Efficient master production scheduling for manufacturing systems using an enhanced SARSA algorithm

Yuehan Liu*

Shenyang Institute of Automation,
Chinese Academy of Sciences,
Shenyang – 110016, Liaoning, China
and
University of Chinese Academy of Sciences,
Beijing – 100049, China
Email: liuyuehan221@mails.ucas.ac.cn
*Corresponding author

Haibo Shi and Chang Liu

Shenyang Institute of Automation,
Chinese Academy of Sciences,
Shenyang – 110016, Liaoning, China
Email: hbshi@sia.cn
Email: changl@sia.cn

Abstract: Efficient master production scheduling is crucial in production planning, yet automated solutions remain scarce. This paper introduces a novel scheduling method, master production scheduling with improved state-action-reward-state-action (MPS-ISARSA) algorithm, based on an enhanced state-action-reward-state-action (SARSA) framework. Using a Markov decision process model, the method optimises scheduling through innovative reward shaping and a linearly decaying epsilon-greedy (ε_{lin}-greedy) policy to accelerate training. An overtime penalty ensures alignment between production capacity and demand. Extensive simulations validate the model's effectiveness, demonstrating superior convergence and efficiency compared to traditional methods and other reinforcement learning algorithms. This approach offers a scalable, intelligent framework for capacity-constrained production scheduling, with practical applications for manufacturing industries aiming to enhance operational efficiency.

Keywords: reinforcement learning; SARSA algorithm; production scheduling; master production scheduling; reward shaping; Markov decision process; capacity-constrained scheduling.

Reference to this paper should be made as follows: Liu, Y., Shi, H. and Liu, C. (2025) 'Efficient master production scheduling for manufacturing systems using an enhanced SARSA algorithm', *Int. J. Simulation and Process Modelling*, Vol. 22, Nos. 1/2, pp.60–74.

Biographical notes: Yuehan Liu received his BS degree in Building Electrical Engineering and Intelligentization from Shenyang Jianzhu University (SJZU), Shenyang, China in 2017 and MS degree in Control Engineering from SJZU, in 2021. He is currently pursuing his PhD at Shenyang Institute of Automation (SIA), Chinese Academy of Sciences. His research interests include deep reinforcement learning, pattern recognition, swarm intelligence algorithms, and production planning and scheduling.

Haibo Shi received his PhD degree in Mechanical and Electrical Engineering from the Harbin University of Science and Technology, Harbin, China in 2003. His current research interests include data intelligent analysis and mining, intelligent information processing platform software, and industrial process modelling.

Chang Liu received her BS degree in Computer Science and Application from Shenyang University of Technology in 1996, and her PhD degree in Mechatronic Engineering from the University of Chinese Academy of Sciences (UCAS) in 2008. She is currently a Researcher. Her research interests include manufacturing system modelling, production scheduling, intelligent optimisation, and intelligent decision-making.

1 Introduction

The master production scheduling (MPS) is a core component of production planning in manufacturing enterprises. These enterprises receive customer orders and market forecasts, then use sales and operations planning (S&OP) to formulate annual, quarterly, and monthly demand plans. These plans are issued to the production control (PC) department, where the planning manager disaggregates the demand and creates the MPS based on monthly demand and production line data. The MPS specifies the products, quantities, and production timelines. It is typically developed monthly and provides detailed weekly/daily plans, which are issued to the manufacturing execution system for execution. The MPS must balance enterprise needs, customer requirements, and production capacity constraints to ensure feasible daily plans.

The MPS is required to split demand orders from the demand plan and aggregate these split orders. Formulating the MPS is a combinatorial optimisation problem. Traditionally, production planning managers manually split demand orders based on experience or predefined rules. These split orders are then converted into production orders and assigned to the MPS. Although some enterprises have deployed enterprise resource planning systems, the MPS that is typically calculated in such systems is based on the assumption of 'infinite capacity' (Panwalkar and Iskander, 1977), which is unrealistic for capacity-constrained enterprises.

Due to the importance of MPS for manufacturing enterprises, many scholars have conducted research on related issues.

Heuristic and evolutionary algorithms have been widely explored for MPS optimisation. Akhoondi and Lotfi (2016) proposed a scenario-based heuristic to address demand uncertainty, while Astantı and Ai (2024) developed clustering heuristics for aligning production and delivery schedules. Gahm et al. (2014) introduced a multi-criteria approach for special-purpose MPS, and Vieira and Favaretto (2011) designed rule-based heuristics for practical MPS creation. Evolutionary methods, such as genetic algorithms (GA), have been applied to solve combinatorial optimisation in MPS by mimicking natural selection processes (Soares and Vieira, 2009). Differential evolution further extends this paradigm through mutation and crossover operations (Sajja et al., 2013). Recent extensions include memetic algorithms that hybridise evolutionary search with local optimisation (Sadiq et al., 2020), and metaheuristics like NSGAII-SMPSO for multi-objective factory planning (Maalouf et al., 2022).

However, these methods face critical limitations. Heuristic approaches rely on manually crafted scheduling rules that must be tailored to specific production line configurations (Gahm et al., 2014). Evolutionary algorithms, despite their global search capabilities, require extensive iterative calculations (Vieira and Favaretto, 2011). For example, GA-based MPS methods generate new solutions from scratch for each planning cycle and do not make use of previous optimisation results. This leads to redundant computations and prolonged response times, especially for enterprises with complex operational constraints.

Hybrid AI approaches combine traditional optimisation with advanced techniques to mitigate these issues. Hubbs et al. (2020) integrated reinforcement learning (RL) with integer programming for chemical production scheduling, while Song et al. (2023) fused M5P algorithms with master data analysis. Although such hybrid methods improve flexibility, they often involve intricate parameter tuning (e.g., learning rates, exploration policies) and scenario-specific model architectures (Hubbs et al., 2020). For instance, integrating mathematical programming with heuristic rules requires balancing constraint weights and exploration strategies, increasing implementation complexity and limiting generalisability.

RL provides a paradigm shift through data-driven policy learning, directly optimising decisions from historical interactions. Early applications focused on workshop-level scheduling, such as Q-learning for dynamic job shops (Shahrabi et al., 2017) and state-action-reward-state-action (SARSA) for task allocation (Momenikorbekandi and Abbod, 2023). Recent studies extend RL to enterprise-level planning: Serrano-Ruiz et al. (2024) proposed a deep Q-network for MPS, and Esteso et al. (2022) reviewed RL applications in production control. Nevertheless, key challenges persist. Sparse reward signals in complex MPS state-action spaces impede efficient exploration (Serrano-Ruiz et al., 2021). While deep reinforcement learning (DRL) enhances function approximation, its computational demands are prohibitive for medium-scale MPS. Training DRL models requires substantial computational resources and time, even for relatively small-scale scheduling instances (Wang, 2020). Additionally, industrial decision-making prioritises interpretability for safety-critical scenarios, but DRL's 'black-box' nature obscures the rationale behind scheduling actions (Li et al., 2023).

In addition to the challenges discussed above, the selection of an appropriate RL algorithm for MPS requires careful consideration of various factors. To this end, Table 1 provides a detailed comparison of SARSA, Q-learning, and DRL based on computational cost, interpretability, and handling of sparse rewards.

Table 1 Comparison of RL algorithms for industrial production scheduling

Criteria	*SARSA*	*Q-learning*	*DRL (e.g., DQN/PPO)*
Policy type	On-policy (learns from actual actions)	Off-policy (learns from optimal actions)	Off-policy with neural networks
Computational efficiency	Lightweight Q-table updates	Moderate Q-table updates	High computational cost (GPU/TPU required)
Interpretability	Transparent Q-table structure	Transparent Q-table structure	Black-box neural networks
Sparse reward handling	Stable via on-policy updates	Prone to overestimation	Struggles without dense reward signals
Real-time decision	Fast inference (Q-table lookup)	Fast inference (Q-table lookup)	Slower due to neural network inference
Safety in industrial use	Explores safely with on-policy updates	Risk of overcommitting to untested actions	Unpredictable exploration risks

The key gaps in the research area are identified as follows:

1 capacity constraints ignorance: traditional MPS methods frequently assume infinite capacity, leading to infeasible plans

2 knowledge reuse deficiency: heuristic and evolutionary algorithms require repetitive calculations without leveraging historical scheduling data

3 RL limitations at enterprise level: existing RL approaches inadequately model hierarchical demand-production relationships and struggle with sparse rewards

4 DRL practicality barriers: high computational costs and interpretability issues limit DRL's applicability to MPS despite theoretical advantages.

Considering SARSA's advantages such as relatively low computational complexity, better interpretability, and its robustness in handling sparse rewards (as summarised in Table 1), it is well suited for industrial scenarios and the master production scheduling problem (MPSP). Based on these strengths, we select SARSA as the foundational algorithm for this study. Therefore, we propose a master production scheduling with improved SARSA algorithm (MPS-ISARSA).

The main contributions in this research work involve:

1 Application of RL to MPS with limited capacity: traditional MPS methods often rely on manual scheduling or heuristics, which can be limited in adaptability and scalability. This study applies RL to the MPSP under finite capacity constraints. Unlike traditional methods that commonly assume infinite capacity, our RL approach incorporates production limitations, ensuring scheduling decisions align with actual capacity, leading to feasible production plans. RL provides a data-driven approach, learning from historical data and adapting to changes, offering a solution for capacity-constrained scheduling.

2 Development of a Markov decision process (MDP) model for MPS: this research formulates the MPSP as an MDP to enable the application of RL. An MDP model is developed, featuring a state space that represents key production variables and an action space that encompasses relevant scheduling decisions. By explicitly defining the state and action spaces, this MDP provides a framework for the RL agent to learn scheduling policies through interactions with the environment, thereby understanding the relationships between states, actions, and their consequences.

3 Enhanced reward mechanism via reward shaping: sparse rewards are a known challenge in RL, particularly in complex scheduling. This study addresses this through reward shaping, incorporating intermediate rewards to guide learning. This shaping encourages exploration while maintaining focus on minimising overtime and meeting deadlines. An overtime penalty factor prevents production from exceeding capacity. The shaped reward function provides denser feedback signals, accelerating training convergence and improving scheduling efficiency.

4 Implementation of a linearly decaying epsilon-greedy (ε_{lin}-greedy) policy: the exploration-exploitation trade-off is a fundamental consideration in RL. This study utilises a ε_{lin}-greedy policy. Initially prioritising exploration, this policy promotes the discovery of effective scheduling policies. As training progresses, the policy gradually shifts towards exploiting learned knowledge, refining decisions based on accumulated experience. This adaptive approach contributes to faster convergence and stable learning.

The remainder of the paper is structured as follows: Section 2 defines the MPSP and presents the underlying assumptions, objective function, and constraints. Section 3 details the proposed methodology, including the MDP model and SARSA algorithm enhancements. Section 4 presents simulation experiments and performance analysis. Finally, Section 5 concludes the study, highlighting its contributions and suggesting future research directions.

2 MPSP description

The master production scheduling process is typically managed by the PC department within an enterprise. The PC department receives the monthly demand plan from the S&OP, which includes the number of demand orders per week, required quantities, and delivery dates.

In the supply chain, production is organised by the smallest task unit, the stock keeping unit (SKU), which contains product type and quantity information and serves as the basic unit of production management. Based on demand orders, the PC department calculates the types and quantities of SKUs to be produced weekly. Production time information for each SKU is obtained from the production line to estimate the minimum and maximum number of SKUs that can be produced daily for each demand order.

Figure 1 Overview of the demand and production planning workflow (see online version for colours)

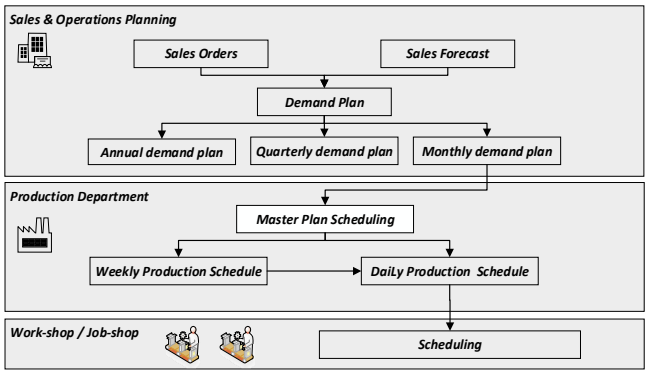
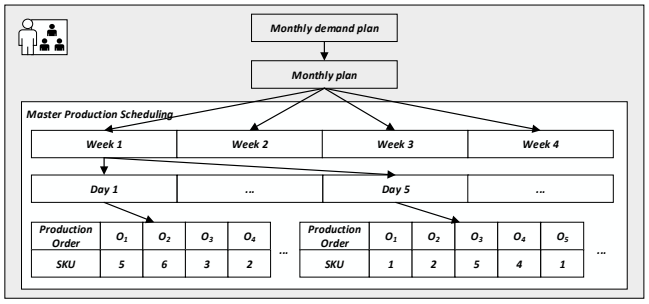


Figure 2 Example of a MPS result



The production planning manager then considers the planning cycle, daily working hours, and allowable overtime from the work calendar to allocate and schedule demand orders across each day of the planning cycle. This constitutes the master production scheduling process. The resulting daily production plan is forwarded to the workshop for detailed scheduling. An effective MPS aims to balance production tasks evenly throughout the week to minimise excessive overtime on any given day.

This scheduling challenge is formally defined as the MPSP, which involves determining the optimal allocation of SKUs to daily production schedules while satisfying demand and operational constraints.

2.1 MDP for MPS

In the master production scheduling process, the production planning manager selects a number of SKUs from each demand order to include in the daily production plan. After each selection action, the scheduled quantity of each demand order is updated, reflecting the current state. The choice of which demand order to prioritise and how many SKUs to schedule at each step is stochastic. Each action transitions the environment to a new state, and this process repeats until a terminal state is reached, producing a one-week MPS. Therefore, this scheduling process forms a complete MDP. The state space, initial state, terminal state, action space, reward function, and action selection policy of this MDP are defined below.

2.2 Problem definition and key variables

We are given a planning cycle with a duration of D days. i^{th} day in the planning cycle is represented as d_i. Additionally, there is a set of demand orders with a total number of demand orders denoted by O. Each demand order is indexed by j, where j^{th} demand order is represented as o_j, $j \in \{1, \ldots, O\}$.

In addition, the following variables and set, as detailed in Tables 2 and 3, are defined to facilitate a detailed elaboration of the MPSP under consideration.

Table 2 Definitions of variables used in the MPSP

Variables	*Description*
a_c	The action chosen by the agent, $c \in [1, AC]$.
AC	The length of the action code.
BNH	The maximum number of SKUs that can be scheduled per demand order per day.
BNL	The minimum number of SKUs that can be scheduled per demand order per day.
BS_j	The minimum batch size for demand order o_j.
cn_i	Time required to complete the plan for day d_i in MPS, unit in hours.
D	Days of planning cycle.
d_i	The i^{th} day in the planning cycle, $i \in \{1, \ldots, D\}$.
$decay\_ratio$	The ratio of decayed episodes to total episodes.
$episodes$	Cumulative number of episodes run by the algorithm.
ε_{lin}	The linear decay factor of ε_{lin}-greedy policy.
i	Index of day in planning cycle, $i \in \{1, \ldots, D\}$.
$init\_\varepsilon$	The initial value of the linear decay factor ε_{lin}.
j	Index of demand order, $j \in \{1, \ldots, O\}$.
k_C	The count of times action a_c is chosen.
$MDOT$	Maximum of overtime per day, unit in hours.
$MDWT$	Maximum daily working time for production within an enterprise, unit in hours.

Table 2 Definitions of variables used in the MPSP (continued)

Variables	*Description*
min_ε	Minimum value of the linear decay factor ε_{lin}.
MT_i	The maximum machine operating time on day d_i, aligned with *MDWT*.
n_j	The number of SKUs needed to complete o_j, $j \in \{1, \ldots, O\}$.
na_j	The average number of SKUs that need to be completed per day during the planning cycle for a demand order o_j.
n_episodes	The total number of episodes set for the algorithm.
o_j	The j^{th} demand order in the demand plan, $j \in \{1, \ldots, O\}$.
O	Number of demand orders included in the demand plan.
ot_i	The overtime on day d_i, $ot_i \le MDOT$, $i \in \{1, \ldots, D\}$, unit in hours.
OTT	The cumulative overtime within the planning cycle, unit in hours.
$para_{low}$	The lower bound parameter of the step reward.
$para_{up}$	The upper bound parameter of the step reward.
$r(a_C)$	The step reward when the agent chooses action a_C
r_T	The terminal reward.
s_t	The t^{th} state of agent in an episode.
$SDWT$	Standard daily working time for production within an enterprise, unit in hours.
$SKUS_j$	The order o_j has accumulated the number of scheduled SKUs, when the agent is in state s_t.
$sn_{i,j}$	Number of SKU selected for demand order o_j on day d_i, $i \in \{1, \ldots, D\}, j \in \{1, \ldots, O\}$.
$sn(a_C)$	The number of SKUs that are scheduled when the agent chooses the action a_C.
t	Time step index of agent's state.
t_{SKU}	The time required to complete a SKU, unit in hours.
td_j	Delivery date for demand order o_j, $j \in \{1, \ldots, O\}$.
tdp_j	Scheduled delivery dates for demand order o_j, $j \in \{1, \ldots, O\}$.

Table 3 Definitions of sets used in the MPSP

Set	*Description*
A	The action space set of the algorithm.
CN	Daily production durations within the MPS.
N	The number of SKUs needed to complete all demand orders.
S	The state space set of the algorithm.

2.3 Assumptions underpinning the MPSP

The start date for a demand order cannot be earlier than the date agreed upon with the customer; scheduling starts earlier than the start of production; during the production cycle, it is assumed that all production materials will be delivered in a timely manner; daily tasks for each demand order, once started, must continue until completion; no changes will be made to the production line during the planning cycle; production tasks are scheduled without considering unforeseen circumstances (e.g., power outages, machine breakdowns, component returns, accidents).

2.4 Optimisation objective for MPSP

MPS focuses on strategic allocation and balancing of production capacity, whereas job shop scheduling emphasises the completion times of individual orders. PC department typically receives demand plans from S&OP, which have already accounted for the alignment between the enterprise's demand capacity and its available production capacity. Demand capacity encompasses both customer order data and forecasts of future market sales. A primary objective of MPS is to optimise the utilisation of the company's production capacity, thereby avoiding idle or overburdened production resources and ensuring the efficient use of equipment and personnel. MPS focuses more RL on the overall allocation and balance of capacity rather than the completion times of individual orders. Therefore, overtime during the planning period is used as the objective function for the MPSP.

The production time in the MPS as CN:

$$CN = \{cn_1, \ldots, cn_i, \ldots, cn_D\} \tag{1}$$

$$cn_i = \sum_{j=1}^{O} sn_{i,j} \times t_{SUK} \tag{2}$$

The overtime ot_i on d_i are defined as follows:

$$ot_i = \begin{cases} cn_i - SDWT & \\ \quad = \sum_{j=1}^{O} sn_{i,j} \times t_{SUK} - SDWT & cn_i \ge SDWT \\ 0 & cn_i < SDWT \end{cases} \tag{3}$$

The cumulative overtime OTT within the planning cycle is given by:

$$OTT = \sum_{i=1}^{D} ot_i \tag{4}$$

2.5 Constraints in MPSP

1 Daily working time constraints:

$$t_{SKU} \times \sum_{j=1}^{O} sn_{i,j} \le SDWT + MDOT \tag{5}$$

Each day, the SKUs scheduled for any demand order must be between the minimum and maximum allowed quantities.

2 Maximum daily production constraints:

$$BNL \le sn_{i,j} \le BNH \tag{6}$$

Each day, for any demand order, the number of SKUs scheduled must be greater than or equal to the minimum quantity and less than or equal to the maximum quantity.

3 Delivery date constraints:

$$tdp_j \le td_j \tag{7}$$

The scheduled delivery date should be earlier than the delivery date for demand order.

4 Demand order completion constraints:

$$\sum_{i=1}^{tdp_j} sn_{i,j} \ge n_j \tag{8}$$

The number of SKUs completed by the planned delivery date must be greater than or equal to the total number of SKUs required by the demand order.

5 Machine availability constraint:

$$MT_i = SDWT + MDOT \tag{9}$$

$$\sum_{j=1}^{O} sn_{i,j} \times t_{SUK} \le MT_i \tag{10}$$

All machines are assumed available, with daily machine operating time equal to the enterprise's maximum working hours. The total production time on each day must not exceed this limit.

6 Batch size constraint:

$$BS_j = SKU \tag{11}$$

The minimum batch size for each demand order is one SKU.

3 MPS-ISARSA description

RL, a major branch of machine learning, is based on an agent that takes feedback information from the environment as input. Agents in the RL approach mimic biological learning processes. In an environment, behaviours that are 'rewarded' are reinforced, while those that are 'punished' are weakened, allowing the agent to learn to choose desired behaviours without relying on preparatory knowledge or expert assistance. The SARSA algorithm, proposed by Rummery and Niranjan in 1994, is similar to Q-learning. It adopts the form of a Q-table to store the value function Q of an action and uses this value function to make decisions, so it is a value-based RL method. However, the SARSA updating policy differs from Q-learning in that it uses on-policy updating, meaning the action is selected first and then the value function is updated accordingly.

The core of the master production scheduling process lies in the dynamic selection of daily processing demand orders and the number of SKUs. This dynamic selection necessitates continual strategy adjustments within a variable environment to address diverse production demands and resource constraints. SARSA, an on-policy algorithm, ensures that during the learning process, actions are consistently chosen based on the current policy, maintaining policy consistency at each decision step within an episode. Additionally, the master production scheduling process is often directly influenced by the selection of actions. For example, choosing a specific demand order may impact the subsequent allocation of production resources. This requires the algorithm to consider the effects of the actions taken on the environment during the learning process to adjust strategies. Consequently, SARSA is more suitable for solving the MPSP.

The MPSP is abstracted into a MDP with five elements: $\{s, a, r, s', a'\}$. By selecting an action from the action space $A \to a$, the number of tasks scheduled to be given in a week changes, and the system changes to the next state $s \to s'$, continuing to select the next action in the next action space $A \to a'$, and repeating until the termination state s_T.

3.1 State space

In the context of MPSP, s_t represents the cumulative quantity of each SKU that has been scheduled for every demand order up to time step t within the planning cycle, as illustrated in equation (12).

$$s_t = \{SKUS_1, \ldots, SKUS_j, \ldots, SKUS_O\} \tag{12}$$

$$SKUS_j = \sum_{i=1}^{D} sn_{i,j} \tag{13}$$

The state space S is composed of all states s_t.

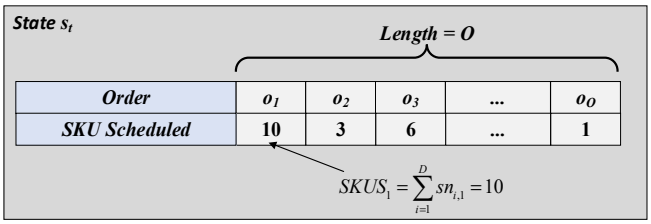
Initial state s_0: the condition where no SKUs are scheduled for any demand order, meaning the scheduled quantity of all tasks for all orders is zero, as illustrated in equation (14).

$$\begin{aligned} s_0 &= \{SKUS_{1,0}, \ldots, SKUS_{j,0}, \ldots, SKUS_{O,0}\} \\ &= \{0, \ldots, 0, \ldots, 0\} \end{aligned} \tag{14}$$

Terminated state s_T: the condition where all tasks for all demand orders have been scheduled, as illustrated in equation (15).

$$\begin{aligned} s_T &= \{SKUS_{1,T}, \ldots, SKUS_{j,T}, \ \ldots, SKUS_{O,T}\} \\ &= \{n_1, \ldots, n_j, \ldots, n_O\} \end{aligned} \tag{15}$$

Figure 3 Representation of state (s_t) in the state space for MPSP (see online version for colours)



3.2 Action space

The action space is discrete, as illustrated in Figure 4. Each action in action space is represented by a one-hot encoded vector of length AC, i.e., $AC = O \times BNH$, where a_C indicates the selected action. This selected action encompasses both the chosen demand order and the number of SKUs scheduled. Specifically, $a_C = a_4$ indicates the scheduling of one SKU to demand order o_2.

We denote the action space by A:

$$A = \{a_1, a_2, \ldots, a_c, \ldots, a_{AC}\}, c \in [1, AC] \tag{16}$$

Additionally, we interpret the selection count k_C of a selected action a_C as the selected day d_i when that action is chosen. For instance, when action $a_C = a_4$ is selected for the third time, i.e., $k_4 = 3$, it signifies that on day 3, the scheduling of one SKU to demand order o_2 is executed. A detailed explanation of this process is provided in the algorithmic steps described in Section 3.5.

3.3 Reward design

In the SARSA algorithm, reaching the terminal state and generating the 'feasible master production scheduling result' is a rare occurrence. Consequently, the agent typically needs to undertake extensive exploration to achieve substantial rewards. In other words, the agent can only reach the terminal state, where the 'feasible master production scheduling result' is generated, at the final step of each episode, thereby receiving the 'rare reward'. If the agent receives zero rewards for all experienced states during exploration, it lacks the incentive to explore further. This results in the agent finding it difficult to receive frequent and useful feedback, leading to slow and inefficient model training. Therefore, it is necessary to design additional rewards at non-terminal steps to motivate the agent during the exploration process. Moreover, if a demand order is scheduled for too many SKUs, it may cause insufficient capacity to schedule other demand orders in the master production scheduling process. Conversely, scheduling too few SKUs may prevent demand orders from being fulfilled in a timely manner. Therefore, reward shaping is employed to effectively design the reward function.

- *Terminal reward*

  The overtime factor β represents the ratio of cumulative overtime to the maximum allowable overtime per day, it is obtained as follows:

$$\beta = \frac{OTT}{MDOT \times D} = \frac{\sum_{i=1}^{D} ot_i}{MDOT \times D} \tag{17}$$

  The terminal reward r_T is achieved based on the overtime factor β through equation (18):

$$r_T = 10 \times (1 - \beta) = 10 \times \left(1 - \frac{\sum_{i=1}^{D} ot_i}{MDOT \times D}\right) \tag{18}$$

- *Step reward*

  The average number of SKUs that need to be completed per day for a demand order o_j during the planning cycle, denoted as na_j, is obtained as follows:

$$na_j = \frac{n_j}{td_j} \tag{19}$$

  The step reward $r(a_c)$, which depends on na_j from equation (19), is obtained as follows:

$$r(a_c) = \begin{cases} -0.1 & sn(a_C) \le na_j \times (1 + para_{low}) \\ 0.1 & na_j \times (1 + para_{low}) < sn(a_C) \le na_j \times (1 + para_{up}) \\ -0.1 & sn(a_C) > na_j \times (1 + para_{up}) \end{cases} \tag{20}$$

  The step rewards are configured with values of ±0.1, significantly smaller in magnitude than the terminal reward. This ensures that the agent prioritises task completion rather than merely optimising local step rewards. Additionally, the small absolute values of the step rewards prevent the accumulation of multi-step rewards from quickly outweighing the influence of the terminal reward.

During the training process, each time the agent executes an action a_C, the Q-table is updated using the step reward $r(a_C)$. When the agent reaches the terminated state, both the step reward $r(a_C)$ and the terminal reward r_T are used to update the Q-table.

Figure 4 Illustration of the action space for MPSP (see online version for colours)

Action Space											
Demand Order	***SKU***	a_1	a_2	a_3	a_4	a_5	a_6	…	a_{AC-2}	a_{AC-1}	a_{AC}
o_1	1	1	0	0	0	0	0	…	0	0	0
	2	0	1	0	0	0	0	…	0	0	0
	3	0	0	1	0	0	0	…	0	0	0
o_2	1	0	0	0	1	0	0	…	0	0	0
	2	0	0	0	0	1	0	…	0	0	0
	3	0	0	0	0	0	1	…	0	0	0
	…	…	…	…	…	…	…		…	…	…
	…	…	…	…	…	…	…		…	…	…
	…	…	…	…	…	…	…		…	…	…
o_j	1	0	0	0	0	0	0	…	1	0	0
	2	0	0	0	0	0	0	…	0	1	0
	3	0	0	0	0	0	0	…	0	0	1

3.4 Action selection policy

In the initial episodes, it is crucial for the agent to thoroughly explore the environment, as its knowledge of the action-value function is still limited. As training progresses and the agent's estimation of the action-value function improves, a shift towards exploiting the acquired knowledge becomes desirable. To balance exploration and exploitation efficiently, our algorithm adopts an ε_{lin}-greedy policy for action selection. Specifically, the value of ε is initialised to 1 and then linearly decayed with each episode, gradually reducing the probability of random action selection as the agent gains experience.

For comparison, the standard epsilon-greedy (ε-greedy) policy maintains ε at a fixed value throughout training, which may lead to either excessive exploration or premature exploitation depending on the value chosen. By contrast, the ε_{lin}-greedy policy approach allows the agent to better adapt its exploration-exploitation trade-off over time.

The linear decay factor ε_{lin} of ε_{lin}-greedy policy is got as shown in equation (21):

$$\varepsilon_{\text{lin}} = \begin{cases} \left[1 - \dfrac{episodes}{n\_episodes \times decay\_ratio}\right] \\ \times (init\_\varepsilon - min\_\varepsilon) + min\_\varepsilon, \\ \qquad episodes \le decay\_episodes \\ min\_\varepsilon, \\ \qquad episodes > decay\_episodes \end{cases} \tag{21}$$

The variable *decay_episodes*, which is utilised in equation (21), is computed as shown in equation (22):

$$decay\_episodes = n\_episodes \times decay\_ratio \tag{22}$$

The action selection policy π^* is given by:

$$\pi^* = \begin{cases} rand(s_t) & \varepsilon_{\text{lin}} \\ \arg\max V^{\pi}(s_t) & 1 - \varepsilon_{\text{lin}} \end{cases} \tag{23}$$

In state s_t, the agent explores with probability ε_{lin} by selecting a random action and exploits with probability 1-ε_{lin} by selecting the action with the highest value.

3.5 Procedure for MPS-ISARSA

1 Obtain scheduling information and initialise the MPSP environment parameters. The production information includes the planning cycle of the MPS, the daily working hours of the production line, and the maximum allowable overtime. Proceed to step 2.

2 Set the parameters for the MPS-ISARSA. These include the decay ratio, learning rate, number of episodes, and the upper and lower bound parameters of the step reward. Proceed to step 3.

3 Define the state and action spaces for the MPSP, and initialise the Q-table based on the demand orders scheduled within a week, as outlined in the monthly demand plan provided by S&OP. Proceed to step 4.

4 Reset environment state $s \rightarrow s_0$. Proceed to step 5.

5 Determine the current number of episodes and check whether it has reached the *decay_episodes*. If not, proceed to step 6; otherwise, proceed to step 7.

6 Calculate the linear decay factor ε_{lin}, apply it to update the ε_{lin}-greedy policy, and then proceed to step 8.

7 Use the minimum factor *min_ε*, update the ε_{lin}-greedy policy. Proceed to step 8.

8 $\varepsilon_{\text{line}}$-greedy is employed to select an action a_C within the action space. And the selection count for the action is updated as $k'_C = k_C + 1$. Proceed to step 9.

9 Incorporate the demand orders and the corresponding quantities of scheduled SKUs, as determined by action a_C, into the MPS for day k_C, resulting in the updated state s'. Proceed to step 10.

10 Judge whether state s' exists in Q-table, if not, proceed to step 11, otherwise, proceed to step 12.

11 Record the state s' in Q-table, proceed to step 12.

12 Judge whether state s' satisfies the condition of 'terminated state'. If not, execute step 13, otherwise, execute step 14.

13 Calculate step reward r and update Q-table, $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$, $s \leftarrow s'$, $a \leftarrow a'$. Proceed to step 8.

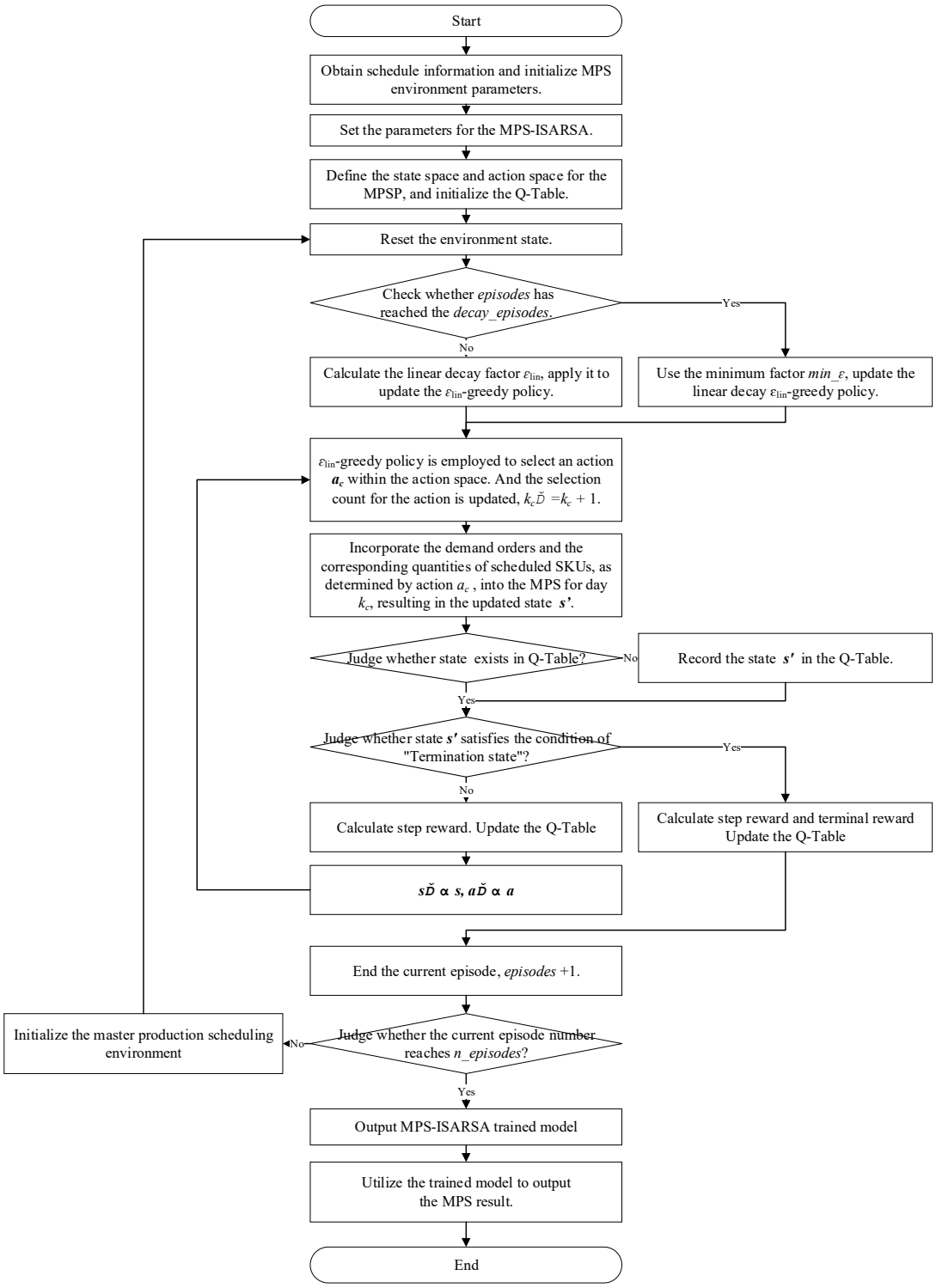
14 Calculate step reward r and terminal reward r_T, then update Q-table, $Q(s, a) \leftarrow Q(s, a) + \alpha[r + r_T + \gamma Q(s', a') - Q(s, a)]$. Proceed to step 15.

15 End the current episode. Judge whether the current episode number reaches *n_episodes*. If not reach the preset value, the number of *episodes* + 1, return to step 3, otherwise, proceed to step 16.

16 Output MPS-ISARSA trained model. Proceed to step 17.

17 Utilise the trained model to output the MPS result.

Figure 5 Flowchart of the MPS-ISARSA for MPSP



4 Simulation and performance evaluation

To verify the effectiveness of the MPS-ISARSA for solving the MPSP, we evaluated the convergence and optimisation performance of the algorithm using accumulated overtime as the evaluation index. The simulation was tested using a monthly demand plan case, with the weekly demand orders shown in Table 4. All simulation experiments were conducted on a system with an Intel(R) Core (TM) i7-9750H CPU @ 2.60GHz, 16 GB RAM, running Windows 10 64-bit, in a Python 3.11 programming environment.

4.1 Simulation data construction

A manufacturing enterprise operates a production line 5 days a week for 8 hours per day, with a maximum of 2 hours of overtime per day. Each day, 1 to 3 SKUs can be put into production per demand order, with each SKU taking 1 hour to complete.

The PC department receives the monthly demand plan from S&OP. The monthly demand plan includes weekly demand orders specifying the number of SKUs to be produced, delivery dates, and lead times, which also represents the earliest production start date.

Table 4 Monthly demand plan information

		Demand orders	*SKU*	*Lead time*	*Delivery date*
Monthly demand plan	Week 1	$o_{1,1}$	4	0	4
		$o_{1,2}$	5	1	5
		$o_{1,3}$	5	0	5
		$o_{1,4}$	5	0	5
		$o_{1,5}$	6	0	5
		$o_{1,6}$	6	0	5
	Week 2	$o_{2,1}$	6	0	5
		$o_{2,2}$	8	0	5
		$o_{2,3}$	9	0	5
		$o_{2,4}$	10	0	5
	Week 3	$o_{3,1}$	5	1	3
		$o_{3,2}$	7	1	4
		$o_{3,3}$	13	0	5
		$o_{3,4}$	5	0	5
		$o_{3,5}$	6	0	5
	Week 4	$o_{4,1}$	5	0	2
		$o_{4,2}$	5	0	3
		$o_{4,3}$	6	0	3
		$o_{4,4}$	6	0	3

4.2 Valuation indicators

The objective is to minimise accumulated overtime while fulfilling all production tasks specified in the weekly demand plan. Therefore, cumulative overtime $OTT = \sum_{i=1}^{D} ot_i$ is used as the evaluation indicator.

4.3 Convergence and effectiveness analysis

As an example, the weekly demand plan for the week 4 of the monthly demand plan contains four demand orders $\{o_{4,1}, o_{4,2}, o_{4,3}, o_{4,4}\}$. The SKUs contained in these demand orders are {5, 5, 6, 6}. The lead time of these demand orders are {0, 0, 0, 0}. The delivery dates are {2, 3, 3, 3}.

Figure 6 shows the probability of generating feasible MPS by MPS-ISARSA during training.

The probability P_o is defined as $P_o = \frac{Num_o}{100}$, where Num_o indicates the number of times the MPS results were achieved in every 100 episodes.

The horizontal axis represents the number of episodes, while the vertical axis indicates the probability P_o. The algorithm runs for a total of 1.0×10^5 episodes:

As shown in Figure 6, the probability that the agent successfully generates a feasible MPS result increases progressively throughout the training process. Around 6,000 episodes, this probability experiences a sharp rise to nearly 0.6, after which it stabilises, indicating convergence in the agent's performance.

Figure 6 Probability of generating feasible master production scheduling results averaged over every 100 episodes during the training process (see online version for colours)

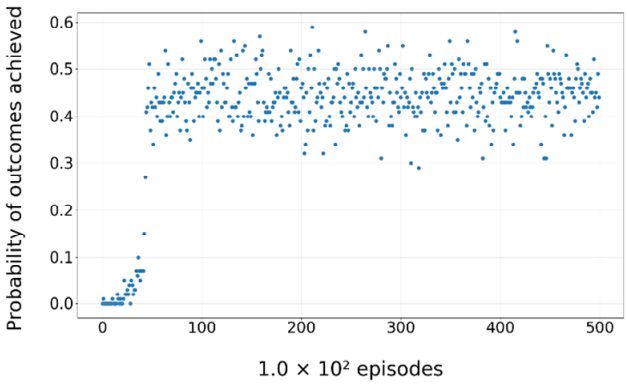


Figure 7 illustrates the relationship between the average overtime and the number of training episodes, considering only episodes that produce feasible MPS results. The horizontal axis represents the number of training episodes, while the vertical axis represents the average overtime. Figure 8 depicts the relationship between the average reward and the number of training episodes. The horizontal axis represents the number of episodes, while the vertical axis corresponds to the average reward achieved.

Figure 7 Relationship between average overtime and the number of training episodes (see online version for colours)

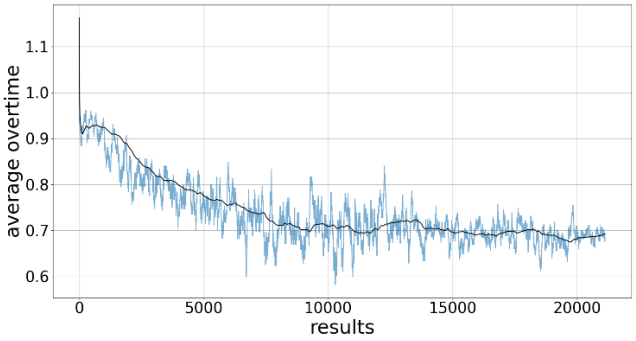
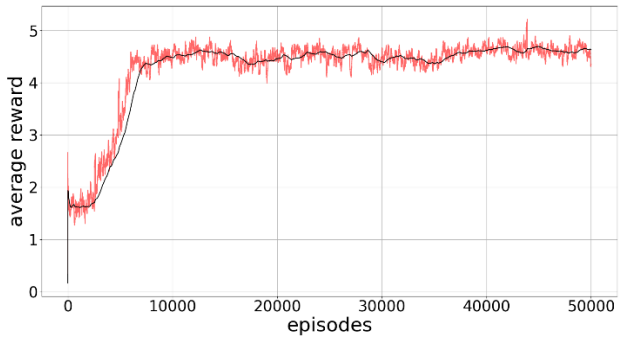


Figure 8 Relationship between average reward and the number of training episodes (see online version for colours)



As seen in Figure 7, the average number of overtime hours decreases as the number of episodes increases. In Figure 8, at the beginning of the training period, the agent does not explore sufficiently and lacks enough experience. However, as the agent explores the state space and undergoes training, the average reward gradually stabilises. By 10,000 episodes, the average reward stabilises at 4.83, and the average overtime is 0.66.

The training results for the data from the other three weeks of the demand plan are also shown in Figure 9, all of which yielded better outcomes. The average overtime stabilises at [0.59, 0.78, 0.71] and the average reward stabilises at [8.33, 13.87, 13.55], demonstrating that the algorithm converges effectively.

4.4 Comparative analysis

4.4.1 Policy comparison

We compare two action selection policies using the demand order data from the fourth week of the demand plan: ε-greedy policy with $\varepsilon = 0.01$ and the ε_{lin}-greedy policy with *decay_ratio* = 0.1, *init_ε* = 1 and *min_ε* = 0.01. Both policies were trained for 50,000 episodes, with results shown in Figure 10.

In Figure 10, the ε-greedy policy maintains a fixed ε value of 0.01, balancing exploration and exploitation throughout the training. On the other hand, the ε_{lin}-greedy policy starts with $\varepsilon = 1$ and gradually reduces it to 0.01, emphasising exploration in the early stages and shifting towards exploitation as training progresses. Consequently, the ε_{lin}-greedy policy achieves a higher final average reward of 4.83 compared to 3.98 for the ε-greedy policy, demonstrating its superior exploration and policy discovery capabilities.

Figure 9 Training results using demand planning data from weeks 1 to 3 (see online version for colours)

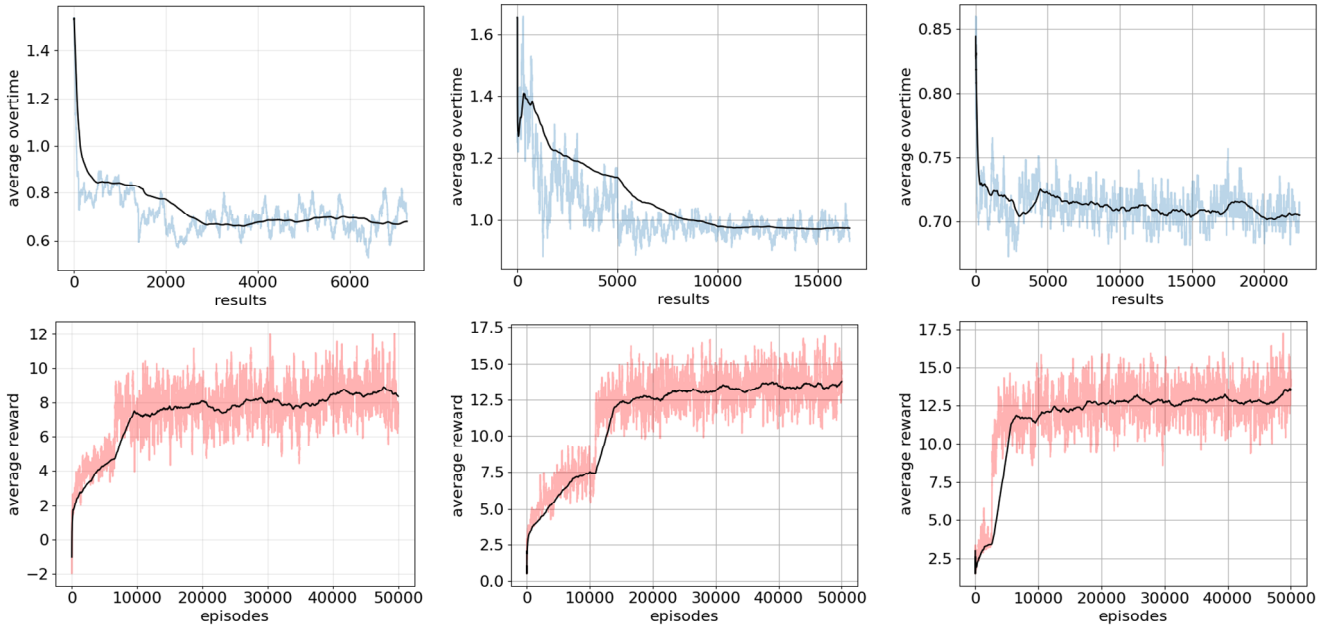
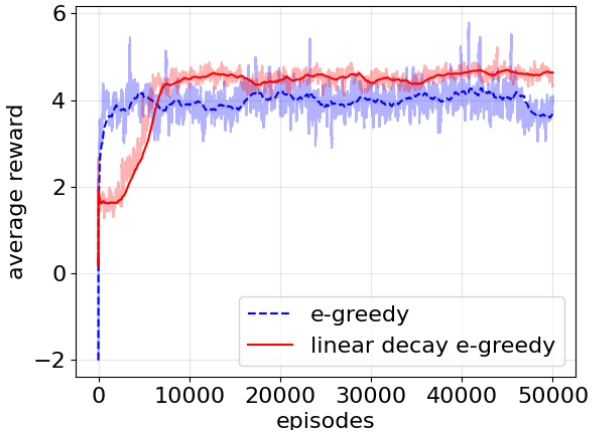


Figure 10 Comparison of average rewards between different action selection policies (see online version for colours)



4.4.2 Reward parameters comparison

Three sets of reward boundary parameters were used to compare the training. The three sets of parameters are:

$$(para_{up}, para_{low}) = \{(-0.1, 0.1), (-0.2, 0.2), (-0.5, 0.5)\}$$

The comparison results are shown as Figure 11.

From Figure 11, we can observe that the reward boundary parameter setting significantly influences the agent's learning process. When the boundary parameter is set to $(para_{up}, para_{low})$ = (–0.1, 0.1), the average reward curve rises more slowly and fluctuates more, making it difficult for the selected actions to yield positive rewards. When the boundary parameter is set to $(para_{up}, para_{low})$ = (–0.5, 0.5), the reward curve rises faster, but the final average reward is smaller because many actions in the action space can earn positive rewards, which may dilute the overall effectiveness. Conversely, when the boundary parameter is set to $(para_{up}, para_{low})$ = (–0.2, 0.2), the average reward stabilises more quickly, convergence is faster, and the average reward is the highest.

Figure 11 Comparison of average rewards obtained for different boundary parameters (see online version for colours)

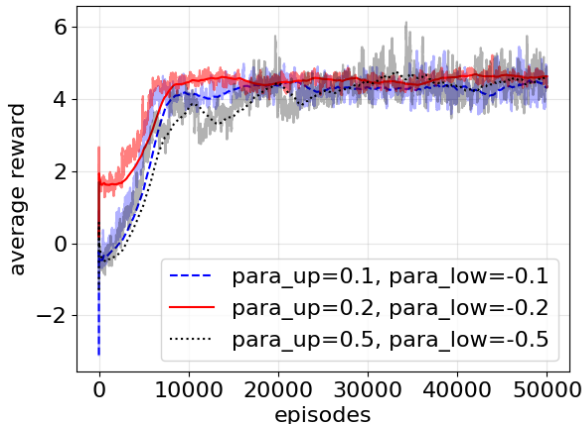


Table 5 Simulation schemes information

	Algorithm	*Policy*	*Reward shaping*
Scheme 1	Q-learning	ε-greedy	No
Scheme 2	SARSA	ε-greedy	No
Scheme 3	SARSA	ε-greedy	Yes
Scheme 4	SARSA	ε_{lin}-greedy	No
Scheme 5	SARSA	ε_{lin}-greedy	Yes

Table 6 Simulation results of five groups of schemes

	Week 1			*Week 2*		
	Number of feasible MPS results generated	*Average reward*	*Average overtime*	*Number of feasible MPS results generated*	*Average reward*	*Average overtime*
Scheme 1	3,560	6.25	0.66	9,785	9.56	0.93
Scheme 2	5,167	6.85	0.6	12,357	9.61	0.85
Scheme 3	7,151	7.93	0.61	15,997	10.58	0.82
Scheme 4	6,336	7.94	0.6	12,685	11.09	0.78
Scheme 5	7,245	8.37	0.59	16,588	13.79	0.78
	Week 3			*Week 4*		
	Number of feasible MPS results generated	*Average reward*	*Average overtime*	*Number of feasible MPS results generated*	*Average reward*	*Average overtime*
Scheme 1	10,356	9.63	0.81	11,374	4.01	0.81
Scheme 2	16,552	9.77	0.8	18,579	4.11	0.79
Scheme 3	21,151	11.33	0.77	23,985	3.98	0.75
Scheme 4	17,005	13.01	0.72	19,856	4.65	0.67
Scheme 5	22,480	13.55	0.71	25,625	4.65	0.66

4.4.3 Comparison with other RL algorithms

To further illustrate the performance of MPS-ISARSA, we established five comparison simulation schemes, as shown in Table 5. The data from Table 4 is utilised, and all simulations were trained for 50,000 episodes. The reward function's boundary parameters are set to ($para_{up}$, $para_{low}$) = (–0.2, 0.2), and the linearly decaying ratio is set to 0.1. The comparison metrics include the total number of feasible MPS results generated during the 50,000 training episodes, the average reward calculated over the last 1,000 episodes, and the average overtime of feasible solutions during the same period, as presented in Table 6.

Since the Q-learning algorithm is off-policy, it has separate action selection and target policies, which makes it less suitable for the master production scheduling process. Consequently, the number of feasible MPS results generated is lower than that of the SARSA algorithm. After 50,000 training episodes, the average rewards are [6.25, 9.56, 9.63, 4.01], which are the lowest among all schemes.

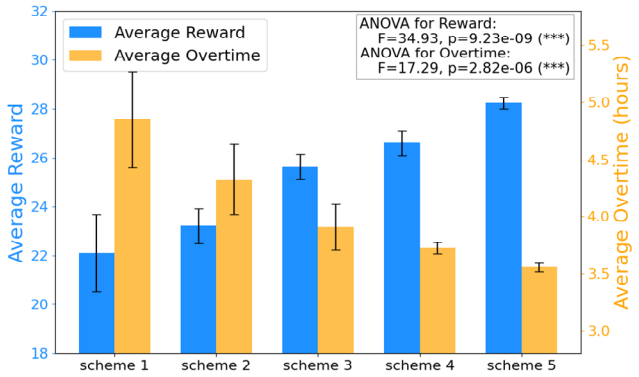
Comparing schemes 2 and 3, both employ the SARSA algorithm, but scheme 3 incorporates reward shaping in the design of the reward function, enabling it to generate more feasible MPS results. Additionally, the average overtime over the four weeks in scheme 3 is less than that in scheme 2 because scheme 3 provides step rewards to the agent based on action selection in each episode, allowing the agent to prioritise better actions and achieve improved outcomes.

Comparing scheme 4 and scheme 2, the ε_{lin}-greedy policy in scheme 4 focuses more on exploration in the early steps, facilitating the discovery of a better behavioural policy. The four-week average rewards for scheme 4 are [6.94, 11.09, 13.01, 4.65], compared to [6.85, 9.61, 9.77, 4.11] for scheme 2, indicating that scheme 4 outperforms scheme 2 in all four weeks.

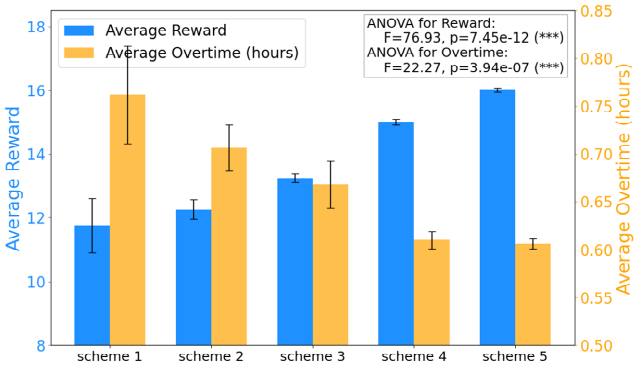
Scheme 5 combines the ε_{lin}-greedy policy and reward shaping. Among the five schemes, it achieves the best performance across all three indicators: the number of feasible MPS results generated, average reward, and average overtime. This verifies the superior performance of the MPS-ISARSA proposed in this paper.

To evaluate how the proposed MPS-ISARSA performs on larger problem sizes, we conducted additional experiments using demand order quantities of 10, 50, and 100, all constrained by a weekly capacity of 50 hours (5 days × 10 hours/day). To generate larger order volumes within the same weekly time frame, we adjusted the production time granularity for each SKU from 1 hour to 10 minutes. Orders were randomly generated with the following conditions: each order randomly between 1 to 10 SKUs (uniform distribution), with 80% of the orders having a delivery date on day 5, and 20% due on day 4. The total number of SKUs did not exceed 300. The performance of schemes 1–5 on larger-scale data for the MPSP is shown in Figure 12.

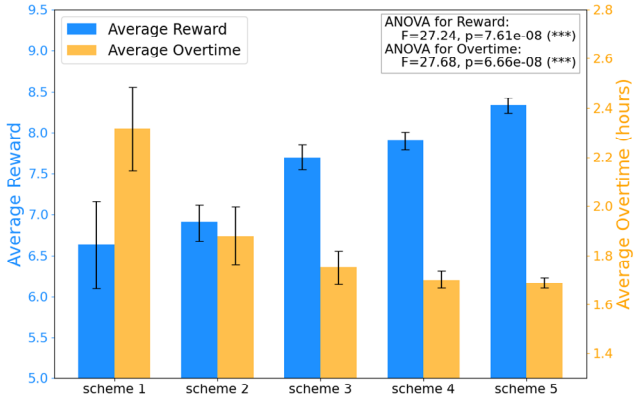
Figure 12 Comparison of average reward and overtime for various demand order quantities (mean of 20 trials), (a) demand order quantity: 10 (b) demand order quantity: 50 (c) demand order quantity: 100 (see online version for colours)



(a)



(b)



(c)

Scheme 5 performs best across all tested demand sizes in terms of average reward and average overtime. For 10 demand orders, scheme 5 achieves an average reward of 16.02 ± 0.06 and an average overtime of 0.61 ± 0.01 hours, which are better than all other schemes. At 50 demand orders, scheme 5 still leads with an average reward of 8.33 ± 0.19 and overtime of 1.69 ± 0.02 hours, outperforming scheme 4. When the demand order quantity increases to 100, scheme 5 obtains the highest average reward of 28.24 ± 0.24 and the lowest overtime of 3.56 ± 0.04 hours, demonstrating good scalability. One-way ANOVA tests

show that these differences are statistically significant ($p < 0.001$) for both reward and overtime at all demand sizes. The low standard deviations indicate stable results.

Overall, scheme 5 effectively maximises rewards while minimising overtime, demonstrating its effectiveness and scalability in solving the MPSP with varying demand orders.

4.4.4 Comparison with evolutionary algorithms

To compare traditional evolutionary algorithms for solving the MPSP, a horizontal comparison was conducted among MPS-ISARSA, GA, and whale optimisation algorithm (WOA), as shown in Table 7. The MPS-ISARSA model was pre-trained for 20,000 episodes with a training time of 38.0241 seconds. GA (population size: 100) and WOA (search agents: 50) both ran for 300 iterations. The comparison uses demand planning data from week 4 (Table 4), focusing on overtime results. Simulation outcomes are presented in Figure 13. The curves are smoothed using a 2-point moving average.

Table 7 Simulation schemes for MPS-ISARSA (trained), GA, and WOA

Optimisation method	*Number of iterations*	*Iteration progress*	*Training time/runtime (seconds)*
MPS-ISARSA (trained)			Training time: 38.0241 Runtime: 0.0119
GA	300	Population: 100	Runtime: 5.1631
WOA	300	Search agents: 50	Runtime: 3.6842

Figure 13 Comparison of overtime reduction for MPS-ISARSA, GA, and WOA (see online version for colours)

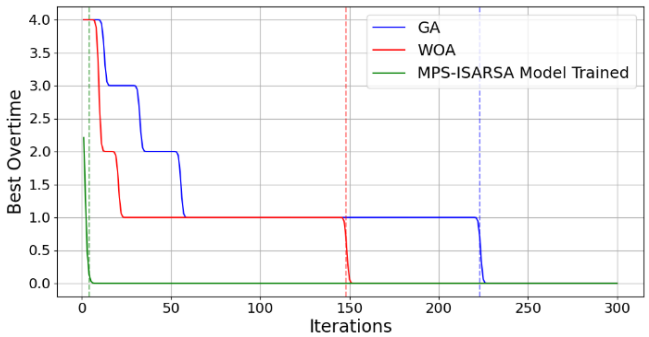


Figure 13 shows that the GA reaches zero best overtime at iteration 223, and the WOA at iteration 148. Compared to traditional evolutionary algorithms, RL approaches for MPSP can effectively leverage historical scheduling experience without the need for complex evolutionary computations. The MPS-ISARSA method achieves zero best overtime after only 4 iterations. The runtime to obtain the best zero overtime solution for MPS-ISARSA is 0.0119 seconds, compared to 5.1631 seconds for GA and 3.6842 seconds for WOA (Table 7). These results indicate that MPS-ISARSA requires fewer iterations and less computational time to reach the optimal solution. Although the initial training of MPS-ISARSA takes 38.0241 seconds, the trained model can be reused for fast inference without retraining. This feature makes it suitable for dynamic scheduling scenarios requiring quick responses.

5 Conclusions

In this paper, we propose the MPS-ISARSA to address the MPSP. We model the MPS as a MDP and define the corresponding mathematical model, with cumulative overtime as the optimisation objective. Next, we establish the MPSP environment, design the state space, and specify the initial and terminal states. One-hot encoding is used to define actions. To enhance the reward function, we employ reward shaping, defining both step rewards and the terminal reward, and introduce an overtime factor to adjust the terminal reward. To boost the algorithm's exploration speed during the early training stages, we use the ε_{lin}-greedy policy for action selection, encouraging the agent to focus more on exploration initially.

Simulations have verified the effectiveness of MPS-ISARSA. We conducted multiple simulation schemes to validate the performance of MPS-ISARSA comprehensively.

Building upon this work, future research will focus on enhancing the scalability of the algorithm, to handle more complex production environments. A key focus will be on investigating the incorporation of uncertainty into the model. Furthermore, we plan to conduct field tests and evaluations in industrial environments to validate the practical applicability of MPS-ISARSA.

Acknowledgements

This work was supported by the Liaoning Provincial Basic Applied Research Program, China (2023JH2/101300184).

References

Akhoondi, F. and Lotfi, M.M. (2016) 'A heuristic algorithm for master production scheduling problem with controllable processing times and scenario-based demands', *International Journal of Production Research*, Vol. 54, No. 12, pp.3659–3676, https://doi.org/10.1080/00207543.2015.1125032.

Astantı, R.D. and Ai, T.J. (2024) 'Clustering-based heuristics for aligning master production schedule and delivery schedule', *Management Systems in Production Engineering*, Vol. 32, No. 3, pp.401–408, https://doi.org/10.2478/mspe-2024-0037.

Esteso, A., Peidro, D., Mula, J. and Díaz-Madroñero, M. (2022) 'Reinforcement learning applied to production planning and control', *International Journal of Production Research*, Vol. 61, No. 16, pp.5772–5789, https://doi.org/10.1080/00207543.2022.2104180.

Gahm, C., Dünnwald, B. and Sahamie, R. (2014) 'A multi-criteria master production scheduling approach for special purpose machinery', *International Journal of Production Economics*, Vol. 149, No. 1, pp.89–101, https://doi.org/10.1080/00207543.2015.1125032.

Hubbs, C.D., Li, C., Sahinidis, N.V., Grossmann, I.E. and Wassick, J.M. (2020) 'A deep reinforcement learning approach for chemical production scheduling', *Computers & Chemical Engineering*, Vol. 141, No. 4, p.106982.

Li, C., Zheng, P., Yin, Y., Wang, B. and Wang, L. (2023) 'Deep reinforcement learning in smart manufacturing: a review and prospects', *CIRP Journal of Manufacturing Science and Technology*, Vol. 40, No. 2, pp.75–101.

Maalouf, E., Daaboul, J., Le Duigou, J. et al. (2022) 'Production management for mass customization and smart cellular manufacturing system: NSGAII and SMPSO for factory-level planning', *International Journal of Advanced Manufacturing Technology*, Vol. 120, No. 10, pp.6833–6854.

Momenikorbekandi, A. and Abbod, M. (2023) 'Intelligent scheduling based on reinforcement learning approaches: applying advanced Q-learning and state-action-reward-state-action reinforcement learning models for the optimisation of job shop scheduling problems', *Electronics*, Vol. 12, No. 23, p.4752.

Panwalkar, S.S. and Iskander, W. (1977) 'A survey of scheduling rules', *Operations Research*, Vol. 25, No. 1, pp.45–61.

Rummery, G.A. and Niranjan, M. (1994) 'On-line Q-learning using connectionist systems', *Technical Report*, Vol. 1, No. 1, pp.1–20.

Sadiq, S., Abdulazeez, A. and Haron, H. (2020) 'Solving multi-objective master production schedule problem using memetic algorithm', *Indonesian Journal of Electrical Engineering and Computer Science*, Vol. 18, No. 2, pp.938–945.

Sajja, R., Rao, C.S. and Pavan, K. (2013) 'A study and performance evaluation of evolutionary optimization techniques for multi-objective master production scheduling problems', *Journal of Production Research and Management*, Vol. 3, No. 2, pp.12–22.

Serrano-Ruiz, J., Mula, J. and Poler, R. (2021) 'Smart master production schedule for the supply chain: a conceptual framework', *Computers*, Vol. 10, No. 12, p.156.

Serrano-Ruiz, J.C., Mula, J. and Poler, R. (2024) 'Smart master production scheduling by deep reinforcement learning: an exploratory analysis', *Working Conference on AI and Robotics for Industry 4.0*, Springer, pp.1–15.

Shahrabi, J., Adibi, M.A. and Mahootchi, M. (2017) 'A reinforcement learning approach to parameter estimation in dynamic job shop scheduling', *Computers & Industrial Engineering*, Vol. 110, No. 1, pp.75–82.

Soares, M.M. and Vieira, G.E. (2009) 'A new multi-objective optimization method for master production scheduling problems based on genetic algorithm', *International Journal of Advanced Manufacturing Technology*, Vol. 41, pp.549–567.

Song, H., Gi, I., Ryu, J., Kwon, Y. and Jeong, J. (2023) 'Production planning forecasting system based on M5P algorithms and master data in manufacturing processes', *Applied Sciences*, Vol. 13, No. 13, p.7829.

Vieira, G.E. and Favaretto, F. (2011) 'A new and practical heuristic for master production scheduling creation', *International Journal of Production Research*, Vol. 44, Nos. 18–19, pp.3607–3625.

Wang, Y. (2020) 'Adaptive job shop scheduling strategy based on weighted Q-learning algorithm', *Journal of Intelligent Manufacturing*, Vol. 31, No. 2, pp.417–432.