International Journal of Industrial Engineering Computations 17 (2026) 1835–1850

Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

A genetic algorithm-based approach for solving the production scheduling problem in unrelated parallel machines

Yung-Chia Chang[a], Kuei-Hu Chang[b*], Te-Chi Kong[a] and Austin Lu[c]

[a]*Department of Industrial Engineering and Management, National Yang Ming Chiao Tung University, Hsinchu 300, Taiwan*
[b]*Department of Management Sciences, R.O.C. Military Academy, Kaohsiung 830, Taiwan*
[c]*Department of Computer Science, Cornell University, Ithaca, NY, USA*

C H R O N I C L E

Article history:
Received December 14 2025
Received in Revised Format May 26 2026
Accepted June 29 2026
Available online June 29 2026

Keywords:
Machine eligibility restrictions
Q-learning
Reinforcement learning
Unrelated parallel machine scheduling problem
Variable neighborhood search
Sequence-dependent setup times

A B S T R A C T

The scheduling of unrelated parallel machines (UPM) is a critical aspect of production planning and a major challenge in the manufacturing industry. Scheduling outcomes directly impact a business's profitability and sustainability. In the UPM problem, the processing times for each work order vary across machines and are not proportionally consistent. This becomes a common scheduling challenge for medium- to large-sized enterprises due to its complexity. To address the UPM scheduling problem, this study aims to minimize the makespan while considering sequence-dependent setup times for both machines and work orders. It then proposes a genetic algorithm integrating reinforcement learning and variable neighborhood search. To assess the feasibility and effectiveness of the proposed algorithm, real work orders from a renowned international semiconductor copper foil manufacturer (Company D) are used as test data to assess the proposed method. The results indicate that the proposed method can reduce the makespan by 22.45% compared to other existing methods used by Company D. Additionally, randomly generated test data were used to validate the effectiveness and feasibility of the method.

© 2026 by the authors; licensee Growing Science, Canada

1. Introduction

Production scheduling has been a critical issue in the manufacturing industry, involving the decision-making process of assigning work orders to machines. Efficient scheduling optimizes the allocation and utilization of production line resources, reduces production costs, increases production efficiency, and enhances a company's ability to respond to market demands and gain a competitive advantage. Pinedo (2022) categorized the machine environment for production scheduling into five types, namely single machine, parallel machine, flow shop, job shop, and open shop. Among these, the parallel machine environment consists of multiple machines operating simultaneously and independently within a process. Each work order is processed on a single machine, requiring only one processing step. Parallel machines are further classified into equivalent parallel machines, non-equivalent parallel machines, and unrelated parallel machines (UPM) (Pinedo, 2022). In the UPM case, the processing time for each work order varies across different machines without following a specific proportional relationship. To date, extensive research has addressed the UPM scheduling problem (Xie et al., 2025; Silva et al., 2025; Shafiee et al., 2025; Yan et al., 2025). Since each machine processes multiple work orders, the UPM scheduling problem involves determining the machine on which each work order should be processed and establishing the processing sequence for each machine. Previous studies employed mathematical programming methods and general-purpose heuristic algorithms to obtain approximate optimal solutions. Mathematical programming involves constructing a mixed-integer programming model to solve the problem, typically using the branch-and-bound method (Rocha et al., 2008; Pei et al., 2020). Since the UPM scheduling problem, which aims to minimize the makespan, is NP-hard, the time needed to solve it increases exponentially as the problem grows. Therefore, some studies adopted general-purpose heuristic algorithms, such as variable neighborhood search, tabu search, particle swarm optimization, simulated annealing, genetic algorithm (GA), and ant colony optimization, which are inspired by natural animal behaviors (Wang & Alidaee, 2019; Thevenin & Zufferey, 2019; Bernal et al., 2025; Fallahi et al., 2025). General-purpose heuristic algorithms can provide approximate optimal solutions within a reasonable time frame. However, parameter design and search direction often depend on researchers' understanding of the problem and require extensive testing to determine the most suitable parameters.

* Corresponding author
E-mail evenken2002@yahoo.com.tw (K.H Chang)
ISSN 1923-2934 (Online) - ISSN 1923-2926 (Print)
2026 Growing Science Ltd.
doi: 10.5267/j.ijiec.2026.6.010

Artificial intelligence (AI) has recently driven transformations across various industries. The rapid advancement of hardware and the development of cloud computing, combined with AI's self-learning capabilities, has accelerated its application. Machines have transitioned from merely assisting humans in decision analysis to making decisions that surpass human abilities. Many companies have integrated AI into their supply chains to enhance efficiency and evolving market demands. Currently, one of the most influential topics in AI is machine learning (ML). Sultan and Pande (2018) categorized ML into three types based on the characteristics of the problem's data: supervised learning, unsupervised learning, and reinforcement learning (RL). Among them, RL models consist of two key components: the agent and the environment. The environment provides information to the agent in the form of a state. The agent, as the decision-maker, observes the environment and takes action. Each action generates feedback, referred to as a reward, which can be positive or negative based on the environment. The reward or penalty obtained from the environment is used to evaluate the action. The agent uses this reward or penalty to evaluate its actions and determine the actions that would yield the highest return. RL has been successfully applied to various combinatorial optimization problems, including the traveling salesman problem (Duan et al., 2025; Liu et al., 2025), the vehicle routing problem (Peng et al., 2025), and scheduling problems (Malekshahi & Hoseinpour, 2025; Wang et al., 2025).

Recent studies have also shown that integrating AI with general-purpose heuristic algorithms can improve search efficiency and enhance solution quality. By leveraging RL's flexibility and learning capability, it can guide the parameter selection and search direction of heuristic algorithms. For example, RL methods can determine the crossover and mutation rates in GA or optimize the neighborhood search sequence in variable neighborhood search (VNS) (Chen et al., 2020). This integration allows heuristic algorithms to acquire self-learning capabilities and adapt to different problem environments, addressing the challenge of requiring numerous experiments to determine optimal parameter settings. However, scanty studies applied this approach to solve the UPM scheduling problem. Therefore, this study examines the UPM environment while considering practical production constraints, such as machine eligibility restrictions and sequence-dependent setup times for both machines and jobs, to minimize the makespan. This study aims to incorporate RL to guide the neighborhood search sequence in VNS and integrate it into a GA, so that the algorithm can develop self-learning capabilities and enhance its search performance.

The remainder of this paper is organized as follows: Section 2 is the literature review on the UPM scheduling problem, hybrid general-purpose heuristic algorithms, and the integration of RL with heuristic algorithms. Section 3 describes the research methodology and process. Section 4 presents the implementation and analysis of the results. Finally, Section 5 provides the conclusions and recommendations.

2. Literature Review

2.1 UPM scheduling problem

Pinedo (2022) proposed using three parameters, namely α, β, and γ, to define scheduling problems. The parameter α denotes the machine environment setup information, which has a single setting for a given problem, such as single machine (denoted as 1), job shop (denoted as *J*), equivalent parallel machine (denoted as P_m), or UPM (denoted as R_m). The parameter β denotes the characteristics and related constraints of the production environment, and there can be multiple settings for a given problem. If no production constraints exist, this parameter is left blank. Common characteristics and constraints include product processing order restrictions (denoted as *prec*), sequence-dependent setup times (denoted as s_{ik}), and ready times (denoted as r_j). The parameter γ denotes the scheduling objective, which is typically a minimization problem. Common objective functions include makespan (denoted as C_{max}), total tardiness (denoted as $\sum T_j$), and maximum tardiness (denoted as L_{max}), among others. When a factory acquires new machines to address production bottlenecks or accommodate varying parameter setups, multi-machine scheduling problems arise. Based on how work orders are processed, these problems are classified into parallel machine scheduling, flow shop scheduling, and job shop scheduling. Parallel machine scheduling can be further categorized based on machine processing capabilities: equivalent, non-equivalent, and UPM. The UPM scheduling problem involves assigning N independent work orders to M parallel machines. Due to variations in machine modules or parameter settings, the processing time for each work order varies between machines and is independent. Each machine can process only one work order at a time. Once it starts processing, it must continue uninterrupted until completion. Generally, solutions to the UPM scheduling problem can be divided into mathematical programming methods and general-purpose heuristic algorithms. Mathematical programming methods involve constructing a MIP model to solve the problem, with studies frequently employing branch-and-bound algorithms. For example, Rocha et al. (2008) used a greedy algorithm as an upper bound while considering machine-related and sequence-dependent setup times to minimize the makespan. Given that minimizing makespan in the UPM scheduling problem is NP-hard (Pinedo, 2022), the time required to solve it increases exponentially with the problem size. For large-scale scheduling problems, heuristic or general-purpose heuristic algorithms are often used to obtain near-optimal solutions within a reasonable timeframe. To address the UPM scheduling problem while minimizing the makespan, Balin (2011) compared the performance of a GA to that of the longest processing time dispatching rule, and found that GA provides better solutions for large-scale scheduling problems within shorter timeframes. Similarly, Abreu and Prata (2018) applied GA to scheduling problems with sequence-dependent setup times and compared the results with those obtained using the earliest due date (EDD), shortest processing time (SPT), and first-in-first-out (FIFO) dispatching rules. They found that GA outperformed these three dispatching rules in terms of solution quality.

2.2 Hybrid general-purpose heuristic algorithms

Recent studies have shown that hybridizing two general-purpose heuristic algorithms or incorporating local search methods can enhance solution quality and improve computational efficiency. In terms of hybrid approaches, Mir and Rezaeian (2016) combined particle swarm optimization and GAs to solve the UPM scheduling problem, aiming to minimize the total machine load. Two particles are paired in each iteration, and the best solution is then subjected to mutation. They found that the mating and mutation logic increased solution diversity, thus leading to better solutions. Similarly, Ewees et al. (2021) addressed sequence-dependent and machine-dependent setup time constraints to minimize the makespan. They initially applied the salp swarm algorithm (SSA) to address the problem and subsequently employed the firefly algorithm (FA) to refine the initial solution obtained from SSA with a certain probability. They suggested that hybrid algorithms improve the algorithm's solution capability while reducing computational time.

For methods that integrate local search to enhance general-purpose heuristic algorithms, two primary approaches exist. The first involves embedding local search within the iterative steps of the basic general-purpose heuristic algorithm, while the second entails using a general-purpose heuristic algorithm to generate an initial solution, followed by applying a local search for further optimization. Vallada and Ruiz (2011) based their approach on GA and integrated a multiple-insertion local search method within the iteration steps. They studied the UPM scheduling problem with sequence-dependent setup time constraints and found that adding the local search method led to better solutions. Ezugwu and Akutsah (2018) addressed sequence-dependent and machine-dependent setup time constraints, initially using FA to obtain a solution and then applying a local search method for further optimization. Since the initial solution obtained by FA is influenced by the first generation of the population, the local search method, by swapping job sequences, ensures a more stable approximate optimal solution. They suggested that integrating the local search method effectively solves large-scale scheduling problems and improves computational efficiency (Xie et al., 2025; Shen et al., 2025).

2.3 Integration of reinforcement learning and universal heuristic algorithms

2.3.1 Reinforcement learning

RL developed from neuroscience, computer science, psychology, and cybernetics. RL mimics the learning mechanisms of animals or humans, who tend to perform behaviors that lead to rewards. RL attempts to identify actions that generate the highest return or reward. It is a learning mechanism that interacts with the environment and learns through continuous trial and feedback. Unlike other learning methods, the RL does not specify the best action but evaluates the overall feedback of an action in its current state. RL models have two main components: the agent and the environment. At each decision step t, the agent observes the system state S_t, performs an action A_t, and receives feedback R_t. After executing the action, the environment's state changes, and the agent observes the next state S_{t+1} and takes the next action A_{t+1}. RL is typically represented as a Markov decision process, where the Markov property states that all processes involve randomness, and the transition to the next state depends only on the current state. In real life, it is impossible to predict the exact next state. However, the current state contains information about all past states. In other words, past states can be ignored, allowing a focus solely on the current state. This reduces computation and ensures the use of simple iterative methods to obtain results. Therefore, when solving RL problems, the Markov property is often assumed to achieve approximate results. According to the use of the environment model, RL can be divided into two categories: model-based and model-free. Model-based RL requires known state transition probabilities and predicted rewards. As it is challenging to obtain state transition probabilities in real-world problems, model-free temporal difference learning algorithms can be used. These methods learn by interacting with the environment without requiring state transition probabilities. Model-free temporal difference learning algorithms are divided into two types: on-policy and off-policy learning strategies. The key difference is that, in on-policy learning, the policy followed when choosing an action is the same as the policy used to update the action value function. This method estimates actions that lead to the maximum reward while directly selecting those actions for the next step. In off-policy learning, the algorithm estimates the actions that could potentially lead to the maximum reward. However, the action chosen does not necessarily correspond to this estimated action. The representative algorithms for these two strategies are Sarsa and Q-learning. The Sarsa algorithm updates the action value function and selects actions using the target policy, which is typically ε-greedy. This makes it relatively conservative and suitable for problems with high risk, with no room for failure. In contrast, Q-learning updates the action value function using the maximum possible reward, making its decision process bolder and more tolerant of penalties. Hence, Q-learning is particularly suitable for problems that require quickly finding the optimal path. Since RL methods can model scheduling problems, agents that learn from previous experiences can solve new problems more efficiently when provided with new information. Most current research applies the Q-learning algorithm to solve UPM scheduling problems (Kayhan & Yildiz, 2023; Yi et al., 2023).

2.3.2 Q-learning algorithm

The Q-learning algorithm is a model-free, temporal difference learning method. This method does not require assuming an environment model but instead directly receives information from the environment and selects the optimal action policy in a finite Markov decision process. The Q-learning algorithm aims to determine the optimal action that yields the maximum

reward in each state. The algorithm uses the long-term expected reward $Q(S_t, A_t)$ value to determine the best action. During the agent's interaction with the environment, it continuously updates the Q-values based on the received feedback. The function used to update the Q-value is given in Equation (1) (Watkins, 1989):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{a \in A} Q(S_{t+1}, a) - Q(S_t, A_t) \right] \quad (1)$$

where:

S: The set of possible states in the process;
A: The set of possible actions in the process;
R: The set of rewards obtained when transitioning from one state to another;
γ: The discount factor;
α: The learning rate.

In Eq. (1), α defines the learning rate. When α=0, the value function is not updated, and the Q-value remains unchanged, indicating no learning capability. When α=1, the agent completely ignores previously learned knowledge and directly updates the knowledge acquired from the current action. A higher learning rate causes the Q-values to change more quickly, which carries a high risk. Therefore, it is important to avoid a learning rate that is too low, which would cause the agent to learn too slowly, while also avoiding a rate that is too high, which could lead to overcorrection. The discount factor γ, which lies between 0 and 1, prevents the agent from being myopic. When γ=1, the agent is like wearing glasses and can clearly see the benefits of each subsequent step. Conversely, when γ=0, the agent removes the glasses and can only recognize the immediate benefits. Thus, as γ increases from 0 to 1, the "degree" of the glasses increases, and the agent gradually becomes more farsighted and focuses on long-term rewards.

Since the Q-learning algorithm aims to maximize long-term expected rewards, the agent estimates future rewards when selecting actions rather than focusing solely on immediate benefits. Existing studies have applied Q-learning to scheduling problems. The number of actions remains fixed for dispatching rules, ensuring that the Q-table does not grow excessively as the problem size increases, thereby reducing memory requirements. However, the algorithm's performance depends on the selected dispatching rules. Wang and Usher (2004) examined a single-machine scheduling problem aimed at minimizing delays, applying Q-learning to dynamically select among three dispatching rules: EDD, SPT, and FIFO. The results demonstrated that, compared to using a single dispatching rule, this method achieved solutions with shorter delays and provided suggestions for setting parameters. Yuan and Xu (2015) applied Q-learning to dynamic scheduling problems to minimize the maximum tardiness and the percentage of tardy jobs. They used SPT, EDD, and First-Come, First-Served dispatching rules as actions. The results indicated that Q-learning outperformed each objective's most effective scheduling rules. Guo et al. (2014) investigated the multi-objective dynamic UPM scheduling problem, in which machine speeds vary and task arrival times are uncertain. They proposed a Q-learning-based multi-stage RL method that repeatedly trains in multiple stages to achieve different objectives. Their method was compared with three dispatching rules and individual single-objective Q-learning methods. The experimental results showed that the proposed multi-stage RL method effectively balanced the performance of the three objectives. For problems involving machine allocation and job sequence adjustments as actions, this approach is suitable for problems with fewer constraints, thus reducing the risk of generating infeasible solutions. Li et al. (2021) defined the length of waiting for jobs as the state and the selection of unprocessed jobs as the action. They employed Q-learning to solve the single-machine scheduling problem and minimize delay times. The results indicated that the Q-learning algorithm outperformed the EDD dispatching rule, producing solutions with shorter delays.

2.3.3 Combining RL with general heuristic algorithms

Recent studies have found that introducing RL to assist general heuristic algorithms can improve search efficiency or enhance solution quality. The flexibility and learning ability of RL guide parameter selection and search direction in general heuristic algorithms, so that these algorithms self-learn and adapt to the problem environment. This approach can overcome the issue of general heuristic algorithms requiring numerous experiments to determine optimal parameter settings. When using RL to guide parameter selection in general heuristics, the RL agent first learns parameter settings through interaction with the environment using test data. During the solution process, the trained agent then guides the general heuristic algorithm to self-adjust. Shahrabi et al. (2017) used the Q-learning algorithm to guide the selection of parameters for VNS, in order to dynamically adjust the parameters during the iterative process to improve search efficiency. Their method solved the job-shop production scheduling problem, aiming to minimize the total makespan, and showed that the algorithm outperformed traditional VNS by finding shorter total makespan solutions in less time. Chen et al. (2020) designed a self-learning GA and introduced two RL methods, Sarsa and Q-learning, to adjust the parameters of the GA and improve search capabilities at different stages.

For applying RL to guide the search space of general heuristics, Liu and Zeng (2009) introduced a reinforcement learning-based genetic algorithm (GA with reinforcement mutation) to tackle the traveling salesman problem. Silva et al. (2019)

proposed a Q-learning algorithm to train multiple agents to determine the neighborhood structure of VNS, aiming to solve the UPM scheduling problem and minimize the makespan. Although this approach converges quickly, its solution quality is less stable and performs poorly in some cases. Alicastro et al. (2021) introduced a reinforcement learning-based iterative local search algorithm to tackle the UPM scheduling problem, aiming to minimize the makespan. In the search process, RL assists in selecting the neighborhood structure to improve solution efficiency. Experimental results show that this method produces solutions that are not significantly different from the best-known solutions in the database, achieving faster solution times.

3. Research methodology and process

3.1 Problem description

The UPM scheduling problem is a typical allocation problem involving n independent jobs that must be processed on m parallel machines. The processing time for each job is a known, finite, and deterministic positive value. These processing times vary across different machines and are independent of each other, without any specific proportional relationship. Each job must be assigned to one of the m parallel machines for processing, with only one job being processed per machine at a time, and processing cannot be interrupted.

Pinedo (2022) proposed that production scheduling problems can be described using three parameters: $\alpha|\beta|\gamma$, where α is the machine environment configuration, β is the production scheduling constraints, and γ is the objective of the production scheduling problem. Based on the problem explored in this study and the objective of minimizing the maximum completion time, it can be expressed as $R_m|M_j, S_{ik}|C_{max}$. This study makes the following assumption regarding this problem:

(1) Each machine can process only one job at a time.
(2) The jobs must not be interrupted once processing begins; they must be completed once started.
(3) The processing time for each job on the available machines is fixed and known in advance.
(4) The setup time between different jobs on each machine is deterministic and known.
(5) Rework characteristics are not considered.
(6) The machines are assumed to be continuously available for processing without considering issues such as machine breakdowns or maintenance.

To solve this production scheduling problem, this study proposes a GA with self-learning capabilities, incorporating innovative features that combine RL and local search methods. These features effectively address the aforementioned problem.

3.2 Construction of mathematical model

A mixed integer programming model is constructed to explore the UPM scheduling problem. The decision variables and parameters involved in the model are summarized in Table 1.

Table 1
Description of the decision variables and parameter symbols used in the model

Symbol	Description	
Subscript		
i	Machine number, $i \in M$	
h, j, k	Job number, $h, j, k \in N$	
t	Time point, $t \in H$	
0	Virtual job	
Set		
M	Set of all machines, $i = \{1,2, \dots, \lvert M \rvert\}$	
N	Set of all jobs, $j, h, k = \{1,2, \dots, \lvert N \rvert\}$	
H	Set of time points, $t = \{1,2, \dots, \lvert H \rvert\}$	
Parameters		
p_{ij}	Processing time of job j on machine i	
s_{ijk}	Setup time between job j and the next job k on machine i	
E_{ij}	Equals 1 if job j can be processed by machine i, otherwise 0	
G	A large constant value	
Decision variables		
$x_{ij} = \begin{cases} 1, \text{if job } j \text{ is assigned to machine } i \\ 0, \ otherwise \end{cases}$		$i \in M; j \in N$
$x_{ijk} = \begin{cases} 1, \text{if job } j \text{ is assigned to machine } i \text{ and is followed by job } k \\ 0, \quad otherwise \end{cases}$		$i \in M; j, k \in N$
$x_{i0j} = \begin{cases} 1, \text{if job } j \text{ is assigned to machine i and is the first job} \\ 0, \quad otherwise \end{cases}$		$i \in M; j \in N$
$z_{jt} = \begin{cases} 1, \text{if job } j \text{ is completed at time } t \\ 0, \quad otherwise \end{cases}$		$j \in M; t \in H$

The objective function and constraints of the model are described as follows:

$$min\ C_{max} \tag{2}$$

subject to

$$\sum_{i\in M} x_{ij} = 1\ ,\ \forall\, j \in N \tag{3}$$

$$\sum_{i\in M}\sum_{j\in\{0\}\cup\{N\}} x_{ijk} = 1\ ,\ \forall k \in N \tag{4}$$

$$\sum_{\substack{k\in N\\ j\neq k}}(x_{ikj} + x_{i0j}) = x_{ij},\ \forall i \in M;\ \forall j \in N \tag{5}$$

$$\sum_{t=p_{ik}}^{|H|}(t * z_{kt}) \geq \sum_{t=p_{ij}}^{|H|}(t * z_{jt}) + \sum_{i\in M}\left(\sum_{\substack{k\in N\\ k\neq j}} s_{ijk} * x_{ijk}\right) + \sum_{i\in M}(p_{ik} * x_{ik}) + G\left(\sum_{i\in M} x_{ijk} - 1\right), \forall j,k \in N;\ k \neq j \tag{6}$$

$$\sum_{t=p_{ik}}^{|H|}(t * z_{kt}) \geq \sum_{i\in M}\left(\sum_{\substack{k\in N\\ k\neq j}} s_{ijk} * x_{ijk}\right) + \sum_{i\in M}(p_{ik} * x_{ik}),\ \forall j,k \in N, k \neq j \tag{7}$$

$$\sum_{t\in H} z_{jt} = 1, \quad \forall\, j \in N \tag{8}$$

$$x_{ijk} \leq E_{ik}, \quad \forall i \in M\ ;\ \forall j,k \in N, k \neq j \tag{9}$$

$$x_{i0j} \leq E_{ij},\ \forall i \in M\ ;\ \forall j \in N \tag{10}$$

$$x_{ij} \leq E_{ij},\ \forall i \in M\ ;\ \forall j \in N \tag{11}$$

$$C_{max} \geq t * z_{jt}, \quad \forall j \in N;\ \forall t \in H \tag{12}$$

$$C_{max} \geq 0,\ x_{ij}, x_{ijk}, x_{i0j}, z_{jt} \in \{0,1\} \quad \forall i \in M\ ;\ \forall\, j,k \in N, t \in H \tag{13}$$

Eq. (2) defines the objective function, which aims to minimize the maximum completion time. Eq. (3) ensures that each job is assigned to exactly one machine. Eq. (4) ensures that no job is processed more than once. Additionally, it enforces that once assigned to a machine, each job can have at most one subsequent job following it in processing. Eq. (5) enforces the constraint that jobs cannot be split and must be processed continuously on the machine. If job *j* is assigned to machine *i*, there must be a job *k* on the same machine processed before job *j*, or job *j* must be the first job on that machine. Eq. (6) ensures that the completion time of the subsequent job *k* is at least as large as the completion time of job *j*, plus the setup time between jobs *j* and *k*, and the processing time of job *k*. In other words, job *k* can only begin after job *j* finishes, considering both the setup time and its own processing time. Eq. (7) restricts the start time of job *k* by ensuring that job *k* can only finish after its own processing time and the setup time between it and the previous job *j*. Eq. (8) ensures that each job is processed exactly once within the scheduling period. Eq. (9) to Eq. (11) impose machine eligibility restrictions, ensuring that each job is only processed on permitted machines. Eq. (12) calculates the completion time of each job. Equation (13) ensures that the completion time is non-negative and that the decision variables are binary

3.3 Genetic algorithm with RL and VNS execution process

This study develops a GA that integrates RL and VNS to solve the UPM scheduling problem. The algorithm is based on a GA, with VNS incorporating RL during iterative steps.

3.3.1 Genetic algorithm execution process

Step 1: Job encoding

In this study, the job number is treated as a unique and non-repetitive gene, and integer coding is used for gene representation. A chromosome represents a scheduling combination. Since the UPM scheduling problem involves both machine assignment and job sequencing, the chromosome is structured as a two-dimensional matrix.

Step 2: Generate the initial population of chromosomes

All machines randomly select jobs sequentially, ensuring uniqueness in job encoding so that each job is assigned to only one machine. This process is then converted into the chromosome format from Step 1, and generates a chromosome that represents a solution. This process is repeated iteratively until the specified population size is reached.

Step 3: Calculate fitness values

To evaluate the quality of each chromosome, a fitness function that aligns with the problem's objective is defined. This function calculates the fitness value for each chromosome. In this study, the maximum completion time is used as the performance indicator. The fitness function is defined in Eq. (2), where the fitness value is determined by the maximum completion time in the chromosome.

Step 4: Select parent chromosomes

To enhance the likelihood of mating higher-performing chromosomes while maintaining population diversity, this study employs an elitism selection technique alongside roulette-wheel selection to construct the mating pool. First, the chromosome

with the highest fitness value is added to the pool. The remaining chromosomes are then selected using the roulette-wheel method.

Step 5: Crossover

To improve fitness values in each iteration, this study applies self-crossover to generate offspring. After selecting the parent chromosomes, the machine with the maximum completion time in the chromosome is identified. A job number for that machine is randomly selected and moved to another machine while ensuring feasibility constraints are met. The new processing sequence is then assigned randomly.

Step 6: Mutation

To maintain population diversity and prevent premature convergence, where similar individuals dominate early and increase the risk of falling into a local optimum, offspring chromosomes undergo mutation with a low probability after crossover. This study applies a single-point mutation, where a randomly selected gene is reassigned to another randomly chosen feasible machine, completing the mutation process.

Step 7: Self-learning local search

Step 8: Check if iteration conditions are met

Considering the solution time, this study sets the maximum number of iterations as the stopping criterion. If the iteration count reaches the upper limit, the algorithm terminates.

3.3.2 Integrating RL with VNS

This study utilizes a trained agent to intelligently select the neighborhood search strategy of VNS, aiming to improve solution quality. The process of the self-learning regional search method is as follows:

Step 1: Define the action set and state set

In this study, both actions and states correspond to the neighborhood structures of VNS. The concept of an action is defined as "go to", which means transferring from the current state to the next state by applying the selected search method.

The action and state sets consist of four neighborhood structures:

(1) Insertion within the same machine: Insert the job order into a different position on the same machine.
(2) Swap within the same machine: Swap the job order with another job order on the same machine.
(3) Insertion to a different machine: Insert the job order into a different position on a different machine.
(4) Swap between different machines: Swap the job order with another job order on a different machine.

Step 2: Define the initial state S_0 and feedback mechanism

Due to the design of states and actions, the algorithm requires an initial "warming up" phase, where the initial state S_0 is randomly selected. The search method corresponding to this state is then applied to the initial solution, followed by iterative searching.

To minimize the makespan (i.e., the maximum completion time), this study uses the feedback mechanism to evaluate the difference in fitness values between the previous solution and the one obtained using the current search method, as shown in Equation (14). A better solution receives a higher reward, while a worse solution is penalized.

$$reward = f(x_{t-1}) - f(x_t) \tag{14}$$

Step 3: Initialize the Q-table and determine the target policy

First, the agent explores actions and interacts with the environment. All Q-values are set to 0, allowing the agent to randomly select an initial action. To ensure the agent selects actions with diversity, Watkins (1989) defined the ε-greedy function as the target policy, as shown in Eq. (15). Here, ε denotes the exploration rate. The agent selects an action randomly with a probability of ε, while with a probability of 1-ε, it chooses the action with the highest Q-value in the Q-table, denoted as a^*.

$$\pi(S) = \begin{cases} a^*, & with\ probability\ 1-\varepsilon \\ any\ other, & with\ probability\ \varepsilon \end{cases} \tag{15}$$

Step 4: Execute the action and update the Q-table based on the next state and feedback

The agent applies the action selected according to the target policy to the current solution, which is equivalent to performing

a single neighborhood search. At this point, the state transitions to the next state, and based on the degree of improvement in the solution, the environment provides feedback, completing one learning iteration. The "experience" gained from learning is recorded in the Q-table. The Q-value for this learning step is calculated according to Equation (1), and the Q-table is then updated.

Step 5: VNS search steps

After the GA generates offspring, the agent selects an appropriate neighborhood search based on the current state. If a better solution is found, the current solution is updated. If no improvement is found, only the Q-value is updated, while the current solution remains unchanged. The agent then proceeds to select the next neighborhood search. This design ensures that the algorithm continuously searches in a direction that improves solutions.

Step 6: Iteration termination condition

The self-learning regional search terminates when all neighborhood search methods have been executed, but no better solutions can be found. The current solution is then used to update the next generation of the GA. Fig. 1 is a flowchart illustrating the integration of GA, RL, and VNS.

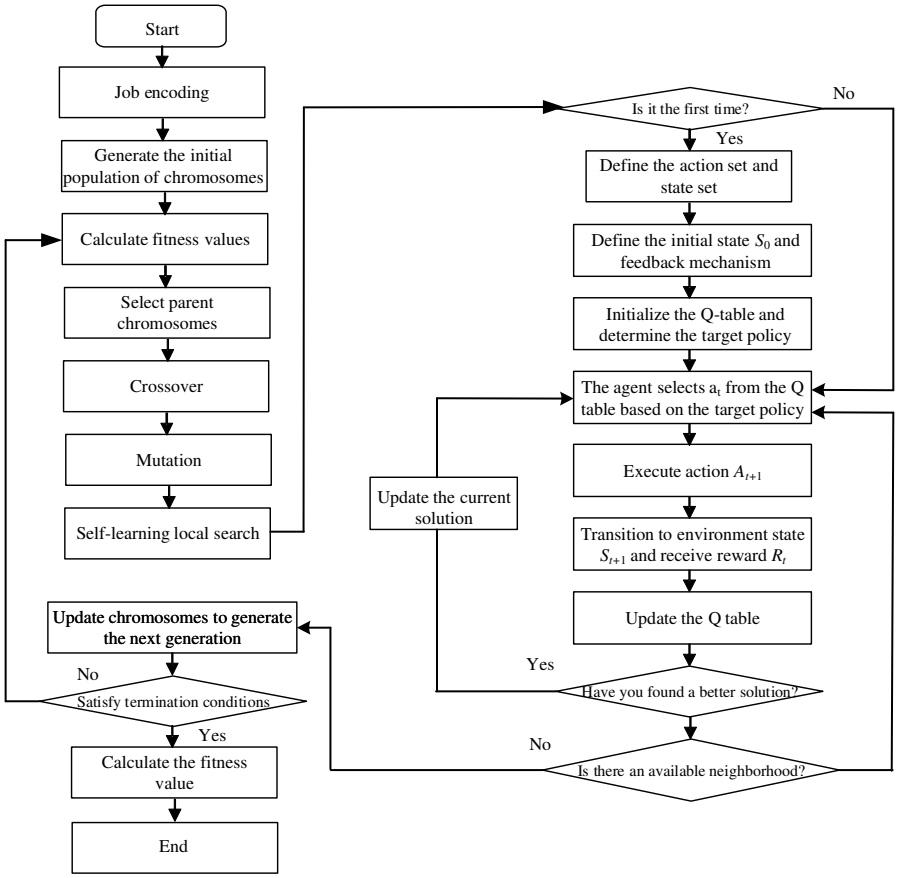

Fig. 1. Flowchart of the GA combining RL with VNS

4. Implementation and result analysis

This study used randomly generated data and actual job scheduling data from the slitting process of D Company, a globally recognized semiconductor manufacturer. The experiments were conducted using Python 3.8, with the following computer specifications: Intel i5-8250U processor and 12GB RAM.

4.1 Data sources

To evaluate the performance and practicality of the proposed algorithm, small-scale test problems provided by Vallada and Ruiz (2011) were first used. Their solutions were compared with the optimal solutions to validate the algorithm's effectiveness. Next, large-scale test data generated randomly were tested and compared with the standard GA and the GA combined with VNS. To verify the practical applicability of the proposed algorithm, 20 job orders from the week of January 2022, provided

by Company D, were used for actual testing. The results were compared with those of the heuristic algorithm based on Company D's on-site scheduling logic design.

The processing time for the small-scale test problems provided by Vallada & Ruiz (2011) was randomly generated using a discrete uniform distribution U(1, 99). In each problem scenario, different subsets of setup times were generated using four different uniform distributions: U(1, 9), U(1, 49), U(1, 99), and U(1, 124). To assess the performance of the proposed method on large-scale problems, large-scale test problems were generated using three factors. The first factor is the number of job orders (N), representing the number of scheduled job orders, at four levels: 30, 50, 70, and 90. The second factor is the number of machines (M), representing the number of machines available, with three levels: 5, 7, and 10. The third factor is machine availability, which simulates the feasibility constraint of machines. It represents the probability that a job can be processed on each machine, with two levels: 0.5 and 0.7. If processing is not feasible, the processing time is set to 99,999 hours. In these test problems, processing time and setup time are randomly generated using discrete uniform distributions U(1, 99) and U(1, 9), respectively.

The scheduled job orders, provided by Company D, consist of 20 job orders from the week of January 2022, as shown in Table 2. Company D's slitting process involves taking semi-finished products from the previous stage and cutting them to the required size. The process is divided into upper and lower axes, with the cutting tools, modules, and settings adjusted based on the product and size. Each week, the production control engineer at Company D manually assigns machines and determines the processing order for each job order based on production environment constraints, experience, and weekly production capacity. If, after scheduling, the development team requires a machine or an unforeseen event affects a machine or job order, the production control personnel must adjust the schedule. However, machine idle time may increase without a quick and appropriate scheduling adjustment, leading to reduced production capacity or job order delays. Therefore, a fast and efficient algorithm is needed to quickly adjust the schedule and improve Company D's production efficiency.

Table 2
Company D job orders to be scheduled

Order	Target qty	Basic fin.	Type	Width	M1-4	M1-5	M2-1	M2-2	M2-3
1	1850	01/02/2021	G	1000	23	16	23	-	-
2	120	01/02/2021	G	1000	2	2	2	-	-
3	450	01/03/2021	G	1000	6	4	6	-	-
4	1800	01/03/2021	G	1000	22	15	22	-	-
5	2550	01/04/2021	G	1000	31	31	31	-	-
7	2080	01/05/2021	G	520	26	19	26	-	-
8	5000	01/05/2021	G	1000	61	43	61	-	-
9	1500	01/06/2021	G	1000	19	15	19	-	-
10	1800	01/06/2021	G	1000	22	16	22	-	-
11	2400	01/06/2021	G	1000	29	21	29	-	-
12	1092	01/06/2021	G	520	14	9	14	-	-
13	900	01/07/2021	G	1000	11	8	11	-	-
14	4004	01/07/2021	B	520	-	-	-	49	49
15	100	01/07/2021	C	1000	-	-	-	2	2
16	1250	01/09/2021	C	250	-	-	-	24	24
17	1050	01/09/2021	G	1000	13	9	13	-	-
18	2000	01/09/2021	G	250	26	21	26	-	-
19	1100	01/09/2021	A	1000	-	-	-	-	66
20	6550	01/09/2021	G	1000	79	60	79	-	-

Table 3
Explanation of the fields for job orders to be scheduled

Name	Description
Order	Job order code
Target qty	Target production quantity
Basic fin.	Delivery time
Type	Product category for processing
width	Width of the product being processed
M1-4~M2-3	Abbreviated as M1 to M5, represents the processing time for the corresponding job order on each machine (M1 to M5)

Table 3 shows the fields for job orders to be scheduled. Company D applies sequence-dependent setup times, meaning that when different product types are processed consecutively, setup times vary based on the product category. Additionally, setup time is required if the width of two consecutive job orders differs. Machines M1-4 to M2-3 represent the machine sequence within Company D, abbreviated as M1 to M5. The field content indicates the processing time required for each job order on the corresponding machine, measured in hours. Due to limitations in machine capabilities or considerations related to worker skill levels on each shift, some products cannot be processed on certain machines. In such cases, the corresponding machine field displays a "–", indicating that the job order cannot be processed on that machine. This constraint restricts the availability of machines. For ease of program execution, when a job cannot be processed on a machine, its processing time is set to a significantly large value. During fitness value calculations, these infeasible machines are directly eliminated from

consideration. This study compared the results of the proposed algorithm across different problem scales and setup time data subsets with a standard GA and a GA combined with VNS. Additionally, the algorithm was implemented using 20 actual job orders and data from five machines, with results compared against the company's current scheduling method.

4.2 Algorithm parameter settings

The parameters of the GA significantly influence both the computational efficiency and the quality of the optimal solution. Therefore, it is essential to configure the algorithm's parameters, including population size, crossover rate, mutation rate, and iteration termination criteria. This study uses actual job orders from Company D to conduct experimental tests under various parameter combinations to determine the optimal settings. The diversity of feasible solutions in this algorithm is directly proportional to the population size. As the global search time increases, diversity also increases, and vice versa. The number of chromosomes selected for crossover is determined by the crossover rate, which significantly affects the algorithm's computation time. Therefore, this study analyzed the interaction between these two parameters, explored the relationship between their variations, and assessed the computational efficiency based on changes in fitness values.

(1) Population size and crossover rate settings

A larger population size increases solution diversity, and enhances the algorithm's ability to perform global search. However, it also extends execution times. A smaller population size reduces execution time but may lower the diversity of feasible solutions, potentially leading to premature convergence on a local optimum. The number of chromosomes selected for crossover is determined by the crossover rate, which significantly affects the computational time. Therefore, this study experimented with different combinations of these two parameters and observed the changes in fitness to identify the most suitable parameter settings. Grefenstette (1986) noted that when the population size exceeds 30, better solutions are more likely to be obtained. Accordingly, this study evaluated the impact of population sizes set at 20, 30, and 40 on the fitness values. Regarding the effect of the crossover rate on fitness values, Sriniva and Patnaik (1994) suggested that crossover rates typically range between 0.5 and 1.0, while Wang et al. (1999) recommended a crossover rate of 0.8 for improved solutions. Based on these prior studies, this study tested crossover rates of 0.5, 0.8, and 1.0, resulting in nine different parameter combinations. The corresponding fitness values are analyzed, with each combination repeated three times. The experimental results are presented in Table 4.

Table 4
Algorithm parameter settings

Population size	Reproduction rate		
	0.5	0.8	1
20	118	117	115
	117	116	115
	116	114	114
30	115	114	114
	115	114	114
	114	114	114
40	114	114	114
	114	114	114
	114	114	114

The results in Table 4 indicate that when the population size is set to 20, 30, and 40, and the algorithm is executed with different crossover rates, the solutions obtained are measured in hours. Table 4 shows that when the population size is set to 30 or 40, and the crossover rate is either 0.8 or 1.0, the algorithm achieves a smaller makespan, leading to better solutions. When the population size is 20, and the crossover rate is set to 0.8 or 1.0, one of the three experiments produced a fairly good solution. However, due to the inconsistency in the solution quality, this study concluded that the most suitable parameters are a population size of 30 and a crossover rate of 1.0, as this combination consistently yields better-quality solutions.

(2) Mutation rate

Srinivas and Patnaik (1994) proposed that mutation rates are generally set between 0.005 and 0.05. Given the relatively small population size in this study, the risk of the algorithm converging to a local optimum increase. Therefore, the mutation rate is set to 0.05 to enhance the algorithm's ability to escape local optima and increase the diversity of feasible solutions through mutation.

(3) Iteration stopping condition

The proposed algorithm in this study terminates when the maximum number of iterations is reached. Different iteration numbers are tested to determine the appropriate maximum iteration count, and the reduction in fitness values is observed to assess whether the algorithm has converged. The experimental results, shown in Fig. 2, indicate that the makespan gradually stabilizes after 25 iterations, with full convergence achieved by the 30th iteration. Therefore, the maximum number of iterations for the algorithm is set to 30.

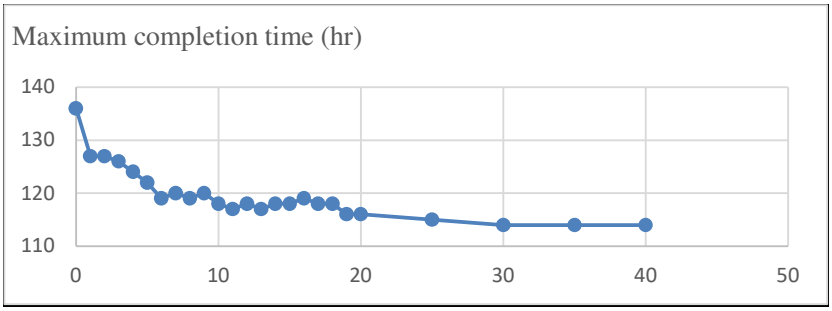

Fig. 2. Trend of the makespan reduction during the algorithm's iteration process

(4) Parameter settings for RL

In this study, the parameter settings for the Q-learning algorithm, as proposed by Watkins (1989), were applied. The parameters were configured as follows: the population size is set to 50, the crossover rate to 1, the mutation rate to 0.05, and the maximum number of iterations to 30. The learning rate (α) is 0.5, the discount factor (γ) is 0.95, and the exploration rate (ε) is 0.1. These settings were used to conduct experiments on various problem scales and the job orders provided by Company D.

4.3 Experiment results analysis

4.3.1 Small-scale problem test data experiment results

This study utilized small-scale test data provided by Vallada and Ruiz (2011) . The dataset, named "I_6_3_S_1-9", consists of six job orders, three machines, and setup times that follow a uniform distribution U(1, 9). A total of 32 different combinations were tested, and the results were compared with the optimal solutions obtained from the CPLEX solver to evaluate the algorithm's performance. The experimental results are presented in Table 5, where C_max denotes the maximum makespan. The difference is calculated using the following formula to determine the average relative percentage deviation (RPD) for each data set:

$$\text{RPD} = \frac{\text{Our method objective value} - \text{Best solution objective value}}{\text{Best solution objective value}} \times 100\%$$

The experimental results indicate that the algorithm proposed in this study consistently finds the optimal solution across all 32 scenarios, demonstrating its effectiveness.

Table 5
Experimental results for small-scale problem test data

No.	Instance	Proposed method	Optimal solution	Difference (%)
		C_{max}	C_{max}	
1	I_6_3_S_1-9	94	94	0%
2	I_6_3_S_1-49	70	70	0%
3	I_6_3_S_1-99	64	64	0%
4	I_6_3_S_1-124	81	81	0%
5	I_6_5_S_1-9	49	49	0%
6	I_6_5_S_1-49	33	33	0%
7	I_6_5_S_1-99	56	56	0%
8	I_6_5_S_1-124	53	53	0%
9	I_8_3_S_1-9	100	100	0%
10	I_8_3_S_1-49	71	71	0%
11	I_8_3_S_1-99	94	94	0%
12	I_8_3_S_1-124	140	140	0%
13	I_8_5_S_1-9	58	58	0%
14	I_8_5_S_1-49	77	77	0%
15	I_8_5_S_1-99	60	60	0%
16	I_8_5_S_1-124	67	67	0%
17	I_10_3_S_1-9	64	64	0%
18	I_10_3_S_1-49	128	128	0%
19	I_10_3_S_1-99	181	181	0%
20	I_10_3_S_1-124	169	169	0%
21	I_10_5_S_1-9	69	69	0%
22	I_10_5_S_1-49	66	66	0%
23	I_10_5_S_1-99	83	83	0%
24	I_10_5_S_1-124	80	80	0%
25	I_12_3_S_1-9	115	115	0%
26	I_12_3_S_1-49	157	157	0%
27	I_12_3_S_1-99	135	135	0%
28	I_12_3_S_1-124	173	173	0%
29	I_12_5_S_1-9	52	52	0%
30	I_12_5_S_1-49	64	64	0%
31	I_12_5_S_1-99	80	80	0%
32	I_12_5_S_1-124	97	97	0%

4.3.2 Large-scale problem test data experiment results

Experiments were conducted using large-scale test data, comparing the results of the GA with those of the GA combined with VNS. The experimental results are shown in Table 6, where C_max denotes the maximum makespan. The difference is calculated using the following formula to compute the average relative percentage deviation (RPD):

$$\text{RPD} = \frac{\text{Algorithm's objective value} - \text{Our method's objective value}}{\text{Our method's objective value}} \times 100\%$$

Therefore, a larger difference indicates a better performance of the proposed method. The experimental results show that the algorithm proposed in this study outperforms other algorithms across all 24 scenarios, highlighting its feasibility and effectiveness.

Table 6
Experimental results for large-scale test problems

No.	N×M	Machine availability rate	Proposed method	GA+VNS		GA	
			C_{max}	C_{max}	Difference (%)	C_{max}	Difference (%)
1	30×5	0.5	187	190	1.60	219	17.11
2	30×5	0.7	149	162	8.76	211	41.61
3	30×7	0.5	148	158	6.76	163	10.14
4	30×7	0.7	98	104	6.12	145	47.96
5	30×10	0.5	87	97	11.49	106	21.84
6	30×10	0.7	71	85	19.72	107	50.70
7	50×5	0.5	370	391	5.68	438	18.38
8	50×5	0.7	242	277	14.46	378	56.20
9	50×7	0.5	212	235	10.85	290	36.79
10	50×7	0.7	175	195	11.43	254	45.14
11	50×10	0.5	144	197	36.81	221	53.47
12	50×10	0.7	82	120	46.34	184	124.39
13	70×5	0.5	400	450	12.50	538	34.50
14	70×5	0.7	319	389	21.94	553	73.35
15	70×7	0.5	291	313	7.56	441	51.55
16	70×7	0.7	236	266	12.71	424	79.66
17	70×10	0.5	216	256	18.52	354	63.89
18	70×10	0.7	207	243	17.39	301	45.41
19	90×5	0.5	551	614	11.43	727	31.94
20	90×5	0.7	558	673	20.61	829	48.57
21	90×7	0.5	407	465	14.25	596	46.44
22	90×7	0.7	330	338	2.42	583	76.67
23	90×10	0.5	229	306	33.62	430	87.77
24	90×10	0.7	187	283	51.34	407	117.65

4.3.3 Actual work order scheduling experiment results

To assess the effectiveness and feasibility of the proposed algorithm in real-world applications, this study conducted tests using actual work orders provided by Company D and compared the results with Company D's current scheduling logic, heuristic algorithm, and GA analysis. Currently, Company D's scheduling method is divided into two steps, as described below:

Step 1: Work order sorting

The work orders are sorted based on delivery time and the number of items to be processed, which determines the scheduling priority. Specifically, work orders with earlier delivery times are assigned higher priority. If multiple work orders share the same delivery time, those with larger processing quantities are prioritized for scheduling.

Step 2: Work order scheduling

After sorting, the work orders are scheduled sequentially. Before each work order enters the scheduling process, its completion time on each of the five machines is calculated, including the setup time required between the previous work order and the work order to be scheduled on each machine. The work order is then assigned to the machine, which minimizes the current completion time. This process is repeated until all work orders have been allocated. After executing Company D's current scheduling method, the resulting maximum completion time was 147 hours. The Gantt chart depicting the scheduling results is shown in Figure 3. In the chart, the colored blocks represent the processing times of different work orders, the gray blocks indicate the setup times between work orders, and the blank areas denote the periods during which the machines are idle. The data indicate that Machine M4 has longer idle times due to fewer work orders being assigned to it.

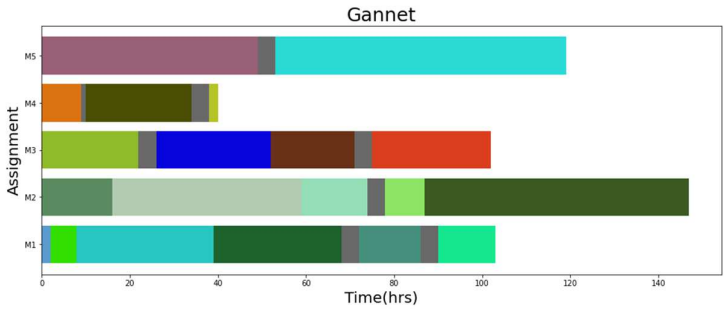

Fig. 3. Gantt chart of Company D's current scheduling method

To evaluate the effectiveness of the self-learning neighborhood search method proposed in this study, experiments were conducted using the GA without this method. The resulting maximum completion time after execution was 120 hours. The Gantt chart depicting the scheduling results is presented in Fig. 4.

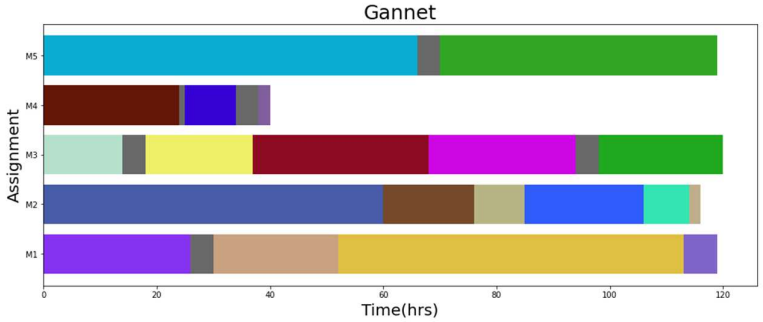

Fig. 4. Gantt chart of the GA execution

After executing the proposed GA combined with RL and VNS, the maximum completion time obtained was 114 hours. The Gantt chart depicting the scheduling results is presented in Fig. 5.

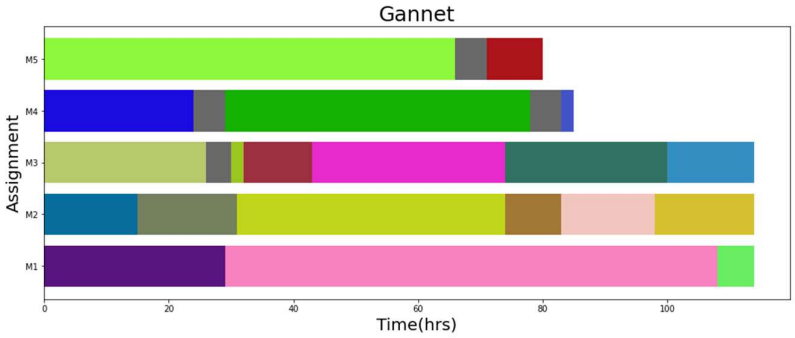

Fig. 5. Gantt chart of the execution of the GA combined with RL and VNS

Based on the scheduling results obtained from the three different algorithms, this study compared the maximum completion time using Company D's current scheduling method as the baseline. Table 7 presents the summarized results, with the differences computed using the average relative percentage deviation (RPD), calculated as follows:

$$\text{RPD} = \frac{\text{Comparative method target value} - \text{Current scheduling method target value}}{\text{Current scheduling method target value}} * 100\%$$

As shown in Table 7, the proposed method reduces the maximum completion time by 22.45%. Therefore, the GA combined with RL and VNS proposed in this study is both feasible and effective.

Table 7
Comparison of scheduling results among three methods

Method	Maximum completion time	Difference (%)
Current scheduling method (D company)	147	-
GA method	120	18.37%
Proposed method	114	22.45%

5. Conclusion and recommendations

To address scheduling problems in the manufacturing industry, this study focused on the UPM scheduling problem while considering machine feasibility constraints and machine-job dependent setup times. Since this problem is NP-hard, the solution time increases exponentially with the problem size, making it impractical for real-world applications. Previous studies employed general-purpose heuristic algorithms to minimize the maximum completion time. To enhance the efficiency and performance of these algorithms, hybrid algorithms or general-purpose heuristic algorithms incorporating local search methods have been employed to achieve effective and rapid solutions. This study integrated the concept of RL with VNS and introduced it into a GA, and proposes a method to acquire self-learning capabilities and adapt to the problem environment.

Prior studies either incorporated local search methods into general-purpose heuristic algorithms or introduced RL to select the neighborhood structure for local search, without combining RL, VNS, and GA in a single framework. Differing from previous studies, this study validated, through test problems, that the proposed method yields better solutions compared to past approaches. In addition to test problems, the algorithm proposed in this study was applied to real-world factory scheduling. When evaluated against the current scheduling results at Company D, the proposed algorithm successfully reduced up to 22.45% in the maximum completion time, demonstrating its practical viability and efficiency in real-world settings. It has accommodated different machine configurations and production constraints to maximize production capacity, and reduced the time required for manual scheduling.

This study integrated the concept of RL with VNS, and applied to a GA to enhance the algorithm's performance. The suggestions for future research are as follows:

(1) While this study integrated the concept of RL with VNS, further experimentation is needed to determine the optimal parameters for the GA. Therefore, if future research employs RL to select the parameter values for the GA, parameter experimentation may no longer be necessary, potentially reducing search time.
(2) This study considered only two common production constraints. Future research could explore additional production constraints to enhance the applicability of the proposed method to practical problems. Other RL methods could also be considered to improve the quality of the solution.
(3) This study focused solely on minimizing the maximum completion time. However, in practice, various performance indicators, such as delay time and the number of delayed jobs, must also be considered. Future research could expand this problem into a multi-objective problem to better align with real-world requirements.
(4) In practice, manufacturers could apply the proposed method to solve scheduling problems for bottleneck processes, thus improving overall production efficiency. Additionally, they could use the scheduling results of the bottleneck process to predict the material input time or calculate the output time, enabling more accurate delivery deadlines for customer orders.

Acknowledgments

The authors would like to thank the National Science and Technology Council, Taiwan, for financially supporting this research under Contract No. NSTC 112-2221-E-A49-107-MY2, NSTC 114-2221-E-A49-183-MY2, NSTC 114-2221-E-145-001-MY3, and NSTC 113-2221-E-145-002-MY2.

Conflicts of Interest: The authors declared that they have no conflicts of interest to this work.

ORCID:

https://orcid.org/0000-0001-6272-3560 (Yung-Chia Chang)
https://orcid.org/0000-0002-9630-7386 (Kuei-Hu Chang)

References

Abreu, L.R., & Prata, B.A. (2018). A hybrid genetic algorithm for solving the unrelated parallel machine scheduling problem with sequence dependent setup times. *IEEE Latin America Transactions, 16*(6), 1715-1722.

Alicastro, M., Ferone, D., Festa, P., Fugaro, S., & Pastore, T. (2021). A reinforcement learning iterated local search for makespan minimization in additive manufacturing machine scheduling problems. *Computers & Operations Research, 131*, 105272.

Balin, S. (2011). Non-identical parallel machine scheduling using genetic algorithm. *Expert Systems with Applications, 38*(6), 6814-6821.

Bernal, C.P., Salido, M.A., & Moya, C.M. (2025). Optimizing energy efficiency in unrelated parallel machine scheduling problem through reinforcement learning. *Information Sciences, 693,* 121674.

Chen, R.H., Yang, B., Li, S., & Wang, S.L. (2020). A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem. *Computers & Industrial Engineering, 149*, 106778.

Duan, H.P., Xiao, K.M., Liu, L.H., Chen, H.W., & Huang, H.B. (2025). REINFORCE with bound-guided gradient estimator

for the traveling salesman problem toward scale generalization. *Engineering Applications of Artificial Intelligence, 141*, 109819.

Ewees, A.A., Al-qaness, M.A.A., & Abd Elaziz, M. (2021). Enhanced salp swarm algorithm based on firefly algorithm for unrelated parallel machine scheduling with setup times. *Applied Mathematical Modelling, 94,* 285-305.

Ezugwu, A. E., & Akutsah, F. (2018). An improved firefly algorithm for the unrelated parallel machines scheduling problem with sequence-dependent setup times. *IEEE Access, 6,* 54459-54478.

Fallahi, A., Bani, E.A., & Varmazyar, M. (2025). Towards sustainable scheduling of unrelated parallel batch processors: A multiobjective approach with triple bottom line, classical and data-driven robust optimization. *Computers & Operations Research, 173,* 106863.

Grefenstette, J.J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics, 16*(1), 122-128.

Guo, Z.Y., Duan, J., & Yang, Y.Y. (2014). On-line multicast scheduling with bounded congestion in fat-tree data center networks. *IEEE Journal on Selected Areas in Communications, 32*(1), 102-115.

Kayhan, B.M., & Yildiz, G. (2023). Reinforcement learning applications to machine scheduling problems: a comprehensive literature review. *Journal of Intelligent Manufacturing, 34*(3), 905-929.

Li, Z.P., Wei, X.M., Jiang, X.S., & Pang, Y.W. (2021). A kind of reinforcement learning to improve genetic algorithm for multiagent task scheduling. *Mathematical Problems in Engineering, 2021,* 1796296.

Liu, C., Feng, X.F., Li, F., Xian, Q.L., Jia, Z.H., Wang, Y.H., & Du, Z.D. (2025). Deep reinforcement learning combined with transformer to solve the traveling salesman problem. *Journal of Supercomputing, 81*(1), 161.

Liu, F., & Zeng, G.Z. (2009). Study of genetic algorithm with reinforcement learning to solve the TSP. *Expert Systems with Applications, 36*(3), 6995-7001.

Malekshahi, M.S., & Hoseinpour, P. (2025). Integrating pricing and scheduling decisions in a production network with distance-price-sensitive orders. *International Journal of Systems Science: Operations & Logistics, 12*(1), 2443148.

Mir, M.S.S., & Rezaeian, J. (2016). A robust hybrid approach based on particle swarm optimization and genetic algorithm to minimize the total machine load on unrelated parallel machines. *Applied Soft Computing, 41,* 488-504.

Pei, Z., Wan, M.Z., & Wang, Z.T. (2020). A new approximation algorithm for unrelated parallel machine scheduling with release dates. *Annals of Operations Research, 285*(1-2), 397-425.

Peng, W.H., Wang, D.J., Yin, Y.Q., & Cheng, T.C.E. (2025). Multi-agent deep reinforcement learning-based truck-drone collaborative routing with dynamic emergency response. *Transportation Research Part E: Logistics and Transportation Review, 195*, 103974.

Pinedo, M.L. (2022). *Scheduling: theory, algorithms, and systems.* Springer Nature, Switzerland AG.

Rocha, P.L., Ravetti, M.G., Mateus, G.R., & Pardalos, P.M. (2008). Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times. *Computers & Operations Research, 35*(4), 1250-1264.

Shafiee, M., Amiri-Aref, M., & Klibi, W. (2025). The integration of shared renewable resources considering setup times for the parallel machine scheduling problem. *Computers & Industrial Engineering, 200*, 110828.

Shahrabi, J., Adibi, M. A., & Mahootchi, M. (2017). A reinforcement learning approach to parameter estimation in dynamic job shop scheduling. *Computers & Industrial Engineering, 110*, 75-82.

Shen, X., Lu, Z.Z., Li, L.T., Chen, Y.X., Li, X.., Wang, J.Y., & Yao, W. (2025). Multistrip stitching imaging mission planning method for SAR satellite regional mapping considering onboard energy consumption. *IEEE Transactions on Geoscience and Remote Sensing, 63*, 4700213.

Silva, M.A.L., da Silva, J.F., de Souza, S.R., & Souza, M.J.F. (2025). A scalability analysis of a Multi-agent framework for solving combinatorial optimization via Metaheuristics. *Engineering Applications of Artificial Intelligence, 142,* 109738.

Silva, M.A.L., de Souza, S.R., Souza, M.J.F., & Bazzan, A.L.C. (2019). A reinforcement learning-based multi-agent framework applied for solving routing and scheduling problems. *Expert Systems with Applications, 131*, 148-171.

Srinivas, M., & Patnaik, L.M. (1994). Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics, 24*(4), 656-667.

Sultan, M.M., & Pande, V.S. (2018). Automated design of collective variables using supervised machine learning. *Journal of Chemical Physics, 149*(9), 094106.

Thevenin, S., & Zufferey, N. (2019). Learning Variable Neighborhood Search for a scheduling problem with time windows and rejections. *Discrete Applied Mathematics, 261,* 344-353.

Vallada, E., & Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research, 211*(3), 612-622.

Wang, H., Lin, W.Z., Peng, T., Xiao, Q.G., & Tang, R.Z. (2025). Multi-agent deep reinforcement learning-based approach for dynamic flexible assembly job shop scheduling with uncertain processing and transport times. *Expert Systems with Applications, 270*, 126441.

Wang, H.B., & Alidaee, B. (2019). Effective heuristic for large-scale unrelated parallel machines scheduling problems. *Omega-International Journal of Management Science, 83*, 261-274.

Wang, W., Nelson, P. C., & Tirpak, T. M. (1999). Optimization of high-speed multistation SMT placement machines using evolutionary algorithms. *IEEE Transactions on Electronics Packaging Manufacturing, 22*(2), 137-146.

Wang, Y.C., & Usher, J.M. (2004). Learning policies for single machine job dispatching. *Robotics and Computer-Integrated Manufacturing, 20*(6), 553-562.

Watkins, C.J.C.H. (1989). Learning from delayed rewards. PhD thesis, University of Cambridge, England.

Xie, F.L., Li, K., Chen, J.F., Xiao, W., & Zhou, T. (2025). An adaptive large neighborhood search for unrelated parallel machine scheduling with setup times and delivery times. *Computers & Operations Research, 177,* 106976.

Yan, X., Wang, T., & Shi, X.F. (2025). Optimal scheduling on unrelated parallel machines with combinatorial auction. *Annals of Operations Research, 344*(2-3), 937-963.

Yi, T., Li, M.B., & Lei, D.M. (2023). A shuffled frog-leaping algorithm with Q-learning for unrelated parallel machine scheduling with additional resource and learning effect. *Journal of Intelligent & Fuzzy Systems, 44*(3), 5357-5375.

Yuan, Y., & Xu, H. (2015). Multiobjective flexible job shop scheduling using memetic algorithms. *IEEE Transactions on Automation Science and Engineering, 12(*1), 336-353.

© 2026 by the authors; licensee Growing Science, Canada. This is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC-BY) license (http://creativecommons.org/licenses/by/4.0/).