled first, and five dynamic jobs are inserted during schedule execution. The arrival times and due-date settings of inserted jobs are generated using the online arrival rules defined in this study, so that the comparisons in this section primarily reflect the effect of load-structure differences on scheduling performance.

In Table 15, S1 (CNC-heavy) makes the middle-stage CNC the primary bottleneck. This setting may cause waiting accumulation before CNC after upstream operations are completed, increasing the difficulty of coordinating processing pace and robotic-arm transportation. S2 (EDM-heavy) makes the final-stage EDM the bottleneck. Since the per-job processing time at the final stage is longer, delays tend to concentrate at the end of the process route, and the room for scheduling adjustment is relatively limited. S3 (EW-heavy) makes the initial-stage EW the bottleneck. In this case, work-in-process accumulation occurs early in the route. If transportation decisions and machine assignment decisions are not integrated, waiting effects may propagate downstream, increasing the risk of downstream machine idleness and blockage.

Table 15 Parameter settings for the three load distribution scenarios (Load Distribution Settings).

Scenario Code	Scenario Name	Processing Time Ratio (EW:CNC:EDM)	Main Bottleneck Process
S1	CNC-heavy	3:5:2	CNC
S2	EDM-heavy	2:3:5	EDM
S3	EW-heavy	5:2:3	EW

Table 16 summarizes the performance of EDD and RLSSO under the three load distribution scenarios. Overall, RLSSO outperforms EDD across all three bottleneck-location settings, and the improvement magnitude varies with the bottleneck position. In S1 (CNC-heavy), RLSSO achieves the largest reduction in total tardiness (30.56%), indicating that when mid-stage capacity is constrained and waiting tends to accumulate, RLSSO can suppress tardiness propagation and maintain system pacing. In S2 (EDM-heavy), the performance gap between the two methods is smaller, with a 5.53% reduction in total tardiness. This result is consistent with the limited scheduling flexibility under a final-stage long-processing-time bottleneck, which imposes structural constraints on achievable improvements. Notably, in S3 (EW-heavy), RLSSO reduces total tardiness and the number of tardy jobs by 61.65% and 64.29%, respectively, indicating that when upstream load increases, work-in-process accumulates earlier, and the effect of transportation integration becomes stronger, the proposed method can still coordinate processing and material-handling decisions and improve due-date performance while reducing tardiness risk.

Table 16 Performance comparison under different workload distributions (EDD vs. RLSSO).

Context	Obj	EDD	RLSSO	Improvement Scope
S1	C_{max}	967.67	864.07	10.71%
	TD_j	10583.15	7348.58	30.56%
	N_j	34	30	11.76%
S2	C_{max}	818.03	777.75	4.92%
	TD_j	4020.18	3798.02	5.53%
	N_j	18	17	5.56%
S3	C_{max}	722.73	609.63	15.65%
	TD_j	2133.05	817.94	61.65%
	N_j	14	5	64.29%

Taken together, the three workload settings indicate that RLSSO remains effective under different bottleneck locations, but its benefit is most pronounced in due-date related indicators rather than in makespan alone. In particular, when workload pressure accumulates earlier in the route or propagates more strongly across stages, the integration of learning-guided search and transportation-aware scheduling becomes more beneficial for suppressing tardiness propagation.

4.7 Taguchi-based analysis of online arrival environments

4.7.1 Definition of environmental factors and levels

To further examine the robustness of RLSSO under different online arrival environments, this study conducts a supplementary Taguchi-based environmental analysis. Unlike the main experiments, which focus on the baseline early-arrival insertion scenario, this section is intended to evaluate how the relative performance of RLSSO varies when the dynamic insertion environment changes. Three environmental factors are considered, namely the number of inserted jobs (*N*), the arrival distribution of inserted jobs (Pat.), and due-date tightness (λ_D). Each factor is assigned three representative levels, forming an L9 ($\mathbf{3^3}$) orthogonal design.

1. Number of inserted jobs
   This factor controls the level of dynamic workload. Three levels are considered: 3, 6, and 10, representing low, medium, and high insertion intensity, respectively. Larger values imply stronger rescheduling pressure and heavier competition for shared resources.
2. Arrival distribution of inserted jobs
   This factor controls the release window of inserted jobs within the planning horizon, with three levels: early (0%-30%), mid (30%-70%), and late (70%-100%). These settings represent front-loaded, mid-horizon, and late-concentrated insertion patterns, respectively.
3. Due-date tightness (λ_D)
   To control the urgency of inserted jobs, the due-date of each inserted job is scaled from its baseline lead time using a due-date coefficient $\lambda_D \in \{0.8,1.0,1.2\}$. Let a_j denote the arrival time of inserted job j, and let T_j^{base} denote the baseline lead time inherited from its template job. The due-date is then defined as

$$\tilde{d}_j = a_j + \lambda_D T_j^{base} \quad (28)$$

   where $\lambda_D = 1.0$, represents the baseline setting, $\lambda_D = 0.8$ corresponds to a tighter due-date condition, and $\lambda_D = 1.2$ corresponds to a looser one.

4.7.2 Experimental design and performance evaluation

Based on the above factors and levels, this study adopts an L9 ($\mathbf{3^3}$) orthogonal array to construct nine online arrival environment combinations, as summarized in Table 17. Except for the insertion generation mechanism, all other settings remain consistent with the baseline dynamic insertion experiment, including the base job dataset, machine configuration, robotic-arm parameters, and the definitions of processing and transportation times. This control strategy ensures that the observed differences mainly reflect the online arrival environment itself.

For performance evaluation, this section continues to use the integrated indicator Z (smaller is better). For the *e*-th environment setting, the EDD baseline result and the RLSSO result are denoted by Z_e^{edd} and Z_e^{RLSSO}, respectively. To quantify the relative effect of RLSSO, the absolute difference and relative improvement rate are defined as

$$Z_e = Z_e^{edd} - Z_e^{RLSSO} \quad (29)$$

$$Improve\%_e = \frac{\Delta Z_e}{Z_e^{edd}} \times 100\% \quad (30)$$

Accordingly, $\Delta Z_e > 0$ indicates that RLSSO outperforms EDD under the *e*-th environment setting, whereas $\Delta Z_e < 0$ indicates that RLSSO does not outperform the baseline.

Table 17 shows that the relative performance of RLSSO depends strongly on the online arrival environment. Under low insertion intensity, RLSSO does not consistently outperform EDD. As insertion pressure increases, its relative advantage becomes more evident. In particular, the largest improvement is observed under the setting with 10 inserted jobs, late arrivals, and $\lambda_D = 1.0$, where RLSSO reduces *Z* by 39.37% relative to EDD. However, this does not imply that the late-arrival environment yields the best absolute scheduling quality. The Taguchi analysis should therefore be interpreted as a supplementary robustness test, used to identify the environments in which the relative advantage of RLSSO becomes more or less pronounced, rather than to replace the early-arrival baseline used in the main experiments.

Table 17 Taguchi L9 design and results for online arrival environments.

N	Pat.	λ_D	Z_e^{edd}	Z_e^{RLSSO}	ΔZ_e	$Improve\%_e$
3	early	0.8	17.67	22.62	-4.95	-28.02
3	mid	1.0	16.63	16.76	-0.13	-0.81
3	late	1.2	22.55	27.85	-5.30	-23.50
6	early	1.0	22.37	28.41	-6.04	-27.01
6	mid	1.2	25.35	28.21	-2.86	-11.29
6	late	0.8	32.88	28.07	+4.81	+14.62
10	early	1.2	38.07	37.69	+0.38	+0.99
10	mid	0.8	37.96	37.93	0.02	+0.06
10	late	1.0	44.15	26.76	+17.38	+39.37

Note: N denotes the number of inserted jobs, and Pattern denotes the arrival pattern of inserted jobs.

Overall, RLSSO demonstrates competitive and often superior performance across the dynamic insertion environments considered, with its main advantage appearing in due-date related indicators rather than in universal dominance across all settings. In the main early-arrival baseline scenario, RLSSO achieves the best integrated objective value *Z* among the compared methods while maintaining a makespan level close to the strongest competing solutions. Its main advantage is more clearly reflected in due-date related indicators, including total tardiness and the number of tardy jobs, suggesting that the proposed learning-guided update mechanism is especially effective in suppressing delay accumulation under insertion disturbances. The comparison with SSO further indicates that this advantage is not solely due to the swarm-search backbone, but also to the adaptive operator-selection mechanism introduced by RL.

5 Conclusion and Future Work

This study addresses an integrated scheduling problem that jointly considers machining resources and a single rail-mounted robotic-arm as a shared transportation resource. A unified framework is developed to coordinate processing and transportation decisions under dynamic job arrivals, so that machine availability, robotic-arm movements, and waiting behavior are explicitly represented in the scheduling process. In addition, makespan, total tardiness, and the number of tardy jobs are combined into a composite objective value Z, enabling the evaluation of production efficiency and due-date-related risks within a single criterion. Based on this formulation, RLSSO is proposed as a hybrid approach that combines the exploration capability of simplified swarm optimization with learning-guided operator selection, thereby improving adaptability under dynamic scheduling conditions.

In the main early-arrival dynamic insertion scenario, RLSSO achieved the best integrated objective value among the compared methods. Compared with the representative

NSGA-II solution, RLSSO also achieves better overall schedule quality. It reduces makespan from 704.99 hr to 679.09 hr (3.67%), total tardiness from 2815.60 hr to 2057.50 hr (26.92%), and the number of tardy jobs from 16 to 13 (18.75%), resulting in a Z reduction from 17.09 to 12.95 (24.22%). A Z reduction indicates that the proposed method provides superior overall scheduling performance, with improvements in both production efficiency and due-date related outcomes, under the tested dynamic insertion setting. The supplementary analyses further support the robustness of the proposed framework, especially under stronger insertion pressure, although the main conclusions of this study remain based on the early-arrival baseline scenario.

From a practical perspective, RLSSO is particularly suitable for production settings in which scheduling decisions must be repeatedly updated under shared transportation constraints and due-date pressure. The results suggest that, for multi-stage production lines served by a single robotic-arm, improved scheduling control can reduce tardiness-related losses while also improving overall schedule efficiency, without requiring immediate capacity expansion. In this sense, the proposed framework can serve as a decision-support basis for production planning systems such as MES or APS, especially when frequent job insertions and resource interactions make simple dispatching rules insufficient.

Several limitations remain. The present study considers a single robotic-arm, fixed process precedence, and environments without machine breakdowns or setup-related uncertainties. Future work may extend the framework to multi-robot environments, stochastic disruptions, richer shop-floor constraints, and broader online learning settings for more complex manufacturing systems.

References

1. Gultekin, H., Akturk, M. S., & Karasan, O. E. (2007). Scheduling in a three-machine robotic flexible manufacturing cell. Computers & Operations Research, 34(8), 2463–2477. https://doi.org/10.1016/j.cor.2005.09.015
2. Zhang, S., Tang, F., Li, X., Liu, J., & Zhang, B. (2021). A hybrid multi-objective approach for real-time flexible production scheduling and rescheduling under dynamic environment in Industry 4.0 context. Computers & Operations Research, 132, 105267. https://doi.org/10.1016/j.cor.2021.105267
3. Zisgen, H., Miltenberger, R., Hochhaus, M., & Stöhr, N. (2023). Dynamic scheduling of gantry robots using simulation and reinforcement learning. 2023 Winter Simulation Conference (WSC). https://doi.org/10.1109/WSC60868.2023.10407159
4. Wang, D., Liu, S., Zou, J., Qiao, W., & Jin, S. (2025). Flexible robotic cell scheduling with graph neural network based deep reinforcement learning. Journal of manufacturing systems, 78, 81–93. https://doi.org/10.1016/j.jmsy.2024.11.010
5. Teymourifar, A., Ozturk, G., Ozturk, Z. K., & Bahadir, O. (2020). Extracting new dispatching rules for multi-objective dynamic flexible job shop scheduling with limited buffer spaces. Cognitive Computation, 12(1), 195–205. https://doi.org/10.1007/s12559-018-9595-4
6. Heger, J., Branke, J., Hildebrandt, T., & Scholz-Reiter, B. (2016). Dynamic adjustment of dispatching rule parameters in flow shops with sequence-dependent set-up times. International Journal of Production Research, 54(22), 6812–6824. https://doi.org/10.1080/00207543.2016.1178406
7. Wang, Y., Liu, Y., & Ma, L. (2026). A multi-objective variable-speed flexible job-shop scheduling model and enhanced NSGA-II algorithm considering carbon policies. Evolutionary Intelligence, 19(24), 1–24. https://doi.org/10.1007/s12065-026-01144-z
8. Yang, H., & Fujimura, S. (2026). Multi-objective comprehensive approach for flexible assembly job shop scheduling problem with AGVs. IEEJ Transactions on Electrical and Electronic Engineering, 21(2), 235–248. https://doi.org/10.1002/tee.70114
9. Luo, J., El Baz, D., Xue, R., Hu, J., & Shi, L. (2025). A fully parallel multi-objective genetic algorithm for optimization of flexible shop floor production performance and schedule stability under dynamic environments. Annals of Operations Research, 1–36. https://doi.org/10.1007/s10479-025-06482-2
10. Li, J., Zhang, W., & Li, J. (2025). Solving multi-objective energy-efficient flexible job shop problems by a dual-level NSGA-II algorithm. Memetic Computing, 17, Article 10. https://doi.org/10.1007/s12293-025-00449-3
11. Zisgen Zhang, Y., Zhu, H., Tang, D., Zhou, T., & Gui, Y. (2022). Dynamic job shop scheduling based on deep reinforcement learning for multi-agent manufacturing systems. Robotics and Computer-Integrated Manufacturing, 78, 102412. https://doi.org/10.1016/j.rcim.2022.102412
12. Ribas, I., Leisten, R., & Framiñan, J. M. (2010). Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. Computers & Operations Research, 37(8), 1439–1454. https://doi.org/10.1016/j.cor.2009.11.001
13. Truscott, W. G. (1985). Scheduling production activities in multi-stage batch manufacturing systems. International Journal of Production Research, 23(2), 315–328. https://doi.org/10.1080/00207548508904710
14. Liang T, Zhou L, Jiang Z. Integrated scheduling of production and material delivery for the intelligent manufacturing system. International Journal of Production Research. 2025;63(3):882-903. https://doi.org/10.1080/00207543.2024.2363435
15. Shakeri, Z., Benfriha, K., Varmazyar, M., Talhi, E., & Quenehen, A. (2025). Production scheduling with multi-robot task allocation in a real industry 4.0 setting. Scientific Reports, 15(1), 1795. https://doi.org/10.1038/s41598-024-84240-3
16. Xu, H., Yang, Y., Min, F., Huang, J., Chen, X., Han, Y., & Sun, N. (2022). Toward efficient computing for robotics: From a circuit and system view. IEEE Transactions on Circuits and Systems II: Express Briefs, 69(7), 3051–3056. https://doi.org/10.1109/TCSII.2022.3179871
17. Feng, J., Che, A., Chu, C., Levner, E., & Kats, V. (2024). Scheduling robotic cells with fixed processing times or time windows: Classification, solution approaches, polynomial algorithms and complexity. European Journal of Operational Research, 319(2), 468–483. https://doi.org/10.1016/j.ejor.2024.01.041
18. Kim, H.-J., & Lee, J.-H. (2023). Look-ahead based reinforcement learning for robotic flow shop scheduling. Journal of manufacturing systems, 68, 160–175. https://doi.org/10.1016/j.jmsy.2023.02.002
19. Xiong, H., Fan, H., Jiang, G., & Li, G. (2017). A simulation-based study of dispatching rules in a dynamic job shop scheduling problem with batch release and extended technical precedence constraints. European Journal of Operational Research, 257(1), 13–24. https://doi.org/10.1016/j.ejor.2016.07.030
20. Su, P.-C., Tan, S.-Y., Liu, Z., & Yeh, W.-C. (2022). A Mixed-Heuristic Quantum-Inspired Simplified Swarm Optimization Algorithm for scheduling of real-time tasks in the multiprocessor system. Applied Soft Computing, 131, 109807. https://doi.org/10.1016/j.asoc.2022.109807
21. Byeon, H. (2023). Advances in value-based, policy-based, and deep learning-based reinforcement learning. International Journal of Advanced Computer Science and Applications, 14(8). https://doi.org/10.14569/IJACSA.2023.0140838
22. Bouazza, W., Sallez, Y., & Beldjilali, B. (2017). A distributed approach solving partially flexible job-shop scheduling problem with a Q-learning effect. IFAC-PapersOnLine, 50(1), 15890–15895. https://doi.org/10.1016/j.ifacol.2017.08.2354
23. Luo, S. (2020). Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. Applied Soft Computing, 91, 106208. https://doi.org/10.1016/j.asoc.2020.106208
24. Dulac-Arnold, G., Levine, N., Mankowitz, D. J., Li, J., Paduraru, C., Gowal, S., & Hester, T. (2021). Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. Machine Learning, 110(9), 2419–2468. https://doi.org/10.1007/s10994-021-05961-4
25. Song, W., Chen, X., Li, Q., & Cao, Z. (2022). Flexible job-shop scheduling via graph neural network and deep reinforcement learning. IEEE Transactions on Industrial Informatics, 19(2), 1600–1610. https://doi.org/10.1109/TII.2022.3189725
26. Hu, L., Liu, Z., Hu, W., Wang, Y., Tan, J., & Wu, F. (2020). Petri-net-based dynamic scheduling of flexible manufacturing system via deep reinforcement learning with graph convolutional network. Journal of manufacturing systems, 55, 1–14. https://doi.org/10.1016/j.jmsy.2020.02.004
27. Yuan, E., Wang, L., Song, S., Cheng, S., & Fan, W. (2025). Dynamic scheduling for multi-objective flexible job shop via deep reinforcement learning. Applied Soft Computing, 171, 112787. https://doi.org/10.1016/j.asoc.2025.112787
28. Xu, Y., Song, Y., Pi, D., Chen, Y., Qin, S., Zhang, X., & Yang, S. (2023). A reinforcement learning-based multi-objective optimization in an interval and dynamic environment. Knowledge-Based Systems, 280, 111019. https://doi.org/10.1016/j.knosys.2023.111019
29. Bora, T. C., Mariani, V. C., & dos Santos Coelho, L. (2019). Multi-objective optimization of the environmental-economic dispatch with

reinforcement learning based on non-dominated sorting genetic algorithm. Applied Thermal Engineering, 146, 688–700. https://doi.org/10.1016/j.applthermaleng.2018.10.020
30. Zhang, Z., Wu, F., & Wu, A. (2024). Research on multi-objective process parameter optimization method in hard turning based on an improved NSGA-II algorithm. Processes, 12(5), 950. https://doi.org/10.3390/pr12050950
31. Pei, J., Mei, Y., Liu, J., Zhang, M., & Yao, X. (2025). Adaptive operator selection for meta-heuristics: A survey. IEEE Transactions on Artificial Intelligence. https://doi.org/10.1109/TAI.2025.3545792
32. Mahmud, S., Abbasi, A., Chakrabortty, R. K., & Ryan, M. J. (2022). A self-adaptive hyper-heuristic based multi-objective optimisation approach for integrated supply chain scheduling problems. Knowledge-Based Systems, 251, 109190. https://doi.org/10.1016/j.knosys.2022.109190
33. Ma, K. T., Chou, C. W., Chiu, W. C., & Luo, R. H. (2025). Integrating genetic algorithm and simulation for stochastic flow shop scheduling with delivery risk in System-in-Package assembly. Computers & Industrial Engineering, 111665. https://doi.org/10.1016/j.cie.2025.111665

Declarations

Funding: This work was supported by the National Science and Technology Council (NSTC), Taiwan, under grant numbers NSTC114-2622-E007-021 and NSTC114-2221-E259-013. The recipients of these grants include Jia-Xuan Khoo, Yi-Yun Ni, and Kang-Ting Ma.

Data Availability Statement: The datasets used in this study, including the Excel files before and after job insertion, are publicly available at the following repository: https://github.com/611337004-hub/rlsso-scheduling-data

Conflicts of Interest: On behalf of all authors, the corresponding author states that there are no conflicts of interest.