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.

Energy-Efficient Dynamic Flexible Job Shop Scheduling Using Improved Niching Genetic Programming and Deep Reinforcement Learning

Gang Wang
Xinjiang University
Adilanmu Sitahong
adilanmujx@xju.edu.cn

Xinjiang University
Yiping Yuan
Xinjiang University
Peiyin Mo
Xinjiang University
JunYan Ma
Xinjiang University
YuLong Chen
Xinjiang University

Article

Keywords: dynamic flexible job shop scheduling, niching genetic programming, random immigrants, deep reinforcement learning, energy-efficient scheduling

Posted Date: November 26th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11457724/v1

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

Additional Declarations: No competing interests reported.

Energy-Efficient Dynamic Flexible Job Shop Scheduling Using Improved Niching Genetic Programming and Deep Reinforcement Learning

Gang Wang[a], Adilanmu Sitahong[a], Yiping Yuan[a], Peiyin Mo[a], JunYan Ma[b],YuLong Chen[a]

[a] School of Mechanical Engineering, Xinjiang University, Urumqi 830047, China

[b] School of Economic and Management, Xinjiang University, Urumqi 830047, China

Correspondence: adilanmujx@xju.edu.cn

Abstract

Dynamic flexible job shops require continual adjustment of machine assignments and operation sequences in response to dynamic events while balancing due-date performance and energy consumption. Existing approaches often give insufficient attention to energy efficiency, while rule-library renewal must account for both individual quality and behavioural differences. This paper proposes INichGPDRL, an approach combining improved niching genetic programming with deep reinforcement learning. In the offline stage, dual-tree individuals are evaluated using a joint tardiness–energy fitness function, and a rule library is constructed through niche clearing and random-immigrant screening based on fitness and behavioural novelty. In the online stage, complete individuals are selected according to the shop state, and their routing or sequencing trees are invoked as required. Free float is then used to reduce the speed of eligible non-final operations while retaining machine assignments and processing order. Across four scenarios, INichGPDRL achieves the lowest mean total tardiness in three and the lowest mean total energy consumption in two of these three. Compared with GPDRL, it reduces mean total tardiness by 18.61% in the scenario with high processing-time variability and relatively loose due dates. Speed post-processing reduces energy consumption in all 40 paired runs, with mean savings of 0.86%–1.96% across scenarios and no change in total tardiness. These results show that the approach improves due-date and energy performance under some dynamic production conditions and obtains further energy savings from available time margins.

Keywords: dynamic flexible job shop scheduling; niching genetic programming; random immigrants; deep reinforcement learning; energy-efficient scheduling

1 Introduction

The global transition towards a low-carbon economy has made lower energy consumption and more efficient energy use important priorities for manufacturing. According to the International Energy Agency's Energy Efficiency 2024 report, industry accounted for approximately 39% of global final energy consumption in 2023 [1]. Manufacturing is a major component of this sector, and its shop-floor equipment consumes energy during both processing and waiting. A central challenge in shop scheduling is therefore to arrange production tasks so that energy can be saved while delivery requirements are met.

Dynamic flexible job shop scheduling (DFJSS) is a complex combinatorial optimisation problem with multiple constraints and an important topic in production scheduling [2]. In addition to determining operation sequences, a flexible job shop must assign each operation to a machine from its eligible set. These two decisions jointly determine how production tasks are executed. Compared with static scheduling, dynamic events such as random job arrivals and machine breakdowns continually

change processing requirements and machine availability, requiring decisions to adapt to the evolving shop state [3]. Scheduling methods must therefore respond rapidly to changes while coordinating routing and sequencing to support due-date and energy objectives.

Manually designed dispatching rules make decisions by comparing the priorities of candidate operations or machines. Their low computational cost makes them suitable for dynamic production environments that require rapid responses. However, their design usually relies on expert knowledge or local performance measures, requires considerable effort, and is sensitive to production objectives and operating conditions [4]. Genetic programming (GP) automatically learns dispatching rules by evolving priority expressions, reducing dependence on manual rule design [5]. Nevertheless, GP methods that seek a single rule generally evaluate individuals by their overall performance on training scenarios and use the selected rule throughout subsequent scheduling [6]. Although its priority outputs change with real-time shop features, the decision expression remains fixed and cannot exploit the strengths of other rules in different states. Moreover, declining behavioural diversity during evolution can produce similar candidate rules and further restrict the scope for subsequent rule selection [7].

To exploit different rules as production conditions change, researchers have introduced reinforcement learning (RL) into dynamic scheduling. Deep reinforcement learning (DRL) learns relationships between shop states and action values, allowing rules to be selected according to the current state [8]. In approaches combining GP and DRL, GP generates candidate rules and DRL learns when to use them. Xu et al. used niching GP to evolve sequencing and routing rules with different behaviours, then supplied evolved sequencing rules as candidate actions [9]. Such approaches make the rule library part of the agent's decision space. Library construction must therefore consider both scheduling quality and the different choices available to the subsequent selection policy. High fitness alone does not establish behavioural differences, while different behaviours need not yield useful scheduling choices.

For energy-efficient DFJSS, candidate quality must also reflect the trade-off between due-date performance and energy consumption. Machine assignments and operation sequences alter processing and waiting times, so a schedule with good due-date performance does not necessarily consume less energy. Processing speed creates a further trade-off: lower speeds reduce processing power but extend processing time and may increase tardiness risk [10]. This motivates considering rule evaluation, admission and online use together when constructing a library for both objectives. Once routing and sequencing decisions are fixed, available time margins can be examined separately for additional energy savings without delaying job completion.

This paper proposes INichGPDRL to address rule-library construction and online use for joint tardiness and energy optimisation. The principal modification concerns how new individuals enter the niching GP population. Joint fitness evaluation and admission conditions on quality and behavioural differences determine which candidates are introduced. The resulting library supplies complete rule individuals to a shared Double DQN, making both evolved routing and sequencing rules available during scheduling. A subsequent free-float procedure adjusts eligible operation speeds within the fixed schedule. The main contributions are as follows.

(1) An improved niching GP procedure is developed to construct a rule library for tardiness and energy optimisation. Normalised joint fitness evaluates scheduling quality, while phenotypic characterisation and niche clearing distinguish decision behaviours. Random immigrants are admitted subject to both fitness and behavioural conditions, making the introduction of new behaviours conditional on scheduling quality. This procedure addresses candidate admission alongside the removal of behavioural duplicates.

(2) A shared Double DQN framework is developed for state-dependent selection of complete rule individuals. A decision-type indicator distinguishes routing from sequencing, and the corresponding tree of the selected individual is applied at each decision point. This representation allows both evolved routing and sequencing rules to participate in online scheduling while keeping the action-space size independent of the numbers of candidate machines and operations.

(3) A free-float-based speed adjustment procedure complements rule-based scheduling with a separate energy-saving step. Machine assignments and processing order remain fixed, and each non-final operation is checked against the additional duration required at the lower predefined speed. Eligible operations are slowed without delaying successors or job completion under the recorded event conditions.

The remainder of this paper is organised as follows. Section 2 reviews related work. Section 3 presents the notation and mathematical model for DFJSS. Section 4 describes the proposed approach and its stages. Section 5 details the experimental settings, state features and comparison algorithms. Section 6 analyses overall performance, the effects of individual improvements, behavioural diversity and parameter sensitivity. Section 7 concludes the paper and discusses future research.

2 Related work

2.1 Learning dispatching rules with genetic programming

Shop scheduling methods commonly include exact optimisation, heuristics and metaheuristics. Exact methods solve mathematical models, but their computational cost increases with problem size and constraint complexity. Metaheuristics, including genetic algorithms and particle swarm optimisation, search iteratively for good schedules and must balance solution quality against computation time [11]. Among heuristic methods, dispatching rules use information such as processing times, due dates and machine loads to assign priorities directly to candidate tasks, making them suitable for dynamic production processes with frequent decisions. However, constructing effective rules for different objectives and production conditions generally depends on expert knowledge and requires considerable time and effort [12].

Unlike metaheuristics that search directly for specific schedules, GP evolves dispatching rules themselves. It combines shop features and functions into priority expressions and evaluates individuals using the schedules they generate [13]. Research has mainly addressed rule representations, feature selection, and coordination between routing and sequencing. Nguyen et al. compared alternative representations and showed that incorporating system-state and machine information could improve learned rules [14]. Zhang et al. developed a representation for dynamic flexible job shops to capture the contributions of different features to scheduling decisions [15]. To coordinate routing and sequencing, Xu et al. introduced delayed routing based on updated shop information and evaluated it in dynamic flexible scheduling with mean tardiness and energy-efficiency objectives [16].

Beyond learning a single rule, another line of research learns rule sets to exploit complementarity. Park et al. used GP and cooperative coevolution to construct dispatching-rule ensembles and examined their robustness relative to individual rules in static job shops [17]. For dynamic flexible job shops, Xu et al. included both single-rule and ensemble individuals in the population, extending ensemble evolution by constructing and selecting diverse, complementary members [6]. Li et al. improved population initialisation, population partitioning and mutation to mine composite dispatching rules and used the learned rules as candidate DRL actions [18]. As the learning objective shifts from a single rule to a selectable rule library, individual fitness alone is insufficient: whether the rules make different scheduling choices must also be considered. Different expression structures do not necessarily produce

different decisions, so simply increasing rule counts or preserving structural differences does not establish the effective choice space of a rule set.

To retain different rule behaviours and mitigate premature convergence caused by declining diversity, researchers have improved both selection and behavioural management. Xu et al. introduced lexicase selection for large-scale DFJSS, constructing multiple evaluation cases from a single simulation so that performance across cases informs selection while balancing population diversity and evaluation efficiency [7]. Sitahong et al. combined dynamic diversity management with online feature selection in energy-aware dynamic job shops to investigate the transition from exploration to exploitation and rule interpretability [19]. In contrast, the NichGP method of Xu et al. characterises rules through their choices at reference decision points and uses niche clearing to suppress redundant individuals in the same behavioural region, providing behaviourally different candidate rules for DRL [9]. Lexicase selection primarily distinguishes individuals through case-wise performance, whereas niche clearing directly limits repeated decision behaviours. They therefore address diversity through evaluation performance and decision behaviour, respectively.

These studies extend GP learning from a single rule to sets with different scheduling behaviours, allowing rule complementarity to be exploited. Constructing a library, however, does not by itself determine how its rules should be used. As jobs arrive, machine loads change and equipment availability varies, the relative strengths of rules may also change. Applying a fixed rule selected solely from offline evaluations may therefore underuse the choices available in the set. Automatic rule generation consequently needs to be complemented by learning the relationship between production states and rule selection. RL evaluates decisions through interaction with the environment and offers a means of learning this dynamic selection process.

2.2 Reinforcement learning for scheduling

Traditional approaches to dynamic scheduling either revise an existing schedule through event-driven or periodic rescheduling, or apply dispatching rules directly at each decision point [20]. The former can exploit more complete schedule information, but frequent iterative optimisation increases decision overhead. The latter responds rapidly but requires rules and their conditions of use to be specified in advance. An alternative is to learn a policy during training and select actions directly from the current state during execution, balancing response speed and scheduling quality. RL evaluates the long-term effects of actions through environment interaction, while DRL uses neural networks to represent relationships between shop states and decisions [21].

Existing DRL scheduling approaches mainly involve indirect decisions through rule selection or end-to-end decisions that directly select operations and machines. Rule-selection approaches use dispatching rules as actions and learn when to apply them. Luo used DQN to select composite dispatching rules from production states for flexible job shops with new job insertions, aiming to minimise total tardiness [8]. For multiple objectives, Luo et al. developed a two-hierarchy deep Q-network whose upper level selects an interim objective and whose lower level selects a composite rule for that objective [22]. Direct approaches instead learn choices among candidate tasks and machines. Lei et al. represented operation–machine relationships as a graph and used a multi-pointer graph network with multiple proximal policy optimisation (PPO) agents to learn operation- and machine-selection subpolicies [23]. DRL decision-making therefore depends not only on the learning algorithm but also on the action representation and the organisation of shop information.

For rule-selection methods, candidate-action quality limits the scheduling behaviours available to the agent. To reduce reliance on manually designed rules, researchers have combined GP rule generation with DRL policy learning. Xu et al.'s NichGPDRL first evolves sequencing and routing rules using niching GP, then replaces the sequencing agent's manual actions with evolved sequencing

rules, while a routing agent directly selects candidate machines [9]. In dynamic container-truck scheduling, Chen et al. used RL to coordinate GP-generated heuristics and their ensembles, further investigating automatic rule construction and state-dependent selection [24]. The learning characteristics of GP and RL also affect how they should be combined. Preliminary comparisons by Xu et al. showed differences in scheduling performance and stability under changes in training data, but these findings apply to the algorithms and experimental conditions examined and do not establish universal superiority of either class [25].

Although DRL enables state-dependent scheduling, its effectiveness remains jointly influenced by candidate actions, state representation and the division of decisions. Using rules as actions bounds the choice space, but improvements to the selection policy cannot readily compensate for a library of poor-quality or overly similar rules. Direct operation and machine selection instead needs to handle variable candidate counts and feasible-action constraints [26]. Approaches combining GP and DRL must also coordinate their two stages. For example, NichGPDRL evaluates sequencing and routing rules together during evolution but combines sequencing-rule selection with direct machine selection during DRL [9]. This raises questions about how evolved candidates participate in subsequent learning and whether the learning objectives reflect production requirements. When energy consumption becomes an additional objective, action evaluation must account for the effects of routing, sequencing and equipment states on energy use. Rule selection must consequently be connected to the mechanisms of energy-efficient scheduling.

2.3 Energy-saving strategies in shop scheduling

Assessing the energy effects of scheduling first requires identifying the sources of consumption. Shop-floor energy use includes operation processing and non-processing activities such as idling, standby, startup and shutdown. Studies adopt energy models according to equipment states and problem assumptions. A shorter makespan therefore does not necessarily yield a proportional reduction in energy consumption; processing tasks, equipment states and their durations must be considered together. By decision scope, energy-saving strategies can be grouped into equipment operation control and production scheduling. The former changes equipment energy states through switching, standby or speed adjustment, while the latter changes energy use through machine assignments, operation sequences and production timing. These strategies can be combined [27].

Idle energy consumption is a major concern in equipment operation control. May et al. integrated machine states with production scheduling and used a multi-objective genetic algorithm to examine the trade-off between production efficiency and energy consumption [28]. Switching off idle machines can reduce waiting-related energy use, but restart time and energy must also be included. Gong et al. additionally treated the number of machine restarts as an objective, using a two-stage memetic algorithm and operation-block moves to reduce energy consumption and restarts, and examining schedule adjustments that preserve makespan [29]. The effectiveness of switching strategies therefore depends on the duration and distribution of idle intervals, rather than simply on whether a machine is currently idle.

In addition to reducing idle energy use, equipment control can change processing power and duration through speed adjustment. Wei et al. developed an energy estimation model covering different operating states and considered variable speeds alongside other energy-saving measures in flexible job shops [30]. Qu et al. refined energy consumption to the operation level and incorporated processing-quality requirements and equipment switching intervals into speed selection [31]. Under dynamic disturbances, Duan and Wang jointly studied idle-time allocation and machine-speed levels with breakdowns, using multi-objective search to balance total energy consumption and makespan [32]. These studies show that the power reduction obtained at lower speeds must be assessed alongside the increase in processing time, and feasible speeds must respect technological and equipment conditions.

Production scheduling saves energy mainly through resource selection and task allocation. Zhang and Chiong jointly minimised total weighted tardiness and total energy consumption using a multi-objective genetic algorithm with enhanced local search [33]. For flexible job shops, Zhang et al. modelled machine selection, operation sequencing and switching decisions together, using gene expression programming to learn energy-related rules that incorporate equipment energy features [34]. Park and Ham studied flexible scheduling under time-of-use electricity tariffs and planned downtime, arranging production to optimise makespan and electricity cost [35]. Rui et al. incorporated demand-response information into heterogeneous graph representations and used graph RL to coordinate production and energy demand [36]. Shifting electricity use can change costs and load profiles, but lower electricity costs should not be interpreted directly as lower total energy consumption.

Equipment control changes energy-use states, while production scheduling changes task allocation; combining them requires their respective conditions to be satisfied. Switching depends on idle intervals and startup costs, speed adjustment must respect technological and timing constraints, and dynamic environments require timely responses to new tasks and equipment disturbances [37]. Energy-efficient DFJSS therefore needs to include the energy effects of routing and sequencing in decision evaluation and consider equipment adjustments subject to delivery requirements. Together with the preceding research on rule learning and dynamic selection, this motivates modelling and methods that coordinate rule quality, dynamic decisions and energy-saving constraints.

3 Problem description

3.1 Assumptions and notation

In a dynamic flexible job shop, a set of jobs $\mathcal{J} = \{J_1, J_2, \dots, J_n\}$ is processed on a set of machines $\mathcal{M} = \{M_1, M_2, \dots, M_m\}$. Each job J_i comprises q_i operations with technological precedence relations. Operation O_{ij} can be processed on any machine in its eligible set $M_{ij} \subseteq M$. Since the processing time and energy required for an operation may differ across machines, a schedule must determine both machine assignments and the processing sequence on each machine. Unlike static flexible job shops, this study considers random job arrivals and machine breakdowns, so machine availability and outstanding processing tasks change over time. Total tardiness and total energy consumption are used as objectives, with routing and sequencing coordinated to reduce production energy use while meeting delivery requirements.

Total tardiness measures the accumulated extent to which job completion times exceed due dates and evaluates due-date performance under dynamic arrivals. Total energy consumption includes processing, idling, repair, startup and shutdown, and depends on machine choices, processing sequences and speeds. Online scheduling uses the baseline processing speed; speeds are adjusted using operation time margins after the schedule has been generated. Table 1 defines the main notation and variables.

Table 1 Notation and variables

Symbol	Definition
J, M	Sets of jobs and machines
n, m	Numbers of jobs and machines
i, i'	Job indices
j, j'	Operation indices

k	Machine index
q_i	Number of operations in job J_i
O_{ij}	Operation j of job J_i
O	Set of all operations
M_{ij}	Eligible machine set for operation O_{ij}
r_i, d_i	Arrival time and due date of job J_i
p_{ijk}	Net processing time of operation O_{ij} on machine M_k at the baseline speed
S_{ij}, C_{ij}	Start and completion times of operation O_{ij}
C_i, C_{max}	Completion time of job J_i and schedule makespan
y_{ijk}	Machine-assignment variable: 1 if operation O_{ij} is assigned to machine M_k; 0 otherwise
x_{abk}	1 if distinct operations a, b are both assigned to machine M_k and a precedes b; 0 otherwise
L	A sufficiently large constant exceeding the scheduling horizon considered
B_{ij}	Cumulative breakdown-repair duration between the start and completion of operation O_{ij}
v_{ij}	Processing speed ratio of operation O_{ij}
$P_k, P_k(v)$	Baseline processing power of machine M_k and its processing power at speed ratio v
P_k^I, P_k^R, P_k^S	Idle, repair, and startup/shutdown power of machine M_k
τ_k^I, τ_k^R	Cumulative idle and repair durations of machine M_k within the relevant measurement period
N_k^S, t_k^S	Number of startup/shutdown events of machine M_k within the measurement period and duration per event
T, E	Total tardiness and total energy consumption
E_P, E_I, E_R, E_S	Processing, idle, repair, and startup/shutdown energy consumption
T^{ref}, E^{ref}	Reference scales for tardiness and energy consumption
λ_G, f	Relative energy weight and normalised weighted objective value

The following processing and equipment assumptions are adopted.

(1) Each operation is processed on one eligible machine, without machine changes during processing. A machine processes at most one operation at a time, and at most one operation of a job can be in process at any time.

(2) Operations of the same job follow a prescribed technological sequence. An operation can start only after its predecessor has finished. No additional precedence constraints exist between different jobs.

(3) All machines are initially available, and intermediate buffer capacities are unlimited. Job transportation times and sequence-dependent setup times are not considered.

(4) Jobs arrive randomly according to a Poisson process. Machine failures during processing are modelled using an exponential failure model, and repair durations follow a uniform distribution. Distribution parameters are specified in the experimental settings.

(5) A failed machine stops processing and cannot start another operation until repair is complete. The interrupted operation then resumes its remaining processing, without repeating completed work. No other operation can occupy its interruption interval.

(6) Processing cannot be deliberately pre-empted except by breakdowns. At most one breakdown occurs during the processing of a given operation, while different machines may fail simultaneously.

(7) Repair power equals idle power, both being a fixed proportion of the machine's baseline processing power. Startup and shutdown use the same power coefficient and duration, with energy consumption calculated from the actual number of events.

(8) Online scheduling uses the baseline speed; speed is not an action in online rule selection. Subsequent speed adjustment retains machine assignments and processing order, extending only the net processing times of non-final operations within available time margins. Repair durations do not change with speed.

3.2 Mathematical model

Under these assumptions, DFJSS with random job arrivals and machine breakdowns is modelled as follows:

$$\min\ (T, E) \tag{1}$$

Total tardiness is defined as

$$T = \sum_{i \in J} max\{0, C_i - d_i\} \tag{2}$$

A job's completion time is determined by its final operation. Makespan denotes the completion time of the entire schedule and is defined as

$$C_{max} = \max_{i=1,\ldots,n} C_i \tag{3}$$

Total energy consumption consists of four components:

$$E = E_P + E_I + E_R + E_S \tag{4}$$

(1) Processing energy. The relationship between machine processing power and speed is represented by $P_k(v) = P_k v^3$ [38]. The net processing time of an operation is p_{ijk}/v_{ij}, giving processing energy consumption E_P as

$$E_P = \sum_{O_{ij} \in O} \sum_{k \in M_{ij}} y_{ijk} P_k(v_{ij}) \frac{p_{ijk}}{v_{ij}} = \sum_{O_{ij} \in O} \sum_{k \in M_{ij}} y_{ijk} P_k p_{ijk} v_{ij}^2 \tag{5}$$

(2) Non-processing energy. Idle, repair, and startup/shutdown energy consumption are respectively calculated as

$$\begin{aligned} E_I &= \sum_{k=1}^{m} P_k^I \tau_k^I, \\ E_R &= \sum_{k=1}^{m} P_k^R \tau_k^R, \qquad P_k^I = P_k^R = 0.2P_k. \\ E_S &= \sum_{k=1}^{m} N_k^S P_k^S t_k^S, \end{aligned} \tag{6}$$

Processing energy covers only actual processing intervals; breakdown-repair intervals are included in E_R to avoid double counting. Startup/shutdown energy refers to equipment switching and does not represent operation setup time. The statistical scopes of $\mathcal{J}_c$ and $\mathcal{O}_c$ and the measurement period for equipment-state energy follow the evaluation protocol in Section 5.1. Where whole-run statistics are used, all jobs and the complete simulation period are included.

To account for both tardiness and energy consumption in rule evaluation, the following normalised weighted objective is used:

$$min\ f = \frac{T}{T^{ref}} + \lambda_G \frac{E}{E^{ref}} \tag{7}$$

The reference scales remove differences in units and orders of magnitude between the two measures, while λ_G controls the weight of energy consumption relative to tardiness.

Schedules must satisfy the following basic constraints. Let $a = (i, j)$ and $b = (i', j')$ denote two distinct operations, and let S_a, C_a, y_{ak} represent their start times, completion times and machine-assignment variables, respectively. Their eligible machine sets are M_a and M_b.

$$S_b \geq C_a - L(1 - x_{abk}),\ \forall a \neq b,\ k \in M_a \cap M_b. \tag{8}$$

$$S_{i1} \geq r_i, \qquad \forall i = 1, \dots, n. \tag{9}$$

$$C_{ij} = S_{ij} + \sum_{k \in M_{ij}} y_{ijk} \frac{p_{ijk}}{v_{ij}} + B_{ij},\ \forall i = 1, \dots, n,\ j = 1, \dots, q_i. \tag{10}$$

$$S_{i,j+1} \geq C_{ij}, \quad \forall i = 1, \dots, n,\ j = 1, \dots, q_i - 1. \tag{11}$$

$$\sum_{k \in M_{ij}} y_{ijk} = 1, \quad \forall i = 1, \dots, n,\ j = 1, \dots, q_i. \tag{12}$$

Constraint (8) prevents operations on the same machine from overlapping. When both operations are assigned to machine M_k, exactly one of x_{abk} and x_{bak} equals 1, specifying their order. Constraint (9) permits a job's first operation to start only after arrival. Equation (10) relates operation start and completion times, with B_{ij} denoting the cumulative repair duration actually incurred during processing, or zero without failures. Constraint (11) enforces operation precedence, and constraint (12) assigns each operation to exactly one eligible machine.

4 Methodology

4.1 Overall framework

To address the DFJSS problem described above, we propose INichGPDRL, which comprises rule-library construction, online rule selection and speed adjustment. Figure 1 presents the overall framework. The three stages generate candidate rules, select a rule for the current state, and then exploit time margins in the resulting schedule to reduce energy consumption.

Specifically, Stage 1 uses INichGP to evolve individuals containing sequencing and routing rules. Tardiness–energy evaluation, behavioural clearing and immigrant screening are combined to construct the rule library. Stage 2 treats complete library individuals as actions and invokes the routing or sequencing rule of the selected individual at the corresponding decision point, translating rule selection into machine assignments and operation sequences. Once these are fixed, Stage 3 uses free float to

identify non-final operations that can run at the lower speed without delaying successor operations or job completion. The stages are detailed below.

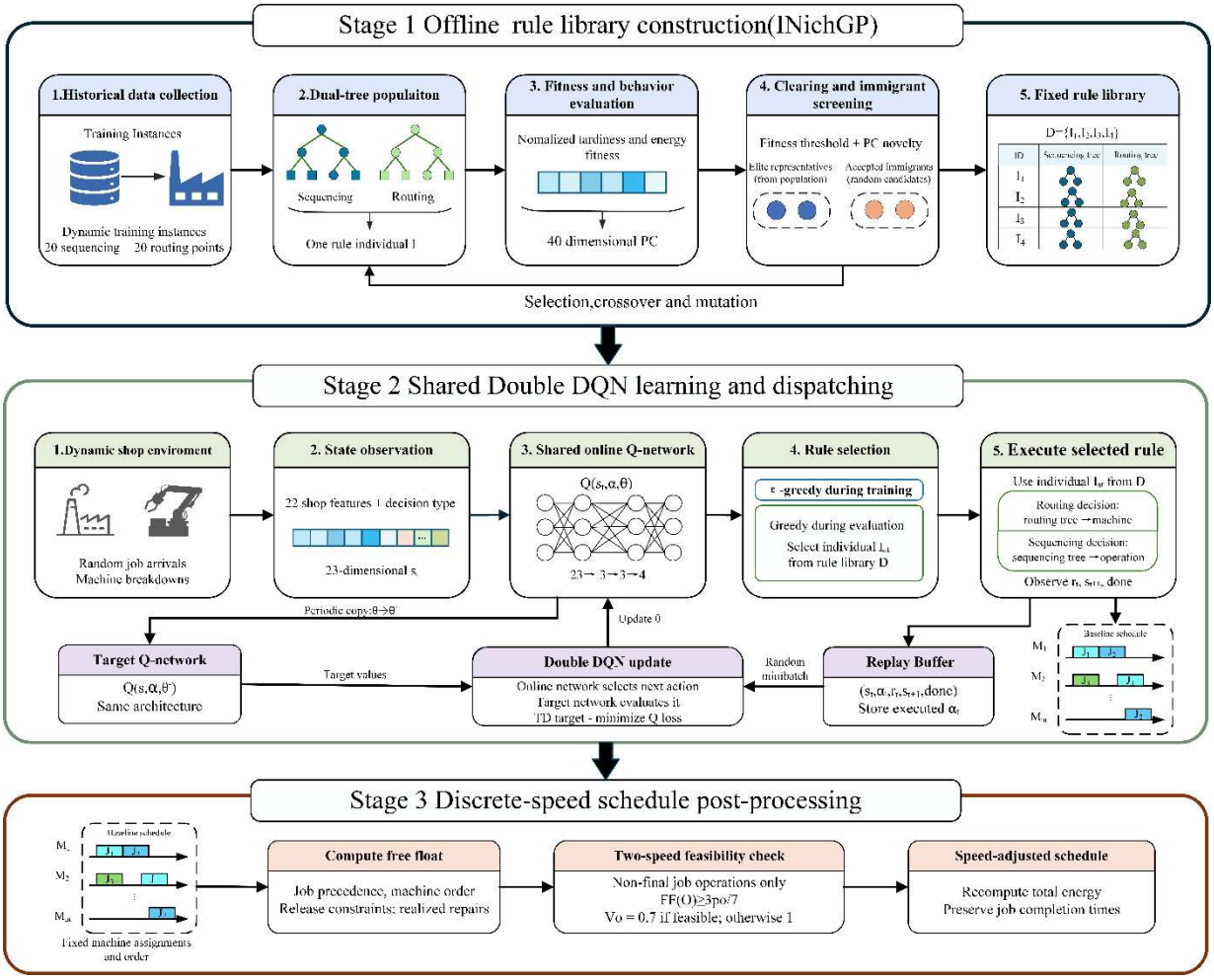

Fig. 1 Overall framework of INichGPDRL

4.2 Rule-library construction using improved niching genetic programming

Traditional GP primarily searches for the best individual, whereas NichGP retains individuals with different scheduling behaviours through a niching mechanism, producing a selectable rule set rather than a single rule [9]. In dynamic flexible job shops, job arrivals and machine breakdowns change candidate operations, machine loads and due-date urgency. A rule suited to one state may therefore lose its advantage in another. Library construction must assess scheduling performance as well as behavioural differences. Building on NichGP's phenotypic characterisation and niche clearing, the proposed INichGP introduces joint tardiness–energy fitness and screens random immigrants for fitness and behavioural novelty, providing behaviourally different dispatching rules for subsequent DRL.

A niche is a group of individuals with identical or similar scheduling behaviours. Unlike traditional GP, individuals first compete within niches, and retained representatives participate in subsequent evolution, reducing repeated occupation of the population by similar rules. Different expressions do not necessarily produce different decisions: two structurally different trees may select the same operation from the same candidate set. Individuals are therefore distinguished using phenotypic characterisation (PC), rather than tree structure.

(1) Phenotypic characterisation and fitness evaluation

PC describes an individual's choices at common decision points. Let A_l be the candidate set at decision point l, R_I the rule associated with individual I, and $a_l^*(I)$ its selected candidate. Candidates are ranked using a common reference rule, with rank 1 assigned to the highest-scoring candidate. Thus,

$$a_l^*(I) = \underset{a \in A_l}{\operatorname{argmax}} R_I(a) \tag{13}$$

$$PC_l(I) = rank_{R^{ref}}(a_l^*(I)) \tag{14}$$

$$PC(I) = (PC_1(I), \ldots, PC_{40}(I)) \tag{15}$$

Sequencing and routing rules are used at their respective decision points. The common reference rule is taken from the fittest individual in the current generation and updated after each generation is evaluated. All individuals within a generation use the same decision points and reference rule; ties are resolved by a fixed candidate index. PC therefore records the position of a selected candidate in the reference ranking, avoiding direct comparison of raw outputs from different rules.

Unlike approaches with a fixed routing rule, this study evolves sequencing and routing rules together. Routing determines which machine queue receives an operation, while sequencing determines its position within that queue; both affect subsequent loads and job completion. With fixed manual routing, even an improved sequencing rule may remain limited by machine assignments. Each individual $I = [\mathcal{T}^{seq}, \mathcal{T}^{rou}]$ therefore contains cooperating sequencing and routing rules, both of which are used to generate schedules. The dual-tree representation separates the two decisions, while fitness evaluation and niche selection jointly determine which individuals enter the library.

Individual fitness uses the normalised weighted tardiness–energy objective defined in Section 3.2, evaluating generated schedules on common training instances. PC characterises differences in selection behaviour. Both fitness and PC inform subsequent niche clearing and immigrant screening.

(2) Decision-point sampling and an example of PC calculation

To cover both types of scheduling decisions, 20 sequencing and 20 routing decision points are randomly sampled from a trajectory generated on an independent reference instance. The trajectory uses earliest-available (EA) routing and first-in, first-out (FIFO) sequencing, independently of the individuals being evaluated. Sampled points remain fixed within an evolutionary run. Sequencing candidates are waiting operations, while routing candidates are machines eligible for the current operation. Each point contributes one PC value, yielding a 40-dimensional vector with 20 sequencing and 20 routing components per individual. This captures both rules and accounts for behavioural differences in operation sequencing and machine assignment. Table 2 illustrates a sequencing point with seven candidate operations, showing scores from the reference rule and two evaluated rules. This constructed example explains PC calculation; seven is not a fixed candidate count in the experiments.

Table 2 Example of PC calculation at a sequencing decision point

Rule or ranking	O_1	O_2	O_3	O_4	O_5	O_6	O_7
Reference scores	70	50	60	30	40	10	20
Reference ranks	1	3	2	5	4	7	6
Rule A scores	50	70	60	10	40	20	30
Rule B scores	60	50	70	40	30	20	10

Rule A selects the highest-scoring operation O_2. Since O_2 has reference rank 3, its PC value at this point is 3. Rule B selects O_3, whose reference rank is 2, giving a PC value of 2. Repeating the calculation at the other 39 points produces the complete PC vector. The seven operations illustrate sequencing only; the number of routing candidates depends on the operation's eligible machine set and is not fixed to the same value.

(3) Niche clearing and immigrant screening

After PC calculation, the behavioural difference between two individuals is measured by Euclidean distance:

$$d(I_u, I_v) = \sqrt{\sum_{l=1}^{40} [PC_l(I_u) - PC_l(I_v)]^2} \tag{16}$$

For example, if rules A and B differ only at the decision point in Table 2 and match at the other 39 points, their distance is $|3 - 2| = 1$. Clearing traverses individuals in ascending fitness order and treats retained representatives as niche centres. An individual joins the nearest niche if its distance to the centre does not exceed radius δ; otherwise, it establishes a new niche. At most κ fitter individuals are retained per niche, and excess individuals do not reproduce. Unlike traditional GP, this procedure follows fitness evaluation in every generation, incorporating behavioural diversity throughout evolution rather than selecting diverse rules only at the end.

This study sets $\delta = 0$ and $\kappa = 1$, retaining only the fittest individual among those with identical PC vectors. This reduces repeated behaviours without prematurely merging rules that make different choices. Since PC uses candidate ranks, non-zero distances still depend on candidate-set sizes. Zero-radius clearing instead directly tests whether behaviours match at the common reference points.

Clearing duplicates alone may still concentrate the search around a few existing rules. To introduce new search directions, INichGP adds random immigrants subject to fitness and behavioural-novelty thresholds. Rather than directly admitting random individuals, it first evaluates candidates on the same c training instances used for the current generation. Only candidates whose fitness is no worse than the population's 95th percentile proceed to behavioural screening. Their PC vectors must then differ in at least one coordinate from each retained elite and accepted immigrant. Eligible candidates are admitted in ascending fitness order until the immigrant limit is reached or candidates are exhausted. The fitness threshold excludes a poorly performing portion of candidates, while the behavioural threshold reduces duplication of existing choices, ensuring that new behaviours are introduced with regard to scheduling quality.

Algorithm 1 summarises rule-library construction with INichGP.

Algorithm 1 Rule-library construction with INichGP

Input: Training and reference instances; population size N; generations G; niche parameters (δ, κ); library size K
Output: Rule library D

1: Randomly initialise population P
2: Sample and fix S: 20 sequencing and 20 routing reference points
3: for g = 1, …, G do
4: Evaluate joint tardiness–energy fitness on common training instances
5: Compute PC on S using the current best individual as reference
6: Apply niche clearing to obtain retained set H
7: if g < G then
8: Retain the fitter elite individuals from H
9: Generate immigrant candidates; evaluate on common instances and S
10: Admit immigrants meeting fitness and behavioural thresholds
11: Select elite parents; generate offspring by crossover and mutation
12: Combine elites, immigrants and offspring into next population P of size N
13: end if
14: end for
15: D ← the K individuals in H with the lowest fitness values
16: return D

4.3 Rule-library-based deep reinforcement learning for scheduling

The library constructed in Section 4.2 contains rules with different selection behaviours. However, offline fitness measures average performance on training instances and does not directly establish suitability for the current shop state. Stage 2 therefore uses DRL to learn the relationship between states and rule selection. We adopt Double DQN for value learning and follow Xu et al.'s idea of using GP-generated scheduling actions [9], with INichGP individuals forming the action set. Unlike DRL approaches based on predefined manual rules, the candidate actions are the rules evolved in Stage 1.

Successive routing and sequencing decisions are represented as a sequential decision process using the following MDP formulation [39]:

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, \gamma, R \rangle. \tag{17}$$

Here, $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, P the state transition probability after an action, γ the discount factor, and R the reward function. Decision step t corresponds to a routing or sequencing choice rather than a fixed physical-time interval. After executing an action, the environment advances processing, job arrivals, breakdowns and repairs until the next decision point. These events jointly determine transitions, and the policy is learned through simulation interaction without explicitly estimating P. The objective is to maximise expected discounted return so that rule selection considers both immediate outcomes and subsequent scheduling effects.

State s_t comprises 22 shop features and one decision-type indicator. The features describe production progress, machine loads, queues, candidate processing times and due-date urgency; the indicator distinguishes routing from sequencing. Definitions are given in Section 5.3 and Table 7. This vector represents the shop state and serves as the policy input in the MDP; its ability to capture the full production history depends on the selected features.

The action space is determined by the Stage 1 library. Let $D = \{I_1, \dots, I_K\}$, giving $\mathcal{A} = \{1, \dots, K\}$. When action k is selected, the routing rule of I_k scores eligible machines at a routing point, or its sequencing rule scores candidate operations at a sequencing point. The highest-scoring candidate is executed. Each action index therefore always denotes a rule individual, although the selected machine or operation varies with the candidate set. Unlike assigning an action to each waiting operation, this representation fixes the network output dimension through library size, independently of dynamic queue length. Figure 2 shows the policy network and its interaction with the shop environment. The online

network receives the 23-dimensional state and passes it through two fully connected hidden layers, each with three nodes and ReLU activation, to output Q-values for $K = 4$ rule actions. The target network has the same architecture and computes learning targets rather than selecting the current action.

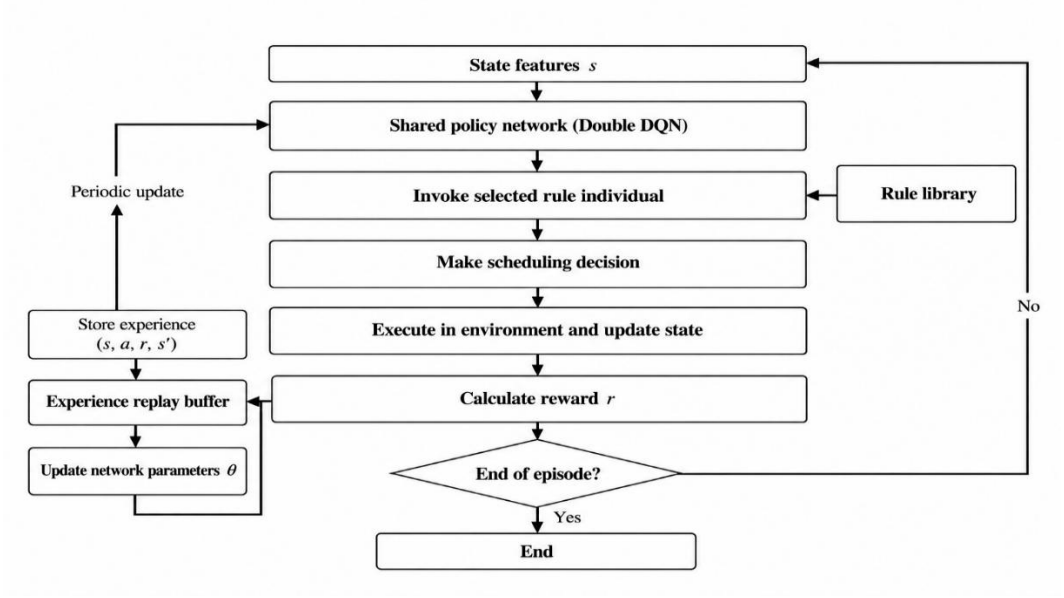

Fig. 2 Shared DRL policy-learning framework based on the rule library

The reference approach trains routing and sequencing agents separately, fixing the other rule during each agent's training. In contrast, this study uses a shared policy network for both rule-selection tasks. Routing and sequencing share individual indices, with the decision type determining which tree is invoked. The network selects one individual at a time, without enumerating machine-assignment–operation-sequencing combinations at a single point; the type indicator distinguishes the two choices. The INichGP library remains fixed during DRL training, and routing and sequencing transitions jointly update the network to learn state-dependent rule use.

At each decision point, the agent observes s_t and selects an action using an ε-greedy policy:

$$a_t = \begin{cases} \text{uniformly sampled from } \mathcal{A}, & \text{with probability } \varepsilon, \\ \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q(s_t, a; \theta), & \text{with probability } 1-\varepsilon. \end{cases} \tag{18}$$

Here, θ denotes the online-network parameters. Exploration probability decreases during training, shifting from trying different rules towards exploiting learned action values; exploration is disabled during evaluation. The environment executes the machine or operation choice made by the selected rule and returns reward r_t and next state s_{t+1}. The agent uses these transitions to update Q-value estimates and thereby subsequent action choices.

Job tardiness is known only upon completion, so using it directly as a reward creates delayed and sparse feedback. To reflect tardiness risk promptly, this study uses slack to represent due-date urgency and includes energy consumption in the reward. Slack is the time remaining until a job's due date minus its remaining processing time; it signals potential tardiness before completion. Liu et al. used slack-based rewards to guide tardiness optimisation [40]. Building on this idea, we add processing energy so that the sequencing reward jointly evaluates candidate urgency, processing time and processing energy. Let the selected operation be s and its slack be S_s. Then,

$$F_s = 1 - \frac{S_s}{|S_s| + \beta} \tag{19}$$

The sequencing reward is defined as

$$r_{seq} = F_s \bar{p}_{-s} - \bar{F}_{-s} p_s + \lambda_S (F_s \bar{e}_{-s} - \bar{F}_{-s} e_s) \tag{20}$$

Here, β is the slack-transformation parameter, p_s the baseline processing time, and e_s the processing energy at the baseline speed. A bar denotes the mean over the other candidate operations; subscript $-s$ excludes the selected operation s. This relative comparison provides immediate feedback on operation choices, with λ_S weighting the energy term. If only one candidate exists, it is executed directly without evaluating a comparison reward over an empty set.

The routing reward combines candidate-machine availability and processing energy according to the current job's due-date urgency. Let the selected machine be k. Then,

$$r_t = F_i(\bar{A} - A_k) + \lambda_E \frac{\bar{e} - e_k}{\bar{e}}. \tag{21}$$

Here, F_i is the urgency factor derived from the current job's slack. Variables A_k and e_k denote the selected machine's estimated availability time and energy required for the current operation; $\bar{A}$ and $\bar{e}$ are the corresponding candidate-machine means, and λ_E weights the energy term. The first term favours assigning urgent jobs to machines available earlier, while the second favours processing energy below the candidate mean. The reward is zero when only one machine is eligible, and the energy term is zero when $\bar{e} = 0$. Both rewards are surrogate evaluations of the scheduling objectives. DRL learns subsequent effects through discounted returns, rather than assuming that maximising immediate reward at every point minimises total tardiness and energy consumption.

With the state, action and reward defined, Double DQN is used to learn action values [41]. Conventional DQN selects and evaluates the maximising action in the next state using the same value estimates, which can introduce overestimation bias. Double DQN separates these steps: the online network selects the next-state action, and the target network evaluates it. The learning target is

$$y_t = r_t + \gamma(1 - d_t) Q\left(s_{t+1}, \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q(s_{t+1}, a; \theta); \theta^-\right) \tag{22}$$

Here, θ^- denotes target-network parameters and d_t the terminal indicator. Future action values are excluded for terminal states. Separating action selection and evaluation reduces overestimation caused by maximisation. The online network is updated by minimising the mean squared error over a minibatch:

$$\mathcal{L}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{t \in \mathcal{B}} [y_t - Q(s_t, a_t; \theta)]^2 \tag{23}$$

Here, $\mathcal{B}$ is a randomly sampled minibatch of transitions. Each interaction stores $(s_t, a_t, r_t, s_{t+1}, d_t)$ in the replay buffer. Once the buffer contains enough samples for a minibatch, random samples are drawn to update the network. Experience replay reuses scheduling transitions and reduces temporal correlation between consecutive decisions [42]. Target-network parameters are synchronised with the online network at a predefined interval; training settings are given in Section 5. Algorithm 2 summarises interaction learning and construction of the execution policy.

Algorithm 2 Rule-library-based Double DQN learning

Input: Fixed rule library D; training environment Env; number of episodes; learning parameters and execution settings
Output: Rule-selection policy π

1: Initialise online and target networks identically; initialise replay buffer B
2: for each training episode do
3: Reset Env and observe initial state s
4: while the episode has not terminated do
5: Select rule individual a using an ε-greedy policy
6: Invoke its routing or sequencing rule according to the decision type
7: Execute the choice; observe reward r, next state s′ and terminal flag d
8: Store (s, a, r, s′, d) in B
9: if B contains at least one minibatch then
10: Sample a random minibatch of transitions
11: Form TD targets using online action selection and target-network evaluation
12: Update the online network by minimising prediction error relative to TD targets
13: Copy online parameters to the target network at the update interval
14: end if
15: s ← s′
16: end while
17: end for
18: Disable exploration to obtain π under the execution settings
19: return π

4.4 Free-float-based two-speed schedule post-processing

The first two stages determine machine assignments and processing sequences through rule generation and online selection. Stage 3 adjusts processing speeds within the resulting schedule to reduce energy consumption further. Under the power model in Section 3, lower speed reduces an operation's processing energy but increases its duration. The key is therefore to identify time margins that can absorb this increase without affecting successor operations or job delivery. To represent equipment with a finite set of operating speeds, this study uses two levels: the baseline speed and 0.7 times that speed. Free float determines whether an operation can switch to the lower level while machine assignments and processing order remain fixed.

The Stage 2 schedule is first represented as a directed acyclic graph. Nodes denote operations, and directed edges encode technological precedence within jobs and processing order on machines. Job arrival times impose release constraints on first operations. Node durations include net processing and realised repair occupancy; only processing duration changes with speed, while repair duration remains fixed. Earliest start times $ES(O)$ and earliest finish times $EF(O)$ are calculated in topological order as the baseline before speed reduction.

Free float is the duration by which an operation can be extended without delaying the earliest start of any immediate successor. Let $succ(O)$ denote the immediate successors of operation O along the job and machine chains. For an operation with successors, its free float is

$$FF(O) = \min_{O' \in succ(O)} ES(O') - EF(O). \tag{24}$$

Rather than merely identifying whether an operation lies on the critical path, free float quantifies the time margin independently available for speed reduction. Critical operations have no available float, but non-critical operations need not have positive free float either. An operation is slowed down only when its free float accommodates the additional processing time at 0.7 speed. Final operations of jobs are excluded even if a gap exists before a machine successor, because their completion times directly determine job completion and hence tardiness.

Let p_O be the net processing time of operation O at the baseline speed. At 0.7 speed, the duration becomes $p_O/0.7$, an increase of $(1/0.7-1)p_O = 3p_O/7$, or approximately 42.9% of the baseline duration. The timing condition for speed reduction is therefore

$$FF(O) \geq \left(\frac{1}{0.7}-1\right)p_O = \frac{3}{7}p_O. \tag{25}$$

Positive free float alone does not guarantee eligibility: it must cover the additional duration required by the predefined lower speed. Let $\mathcal{O}^{last}$ be the set of final operations of all jobs. Speeds are assigned as

$$v_O = \begin{cases} 0.7, & O \notin \mathcal{O}^{last} \text{ and } FF(O) \geq \frac{3}{7}p_O, \\ 1, & \text{otherwise.} \end{cases} \tag{26}$$

Free float therefore determines whether to switch speed levels, rather than continuously determining a speed ratio. According to Eq. (5), processing energy at the lower speed is $0.7^2 = 0.49$ of its baseline value under the adopted power model. This relation applies only to an individual operation's processing energy; total schedule energy is recalculated using the complete model. For illustration, consider a non-final operation with baseline processing time 10 and free float 5. At 0.7 speed, processing takes approximately 14.29, an increase of 4.29 that fits within the available float. Its immediate successors can therefore retain their original earliest starts. If free float were only 2, the operation would fail the condition in Eq. (26) and retain its baseline speed. Eligibility thus depends on sufficient time margin, not merely on being outside the critical path.

Under these constraints, a slowed operation may finish later, but no later than the original earliest start of any immediate successor. Successor processing is therefore not delayed. Since final operations retain their original speeds, propagation along job precedence and machine order preserves job completion times, giving

$$C_i^{'} = C_i, \qquad T^{'} = T, \qquad C_{max}^{'} = C_{max}. \tag{27}$$

Primes denote post-processing values. Equation (27) preserves job completion times, total tardiness and makespan, although slowed operations may finish later themselves. After speed adjustment, operation start and completion times are recalculated and these invariants are checked. Energy changes are then evaluated using the model in Section 3 and the same statistical scope. Algorithm 3 gives the complete procedure.

Algorithm 3 Free-float-based two-speed schedule post-processing

---

Input: Baseline schedule Π including repair occupancy; job arrivals and due dates; processing times and machine powers; speed levels {1, 0.7}
Output: Speed vector v; adjusted schedule Π′; measures (T′, E′, C_max′)

1: Construct the job-precedence and machine-order graph with release constraints
2: Compute baseline operation times and free float FF(O) for non-final operations
3: v_O ← 1 for every operation O
4: for each operation O do
5: if O is not final in its job and FF(O) ≥ (3/7)p_O then
6: v_O ← 0.7
7: end if
8: end for
9: Update durations and rebuild Π′ with assignments, order and repair occupancy fixed
10: Check that all job completion times are unchanged
11: Calculate T′, E′ and C_max′ using the same measurement scope
12: return v, Π′, (T′, E′, C_max′)

This stage follows schedule generation in Stage 2 and uses the job arrivals, realised breakdowns and repair durations recorded in that schedule, without generating new dynamic events. The

completion-time invariance therefore applies to post-processing a fixed schedule. Stage 3 adjusts speeds, while the first two stages make scheduling decisions; together they form the INichGPDRL energy-efficient scheduling procedure.

5 Experimental settings

5.1 Scenarios and data partitioning

The experiments use six machines and 120 jobs. Each job contains two to four operations, each eligible for two to four machines. Processing times on eligible machines are generated independently, and baseline machine powers are specified within each instance. The job arrival rate is calibrated to nominal utilisation $u = 0.90$, an instance-generation parameter rather than the actual mean utilisation including breakdowns and the final completion phase. The failure rate is 0.02, corresponding to a mean time between failures of 50 time units, and repair durations follow $U[3,29]$. Four scenarios combine processing-time variability and due-date tightness, as shown in Table 3.

Table 3 Experimental scenarios

Scenario	Processing time	Due-date factor α_i	Description
HH	$U[5,25]$	$U[1,2]$	High variability; tight due dates
HL	$U[5,25]$	$U[1,3]$	High variability; relatively loose due dates
LH	$U[10,20]$	$U[1,2]$	Low variability; tight due dates
LL	$U[10,20]$	$U[1,3]$	Low variability; relatively loose due dates

Due dates are generated as $d_i = r_i + \alpha_i \sum_j \bar{p}_{ij}$, where $\bar{p}_{ij}$ is the mean processing time over eligible machines. Training seeds start at 2000, validation seeds range from 100 to 199, and evaluation seeds from 500 to 599. Independent calibration seeds starting at 90000 determine energy weights; PC reference instances are independent of all these uses. Each scenario includes ten training runs, each evaluated by mean performance over 100 evaluation instances. The statistical unit is an independent training run: the 100 test instances evaluated by one model are not 100 independent training runs. The first 12 jobs are used for warm-up, and w_{end} denotes the time when all have finished. The main evaluation set is $J = J_i : r_i \geq w_{end}$. Consequently, exclusions may exceed 12 jobs, since subsequent jobs arriving before warm-up completion are also excluded.

5.2 Experimental parameters

INichGPDRL is implemented in Python. NichGP and INichGP use DEAP [43], and the Double DQN networks use PyTorch [44]. Experiments run on a computer with an Intel Core i5-12500H processor and 16 GB of RAM. Tables 4–6 list the GP, niching and DQN parameters used for the algorithm comparisons and improvement assessments in Section 6.

GP constructs candidate individuals by evolving routing and sequencing rules. Table 4 specifies population size, genetic-operation probabilities and tree constraints. Two training instances evaluate each individual per generation, and four elite individuals are retained.

Table 4 GP parameters

Parameter	Value
Population size (N)	200
Number of generations (G)	50
Training instances per generation (c)	2
Number of elite individuals	4
Crossover probability	0.80
Mutation probability	0.15
Maximum tree depth	8
Tournament size	4
Energy fitness weight for HH, HL, LH and LL (λ_G)	0.100, 0.0634, 0.2629, 0.2188

The niching parameters in Table 5 control PC, clearing and immigrant admission. PC contains choices at 20 sequencing and 20 routing decision points, giving 40 dimensions. Four individuals are retained after evolution to form the library used in Stage 2.

Table 5 Niching and rule-library parameters

Parameter	Value
Sequencing reference points	20
Routing reference points	20
PC dimension	40
Clearing radius (δ)	0
Niche capacity (κ)	1
Base number of immigrants	20
Number of immigrant candidates	60
Immigrant admission quantile	0.95
Rule-library size (K)	4

After the library is fixed, Double DQN learns the relationship between shop states and rule selection. Table 6 gives the network architecture and training settings. The target-network update interval is measured in network parameter updates.

Table 6 DQN and action-selection parameters

Parameter	Value
Input dimension	23
Number of hidden layers	2
Nodes per hidden layer	3
Output dimension	4 (corresponding to the 4 library individuals)
Learning rate	10^{-4}
Discount factor (γ)	0.95
Replay buffer capacity	5000
Minibatch size	64
Target-network update interval	150 network updates
Training episodes	100
Initial and minimum exploration probabilities (ε)	0.3, 0.1
Energy weight in sequencing reward (λ_S)	0.041990
Energy weight in routing reward (λ_E)	7.565787

The GP energy weight expresses the objective trade-off between tardiness and energy consumption. Reward energy weights are determined from the variability of the additional and base terms in independent calibration data and remain fixed thereafter. These weights act on rule evolution and local decision evaluation, respectively, and use different calibration criteria.

5.3 State features

The agent receives a state vector comprising 22 shop features and one decision-type indicator. The features describe the global production state, the current machine and candidate operations, enabling rule selection to account for loads, processing requirements and due-date urgency. Table 7 gives their definitions and formulae. Minimum, maximum and mean candidate-operation statistics are computed over the current candidate set.

Let t be the current decision time and m the number of machines. Variable $b_k(t)$ is the cumulative processing time of machine M_k up to t, $u_k(t) = b_k(t)/t$ its cumulative utilisation, and $z_k(t)$ its failure-state indicator. Set $J_u(t)$ contains jobs in the instance that have not finished, while $w_i(t)$ sums the remaining processing times of unfinished operations of job i. For operations without a machine assignment, the mean over eligible machines is used. Index k^* denotes the current decision machine, $Q_{k^*}(t)$ its waiting queue, and $O(t)$ the current candidate-operation set.

For candidate operation O_{ij}, let $\widehat{p}_{ij}(t)$ denote its remaining processing time, ℓ_{ij} its release time, $\sigma_{ij}(t) = d_i - t - w_i(t)$ its slack, $\Delta_i(t) = d_i - t$ the time until its job's due date, and $\omega_{ij}(t) = t - \ell_{ij}$

its waiting time. In Table 7, $\langle x \rangle_O = \frac{1}{|O(t)|}\sum_{O_{ij}\in O(t)} x_{ij}$ denotes the mean over candidates, $\min_{\mathcal{C}}$ and $\max_{\mathcal{C}}$ the corresponding minimum and maximum, and $1\{\cdot\}$ the indicator function. Variable $h_k(t)$ is the remaining processing time of the operation currently running on a machine, or zero when idle. Features are set to zero when there is no current decision machine or the candidate set is empty. Both utilisation features are zero at $t = 0$, and production progress is zero if its denominator is zero.

Table 7 State features and definitions

Category	Feature	Definition	Formula
Global shop	Overall utilisation	Total cumulative machine processing time divided by machine count and current time	$\frac{1}{mt}\sum_{k=1}^{m} b_k(t)$
Global shop	Utilisation range	Difference between maximum and minimum cumulative machine utilisation	$\max_k u_k(t) - \min_k u_k(t)$
Global shop	Failed-machine proportion	Number of failed machines divided by the total machine count	$\frac{1}{m}\sum_{k=1}^{m} z_k(t)$
Global shop	Jobs in the system per machine	Number of arrived but unfinished jobs divided by machine count	$\frac{1}{m}\sum_{i\in J_u(t)} 1\{r_i \le t\}$
Global shop	Normalised production progress	Current time divided by current time plus remaining instance workload per machine	$\frac{t}{t + \frac{1}{m}\sum_{i\in J_u(t)} w_i(t)}$
Current machine	Queue length	Number of operations waiting in the current machine's queue	$\lvert Q_{k^*}(t)\rvert$
Current machine	Queue workload	Sum of remaining processing times of operations in the current machine's queue	$\sum_{O_{ij}\in Q_{k^*}(t)} \hat{p}_{ij}(t)$
Current machine	Remaining time of active operation	Remaining processing time of the active operation; zero if the machine is idle	$h_{k^*}(t)$
Current machine	Cumulative processing time	Cumulative processing time of the current machine up to the decision time	$b_{k^*}(t)$
Candidate operations	Processing-time statistics	Minimum, maximum and mean processing times on the current machine	$(\min_{\mathcal{C}} p_{ij,k^*}, \max_{\mathcal{C}} p_{ij,k^*}, \langle p \rangle_{\mathcal{C}})$
Candidate operations	Remaining-workload statistics	Minimum and maximum remaining workloads of the corresponding jobs	$(\min_{\mathcal{C}} w_i, \max_{\mathcal{C}} w_i)$
Candidate operations	Slack statistics	Minimum, maximum and mean job slack: due date minus current time	$(\min_{\mathcal{C}} \sigma_{ij}, \max_{\mathcal{C}} \sigma_{ij}, \langle \sigma \rangle_{\mathcal{C}})$

		and remaining workload	
Candidate operations	Number of urgent operations	Number of candidate operations with negative slack	$\sum_{O_{ij}\in\mathcal{C}(t)} 1\{\sigma_{ij}(t) < 0\}$
Candidate operations	Time-until-due-date statistics	Minimum and mean differences between job due dates and current time	$(\min_{\mathcal{C}}\Delta_i, \langle\Delta\rangle_{\mathcal{C}})$
Candidate operations	Waiting-time statistics	Minimum and mean waiting times of candidate operations	$(\min_{\mathcal{C}}\omega_{ij}, \langle\omega\rangle_{\mathcal{C}})$
Decision type	Decision-type indicator	1 for routing; 0 for sequencing	$1\{\text{routing decision}\}$

5.4 Comparison algorithms

The comparison includes fixed manual rules, GP, DRL, and combined GP–RL approaches. Each algorithm is assessed using mean measures from repeated runs, as follows.

(1) EA+FIFO: fixed EA routing assigns machines, and FIFO determines operation order, providing a manual-rule baseline.

(2) GP: conventional GP evolves dispatching rules to assess rule learning without a niching mechanism.

(3) GPDRL: a standard GP-generated library forms the action set, and DRL selects rules during scheduling, assessing the combination of conventional GP and RL.

(4) NichGPDRL: the approach of Xu et al. combines niching GP and DRL [9]. NichGP generates a rule set, followed by online rule selection using DRL.

(5) THDQN: the dynamic multi-objective flexible job shop method of Luo et al. [22] provides a DRL scheduling baseline.

(6) INichGPDRL: the proposed method combines improved niching rule learning, Double DQN online rule selection and speed adjustment, with energy terms in both rule fitness and rewards.

NichGPDRL and THDQN are implemented under this study's experimental settings and simulation environment, using the shop size, dynamic events and evaluation measures specified in Section 5.1.

In addition to overall comparisons, ablations examine niche clearing, random immigration and immigrant screening in INichGP, together with paired comparisons of the same schedules before and after speed adjustment. The first group assesses rule-library construction; the second evaluates energy savings from the predefined lower speed and its effect on completion times. Details are provided in Section 6.2.

6 Experimental results and analysis

This section evaluates INichGPDRL through overall performance comparisons, ablation studies, rule-behaviour analysis and parameter sensitivity experiments. Each stochastic learning configuration is independently trained ten times per scenario and tested on 100 common unseen instances per run. Total tardiness and total energy consumption follow the common protocol in Section 5.

6.1 Overall algorithm performance

To assess the ability of INichGPDRL to balance due-date performance and production energy under dynamic disturbances, we compare it with EA+FIFO, GP, GPDRL, Xu et al.'s NichGPDRL [9] and Luo et al.'s THDQN [22]. EA+FIFO provides a manual-rule baseline; GP and GPDRL assess automatic rule learning and online selection, while NichGPDRL and THDQN represent alternative RL scheduling frameworks. All methods use the present environment and parameters and common test instances in HH, HL, LH and LL. Figures 3 and 4 and Tables 8 and 9 report distributions and summaries of total tardiness and total energy consumption, respectively. Two-sided Wilcoxon signed-rank tests [45] compare INichGPDRL with each alternative in all four scenarios. Symbols ↑, ↓ and = indicate significantly better performance of INichGPDRL, significantly worse performance, and no detected significant difference, respectively. This section compares scheduling performance before speed post-processing; its additional energy savings are analysed separately in Section 6.2.

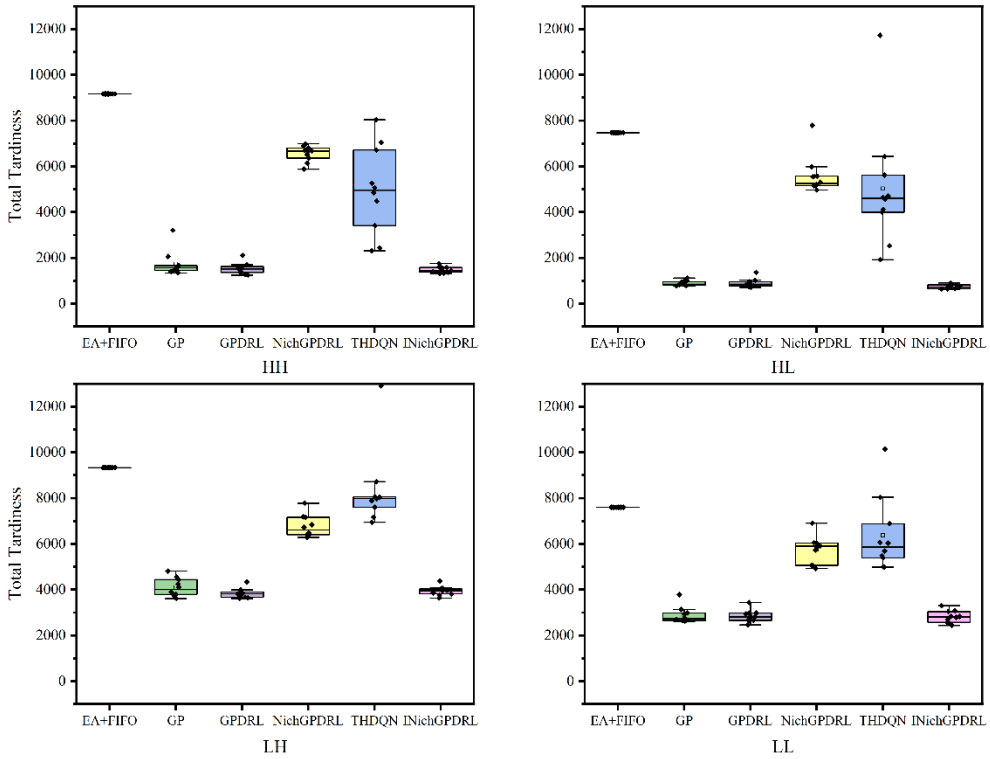

Fig. 3 Total tardiness of six algorithms over ten independent runs in four scenarios

Table 8 Total tardiness of six algorithms over ten independent runs in four scenarios

Scenario	EA+FIFO	GP	GPDRL	NichGPDRL	THDQN	INichGPDRL
HH	9166.0 ± 0.0(↑)	1736.4 ± 553.2(↑)	1533.3 ± 248.3(↑)	6557.4 ± 345.5(↑)	4958.2 ± 1915.7(↑)	**1478.1 ± 141.7**
HL	7468.0 ± 0.0(↑)	888.9 ± 111.7(↑)	902.2 ± 189.7(↑)	5582.2 ± 828.3(↑)	5023.3 ± 2697.3(↑)	**734.3 ± 92.2**
LH	9330.0 ± 0.0(↑)	4100.6 ± 399.7(=)	**3837.6 ± 212.0(↓)**	6768.5 ± 477.6(↑)	8327.7 ± 1682.6(↑)	3942.1 ± 204.2
LL	7598.0	2887.7	2838.2	5753.7	6369.4	**2805.0**

	± 0.0(↑)	± 357.5(=)	± 266.8(↓)	± 604.2(↑)	± 1614.3(↑)	**± 275.2**

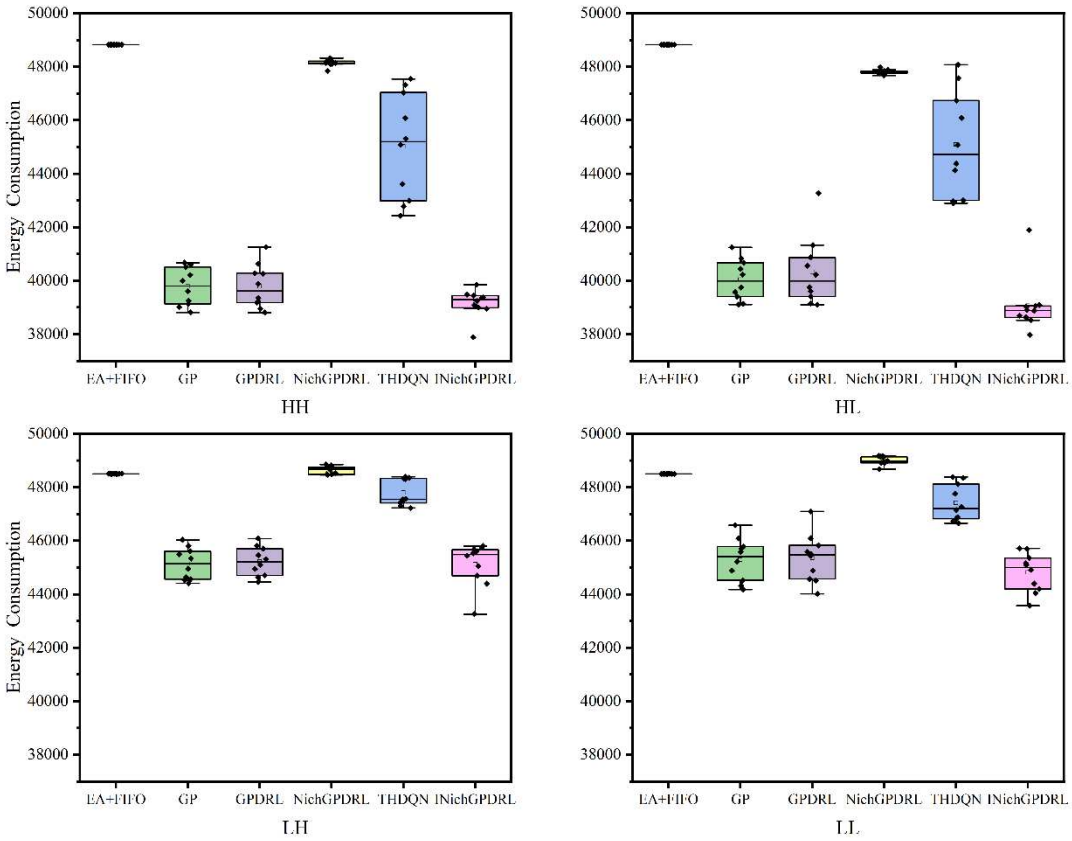

Fig. 4 Total energy consumption of six algorithms over ten independent runs in four scenarios

Table 9 Total energy consumption of six algorithms over ten independent runs in four scenarios

Scenario	EA+FIFO	GP	GPDRL	NichGPDRL	THDQN	INichGPDRL
HH	48826.0 ± 0.0(↑)	**39781.2** **± 705.3(=)**	39781.5 ± 807.7(=)	48143.0 ± 125.2(↑)	45018.5 ± 1965.5(↑)	39915.4 ± 454.7
HL	48826.0 ± 0.0(↑)	40039.2 ± 753.0(↑)	40328.4 ± 1275.4(↑)	47801.0 ± 86.3(↑)	45091.2 ± 1939.6(↑)	**39844.5** **± 935.2**
LH	48504.0 ± 0.0(↑)	**45134.9** **± 592.2(↓)**	45217.2 ± 545.2(↓)	48645.6 ± 147.2(↑)	47794.9 ± 484.4(↑)	45511.1 ± 703.3
LL	48504.0 ± 0.0(↑)	45284.8 ± 800.0(=)	45350.5 ± 896.7(=)	48994.8 ± 157.9(↑)	47409.3 ± 682.2(↑)	**45199.2** **± 599.5**

Figure 3 shows that total tardiness for GP, GPDRL and INichGPDRL is concentrated in a lower range than for the other methods. As reported in Table 8, INichGPDRL achieves the lowest mean total tardiness in HH, HL and LL, reducing it relative to GPDRL by 3.60%, 18.61% and 1.17%, respectively. According to the Wilcoxon signed-rank tests, however, the differences are significant only in HH and HL. In LH, its mean total tardiness is 2.72% higher than GPDRL's. The improvement over the existing GP-based rule-selection framework is therefore mainly observed in HH and HL, where processing-time variability is high, rather than under all conditions.

The energy results differ from the tardiness comparison. In Fig. 4, GP, GPDRL and INichGPDRL occupy the lower energy range. Table 9 shows that INichGPDRL achieves the lowest mean total tardiness and energy consumption simultaneously in HL and LL. In HH, its lower tardiness accompanies energy consumption 0.34% above GPDRL, whereas GPDRL has lower means for both measures in LH. The Wilcoxon signed-rank tests also indicate a similar pattern relative to GP and GPDRL, with superior energy performance detected only in HL.

In contrast, INichGPDRL significantly outperforms EA+FIFO, NichGPDRL and THDQN on both measures in all four scenarios. The gap relative to NichGPDRL may involve coordination between evolved rules and an independently learned routing policy, as discussed in the next subsection. The original THDQN organises decisions around tardiness and machine utilisation, differing from the joint tardiness–energy objective used here; higher utilisation is not directly equivalent to lower energy consumption. Moreover, these reference methods are implemented with adaptations to the present environment. The results therefore reflect their performance under this study's conditions and do not invalidate their effectiveness in the original settings.

6.2 Ablation studies

To distinguish the effects of rule-library construction mechanisms, the complete INichGP configuration A1 is compared with A2 without niche clearing, A3 without random immigrants, and A4 with unscreened immigrants. A2 retains fitness and behavioural screening. A3 fills the population with conventional offspring. A4 uses the same immigrant limit and candidate-generation budget as A1 but removes admission thresholds. Fitness settings, genetic operations, library size and online selection otherwise remain unchanged. Figures 5 and 6 and Tables 10 and 11 report scheduling performance after online selection from the libraries produced by these configurations.

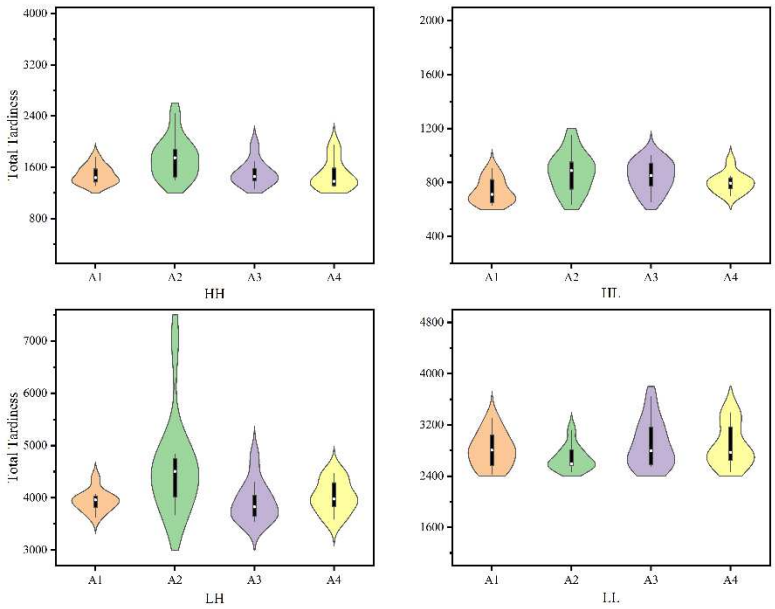

Fig. 5 Total tardiness distributions for complete INichGP and three ablations over ten independent runs in four scenarios

Table 10 Total tardiness for complete INichGP and three ablations over ten independent runs in four scenarios

Scenario	A1	A2	A3	A4
HH	**1478.1±141.7**	1747.5±324.8(↑)	1504.6±199.4(↑)	1488.6±230.6(=)
HL	**734.3±92.2**	873.3±149.3(↑)	844.1±113.9(↑)	799.2±74.6(↑)
LH	3942.1±204.2	4616.2±942.3(↑)	**3929.1±384.5(=)**	4018.8±311.2(=)
LL	2805.0±275.2	**2676.1±191.9(↓)**	2892.2±366.7(=)	2864.0±332.0(=)

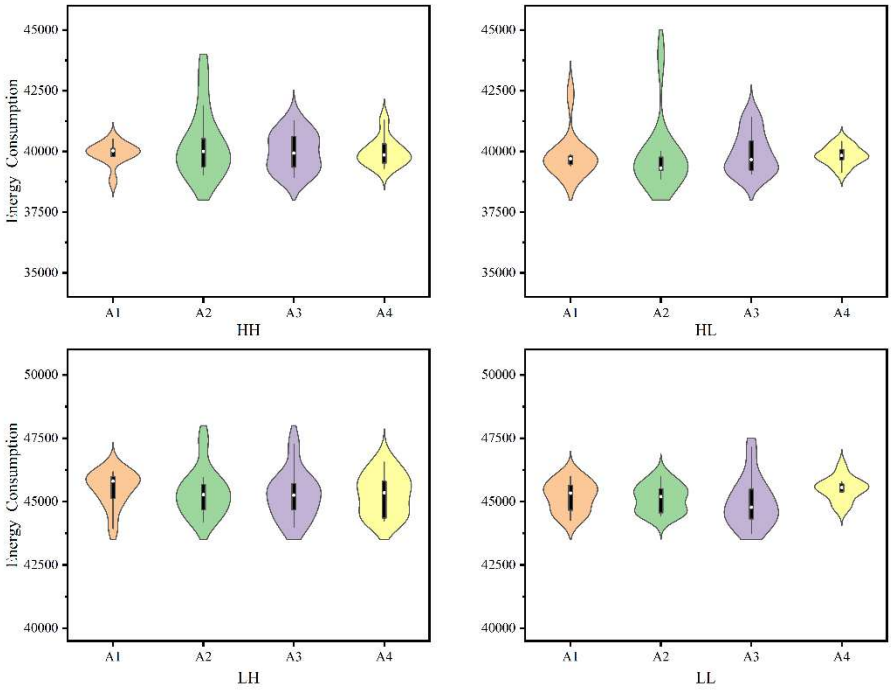

Fig. 6 Total energy consumption distributions for complete INichGP and three ablations over ten independent runs in four scenarios

Table 11 Total energy consumption for complete INichGP and three ablations over ten independent runs in four scenarios

Scenario	A1	A2	A3	A4
HH	**39915.4±454.7**	40309.7±1390.1(=)	39976.7±802.0(=)	39965.3±573.4(=)
HL	39844.5±935.2	39835.0±1493.7(=)	39927.4±857.8(=)	**39826.7±385.6(=)**
LH	45511.1±703.3	45350.5±930.6(=)	**45248.9±957.0(=)**	45278.6±822.5(=)
LL	45199.2±599.5	45123.2±549.3(=)	**44970.5±984.5(=)**	45508.4±457.6(↑)

Removing niche clearing increases A2's mean total tardiness by 18.23%, 18.93% and 17.10% in HH, HL and LH, respectively. In LH, the standard deviation rises from 204.2 to 942.3, and Fig. 5 shows a longer high-tardiness tail. By limiting repeated retention of behaviourally similar individuals, clearing

provides more distinct choices within a small library. These changes are consistent with that purpose and suggest effects on both average due-date performance and the incidence of poor runs. However, A2 has lower mean total tardiness in LL, indicating a continuing trade-off between preserving behavioural differences and concentrating on high-quality individuals.

The effects of random immigrants and their screening are most evident in HH and HL. In HL, mean total tardiness increases by 14.95% for A3 and 8.84% for A4 relative to A1, supporting the mean-performance benefit of screened new individuals in this scenario. Conventional genetic operations generate offspring around existing individuals, while random immigrants introduce expressions from outside the current population. Fitness and behavioural thresholds then determine whether these individuals merit admission. The mechanism therefore aims to introduce useful new choices, rather than simply add randomness.

Energy changes across ablations are smaller than tardiness changes. Distributions overlap substantially in Fig. 6, and Table 11 shows that all mean energy changes relative to A1 are below 1%. This is consistent with a joint fitness function that allows trade-offs: improving combined evaluation does not ensure simultaneous reductions in tardiness and energy in every scenario. The main benefit of the current library-construction mechanisms is improved due-date performance in some scenarios, while their independent energy-saving contribution remains limited.

The distinction between NichGPDRL and the proposed dual-tree decisions concerns how offline-evolved routing knowledge participates in online scheduling. Xu et al. [9] jointly evolve sequencing and routing trees in NichGP, but use only sequencing rules as online candidate actions; an independently trained routing agent assigns machines. FIFO sequencing is fixed when training the routing agent, and EA routing is fixed when training the sequencing agent. Changes in the routing policy can alter machine assignments, queue composition and subsequent waiting times, potentially affecting the performance of evolved sequencing rules. Differences between training and deployment conditions may therefore contribute to the performance gap observed in this study. INichGPDRL instead makes both evolved routing and sequencing rules available to the shared selection network, allowing online decisions to draw on both types of learned scheduling knowledge. However, the comparison also involves differences in rule-library construction, rewards and learning organisation. The results therefore demonstrate the performance difference between the complete frameworks under the present conditions, but do not isolate the effect of routing–sequencing coordination.

Separately from library construction, speed post-processing is assessed through paired comparisons on the same schedules. The baseline retains the original speed. Post-processing changes only eligible non-final operations with sufficient free float to 0.7 speed; machine assignments, processing order, job arrivals and realised repair durations remain identical.

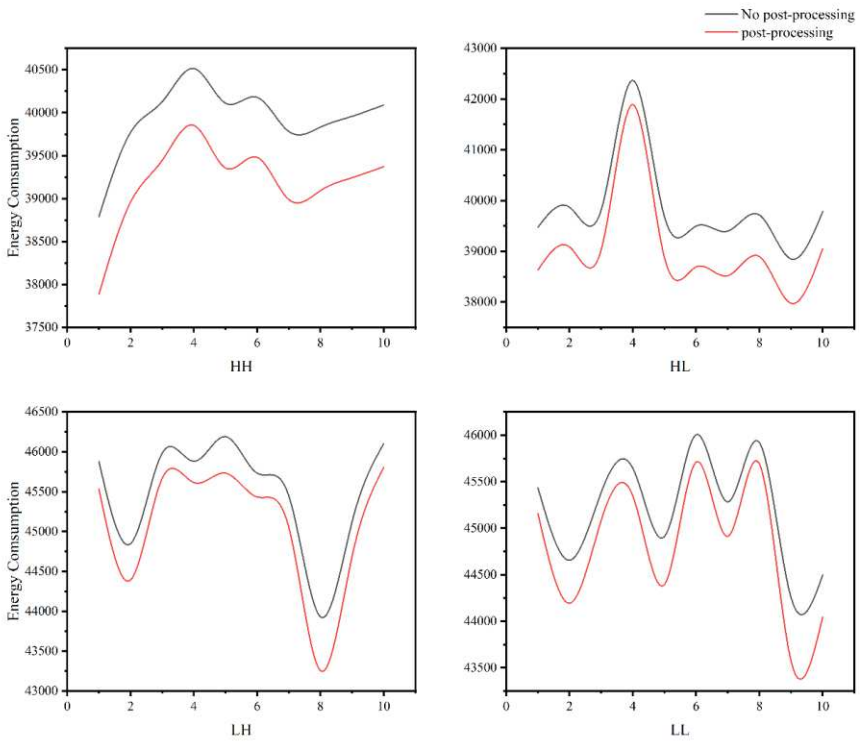

Fig. 7 Total energy consumption with and without speed post-processing over ten independent runs in four scenarios

Figure 7 shows lower post-processing energy consumption than the corresponding baseline in all four scenarios. Mean total energy consumption falls to 39167.9, 39064.4, 45117.0 and 44811.0 in HH, HL, LH and LL, respectively. Mean energy saving rates, calculated for each run before averaging, are 1.87%, 1.96%, 0.87% and 0.86%. Energy consumption decreases in all 40 paired runs, while total tardiness remains unchanged. Under the adopted energy model, the schedules therefore retain processing-time margins that can be used for further savings. Rule learning coordinates due-date and energy performance through routing and sequencing, whereas post-processing exploits free float within the resulting schedule.

6.3 Niche behavioural diversity and action selection frequencies

Different rule expressions need not produce different choices, so structural comparisons alone cannot establish library behavioural diversity. Following Xu et al. [9], 100 sequencing and 100 routing points are sampled from a common trajectory generated on an independent unseen instance. NichGP and INichGP rules then select from identical candidate sets. Each four-individual library yields six rule pairs. Their mean behavioural difference is measured using the normalised Hamming distance in Eq. (28). Sequencing and routing are analysed separately, with means and standard deviations reported across ten independent training runs. This measurement set is independent of the training PC sample and tests whether rule differences persist in unseen decision states. The measure is defined as

$$\rho = \frac{1}{6L}\sum_{1\le a<b\le 4}\sum_{l=1}^{L}\mathbf{1}\{u_a(l) \neq u_b(l)\}, \qquad L = 100. \tag{28}$$

Here, $u_a(l)$ is the candidate selected by rule a at common decision point l, and $\mathbf{1}\{\cdot\}$ is the indicator function. Measure ρ is a proportion and is multiplied by 100 when reported as a percentage. Sequencing and routing are calculated separately, with scenario-level means and standard deviations over ten independent training runs. Independence from the training PC sample allows the persistence of behavioural differences to be assessed in unseen decision states.

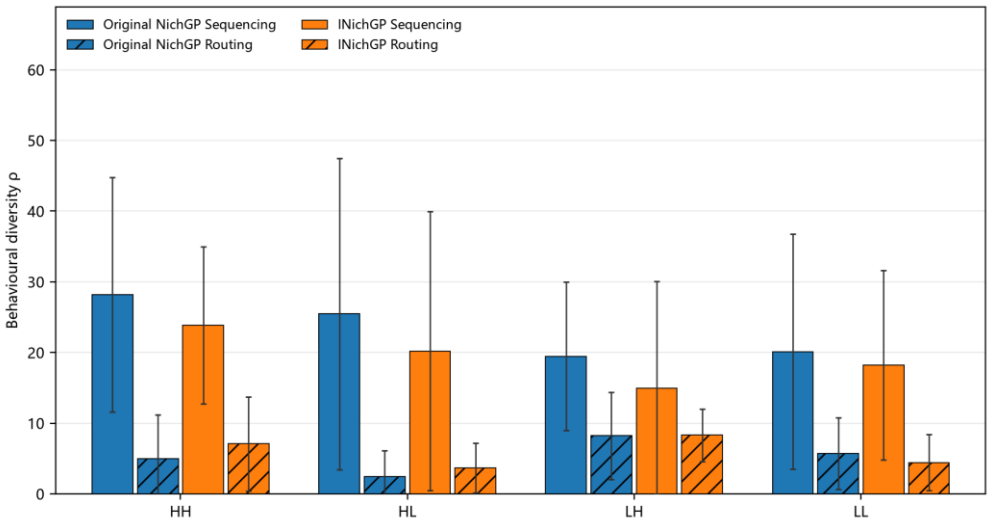

Fig. 8 Behavioural differences ρ of NichGP and INichGP in four scenarios

Figure 8 shows sequencing behavioural differences of 14.97%–23.87% for INichGP, below the corresponding NichGP values in all four scenarios. Routing differences increase in HH and HL, remain similar in LH, and decrease in LL. Together with the preceding performance results, these findings show that greater disagreement between choices does not necessarily imply better scheduling. A behaviourally differentiated library provides alternatives for DRL, but differences alone do not guarantee a better learned schedule; rule quality also directly affects learning.

To examine whether retained rules are actually used, we also measure sequencing-action selection frequencies for GPDRL, NichGPDRL and INichGPDRL. Behavioural differences indicate whether rules make different choices, while selection frequencies describe their use during scheduling; together they inform the analysis of library complementarity. Action counts are first aggregated over 100 test instances for each trained DRL model, and the resulting proportions are averaged over ten training runs. Figure 9 shows that the four INichGPDRL actions account for 20.68%–27.96% of decisions in HH and 17.87%–32.51% in LL, more evenly than NichGPDRL. Use is more concentrated in HL and LH, where the most frequent action accounts for 39.80% and 43.65%, respectively. The aggregated results thus show multiple actions participating in scheduling, with their relative use depending on production conditions.

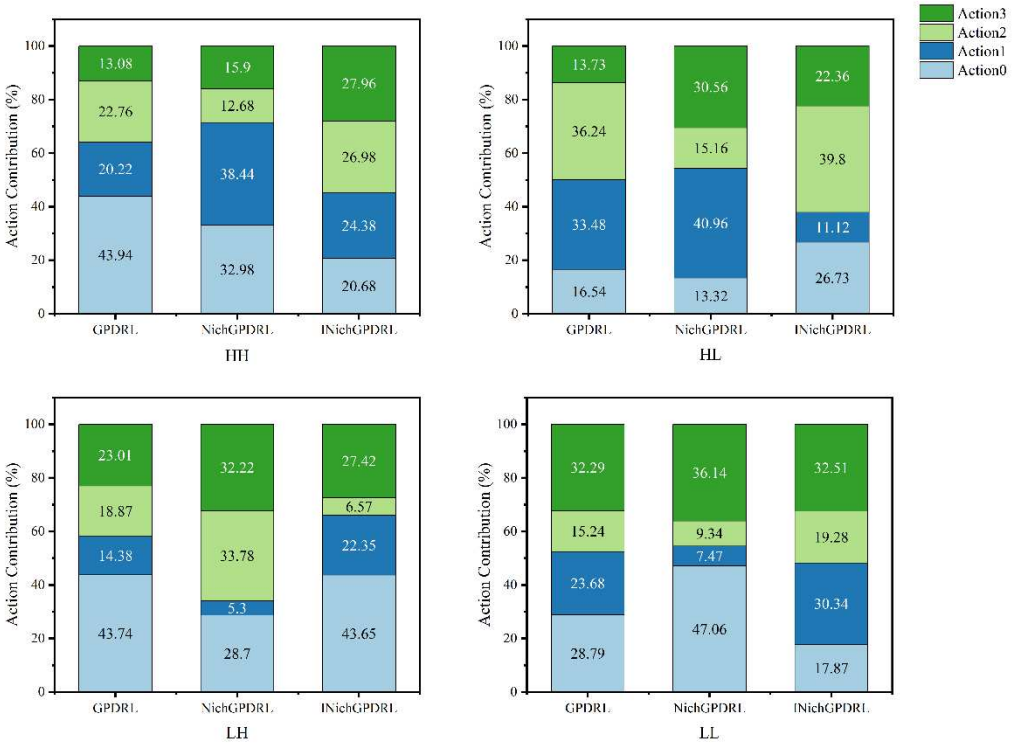

Fig. 9 Action selection frequencies of three algorithms in four scenarios

Behavioural differences and selection frequencies jointly show that the retained rules neither make entirely identical choices nor remain wholly unused. However, selection frequency is not performance contribution. Frequent use does not establish that an action is better, and a rarely selected action may matter in a small number of critical states. Moreover, action indices denote different rules across independently trained models, so aggregate proportions do not establish that every action was used in every run.

6.4 Parameter sensitivity analysis

The number of PC reference points affects behavioural coverage, while immigrant count controls how many new individuals can enter each generation. Sequencing and routing reference-point counts are each set to 10, 20 or 40, and immigrant counts to 0, 10, 20 or 40. One parameter is varied at a time, with all other settings fixed. Each configuration is independently trained ten times in all four scenarios and evaluated on common unseen instances. Tables 12 and 13 report mean tardiness and energy consumption of the library rules to examine effects on rule learning. This differs from Section 6.1, which evaluates schedules produced through online selection.

Table 12 Effects of PC reference-point count on INichGPDRL performance

Level	Scenario	Mean rule tardiness	Mean rule energy	Δ tardiness vs baseline	Δ energy vs baseline
10	HH	1664.8	39941.1	+116	+7
	HL	968.3	40264.2	+200	+546
	LH	4049.7	45170.0	+82	-134
	LL	2856.6	45024.6	-5	-136

20 Baseline	HH	1548.5	39934.2	+0	+0
	HL	768.4	39718.2	+0	+0
	LH	3967.6	45303.8	+0	+0
	LL	2861.7	45160.9	+0	+0
40	HH	1520.6	40472.7	-28	+538
	HL	789.4	39836.0	+21	+118
	LH	4032.0	45337.0	+64	+33
	LL	2967.4	45015.6	+106	-145

Table 13 Effects of immigrant count on INichGPDRL performance

Level	Scenario	Mean rule tardiness	Mean rule energy	Δ tardiness vs baseline	Δ energy vs baseline
0	HH	1549.2	39915.1	+1	-19
	HL	853.1	39852.6	+85	+134
	LH	3928.0	45175.2	-40	-129
	LL	2953.0	44902.3	+91	-259
10	HH	1556.4	40457.3	+8	+523
	HL	946.8	40035.5	+178	+317
	LH	3789.2	45181.6	-178	-122
	LL	2976.0	45232.4	+114	+72
20 (baseline)	HH	1548.5	39934.2	+0	+0
	HL	768.4	39718.2	+0	+0
	LH	3967.6	45303.8	+0	+0
	LL	2861.7	45160.9	+0	+0
40	HH	1564.9	40255.6	+16	+321
	HL	806.2	39813.7	+38	+96
	LH	3957.5	45363.9	-10	+60
	LL	2738.4	44644.7	-123	-516

Table 12 shows that reducing each reference-point count from 20 to 10 increases mean rule tardiness by 7.51% in HH and 26.02% in HL. Increasing it to 40 does not consistently improve

performance: mean tardiness decreases by 1.80% in HH but energy increases by 1.35%, and mean tardiness rises in the other scenarios. Fewer reference points may make some rules indistinguishable, but adding points does not necessarily cover more useful decision states. PC effectiveness therefore depends on sample representativeness as well as sample size; higher dimensionality alone does not guarantee better rules. Immigrant count is similarly scenario-dependent. In Table 13, 20 immigrants per generation give the lowest mean rule tardiness in HH and HL. LH performs better with 10, reducing mean tardiness by 4.50% relative to baseline. In LL, 40 immigrants improve both measures, reducing tardiness by 4.31% and energy by 1.14%. More immigrants broaden the source of new behaviours but change the retained balance between conventional offspring and external individuals. Benefits depend on the need for new choices and the quality of incoming individuals, rather than increasing monotonically with immigrant count.

Immigration also increases evolutionary computation cost, and immigrant count directly affects training time. We therefore examine the computational overhead of different counts.

Table 14 Computational time of INichGPDRL with different immigrant counts

Level	Scenario	Mean total time per run (min)	Mean INichGP evolution time (min)	Mean DRL training time (min)	Complete immigrant evaluations
0	HH	60.4	21.3	6.2	0
	HL	62.9	23.0	6.2	0
	LH	65.8	24.6	6.2	0
	LL	65.3	24.0	6.3	0
10	HH	76.4	30.2	6.1	1500
	HL	77.0	30.2	6.1	1500
	LH	77.4	30.7	6.1	1500
	LL	76.6	30.0	6.1	1500
20 (baseline)	HH	86.5	34.3	6.2	3000
	HL	101.2	36.3	8.1	3000
	LH	114.7	44.2	6.6	3000
	LL	110.9	56.0	6.0	3000
40	HH	116.4	50.3	6.0	6000
	HL	116.6	50.0	6.1	6000
	LH	118.7	51.3	6.1	6000
	LL	109.9	51.8	4.8	6000

In Table 14, immigrant counts of 0, 10, 20 and 40 correspond to 0, 1500, 3000 and 6000 recorded complete candidate evaluations. Mean library-evolution times across the four scenarios are 23.2, 30.3,

42.7 and 50.8 min, respectively, while DRL training times vary less. The additional cost therefore arises mainly in offline rule evaluation, but more candidate evaluations do not consistently improve performance across scenarios. Accordingly, 20 reference points of each type and 20 immigrants per generation are used as the common baseline to balance rule-learning performance and evaluation budget.

7 Conclusions and future work

This paper proposes INichGPDRL for dynamic flexible job shops with random job arrivals and machine breakdowns, jointly considering total tardiness and total energy consumption. Building on niching GP, it introduces joint tardiness–energy fitness and screens random immigrants using fitness and behavioural thresholds to construct a candidate library for online decisions. Complete rule individuals then serve as RL actions, with their routing or sequencing trees invoked at the corresponding decision points so that both evolved rules participate in scheduling. Once a schedule is generated, free float identifies non-final operations that can be slowed down within timing constraints to obtain additional energy savings.

Across four scenarios, INichGPDRL achieves the lowest mean total tardiness in HH, HL and LL, and the lowest mean total energy consumption simultaneously in HL and LL. In HL, mean total tardiness is 18.61% below GPDRL. Its advantages are nevertheless scenario-dependent: the tardiness improvement in HH accompanies slightly higher energy use, while both means exceed GPDRL in LH. The approach thus improves due-date and energy performance under some production conditions without establishing a consistent advantage everywhere. Speed post-processing reduces energy consumption in all 40 paired runs, with mean savings of 0.86%–1.96% across scenarios and unchanged total tardiness, demonstrating further use of schedule time margins under the adopted model.

Ablation and behavioural analyses reveal trade-offs in library construction. Niche clearing improves mean tardiness and run stability in some scenarios, while the mean-performance benefits of random immigrants and screening are mainly observed in HH and HL. Although INichGP has lower sequencing behavioural differences than NichGP, multiple rules still participate in scheduling. Disagreement between rules and improvements in scheduling are therefore distinct: library construction must consider quality, behavioural differences and actual use together. Parameter experiments also show that more reference points or immigrants do not necessarily improve performance, while additional candidate evaluations increase offline evolution cost.

Future research can further investigate coordination between dispatching rules and online decisions, particularly the interaction between machine assignment and operation sequencing. More adaptive rule learning and selection could improve the ability to balance due-date performance and energy consumption in complex dynamic environments. The present experiments have not established applicability to larger shops or different loads, and speed post-processing relies on realised events in a fixed schedule. Further work should extend shop sizes and disturbance conditions, distinguish performance after retraining from transfer across scenarios, and refine the energy model to reflect equipment baseline power, discrete speeds and processing-quality constraints. These extensions can support speed adjustment with unknown future events and closer coordination of rule learning, dynamic scheduling and equipment control for practical delivery and energy requirements.

References

[1] B. Motherway, J. Bilodeau, J. Cohen, E. Mooney, V. Rozite, and M. Slade, “Energy Efficiency 2024,” Paris, France: IEA, Nov. 2024, p. 56. Accessed: Oct. 13, 2026. https://www.iea.org/reports/energy-efficiency-2024.

[2] S. Dauzère-Pérès, J. Ding, L. Shen, and K. Tamssaouet, “The flexible job shop scheduling problem: A review,” *Eur. J. Oper. Res.*, vol. 314, no. 2, pp. 409–432, Apr. 2024, doi: 10.1016/j.ejor.2023.05.017.
[3] C. Destouet, H. Tlahig, B. Bettayeb, and B. Mazari, “Flexible job shop scheduling problem under Industry 5.0: A survey on human reintegration, environmental consideration and resilience improvement,” *J. Manuf. Syst.*, vol. 67, pp. 155–173, Apr. 2023, doi: 10.1016/j.jmsy.2023.01.004.
[4] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, “Survey on Genetic Programming and Machine Learning Techniques for Heuristic Design in Job Shop Scheduling,” *IEEE Trans. Evol. Comput.*, vol. 28, no. 1, pp. 147–167, Feb. 2024, doi: 10.1109/TEVC.2023.3255246.
[5] Y. Mei, S. Nguyen, B. Xue, and M. Zhang, “An efficient feature selection algorithm for evolving job shop scheduling rules with genetic programming,” *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 1, no. 5, pp. 339–353, Oct. 2017, doi: 10.1109/TETCI.2017.2743758.
[6] M. Xu, Y. Mei, F. Zhang, and M. Zhang, “Genetic Programming for Dynamic Flexible Job Shop Scheduling: Evolution With Single Individuals and Ensembles,” *IEEE Trans. Evol. Comput.*, vol. 28, no. 6, pp. 1761–1775, Dec. 2024, doi: 10.1109/TEVC.2023.3334626.
[7] M. Xu, Y. Mei, F. Zhang, and M. Zhang, “Genetic Programming With Lexicase Selection for Large-Scale Dynamic Flexible Job Shop Scheduling,” *IEEE Trans. Evol. Comput.*, vol. 28, no. 5, pp. 1235–1249, Oct. 2024, doi: 10.1109/TEVC.2023.3244607.
[8] S. Luo, “Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning,” *Appl. Soft Comput.*, vol. 91, p. 106208, Jun. 2020, doi: 10.1016/j.asoc.2020.106208.
[9] M. Xu, Y. Mei, F. Zhang, and M. Zhang, “Niching Genetic Programming to Learn Actions for Deep Reinforcement Learning in Dynamic Flexible Scheduling,” *IEEE Trans. Evol. Comput.*, vol. 30, no. 1, pp. 61–75, Feb. 2026, doi: 10.1109/TEVC.2024.3395699.
[10] M. Abedi, R. Chiong, N. Noman, and R. Zhang, “A multi-population, multi-objective memetic algorithm for energy-efficient job-shop scheduling with deteriorating machines,” *Expert Syst. Appl.*, vol. 157, p. 113348, Nov. 2020, doi: 10.1016/j.eswa.2020.113348.
[11] K. Gao, Z. Cao, L. Zhang, Z. Chen, Y. Han, and Q. Pan, “A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems,” *IEEECAA J. Autom. Sin.*, vol. 6, no. 4, pp. 904–916, Jul. 2019, doi: 10.1109/JAS.2019.1911540.
[12] M. Xu, Y. Mei, F. Zhang, and M. Zhang, “Learn to optimise for job shop scheduling: a survey with comparison between genetic programming and reinforcement learning,” *Artif. Intell. Rev.*, vol. 58, no. 6, p. 160, Mar. 2025, doi: 10.1007/s10462-024-11059-9.
[13] S. Nguyen, Y. Mei, and M. Zhang, “Genetic programming for production scheduling: a survey with a unified framework,” *Complex Intell. Syst.*, vol. 3, no. 1, pp. 41–66, Mar. 2017, doi: 10.1007/s40747-017-0036-x.
[14] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, “A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem,” *IEEE Trans. Evol. Comput.*, vol. 17, no. 5, pp. 621–639, Oct. 2013, doi: 10.1109/TEVC.2012.2227326.
[15] F. Zhang, Y. Mei, and M. Zhang, “A New Representation in Genetic Programming for Evolving Dispatching Rules for Dynamic Flexible Job Shop Scheduling,” in *Evolutionary Computation in Combinatorial Optimization*, A. Liefooghe and L. Paquete, Eds., Cham: Springer International Publishing, 2019, pp. 33–49. doi: 10.1007/978-3-030-16711-0_3.
[16] B. Xu, Y. Mei, Y. Wang, Z. Ji, and M. Zhang, “Genetic programming with delayed routing for multiobjective dynamic flexible job shop scheduling,” *Evol. Comput.*, vol. 29, no. 1, pp. 75–105, Mar. 2021, doi: 10.1162/evco_a_00273.
[17] J. Park, S. Nguyen, M. Zhang, and M. Johnston, “Evolving ensembles of dispatching rules using genetic programming for job shop scheduling,” in *Genetic programming*, vol. 9025, P. Machado, M. I. Heywood, J. McDermott, M. Castelli, P. García-Sánchez, P. Burelli, S. Risi, and K. Sim, Eds., in Lecture Notes in Computer Science, vol. 9025. , Cham: Springer International Publishing, 2015, pp. 92–104. doi: 10.1007/978-3-319-16501-1_8.
[18] X. Li *et al.*, “Flexible Job Shop Composite Dispatching Rule Mining Approach Based on an Improved Genetic Programming Algorithm,” *Tsinghua Sci. Technol.*, vol. 29, no. 5, pp. 1390–1408, Oct. 2024, doi: 10.26599/TST.2023.9010141.
[19] A. Sitahong, Y. Yuan, M. Li, J. Ma, Z. Ba, and Y. Lu, “Learning dispatching rules via novel genetic programming with feature selection in energy-aware dynamic job-shop scheduling,” *Sci. Rep.*, vol. 13, no. 1, p. 8558, May 2023, doi: 10.1038/s41598-023-34951-w.

[20] B. Jiang, Y. Ma, L. Chen, B. Huang, Y. Huang, and L. Guan, "A Review on Intelligent Scheduling and Optimization for Flexible Job Shop," *Int. J. Control Autom. Syst.*, vol. 21, no. 10, pp. 3127–3150, Oct. 2023, doi: 10.1007/s12555-023-0578-1.
[21] L. Lv, C. Zhang, J. Fan, and W. Shen, "Deep reinforcement learning for job shop scheduling problems: a comprehensive literature review," *Knowl.-Based Syst.*, vol. 321, p. 113633, Jun. 2025, doi: 10.1016/j.knosys.2025.113633.
[22] S. Luo, L. Zhang, and Y. Fan, "Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning," *Comput. Ind. Eng.*, vol. 159, p. 107489, Sep. 2021, doi: 10.1016/j.cie.2021.107489.
[23] K. Lei *et al.*, "A multi-action deep reinforcement learning framework for flexible Job-shop scheduling problem," *Expert Syst. Appl.*, vol. 205, p. 117796, Nov. 2022, doi: 10.1016/j.eswa.2022.117796.
[24] X. Chen, R. Bai, R. Qu, J. Dong, and Y. Jin, "Deep Reinforcement Learning Assisted Genetic Programming Ensemble Hyper-Heuristics for Dynamic Scheduling of Container Port Trucks," *IEEE Trans. Evol. Comput.*, vol. 29, no. 4, pp. 1371–1385, Aug. 2025, doi: 10.1109/TEVC.2024.3381042.
[25] M. Xu, Y. Mei, F. Zhang, and M. Zhang, "Genetic Programming and Reinforcement Learning on Learning Heuristics for Dynamic Scheduling: A Preliminary Comparison," *IEEE Comput. Intell. Mag.*, vol. 19, no. 2, pp. 18–33, May 2024, doi: 10.1109/MCI.2024.3363970.
[26] I. G. Smit *et al.*, "Graph neural networks for job shop scheduling problems: A survey," *Comput. Oper. Res.*, vol. 176, p. 106914, Apr. 2025, doi: 10.1016/j.cor.2024.106914.
[27] K. Gao, Y. Huang, A. Sadollah, and L. Wang, "A review of energy-efficient scheduling in intelligent production systems," *Complex Intell. Syst.*, vol. 6, no. 2, pp. 237–249, Jul. 2020, doi: 10.1007/s40747-019-00122-6.
[28] G. May, B. Stahl, M. Taisch, and V. Prabhu, "Multi-objective genetic algorithm for energy-efficient job shop scheduling," *Int. J. Prod. Res.*, vol. 53, no. 23, pp. 7071–7089, Dec. 2015, doi: 10.1080/00207543.2015.1005248.
[29] Gong G. *et al.*, "A two-stage memetic algorithm for energy-efficient flexible job shop scheduling by means of decreasing the total number of machine restarts," *Swarm Evol. Comput.*, vol. 75, p. 101131, Dec. 2022, doi: 10.1016/j.swevo.2022.101131.
[30] Wei Z., Liao W., and Zhang L., "Hybrid energy-efficient scheduling measures for flexible job-shop problem with variable machining speeds," *Expert Syst. Appl.*, vol. 197, p. 116785, Jul. 2022, doi: 10.1016/j.eswa.2022.116785.
[31] Qu H., Tong X., Cai M., Shi Y., and Lan X., "Energy-saving scheduling strategy for variable-speed flexible job-shop problem considering operation-dependent energy consumption," *Expert Syst. Appl.*, vol. 256, p. 124952, Dec. 2024, doi: 10.1016/j.eswa.2024.124952.
[32] Duan J. and Wang J., "Energy-efficient scheduling for a flexible job shop with machine breakdowns considering machine idle time arrangement and machine speed level selection," *Comput. Ind. Eng.*, vol. 161, p. 107677, Nov. 2021, doi: 10.1016/j.cie.2021.107677.
[33] Zhang R. and Chiong R., "Solving the energy-efficient job shop scheduling problem: a multi-objective genetic algorithm with enhanced local search for minimizing the total weighted tardiness and total energy consumption," *J. Clean. Prod.*, vol. 112, pp. 3361–3375, Jan. 2016, doi: 10.1016/j.jclepro.2015.09.097.
[34] Zhang L., Tang Q., Wu Z., and Wang F., "Mathematical modeling and evolutionary generation of rule sets for energy-efficient flexible job shops," *Energy*, vol. 138, pp. 210–227, Nov. 2017, doi: 10.1016/j.energy.2017.07.005.
[35] M.-J. Park and A. Ham, "Energy-aware flexible job shop scheduling under time-of-use pricing," *Int. J. Prod. Econ.*, vol. 248, p. 108507, Jun. 2022, doi: 10.1016/j.ijpe.2022.108507.
[36] Z. Rui *et al.*, "Graph reinforcement learning for flexible job shop scheduling under industrial demand response: a production and energy nexus perspective," *Comput. Ind. Eng.*, vol. 193, p. 110325, Jul. 2024, doi: 10.1016/j.cie.2024.110325.
[37] M. Makhoabenyane, S. Guo, and E. Leburu, "Towards Sustainable Factories: A Systematic Review of Energy-Conscious Job-Shop Scheduling Models and Algorithms," *Sustainability*, vol. 17, no. 24, p. 11330, Jan. 2025, doi: 10.3390/su172411330.
[38] X. Wu and A. Che, "A memetic differential evolution algorithm for energy-efficient parallel machine scheduling," *Omega*, vol. 82, pp. 155–165, Jan. 2019, doi: 10.1016/j.omega.2018.01.001.

[39] R. S. Sutton and A. G. Barto, “Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.
[40] R. Liu, R. Piplani, and C. Toro, “Deep reinforcement learning for dynamic scheduling of a flexible job shop,” *Int. J. Prod. Res.*, vol. 60, no. 13, pp. 4049–4069, Jul. 2022, doi: 10.1080/00207543.2022.2058432.
[41] H. van Hasselt, A. Guez, and D. Silver, “Deep Reinforcement Learning with Double Q-learning,” Dec. 08, 2015, *arXiv*: arXiv:1509.06461. doi: 10.48550/arXiv.1509.06461.
[42] V. Mnih *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.
[43] F.-A. Fortin *et al.*, “DEAP: Evolutionary Algorithms Made Easy,” *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, 2012.
[44] A. Paszke *et al.*, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” Dec. 03, 2019, *arXiv*: arXiv:1912.01703. doi: 10.48550/arXiv.1912.01703.
[45] F. Wilcoxon, “Individual Comparisons by Ranking Methods,” *Biometrics Bull.*, vol. 1, no. 6, pp. 80–83, Dec. 1945, doi: 10.2307/3001968.

Acknowledgements

This work was funded by the National Natural Science Foundation of China (No. 72661036); The Key Industry Talent Support Program in the Field of Industry and Information Technology of Xinjiang Uygur Autonomous Region (No. XJRC-2025-GX-ZDCY-13); Autonomous Region Key Research and Development Program: Key Technology Research and Achievement Transformation of Intelligent Manufacturing Execution System for High end Agricultural Machinery Equipment (No. 2025B04005-0001), and Xinjiang Uygur Autonomous Region Key R&D Special Program—Department-Local Joint Initiative (No. 2024B04003-1).

Author Contributions

Gang Wang: Conceptualization, Methodology, Software, Formal analysis, Investigation, Data curation, Validation, Visualization, Writing - original draft. **Adilanmu Sitahong:** Supervision, Methodology, Writing - review & editing. **Yiping Yuan:** Validation, Writing - review & editing. **Peiyin Mo:** Writing - review & editing. **Junyan Ma:** Supervision, Writing - review & editing. **Yulong Chen:** Writing - review & editing.

Data availability statement

Part of the data supporting this study, together with the core algorithm implementation, is publicly available in the GitHub repository at https://github.com/strive676/INichGPDQN. The remaining supporting data are available from the corresponding author upon reasonable request.

Declaration of competing interests

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.

Declaration of generative AI use

During the preparation of this work the authors used DeepSeek to assist with the development of the experimental code. After using this tool, the authors reviewed and revised the resulting code and take full responsibility for its correctness and the results reported in this article.