Journal of Industrial and Management Optimization

https://www.aimspress.com/journal/jimo

JIMO, 22(1): 182–213.
DOI:10.3934/jimo.2026007
Received: 17 July 2025
Revised: 16 October 2025
Accepted: 22 October 2025
Published: 18 November 2025

Research article

A genetic algorithm for tool-constrained scheduling with dynamic expedited orders in smart manufacturing

Yi-Han Chen[1], Hsi-Wen Wang[1], Chiung-Hui Tsai[2] and Chih-Hung Wu[1,*]

[1] Department of Electrical Engineering, National University of Kaohsiung, 700, Kaohsiung University Rd., Nanzih District, 81148, Kaohsiung, Taiwan

[2] Department of Information Engineering, Da-Yeh University, 168, Xuefu Rd., Dacun Township, 515006, Changhua County, Taiwan

* **Correspondence:** Email: Johnw@nuk.edu.tw; Tel: +886-7-5919752; Fax: +886-7-5919374.

Abstract: This paper addresses a practical scheduling challenge in smart manufacturing systems, where dynamic expedited orders and tool change constraints must be handled in real time. While deep learning and multi-objective optimization methods offer strong theoretical capabilities, they are often unsuitable for industrial deployment due to high computational demands, training costs, and lack of transparency. We propose a lightweight, interpretable genetic algorithm framework that incorporates a real-time insertion mechanism and tooling-aware encoding to minimize disruptions when handling expedited orders. The method jointly considers machine-tool compatibility, job priorities, and system stability. It is validated using real production data from a precision screw manufacturing plant, demonstrating superior performance in handling expedited orders and tool constraints. The proposed method requires no training, supports online adjustments, and is well-suited for deployment in real-world production settings.

Keywords: flexible job shop scheduling problem (FJSSP); expedited orders; tool constraints; genetic algorithm; job priority; scheduling decision support
Mathematics Subject Classification: 90B35, 68W50

1. Introduction

The rise of Industry 4.0 has fundamentally transformed the landscape of manufacturing, driving a shift toward smart factories characterized by real-time data exchange, autonomous decision-making, and highly flexible production systems [1, 2]. Smart manufacturing environments usually integrate diverse machine resources across technological generations. Conventional machines offer economic advantages despite their limited flexibility, while advanced CNC (computer numerical control) systems [3] provide

superior processing capabilities and accuracy at a higher cost. In these heterogeneous environments, efficient task allocation is critical for maximizing productivity. Additionally, tool changeovers present significant challenges across all machine types and are crucial in production scheduling. The process involves machine stoppage, tool installation, calibration, and validation, introducing non-productive time and potential quality variations [4]. These activities increase costs and reduce equipment utilization efficiency. In high-variety, low-volume production environments, tool changeover efficiency has a significant impact on overall productivity.

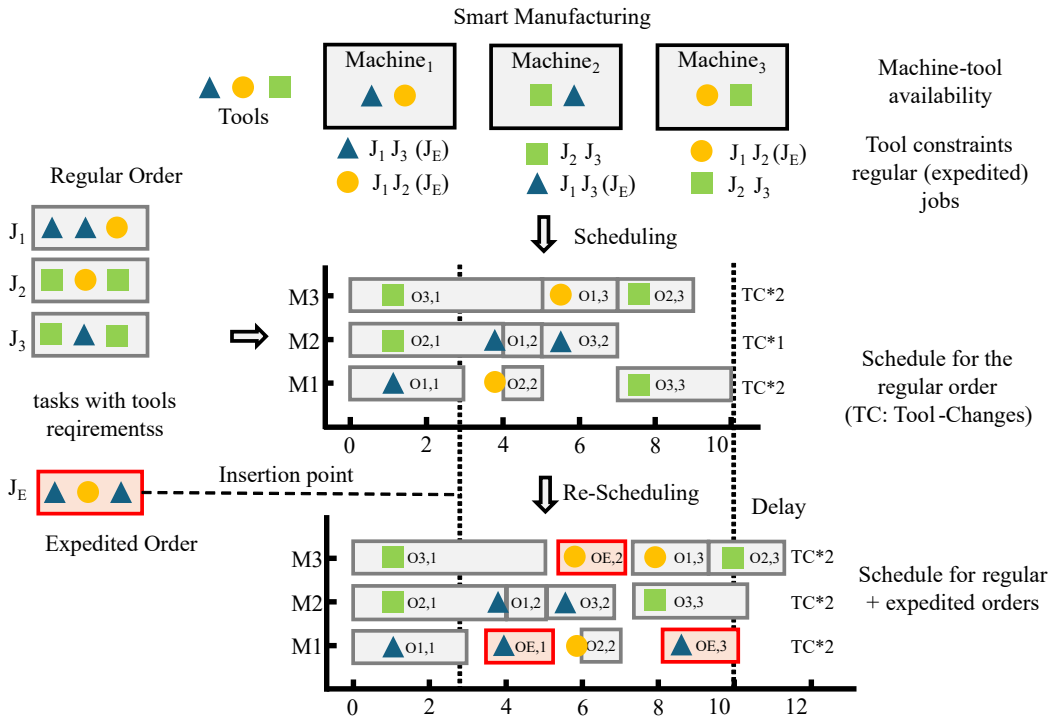


Figure 1. Conceptual framework illustrating the relationship among machine and tool constraints, regular/expedited orders, and scheduling efficiency. Machine M_1 provides two tools (△ and ◯), M_2 provides □ and △, and M_3 provides ◯ and □. Each job has three tasks to be processed, the tasks of J_1 are processed in sequence by tools △, △, and ◯, the ones in J_2 are by □, ◯, and □, and the ones of J_3 are by □, △, and □. The expedited job has three tasks that are processed by △, ◯, and △.

These issues become even more challenging when dealing with *expedited orders*, urgent, high-priority jobs that arrive unpredictably and demand immediate integration into the production schedule. Traditional static scheduling systems struggle to accommodate such disruptions, often leading to complete re-scheduling or inefficient insertion strategies, such as placing expedited orders at the beginning or end of the machining sequence [5, 6]. From a tooling standpoint, expedited orders are particularly problematic because they often introduce new machining requirements that may not align with the current tool setup. As a result, they may trigger unplanned tool changes, disrupt currently optimized tool groupings, increase machine idle time, and elevate the risk of setup errors. These cascading effects highlight the relationship between scheduling decisions and tooling constraints in smart manufacturing environments. Figure 1 illustrates the abovementioned application scenario, where three machines $M1$, $M2$, and $M3$ and three types of tools are used in the manufacturing. Initially, three

regular jobs (J_1–J_3) are scheduled and processed, as shown in the center of Figure 1. An expedited order (J_E) requires scheduling adjustments at time 3, and the re-scheduled result is shown at the bottom of Figure 1.

Graves [7] defines scheduling as "allocating available resources over a given period to achieve specified objectives." In practical manufacturing, this entails coordinating machines, labor, materials, and tools within tight operational constraints, including deadlines, processing routes, and machine capabilities. As the complexity of scheduling problems grows with the number of variables and dynamic conditions, traditional deterministic approaches become less effective. To address this, researchers have explored various metaheuristic algorithms. Previous studies have proposed various approaches to tackle the flexible job-shop scheduling problem (FJSSP), achieving notable success through methods such as teaching-learning-based optimization [8], adaptive rule generation with random forests [9], and memetic algorithms combining local search and crossover strategies [10]. While these methods have yielded promising results, they largely overlook the dynamic integration of expedited orders and fail to explicitly incorporate tool change considerations.

The cases studied in this paper are small and medium-sized manufacturers producing precision screws and fasteners. These factories represent real industrial settings where resources and equipment are limited, which makes scheduling decisions particularly critical. They frequently face expedited orders from customers, and full re-scheduling is often too slow to be practical. Timely completion of urgent and higher-value orders directly affects profitability and customer trust. These factors are essential for the survival of firms operating with narrow margins. Tooling costs further illustrate the practical challenges observed in these cases. Each manual tool change requires removal, calibration, and trial runs before production can resume. This results in downtime, material waste, and potential defects. The motivation for this research arises from these real factory constraints. An effective scheduling system must insert urgent jobs with minimal disruption to ongoing work, and must preserve current tool setups whenever possible. As stated in [11], changeover in production lines is a critical scheduling issue, and their MILP-based (mixed-integer linear programming [12]) framework for lot-sizing and scheduling in a beverage plant demonstrated that buffer tanks can effectively reduce overtime and improve productivity. In our study, the analogous issue is tool changes; thus, minimizing tool changeovers is identified as a key scheduling consideration.

This study addresses this gap by developing a practical scheduling framework that explicitly accounts for expedited orders and tooling costs. Genetic algorithms (GAs) are adopted as the underlying search method because they provide a balance between solution quality and computational efficiency [13], which is important for small and medium-sized factories where computational resources and IT expertise are limited. In our proposed method, the encoding represents both job sequencing and tool usage patterns, the fitness function adapts to shifting production priorities, and an insertion weight mechanism supports urgent job placement with minimal disruption. The framework is validated using real production data from a precision screw manufacturing plant, ensuring practical applicability and industrial relevance.

Recent studies have explored advanced techniques such as multi-objective optimization (MOO) [14, 15] and deep learning-based approaches including reinforcement learning and transformer models [16–19]. While these methods are theoretically powerful, they are less suited to small and medium-sized manufacturers. MOO requires maintaining Pareto fronts with heavy computation, specialized hardware, and dedicated IT staff. Deep learning requires substantial datasets, long training cycles, and continuous expert involvement. These demands create cost and deployment barriers for such manufacturers. In

contrast, the GA framework is simpler to implement, faster to compute, and easier to maintain, making it a practical and reliable scheduling tool for real factory environments.

The goals of this research are threefold:

- To enable rapid and flexible adjustment of production schedules in response to real-time demands, particularly expedited orders.
- To minimize the frequency and cost of tool changes by incorporating tooling constraints into the scheduling process.
- To develop a GA-based framework that optimizes job sequencing and tool utilization under dynamic conditions.

2. Related works

Several scheduling algorithms have been proposed to address various production scheduling problems. In single-machine scheduling, Pei *et al.* [20] derived the structural properties of the problem and applied optimization algorithms to solve it. For parallel-machine scheduling, they introduced a hybrid algorithm combining variable neighborhood search [21] and a gravitational search algorithm [22], demonstrating its efficiency and effectiveness. Lei *et al.* [23] proposed an improved artificial bee colony algorithm to solve the distributed unrelated parallel machine scheduling problem [24]. Their approach combines global and local searches during the employed bee and onlooker bee phases, and introduces a new method in the scout phase to minimize makespan and machine delays. In multi-stage scheduling, Umam *et al.* [25] addressed flow shop scheduling by integrating tabu search [26] with GA. They utilized a partially opposite population initialization to enhance search speed and leveraged tabu search to avoid local optima. The method effectively minimized the makespan across various instances and outperformed other algorithms. Gao *et al.* [27] tackled fuzzy job-shop scheduling problems [28] using differential evolution [29]. They enhanced the differential evolution's search capabilities by introducing a selection mechanism and demonstrated superior performance compared to other methods. Luo *et al.* [30] applied reinforcement learning (RL) through the deep Q-network method to solve the dynamic flexible job shop problem [17]. They proposed six composite scheduling rules and extracted seven state features for re-scheduling points, using deep Q-learning to train the model and validate its feasibility on various instances.

Regarding the methods of MOO for the FJSSP, several studies have been proposed. For example, Burmeister [14] applied a memetic NSGA-II to solve an energy-aware FJSSP in real-time factory settings. Chen *et al.* [15] proposed a multi-objective evolutionary algorithm based on multiple-neighborhoods local search for the multi-objective distributed hybrid flow shop scheduling problem [31]. The method employs a weighted mechanism during initialization to determine the best position for each job and uses multiple neighborhood operators to generate new solutions. Poor solutions are replaced, and an adaptive weight update mechanism is used to avoid local optima. Experimental results show that this method is highly effective for the hybrid flow shop scheduling problem. Although effective, such approaches often suffer from high computational complexity due to the maintenance of the Pareto front, which limits their responsiveness in real-time environments. In parallel, deep reinforcement learning (DRL) has gained popularity for dynamic scheduling. Luo [30] used DRL to handle job insertions in the FJSSP, and Wang [17] proposed a DRL-based multi-objective framework for uncertain

production environments. Ikonen *et al.* [32] applied RL to decide re-scheduling timing and computing resource allocation in dynamic scheduling, improving online decision-making efficiency. The end-to-end framework proposed by Lei *et al.* [16] learns policies that are faster than metaheuristics, and can generalize to large-scale instances for the FJSSP. Another is a hierarchical framework by Lei et al. [18] tackled the dynamic FJSSP in smart manufacturing, enabling near real-time decision-making for large-scale, uncertain production lines. Samouilidou *et al.* [33] integrated RL with MILP to reduce changeover times and computation in production scheduling. Lee *et al.* [34] embedded robust optimization into DRL, improving stability and solution quality under demand uncertainty. A comprehensive review by Ngwu [19] highlights the potential and limitations of DRL in manufacturing scheduling. However, DL-based methods require large-scale training data, long convergence times, and are difficult to interpret, reducing their applicability in real-time shop floor control.

While existing scheduling methods show strong performance, they are usually validated on benchmark datasets and focus on general objectives without explicitly addressing the combined challenges of dynamic expedited orders and tool changeovers. Advanced techniques such as MOO and DRL are theoretically powerful, but their implementation demands specialized hardware, large datasets, and expert personnel, which limits their applicability in small and medium-sized enterprises. Thus, there remains a research gap for a scheduling framework that is computationally efficient, easy to deploy, and able to incorporate the practical costs of both expedited orders and tool constraints. These limitations motivate the need for more lightweight, interpretable, and flexible approaches, such as the GA framework proposed in this study.

3. Problem formulation

3.1. Flexible job-shop scheduling

The flexible job-shop scheduling problem (FJSSP) is an extension of the classical job shop problem, where multiple machines with varying capabilities can process each operation. It involves determining the operation sequence and machine assignment to optimize objectives such as makespan, cost, or resource utilization under complex constraints. In general, an $n \times m$ FJSSP involves n jobs and m available machines, where the set of jobs is denoted as $J = \{J_1, J_2, ..., J_n\}$, and the set of machines as $M = \{M_1, M_2, ..., M_m\}$. Each job consists of multiple operations, and different machines can process each operation. For convenience, the following notations are used:

- $o_{i,j}$: the jth operation of job J_i.
- d_i: the due date of job J_i.
- $t_{i,j,k}$: the processing time required for operation $o_{i,j}$ on machine M_k.
- $s_{i,j,k}$: the start time of operation $o_{i,j}$ on machine M_k.
- wo_k: the operational wear of machine M_k.
- ws_k: the standby wear of machine M_k.
- by_k: the total standby time of machine M_k.
- e_i: the profit gained if job J_i is completed by its due date.

- p_i: the penalty incurred if job J_i is not completed by its due date.
- f_i: indicator of whether job i is completed by its due date.

For completing job J_i, all operations $o_{i,j}$ must be processed on appropriate machines. The processing time $t_{i,j,k}$ for each operation $o_{i,j}$ on machine M_k is fixed and known in advance. Each operation $o_{i,j}$ can be processed on one or more machines. Let $X_{i,j,k}$ denote whether operation $o_{i,j}$ is assigned to machine M_k, i.e., $X_{i,j,k} = 1$ if $o_{i,j}$ is assigned to M_k, otherwise $X_{i,j,k} = 0$. The assignment of operations is subject to the following constraints:

- Each job J_i is independent and has a fixed processing time $t_{i,j,k} > 0$.
- Operations within a job must follow precedence, that is, $s_{i,j,k} + t_{i,j,k} \leq s_{i,j+1,k}$.
- Each operation must have at least one available machine, $\sum_{i=1}^{n} \sum_{j=1}^{h} X_{i,j,k} \geq 1$.
- Each machine can only process one operation at a time, $\sum_{k=1}^{m} X_{i,j,k} = 1$.

This study considers three core objectives: minimizing completion time (C_{min}), minimizing machine wear (M_{min}), and maximizing profit (E_{max}). These metrics are integrated into the fitness function of the proposed GA framework. The objective function for minimizing total completion time is defined as:

$$C_{min} = \min \left\{ \max_{1 \leq i \leq n} \sum_{i=1}^{n} C_i \right\}, \tag{3.1}$$

where

$$C_i = \max_{1 \leq j \leq h} \left\{ \sum_{j=1}^{h} (s_{i,j,k} + t_{i,j,k}) \right\} \tag{3.2}$$

is the completion time for job J_i. The objective for minimizing machine wear is:

$$M_{min} = \min \left\{ \sum_{k=1}^{m} (Wo_k + Ws_k) \right\}, \tag{3.3}$$

where $Wo_k = \sum_{i=1}^{n} \sum_{j=1}^{h} (t_{i,j,k} \times wo_k)$ is the total operational machine wear of machine M_k and $Ws_k = (by_k \times ws_k)$ is the total standby wear for machine M_k. The objective for maximizing profit is:

$$E_{max} = \max \left\{ \sum_{i=1}^{n} E_i \right\}, \tag{3.4}$$

where $E_i = e_i - (f_i \times p_i)$ and $f_i = 1$, if $d_i - C_i \leq 0$ (job completed overdue), otherwise $f_i = 0$ (job completed on-time).

The FJSSP is classified as an NP-hard problem, with two critical decisions: the sequencing of jobs and the selection of machines. While the primary objective of the typical FJSSP is to minimize the makespan, this study further incorporates considerations such as job due dates, machine operating wear, idle energy consumption, etc. In the FJSSP, it is typically assumed that any job operation can be performed on any available machine; however, this study considers machine-tool compatibility, where operations are constrained by the specific tools each machine supports. These additional factors enhance the practical applicability of the FJSSP, but also increase the problem's complexity.

3.2. Tool constraints

The traditional FJSSP primarily focuses on optimizing job-machine assignments and operation sequencing. These models assume that tools are readily available and interchangeable without cost, neglecting the operational consequences of tool changes. In practice, however, tool switching incurs overhead in the form of setup adjustments, idle time, and potential scheduling delays.

To illustrate the impact of tool changes on scheduling efficiency, consider a simple example with two machines, M_1 and M_2, and three tools, t_a, t_b, and t_c. Machine M_1 can operate with tools t_a, t_b, and t_c, while machine M_1 supports tools t_a and t_c only. Assume there are three jobs: J_1 requires tool t_a, J_2 requires t_b, and J_3 also requires t_a. Three scheduling results are listed in Table 1. It can be observed that schedule A requires tool changes twice, schedule B requires one change, and schedule C requires no tool changes. This example demonstrates that different job sequences and machine assignments can result in different numbers of tool changes. Fewer tool changes are preferable, as they reduce non-productive time and increase machine availability.

Table 1. Comparison of scheduling options and tool change frequency.

Scheduling (Machine Assignment)	Tool Change Sequence	Tool Changes
A: $J_1(M_1) \to J_2(M_1) \to J_3(M_1)$	$M_1 : t_a \to t_b \to t_a$	2
B: $J_1(M_1) \to J_3(M_1) \to J_2(M_1)$	$M_1 : t_a \to t_a \to t_b$	1
C: $J_1(M_2) \to J_3(M_2)$, $J_2(M_1)$	No tool change on each machine	0

The cost associated with tool usage includes three components: (1) the number of tool changes, (2) the time required for each change, and (3) the wear or calibration overhead induced by tool switching. To account for these costs, the model introduces the following variables:

- tc_i: number of tool changes for completing Job J_i.
- tt: the average time required per tool change.
- Wt: machine wear per tool change (e.g., calibration or degradation effect).

The total completion time C_i of Job J_i in Eq. (3.2) is updated as:

$$C_i = \max_{1 \le j \le h} \left\{ \sum_{j=1}^{h} (s_{i,j,k} + t_{i,j,k}) + tc_i \cdot tt \right\} \tag{3.5}$$

Similarly, the machine wear objective in Eq. (3.3) is updated as:

$$M_{min} = \min \left\{ \sum_{k=1}^{m} (Wo_k + Ws_k) + \sum_{i} Wt \cdot tc_i \right\} \tag{3.6}$$

3.3. Expedited orders

In the traditional FJSSP, once a machine starts processing an assigned job, it cannot be interrupted and must complete the current job before switching to a new task. Expedited orders, on the other hand, are time-sensitive and demand immediate processing. Most conventional FJSSP models assume that all jobs are known in advance and that scheduling is performed in a static or semi-static manner. In

contrast, real manufacturing environments often encounter expedited orders that arrive dynamically and unpredictably, requiring on-the-fly adjustments to an already active schedule. These urgent jobs must be inserted in a way that minimizes interference with ongoing operations, which adds considerable complexity to the scheduling process. From a scheduling perspective, such scenarios involve not only handling job priorities, but also preserving machine-tool compatibility, avoiding unnecessary tool switches, and maintaining the stability of the existing schedule. A dynamic insertion mechanism is therefore critical to ensure the timely integration of urgent jobs while still satisfying due dates and resource constraints. If expedited orders are scheduled at the end of the production queue, their due dates may be missed. Conversely, inserting them too early can disrupt the planned sequence of regular jobs. Furthermore, it is often necessary to preserve the original machine-tool assignments for operations that have already been scheduled but not yet executed. These operations are partially committed to the production plan, and reassigning them may trigger additional tool changeovers, increased idle time, or setup errors. To address this challenge, the scheduling system must simultaneously consider both incoming expedited jobs and unfinished tasks from the existing schedule, while accounting for due dates, job priorities, and machine-tool constraints. Let J_i' be a new job and $o_{i,j}'$ be the jth operation of J_i'. The formulation of FSSP considering expedited orders uses the following notations.

- $to_{i,j}$: remaining time to complete operation $o_{i,j}$ of job J_i in the on-going schedule.
- $t_{i,j,k}'$, $s_{i,j,k}'$: processing and start time for $o_{i,j}'$ on M_k.
- d_i', e_i', p_i': due date, profit, and penalty of new job J_i'.
- t_s: time point of expedited insertion.

The remaining time for ongoing operations is

$$to_{i,j} = \begin{cases} s_{i,j,k} + t_{i,j,k}, & s_{i,j,k} > t_s \\ t_{i,j,k} - t_s, & s_{i,j,k} \le t_s \le s_{i,j,k} + t_{i,j,k} \\ 0, & s_{i,j,k} < t_s \end{cases} \tag{3.7}$$

Accordingly, the objective for completion time becomes

$$C_{min} = \min\left\{\max(C_i, C_i')\right\}, \tag{3.8}$$

where

$$C_i = \max_{1\le j\le h}\left\{\sum_{j=1}^{h} to_{i,j}\right\} \tag{3.9}$$

$$C_i' = \max_{1\le j\le h}\left\{\sum_{j=1}^{h}(s_{i,j,k}' + t_{i,j,k}') + \sum_{i=1}^{n} tc_i \cdot tt\right\} \tag{3.10}$$

The updated machine wear objective becomes

$$M_{min} = \min\left\{\sum_{k=1}^{m}(Wo_k + Wo_k' + Ws_k) + \sum_i W_t \cdot tc_i\right\}, \tag{3.11}$$

where $Wo_k = \sum_{i=1}^{n}\sum_{j=1}^{h} to_{i,j} \cdot wo_k$ is the machine wear for the remaining operations and $Wo'_k = \sum_{i=1}^{n}\sum_{j=1}^{h} t'_{i,j,k} \cdot wo_k$ is the machine wear for the inserted jobs. Finally, the objective for profit is updated as

$$E_{max} = \max\left\{\sum_{i=1}^{n}(E_i + E'_i)\right\}, \tag{3.12}$$

where $E'_i = e'_i - (f'_i \times p'_i)$, and the definition of f'_i is the same as f_i but for J'_i.

The three objectives, C'_{min}, E'_{max}, and M_{min}, used in this study were identified through discussions with practitioners in precision screw and fastener manufacturing and reflect the main concerns of small and medium factories. Expedited order handling involves not only short-term profit, but also long-term reputation, which cannot be quantified but remains vital. The framework, therefore, allows users to adjust weights so that scheduling decisions reflect practical priorities. The objectives are combined through a weighted sum rather than a Pareto-based MOO approach that demands heavy computation and specialized personnel, which are rarely available in small and medium factories. In contrast, the weighted-sum approach is simple, transparent, and adjustable, enabling users to balance cost, efficiency, and reputation in line with real industrial needs. In this context, generating a full Pareto front through MOO methods is unnecessary, since managers require workable solutions rather than exhaustive trade-off sets.

4. The proposed method

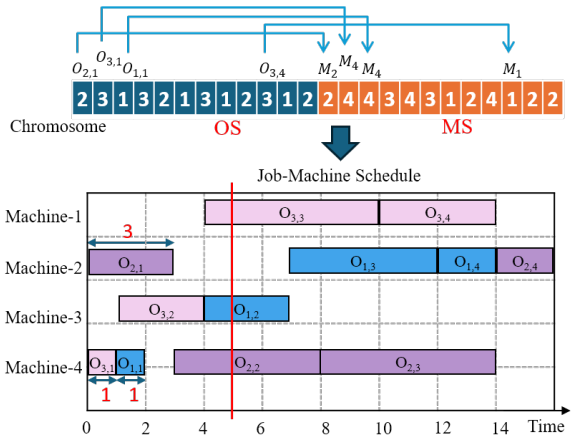
4.1. Chromosome encoding

The first step to implementing GA-based problem-solving is to design an appropriate encoding method. Solving FJSSP requires satisfying the sequential processing of job operations and their assignment to suitable machines. Therefore, the chromosome encoding must account for these two factors simultaneously. This study adopts a chromosome design inspired by the method proposed in [35]. Following the notations given in Section 3, a chromosome ξ for the $n \times m$ FJSSP consists of two parts, OS and MS, each having $\sum_{i=1}^{n} K_i$ integers. The OS part describes the processing sequence of all subtasks, $o_{i,1}, o_{i,2}, \ldots, o_{i,K_i}$, $1 \le i \le n$, wherein each integer i, $1 \le i \le n$, represents a subtask of job J_i to be processed and each integer i appears for K_i times. The jth occurrence of i, $1 \le j \le K_i$, denotes the jth subtask $o_{i,j}$ of J_i. The MS part is a task-machine assignment for the subtask of OS to m machines. Each integer j, $1 \le j \le m$, represents a machine in M. The kth integer j of MS indicates that machine M_j is assigned to process the kth subtask in OS. For example, a 3×4 FJSSP is considered, as shown in Figure 2. In Figure 2(a), three jobs are to be processed by four machines, where the integers in each row denote the operation time of the machine for completing the subtask, and '-' denotes that the machine is inapplicable for the subtask. Figure 2(a) presents a job-machine schedule arranged by a chromosome of length 2×12. Four 2's appear in the 1st, 5th, 9th, and 12th position of OS = [**2**,3,1,3,**2**,1,3,1,**2**,3,1,**2**], respectively, denoting that the J_2's subtasks $o_{2,\mathbf{1}}$, $o_{2,\mathbf{2}}$, $o_{2,\mathbf{3}}$, and $o_{2,\mathbf{4}}$ are processed by the machine represented by the 1st (M_2), 5th (M_4), 9th (M_4), and 12th (M_2) integer of MS = [**2**,4,4,3,**4**,3,4,2,**4**,1,2,**2**], respectively. The same job-machine assignment rule applies to all jobs. A job-machine schedule can be derived from this chromosome by utilizing the processing time information given in Figure 2(a), with the resulting timeline illustrated in the time table of Figure 2(b).

This encoding method directly maps jobs to machines, thereby capturing machine-selection decisions in the scheduling process. It also preserves solution feasibility during GA operations and allows task- or machine-specific information to be embedded, which facilitates the incorporation of tool constraints and expedited orders.

	Tasks	M_1	M_2	M_3	M_4
Job_1	$O_{1,1}$	2	3	-	1
	$O_{1,2}$	5	-	3	-
	$O_{1,3}$	-	5	7	6
	$O_{1,4}$	6	2	-	3
Job_2	$O_{2,1}$	-	3	6	-
	$O_{2,2}$	3	-	4	5
	$O_{2,3}$	2	3	-	5
	$O_{2,4}$	-	2	6	8
Job_3	$O_{3,1}$	1	-	5	7
	$O_{3,2}$	-	6	3	5
	$O_{3,3}$	6	2	-	4
	$O_{3,4}$	4	-	6	3

(a) Processing time of job-task-machine



(b) Chromosome vs. job schedule

Figure 2. Chromosome encoding for FJSSP.

4.2. *Fitness function*

As mentioned in Section 3.1, the objectives of the FJSSP include the minimization completion time C_{min}, and machine wear M_{min}, and the maximization of profit E_{max}, as defined in Eqs (3.1), (3.3), and Eq. (3.4), respectively. The fitness function for evaluating a chromosome ξ can be defined directly as follows:

$$F_{total} = \alpha_1 \times \text{Norm}(\frac{1}{C_{min}}) + \alpha_2 \times \text{Norm}(\frac{1}{M_{min}}) + \alpha_3 \times \text{Norm}(E_{max}), \tag{4.1}$$

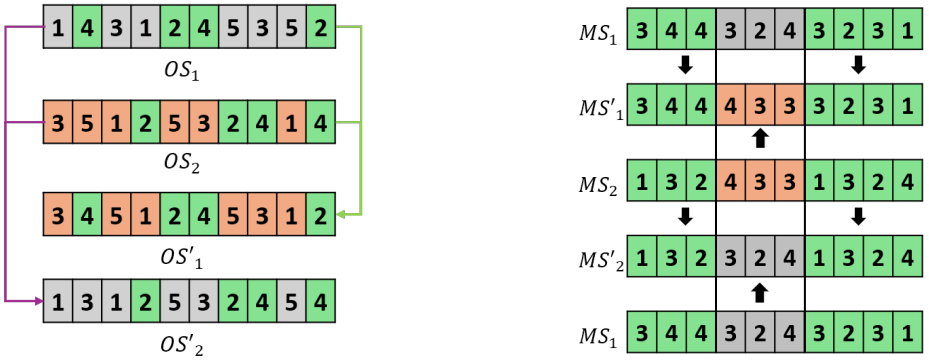
where Norm(·) is a normalization function that re-scales the input value to [0,1] so that C_{min}, M_{min}, and E_{max} are comparable. Moreover, the symbol α_i, $i = 1, 2, 3$ in Eq. (4.1), is the user-defined importance (weight) associated with C_{min}, M_{min}, and E_{max}, $\sum_i \alpha_i = 1$, $0 \leq \alpha_i \leq 1$. A larger weight presents much importance. With Eq. (4.1), a chromosome with a higher fitness value presents a better FJSSP schedule.

4.3. *The Genetic Operators*

Typical genetic operators such as mutation, crossover, and reproduction are also applied to the proposed chromosomes for the FJSSP. However, the task sequence for completing a job and the job-machine assignment must be kept.

- Selection (Reproduction): Chromosomes are ranked by descending fitness, and top individuals are selected using a hybrid of elitism and tournament selection. Elitism retains the top (population size × probability) chromosomes to preserve high-quality solutions. Tournament selection forms a mating pool by favoring fitter chromosomes, thereby preserving critical tasks during crossover and improving solution quality.

- Crossover: Crossover is applied to chromosomes in the mating pool to generate offspring. In the OS part, the precedence operation crossover (POX) is utilized, as shown in Figure 3(a). The crossover preserves task precedence while incorporating ω parameters to adjust task positions based on priority. Tasks with higher priority are more likely to retain their positions. In the MS part, two-point crossover is applied to exchange gene segments between parents, ensuring offspring remain valid solutions. This method enhances diversity by exploring a broader solution space (Figure 3(b)).
- Mutation: Mutation is applied separately for operation sequencing and machine assignment to increase diversity. In the OS part, multi-point mutation randomly alters selected gene positions to adjust task processing orders, as illustrated in Figure 4. In the MS part, a new processing machine is randomly selected from the set of feasible machines for the task.



(a) POX crossover (J_1={2,4}, J_2={1,3,5}) **(b)** Two-point crossover

Figure 3. Chromosome crossover.

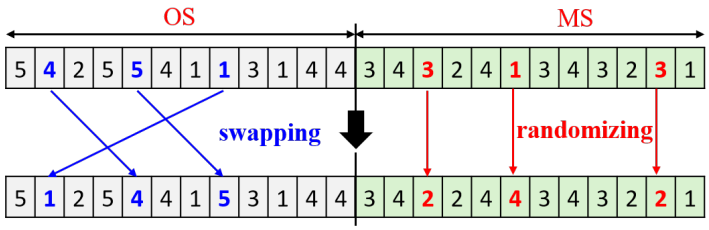


Figure 4. Mutation process illustration.

4.4. Tool constraints

The chromosome representation defined in Section 4.1 does not consider machine-tool constraints; any available machines can process jobs. This study retains the aforementioned chromosome representation and designs a tool-constraint (TC) template to check the job-machine assignment in the chromosome with tool constraints. A TC template has the same structure and length as a chromosome. Each element in the OS part of a TC template is the tool which must be used by task $o_{i,j}$. Each element in the MS part is a link to the machine-tool availability table, denoting the tool provided by the machine. Suppose the machines in Figure 2(a) provide two types of tools, A and B. The example given in Figure 5 presents a TC-template for the job-task-machine status of Figure 2. Referring to the

chromosome of Figure 2(b), Task $o_{1,1}$ is assigned to M_4, which provides both Tool A and Tool B, and is capable of processing $o_{1,1}$. Task $o_{3,2}$ needs to be processed by Tool A; however, it is assigned to M_3, which provides Tool B only, and cannot process $o_{3,2}$. By referring to the TC-template, the feasibility of a job-machine assignment can be checked. A chromosome with a feasible job-machine assignment is reserved; otherwise, it is discarded.

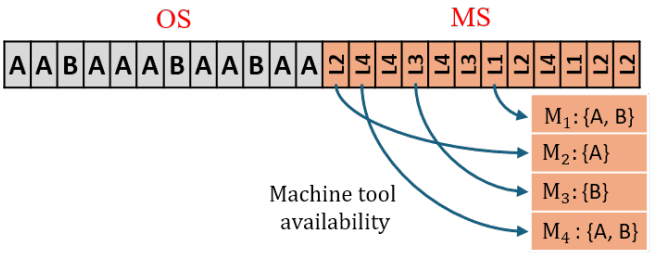


Figure 5. A TC template example

4.5. Re-Scheduling with expedited orders

the typical FJSSP starts with all jobs without being machined, and when using GA for the FJSSP, all chromosomes are initialized randomly. However, expedited orders are considered when the factory is running a determined job-machine assignment/schedule wherein assigned jobs occupy some machines, and the machined tasks can not be suspended or stopped. To make the proposed GA-based method workable for both the FJSSP with typical orders as well as with expedited orders, the following mechanisms are designed.

4.5.1. Job priority

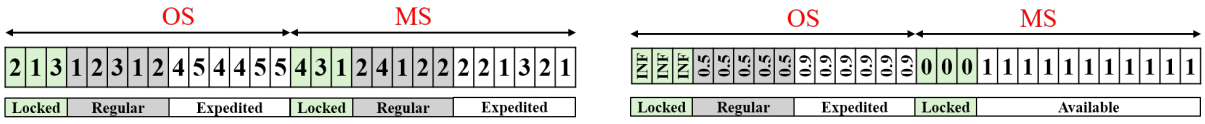
Suppose that the regular orders are scheduled as Figure 2 and an expedited order containing two jobs, J_4 and J_5, is inserted at time 4. When the expedited order is considered, the completed tasks, $o_{1,1}$, $o_{2,1}$, $o_{3,1}$, and $o_{3,2}$, can be ignored in the re-scheduling process. The working-on tasks, $o_{1,2}$, $o_{2,2}$, and $o_{3,3}$, are currently being machined on M_3, M_4, and M_1, respectively. These tasks can be suspended or abandoned to increase the availability of machines.

However, suspending or abandoning working-on tasks may increase cost. Scheduled and waiting-for processing tasks can be re-scheduled, and their assigned machines can be reassigned. Notice that a working-on task that is partially completed does not allow for being interrupted or transferred to another machine, because mid-task transfers would require re-fixturing, recalibration, and additional quality checks and introduce significant costs, delays, and risks of defects. The proposed GA framework only adjusts tasks that have not yet started, while allowing on-going operations to finish uninterrupted. This design is consistent with industrial practice and avoids impractical assumptions about mid-task migration.

In this study, the chromosome considering expedited orders is of the same form as Figure 2. The length of such a chromosome is twice the length of all uncompleted tasks, including those that are being worked on, waiting for, and expedited. Assume that J_4 and J_5 both consist of three tasks. The chromosome for describing such a situation is depicted in Figure 6(a). The locked part is working-on tasks and machines, the regular part is scheduled waiting for tasks and machines, and the expedited part is for expedited tasks.

For scheduling a regular FJSSP, all jobs are assumed to be of equal importance. Expedited orders, which are typically rushed, may be more critical than regular tasks and should be completed as soon as possible. To distinguish the importance of jobs, a weight ω_i is assigned to each job J_i. The value of ω_i is defined by [0,1], with 1 as the most important and 0 the least. Additionally, tasks and machines that are currently in use are possibly locked and should not be changed unless absolutely necessary. In practical applications, some machines may be suspended for maintenance or other purposes. The study considers all these factors by introducing the job-priority (JP) template of chromosomes.

A JP template has the same structure and length as a chromosome. Each element in the OS part of a JP template is the weight $\omega_{i,j}$ of each task $o_{i,j}$, and by default, the subtasks of Job J_i have an equal weight, $\omega_{i,j} = \omega_i$. INF means the task is locked and can not be re-scheduled. The MS part of a JP template presents the availability of machines. An element in the MS part is 0, which means the machine is assigned to a working-on job; 1 means the job-machine assignment is not decided. Figure 6(b) presents a JP-template for the job-task-machine status of Figure 6(a).



(a) Regular and expedited jobs **(b)** JP-template

Figure 6. Chromosome encoding for expedited orders.

4.5.2. Fitness adjustment

When the expedited orders are considered in the GA-encoded JFSSP, several factors are affected by the insertion of new jobs, such as the availability of machines, the delays caused to pre-scheduled jobs, and the impact on profits (or penalties). Due to the occupation of workable machines by scheduled jobs, the availability of machine resources is reduced, and the completion of some jobs may be delayed, resulting in profit loss or penalties. These factors correspond to constraints or adjustments in the GA operations, mainly including job completion time (C_{min}) and profit (E_{max}) in the fitness function (Eq. (4.1)). For machine wear (M_{min}), as it inevitably increases with the addition of more jobs, the original objective function can still be used without modification.

According to the definition of expedited orders, critical jobs should be completed sooner. Thus, jobs with higher importance completed quickly will yield higher fitness values, while those completed more slowly will result in lower fitness values. Based on these characteristics, the design of the two affected objective functions in Eq. (3.8) is tailored accordingly. The job completion time (C_{min}) is first considered if it is affected by the expedited order. If the expedited order does not cause any delays, the original schedule can proceed without modifications. However, if delays occur, the completion time must be extended accordingly.

Let w_i be the priority (weight) associated with job J_i's completion time. The objective function, considering the new completion time, is modified as follows:

$$C'_{min} = \frac{1}{C_{min}} + \sum_{i=1}^{n} f_i \times (\omega_i \times \frac{1}{C_{min}} \times \frac{TC_i}{C_i}) \tag{4.2}$$

$$= \quad (1 + \sum_{i=1}^{n} f_i \times \omega_i \times \frac{TC_i}{C_i})/C_{min}, \tag{4.3}$$

where f_i is a selection function, as used in Eq. (3.4). The value $\frac{TC_i}{C_i}$ is the ratio of J_i's ideal (shortest) competing time, i.e., without any delays or time-waste in waiting for available machines for completing all $o_{i,j}$. TC_i=$\sum_j(\min_k\{t_{i,j,k}\})$ denotes the ideal total time required to complete all operations of job J_i. For example, in Figure 2, Job_1 can theoretically be completed in 11 units of time if each operation is processed consecutively without delays: $O_{1,1}$ on M_4 (1 unit), $O_{1,2}$ on M_3 (3 units), $O_{1,3}$ on M_2 (5 units), and $O_{1,4}$ on M_2 (2 units). This yields $TC_1 = 11$. However, in actual scheduling, the preferred machine may not always be available for a given operation, and waiting times inevitably occur. For instance, $O_{1,2}$ experiences a 2-unit waiting time after $O_{1,1}$, resulting in an actual completion time $C_1 = 13$. The ratio TC_i/C_i is therefore used to indicate whether a job is executed with high priority and minimal delay; a higher ratio reflects a schedule that more closely respects the job's priority. The fraction $\frac{1}{C_{min}}$ is for the adjustment of consistency with other factors.

Expedited orders usually come with higher profits. Thus, the fitness function considers maximizing all profits gained from the completion of all jobs, including the expedited ones. The weighted profit objective E'_{max} is considered, as shown in Eq. (4.4).

$$E'_{max} = E_{max} + \sum_{i=1}^{n} f_i \times (\omega_i \times e_i \times \frac{TC_i}{C_i}) \tag{4.4}$$

Notice that the proposed method can be used directly for regular job scheduling by setting the chromosome and the fitness functions. For example, Eq. (4.3) and Eq. (4.4) can be reduced to Eq. (3.1) and Eq. (3.4), respectively, by setting the parameters associated with expedited jobs, such as C'_i and $W'o_k$, as 0. The proposed fitness function can be used for scheduling both regular and expedited jobs.

4.6. Processing flow

Our proposed method for the FJSSP, considering expedited orders and tool constraints, is depicted in Figure 7 and Algorithm Our Proposed GA Framework "Our Proposed GA Framework". The method can be used for regular job scheduling as well as for expedited orders. For scheduling regular jobs, chromosomes are encoded as OS and MS parts, and GA optimizes schedules based on minimizing makespan, machine wear, and maximizing profit, guided by the TC template to ensure feasibility. Regular jobs are initially treated equally in the same priority in the JP template. Chromosomes are initialized randomly and processed by a regular GA. Upon completion, this phase yields a baseline optimized schedule for regular jobs.

When expedited orders arrive, the scheduling system dynamically adjusts by classifying jobs into completed, on-going, unprocessed, and expedited categories. Completed tasks are locked, on-going tasks are minimally adjusted, and new expedited orders are prioritized explicitly by setting the JP template that assigns higher weights. Chromosomes are extended by considering the on-going and new expedited jobs, and initialized with on-going tasks assigned to the machines in the insertion time and expedited tasks assigned to workable machines randomly. The schedule is then re-evaluated using updated fitness criteria, explicitly incorporating job priority and tool constraints. The genetic operators are applied as regular GA for re-optimizing the schedule, balancing high-priority expedited orders with minimal disruptions, resulting in an integrated, feasible, and optimized production schedule.

Algorithm Our Proposed GA Framework

function GA-FJSSP-EXPEDITED-ORDER-TOOL-CONSTRAINS(J, M, TC, $\alpha_1, \alpha_2, \alpha_3, e, S, t_s$)
  if e **then**
    Classify tasks in S at t_s (completed, ongoing $to_{i,j} > 0$, waiting); Extend J with unfinished + expedited;
    Create JP with $\omega_{i,j}$ (high for expedited), machine availability; ▷ Adjust for expedited mode
  else
    Set equal $\omega_{i,j} = 1$ in JP; ▷ Regular mode
  end if
  Encode chromosomes ξ as OS + MS;
  for $i = 1$ to pop_{size} **do**
    if e **then**
      $\xi \leftarrow$ Extend S + random for expedited (feasible with respect to TC/JP);
    else
      $\xi \leftarrow$ Random feasible OS/MS;
    end if
  end for ▷ Initialize population
  repeat
    for each ξ **do**
      Decode to compute $C_{\min}$, $M_{\min}$, $E_{\max}$;
      if e **then**
        Adjust C'min (Eq.15), E'max (Eq.16) with ω_i;
      end if
      $F_{total} \leftarrow \sum \alpha_i \cdot \text{Norm}(obj_i)$ (Eq.13);
    end for ▷ Fitness evaluation
    Select elites + tournament pairs;
    for each pair **do**
      if rand $< pr(x)$ **then**
        Crossover: POX(OS, ω) + two-point(MS);
      end if
      if rand $< pr(m)$ **then**
        Mutate: Multi-point swaps (feasible w.r.t. TC/JP);
      end if
    end for ▷ Genetic operators
  until max generations or convergence;
  return Best schedule from top ξ;
end function

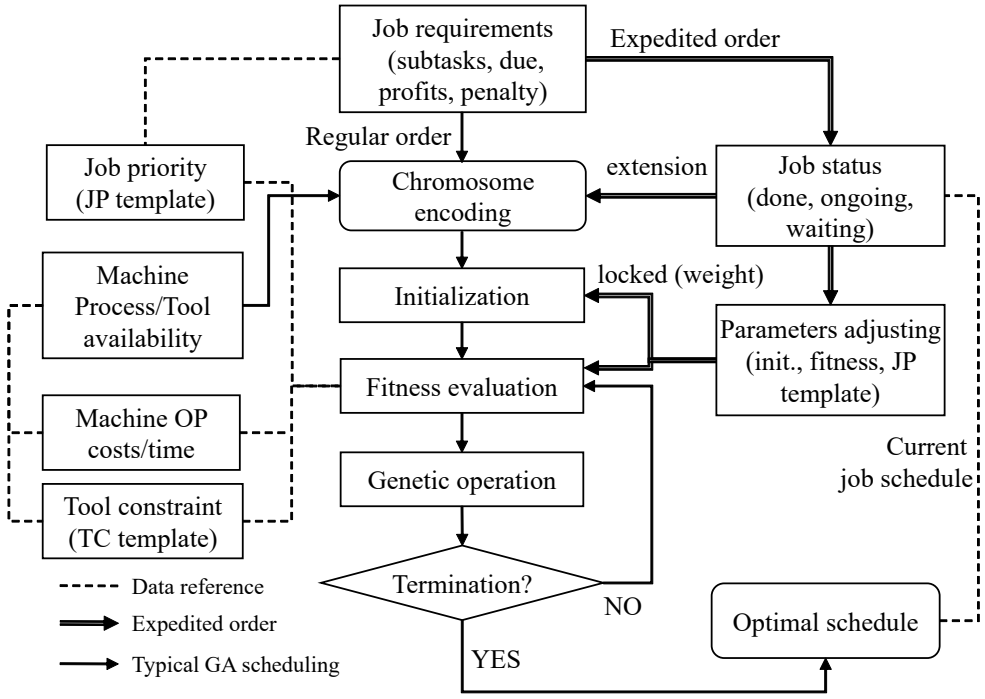


Figure 7. The processing flow of the proposed method

5. Experiments

The following section introduces the experimental environment and the datasets used in this study. Various combinations of GA parameters, fitness function parameters, and the proposed designs from this research were tested to conduct experiments.

5.1. Datasets and test environments

This study addresses a specialized scheduling problem that involves both expedited order insertion and tool change constraints. However, there are currently no publicly available benchmark datasets that reflect these requirements. Existing test instances for the FJSSP are not applicable to this research. Therefore, this study conducts experiments using real-world scheduling data obtained from a precision screw manufacturing factory specializing in aerospace components. The dataset has been sanitized to remove proprietary business information, and has been appropriately transformed for research purposes. Table 2 presents each machine's processing cost along with the set of processes it can perform under different tool types. The factory operates 21 machining units (M_1–M_{21}), each capable of performing different types of processes. These machines support various tool configurations and allow for tool changes depending on the processing requirements. Each machine can handle unique processes using the different tools, depending on its configuration. The processing time and operation cost of each machine vary accordingly. For example, machine-1 (M_1) has a processing cost of 100, and can perform process P_{11} using Tool A in 22 time-units, and process P_{11} using Tool B in the same processing time. M_1, M_2, and M_3 are similar machines for handling tools and processes, but vary in operation costs. Similarly, M_{14} supports only the process P_{15} using Tool A in 57 time-units.

Table 2. Machine operational cost and tool constraints.

Machine (Cost)	**Tool A (Process, Time)**	**Tool B (Process, Time)**
M_1(100), M_2(110), M_3(120)	(P_{11}, 22)	(P_{11}, 22)
M_4(150), M_5(155), M_6(160)	(P_{12}, 36)	(P_{12}, 36)
M_7(165), M_8(160), M_9(165)	(P_{12}, 36)	(P_{12}, 36)
M_{10}(350)	(P_{13}, 6), (P_{15}, 57)	(P_{13}, 6), (P_{14}, 27)
M_{11}(400)	(P_{13}, 6), (P_{14}, 27), (P_{15}, 57)	(P_{13}, 6), (P_{14}, 27)
M_{12}(450), M_{13}(450)	(P_{13}, 6), (P_{14}, 27), (P_{15}, 57)	(P_{13}, 6), (P_{14}, 27), (P_{15}, 57)
M_{14}(250)	(P_{15}, 57)	–
M_{15}(350), M_{16}(300)	(P_{35}, 6), (P_{45}, 6)	(P_{13}, 6)
M_{17}(220), M_{18}(230)	(P_{45}, 6), (P_{55}, 6)	–
M_{19}(300), M_{20}(280), M_{21}(285)	(P_{16}, 30)	(P_{16}, 30)

Table 3 summarizes the job list used in this experiment. A total of ten jobs are included, each defined by its machining type, a sequence of required operations (denoted as OP-1 through OP-6), and parameters such as due date, profit, and late penalty. For example, Job 3 requires being processed by Tool A and must undergo the process sequence, P_{11}, P_{12}, P_{14}, P_{55}, and P_{16}. The job has a due date of 218 time units, yields a profit of 160 if completed on time, and incurs a penalty of 16 if delayed.

Table 3. The test job set.

Job	Type	OP-1	OP-2	OP-3	OP-4	OP-5	OP-6	Tasks	Due	Profits	Penalty
Job_1	A	P_{11}	P_{12}	P_{35}	P_{14}	P_{55}	P_{16}	6	169	300	30
Job_2	A	P_{11}	P_{12}	P_{13}	P_{14}	P_{55}	P_{16}	6	222	330	33
Job_3	A	P_{11}	P_{12}	–	P_{14}	P_{55}	P_{16}	5	218	160	16
Job_4	B	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	6	349	250	25
Job_5	A	P_{11}	P_{12}	P_{35}	P_{45}	P_{55}	–	5	99	210	21
Job_6	B	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	6	215	340	34
Job_7	A	–	P_{12}	P_{35}	P_{45}	P_{55}	P_{16}	5	162	240	24
Job_8	A	P_{11}	P_{12}	P_{13}	P_{14}	P_{55}	P_{16}	6	214	300	30
Job_9	A	P_{11}	P_{12}	P_{35}	P_{45}	P_{55}	P_{16}	6	189	350	35
Job_{10}	A	P_{11}	P_{12}	P_{35}	P_{14}	P_{15}	P_{16}	6	243	310	31

All experiments were conducted on a 12-core computer equipped with an Intel(R) Core(TM) i7-12700 CPU@2.30GHz running Windows 11. The proposed methods were implemented in Python in a virtual environment created using pipenv and the Matplotlib library.

5.2. Experiment A: determining GA parameters

Genetic-based methods work with many parameters. For determining an effective set of GA parameters, thoroughly conducting experiences with all combinations of parameters and weights may be helpful but impractical. In this experiment, various combinations of probabilities of generic operators, $pr(m)$ and $pr(x)$, are used to evaluate the performance of regular scheduling. The length of the chromosome is $57 \times 2 = 114$, which describes the job-machine assignment for the 57 tasks associated with the ten jobs. Each gene in the OS part represents a job, i.e., 1–10, and the one in the MS part represents a machine, i.e., 1–21. According to the tool constraints, job-machine assignments with improper tools are discarded. The time delays and wear cost caused by tool changes are considered in the fitness function, as denoted in Eq. (3.5)–Eq. (3.6). The importance of all tasks is set as 1 in the JP template, as they are considered of equal importance. The time and wear cost for tool changes are set as 2 and 1, respectively.

In the following experiments, PG (performance gain) evaluates the improvement of a schedule against a reference schedule.

$$PG(a, f) \quad = \quad \frac{a - f}{a}, \tag{5.1}$$

where a is a reference value and f is the performance to be compared, and both are for the same object. PG can be applied for C_{min}, E_{max}, and M_{min} as PG_C, PG_M, and PG_E, respectively, and the overall performance gain is denoted as TPG, as shown in Eq. (5.2).

$$TPG = \alpha_1 \times (-1 \times PG_C) + \alpha_2 \times (-1 \times PG_M) + \alpha_3 \times PG_E \tag{5.2}$$

The test dataset listed in Table 3 was evaluated for regular scheduling (no expedited orders) with $\alpha_1 = 0.3$, $\alpha_2 = 0.3$, $\alpha_3 = 0.4$, and 100 populations of chromosomes for 10,000 generic iterations. Each test is repeated 10 times with the same parameter settings. The evaluation results are shown in Table 4. The experimental results show that the best $pr(x)$ is 0.85, and $pr(m)$ is 0.30.

Table 4. Performance evaluation with various $pr(x)$ and $pr(m)$.

(a) $pr(m)$=0.50 and various $pr(x)$

$pr(x)$	C_{min}	M_{min}	E_{max}	PG_C	PG_M	PG_E	TPG
*1.00	300	419	1580	0.000	0.000	0.000	0.000
0.60	352	465	1448	0.173	0.111	-0.084	-0.123
0.65	343	454	1340	0.143	0.085	-0.152	-0.127
0.70	328	426	1480	0.093	0.018	-0.063	-0.058
0.75	335	436	1635	0.117	0.041	0.035	-0.041
0.80	319	**402**	1786	0.063	-0.039	0.130	0.035
0.85	**292**	**402**	**2020**	**-0.027**	**-0.039**	**0.278**	**0.115**
0.90	297	405	1956	-0.010	-0.034	0.238	0.094

*: the reference experiment

(b) $pr(x)$=0.85 and various $pr(m)$

$pr(m)$	C_{min}	M_{min}	E_{max}	PG_C	PG_M	PG_E	TPG
*0.50	**292**	402	2020	0.000	0.000	0.000	0.000
0.05	305	424	1580	0.045	0.053	-0.218	-0.105
0.10	313	422	1495	0.072	0.049	-0.260	-0.127
0.15	297	437	1950	0.017	0.086	-0.035	-0.046
0.20	299	428	1856	0.024	0.063	-0.081	-0.056
0.25	300	402	1756	0.027	-0.002	-0.131	-0.052
0.30	**292**	**400**	**2220**	**0.000**	**-0.005**	**0.100**	**0.035**
0.35	303	426	2018	0.038	0.058	-0.001	-0.032

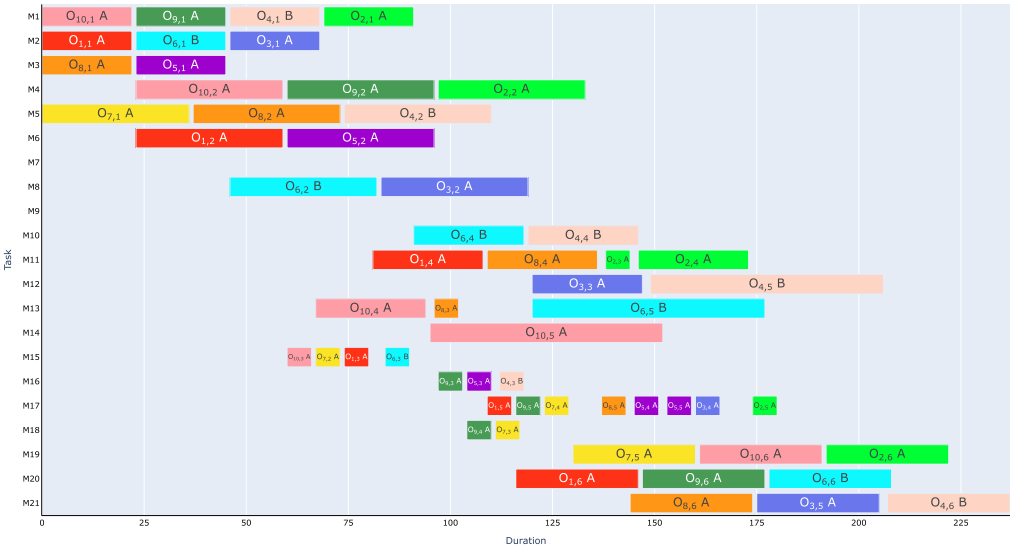


Figure 8. Job-machine schedule generated for the jobs in Table 3.

Figure 8 illustrates the scheduling result for regular orders under the best parameter settings. The earliest completed job was J_1 (at time 146), while the latest was J_4 (at time 237). Machines M_4 and M_5 were the most heavily utilized due to their relatively low operating costs (150 and 155, respectively), while M_7 and M_9, with the highest cost (165), remained idle to minimize total machine expenses. Among machines with similar cost levels, M_{17} (cost 220) received a higher workload than M_{18} (cost 230), reflecting cost-sensitive scheduling behavior. Job J_5, which had the tightest due date (99), was completed at time 159 without delay, and all other jobs were also completed on time. Job J_6 exhibited the highest continuity in processing, with no idle gaps between its operations. A total of 12 tool changes occurred during the schedule at $M_1(2)$, $M_2(2)$, $M_5(1)$, $M_8(1)$, $M_{12}(1)$, $M_{13}(1)$, $M_{15}(1)$, $M_{16}(1)$, $M_{20}(1)$, and $M_{21}(1)$, including both transitions from tool type A to B and vice versa.

5.3. Experiment B: inserting expedited orders

This experiment is designed to validate the proposed method's capability in handling expedited orders and to evaluate its robustness under different levels of job importance. The scheduling is conducted using the optimal GA parameters identified in Section 5.2. Three expedited jobs are assumed to arrive during an ongoing production schedule. The detailed job definitions, including their process routes, due dates, profits, and penalties for lateness, are summarized in Table 5.

Suppose that the insertion of the expedited jobs occurs at time point 50 in the schedule presented in Figure 8. At that moment, some operations (e.g., $o_{10,1}$, $o_{9,1}$, $o_{1,1}$, $o_{6,1}$, $o_{8,1}$, $o_{5,1}$, $o_{7,1}$) have already been completed, while others (e.g., $o_{4,1}$, $o_{3,1}$, $o_{10,2}$, $o_{8,2}$, $o_{1,2}$, $o_{6,2}$) are currently in progress. The remaining tasks are scheduled but have not yet started processing.

Table 5. Expedited orders: Three jobs inserted into the schedule.

Job	Type	OP-1	OP-2	OP-3	OP-4	OP-5	OP-6	Tasks	Due	Profits	Penalty
Job_{11}	A	P_{11}	P_{12}	P_{35}	P_{45}	P_{55}	P_{16}	6	116	250	25
Job_{12}	B	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	–	5	158	240	24
Job_{13}	B	P_{11}	P_{12}	–	P_{14}	P_{15}	P_{16}	5	182	190	19

To accommodate the expedited jobs, the chromosome structure must be expanded. Originally, there were 57 tasks, of which seven had been completed. The insertion adds 16 new tasks, while the number of machines remains unchanged at 21. Thus, the new chromosome length becomes $66 \times 2 = 132$. Tasks currently in progress (i.e., $o_{4,1}$, $o_{3,1}$, $o_{10,2}$, $o_{8,2}$, $o_{1,2}$, $o_{6,2}$) are preserved as locked operations in the chromosome. Each gene in the OS part represents a job identifier from 1 to 13, while each gene in the MS part denotes one of the 21 available machines. Assignments for completed tasks are removed, as are infeasible machine-tool pairings. Also, the time and wear cost of tool changes are considered as those in Experiment A. Based on the JP-template design, the importance of in-progress tasks is set to infinity to prevent re-scheduling. The priority values of tasks that have been scheduled but not yet started, as well as newly inserted expedited jobs, can be adjusted by the user.

The goal of this experiment is to assess the effectiveness of the proposed priority mechanism ω for expedited orders. The fitness function uses $\alpha_1 = 0.4$, $\alpha_2 = 0.3$, and $\alpha_3 = 0.3$, assigning slightly higher weight to minimizing makespan ($C_{\min}$), while still considering machine wear ($M_{\min}$) and overall profit ($E_{\max}$). Three levels of importance are evaluated for the expedited jobs: $\omega = 1$ (very important), $\omega = 0.5$ (equally important to regular jobs), and $\omega = 0.1$ (less important than regular jobs). These weight

combinations are assigned to Job_{11}–Job_{13}. The results are summarized in Table 6. For comparison, the case where $\omega = 0$, meaning that all expedited jobs are ignored until all regular jobs are completed, is treated as the baseline.

Table 6. Priority combinations for expedited orders: ω=1, 0.5, 0.1.

Job importance			Fitness values			Completing time		
ω_{11}	ω_{12}	ω_{13}	C_{min}	M_{min}	E_{max}	Job_{11}	Job_{12}	Job_{13}
0.0	0.0	0.0	227	340	2271	222	185	227
1.0	0.5	0.1	227	340	2271	**134**	172	227
1.0	0.1	0.5	211	336	2415	158	195	185
0.5	1.0	0.1	227	323	2613	164	**134**	186
0.5	0.1	1.0	272	327	2181	182	173	158
0.1	1.0	0.5	257	341	2334	172	**135**	154
0.1	0.5	1.0	271	346	2163	271	176	**135**

The results demonstrate that assigning $\omega = 1$ leads to significantly earlier completion times for expedited jobs, confirming the effectiveness of high-priority insertion under the condition $\alpha_1 > \alpha_2 = \alpha_3$. When $\omega = 0.5$, the expedited jobs are still effectively prioritized, with completion times falling between those of regular jobs. In contrast, when $\omega = 0.1$, only a slight improvement is observed, and job completion becomes more influenced by other factors. Notice that job priorities (ω_i) guide but do not solely determine execution order. In Table 6, Job 12 finishes before Job 11 because it could be processed immediately with the required tool, while scheduling Job 11 first would have caused extra tool changes and idle time. The method provides scheduling recommendations rather than fixed rules, and decision makers can adjust weights to enforce different trade-offs, demonstrating the flexibility of the fitness function design. Due to space limitations, only the scheduling result when $\omega_{11} = 1.0$, $\omega_{12} = 0.5$, and $\omega_{13} = 0.1$ is presented in Figure 9.

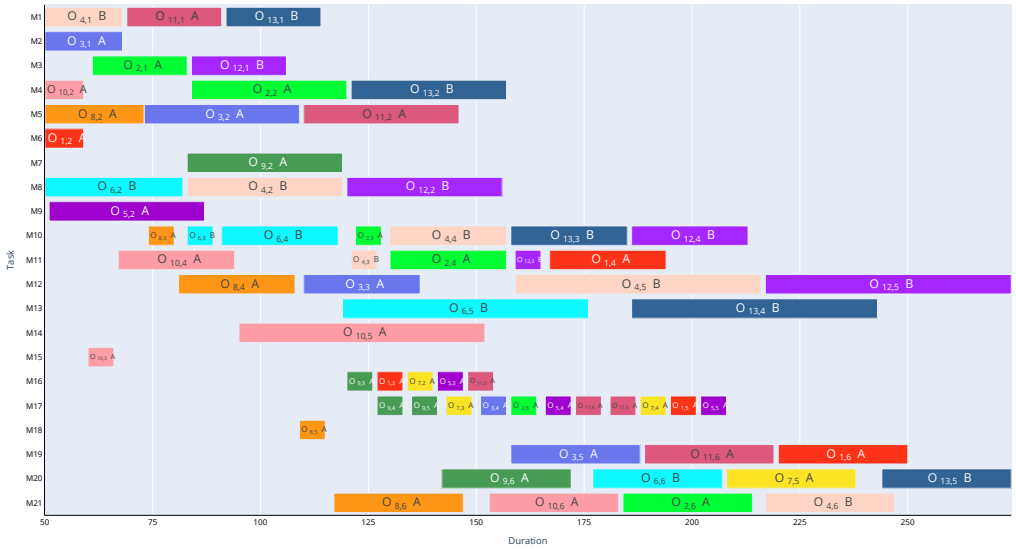


Figure 9. Scheduling for the expedited orders with $\omega_{11} = 1.0$, $\omega_{12} = 0.5$, $\omega_{13} = 0.1$.

Figure 9 presents the scheduling result after expedited orders are inserted, with the re-scheduling process initiated at time 50. As shown on the x-axis, ongoing operations at the moment of insertion-such as $O_{4,1}$, $O_{3,1}$, $O_{10,2}$, $O_{8,2}$, $O_{1,2}$, and $O_{6,2}$ remain unchanged in the revised schedule, reflecting the algorithm's ability to preserve in-process tasks. The insertion of expedited orders led to notable changes in machine utilization and job completion performance. Machines M_7 and M_9, previously unused due to their high operating costs, were activated to ensure overall feasibility and profitability, despite their expense. M_{12} became the most heavily utilized, while M_6 (idle after 160), M_{15} (idle after 350), and M_{18} (idle after 230) remained largely unused. The earliest completed job after re-scheduling was J_8 (at time 147), while the latest were J_{12} and J_{13} (both at time 274). Job J_5, with the shortest due date (99), was completed at time 208, indicating a delay caused by the prioritization of expedited orders. Several jobs (J_1, J_4, J_5, J_7, J_{11}, J_{12}, and J_{13}) originally completed on time were delayed, while the rest remained unaffected. A comparison between Table 3 and Table 6 reveals that jobs remaining on time generally had higher penalty weights (over 30) than those that were delayed, suggesting the algorithm's implicit bias toward minimizing total penalty rather than strictly minimizing lateness. Moreover, the completion sequence of the expedited orders ($J_{11} \rightarrow J_{12} \rightarrow J_{13}$) corresponds precisely with their priority weights ($1 > 0.5 > 0.1$), validating the effectiveness of the job priority template. From a resource perspective, the expedited orders increased demand for B-type operations, leading to higher utilization of machines equipped for B-type tooling and a total of 16 tool changes across the schedule. Conversely, A-type machines such as M_{17} received more A-type workloads to accommodate the shifted task distribution. These changes highlight the system's flexibility in adapting resource allocation to accommodate high-priority orders under tool constraints.

5.4. Experiment C: decision-making for re-scheduling

The decision to insert expedited orders is context-dependent and influenced by multiple factors. Before such orders are accepted into an existing schedule, it is important to evaluate whether their inclusion may degrade overall performance, such as reducing total profit, increasing task delays, or overloading resources. For this purpose, the TPG metric defined in Eq. (5.2) is adopted as the primary evaluation criterion to assist users in making informed re-scheduling decisions.

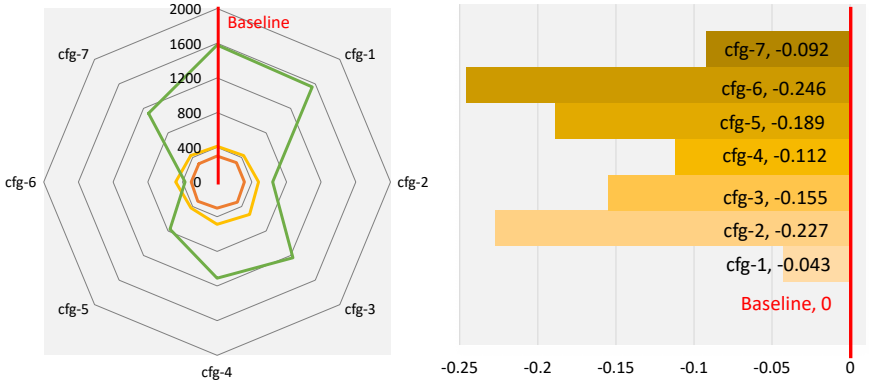
To demonstrate the decision-making process, the three expedited jobs in Table 5 are considered again for insertion into the existing schedule in Figure 8 (regular job scheduling only). Depending on the decision-making objectives, two sets of weight combinations for TPG are examined: $(\alpha_1, \alpha_2, \alpha_3) = (0.6, 0.1, 0.3)$ and $(0.3, 0.1, 0.6)$, corresponding respectively to scheduling preferences focused on minimizing makespan and maximizing profit. This experiment uses the job priority weights ω introduced in Experiment B, and tests all expedited job combinations, as listed in Table 7. The check-mark denotes that the job is considered in the job configuration. The original schedule without any expedited orders serves as the baseline for comparison. The impact of each combination on the TPG value is analyzed, and the results are shown in Figures 10–12.

Table 7. Expedited job insertion combinations.

Jobs	cfg-1	cfg-2	cfg-3	cfg-4	cfg-5	cfg-6	cfg-7
Job_{11}	✓	✓	✓	✓			
Job_{12}		✓	✓		✓	✓	
Job_{13}			✓	✓		✓	✓

Re-scheduling: Scenario-A

Figure 10(a) presents the comparative results of different scheduling configurations (cfg-1 to cfg-7) in terms of makespan (C_{min}), machine wear (M_{min}), and profit (E_{max}). Among all configurations, the baseline schedule without expedited orders achieved the highest profit E_{max} while maintaining competitively low values for both C_{min} and M_{min}. Figure 10(b) shows the overall performance index (TPG), which aggregates the weighted contributions of all three objectives. The TPG values for configurations cfg-1 through cfg-7 were consistently lower than the baseline, with differences ranging from −0.043 to −0.246. The results indicate that inserting the three expedited orders lowers overall scheduling performance. The optimizer consistently scheduled them after all regular jobs, suggesting that this sequence yields the greatest system-level benefit. These findings highlight the framework's effectiveness as a decision-support tool for assessing alternative scheduling strategies across multiple objectives.

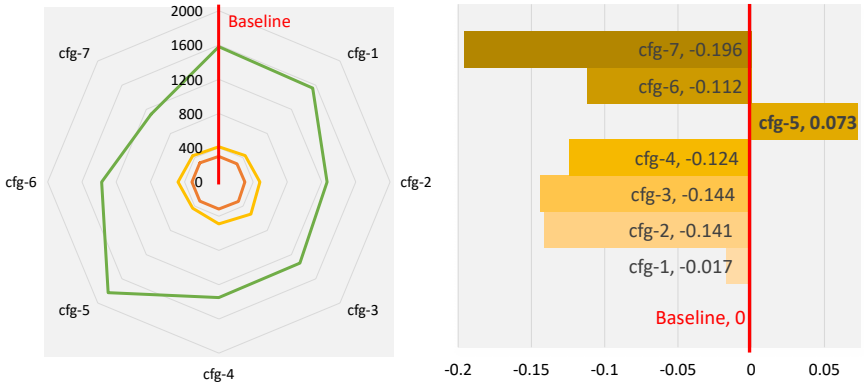


(a) C_{min}, M_{min}, E_{max} of all cfg's **(b)** TPG values

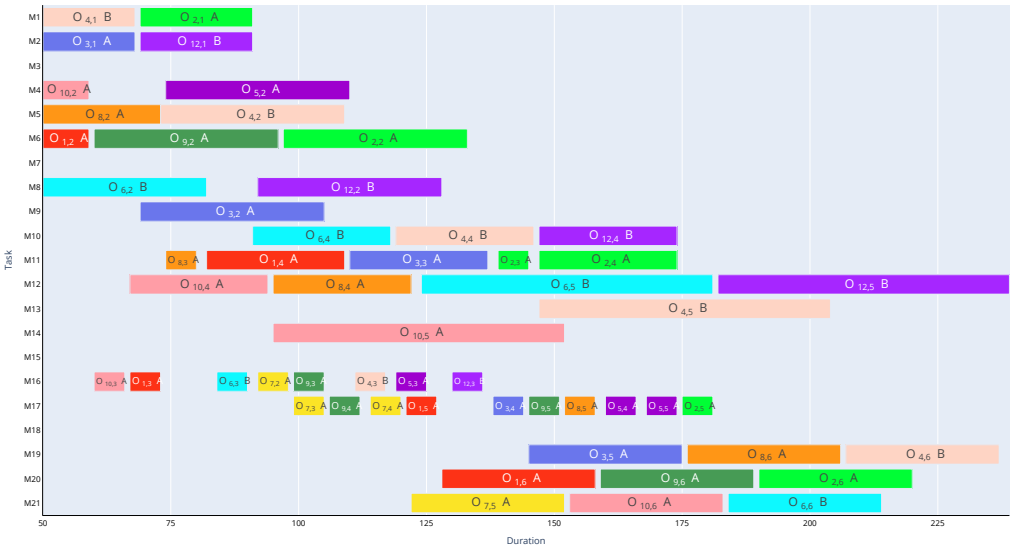
Figure 10. Decision-making for re-scheduling with $\alpha_1 = 0.6$, $\alpha_2 = 0.1$, and $\alpha_3 = 0.3$, and $\omega_{11} = 1.0$, $\omega_{12} = 0.5$, and $\omega_{13} = 0.1$. The best result is the baseline model, no expedited jobs allowed.

Re-scheduling: Scenario-B

Figure 11 presents the experimental results using the weight setting $\alpha_1 = 0.3$, $\alpha_2 = 0.1$, and $\alpha_3 = 0.6$ (emphasizing profit), and job priorities $\omega_1 = 0.5$, $\omega_2 = 1.0$, and $\omega_3 = 0.1$, where Job 12 is considered more important. As shown in Figure 11(a), configuration cfg-5 achieved the highest profit and the lowest machine wear, with a makespan comparable to other configurations. In the overall TPG performance shown in Figure 11(b), cfg-5 was the only configuration that outperformed the baseline (no expedited orders), indicating that selectively inserting only Job_{12} is a favorable scheduling strategy under this setting. The Gantt chart in Figure 11(c) further reveals that Job_{12}, due to its long processing sequence, was completed last. Operation $O_{12,5}$ was assigned to M_{12} instead of M_{13} to avoid interfering with $O_{4,5}$ and ensure the timely completion of Job_4. As a result, M_{12} became the busiest resource in the schedule. In contrast, M_7, M_{15}, and M_{18} were left idle, likely due to their higher operational costs compared to equivalent machines. The total number of tool changes was 11, identical to the baseline schedule, indicating that this configuration did not incur additional tool-switching overhead. These results suggest that selectively inserting only Job_{12}, while deferring Job_{11} and Job_{13}, yields the best trade-off across objectives under profit-prioritized conditions.



(a) C_{min}, M_{min}, E_{max} of all cfg's **(b)** TPG values



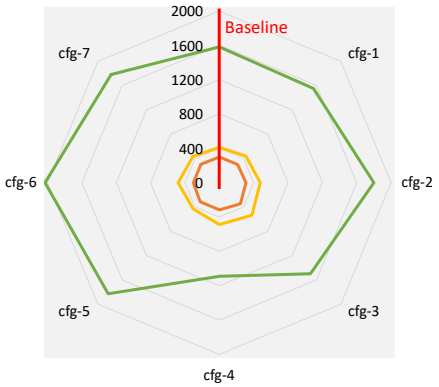
(c) New job-machine schedule

Figure 11. Decision-making for re-scheduling with $\alpha_1 = 0.3$, $\alpha_2 = 0.1$, and $\alpha_3 = 0.6$, and $\omega_{11} = 0.5$, $\omega_{12} = 1.0$, $\omega_{13} = 0.1$. The best result is the cfg-5 model-insert Job_{12} only.

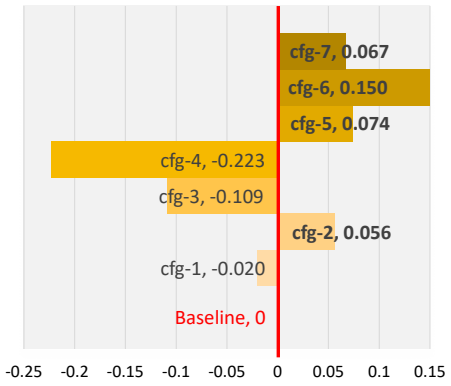
Re-scheduling: Scenario-C

Figure 12 presents the experimental results under the same weight setting as in Scenario-B ($\alpha_1 = 0.3$, $\alpha_2 = 0.1$, $\alpha_3 = 0.6$), but with a new job priority: $\omega_{11} = 0.1$, $\omega_{12} = 0.5$, and $\omega_{13} = 1.0$, indicating that Job_{13} is now considered the most important. As shown in Figure 12(a), multiple configurations (cfg-2, cfg-5, cfg-6, and cfg-7) outperformed the baseline in terms of combined scheduling objectives. Notably, cfg-6 achieved the highest overall performance, with a TPG value of 0.150, as shown in Figure 12(b). The configuration cfg-6 recommends inserting both Job_{12} and Job_{13}. Although this configuration results in a longer makespan, it yields the highest profit while maintaining similar levels of machine wear compared to other configurations. The Gantt chart in Figure 12(c) shows that Job_{12} was completed last due to its longer sequence. Machine M_7 remained idle in this schedule, and a total of 12 tool changes occurred, only one more than in the baseline case, indicating efficient tool usage despite the additional workload. Since both Job_{12} and Job_{13} require B-type tools, machines equipped for B-type

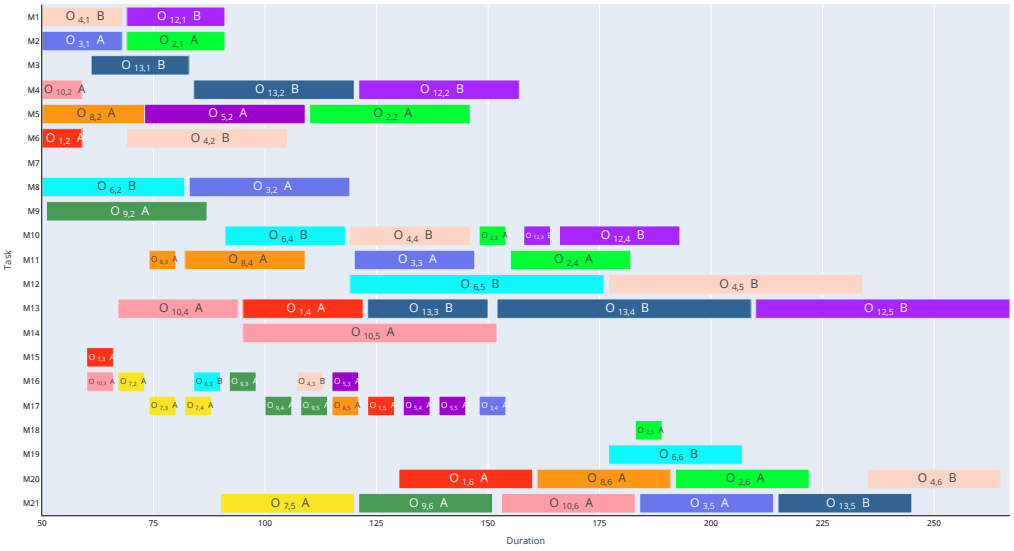
operations exhibited more continuous utilization than in previous scenarios. These results show that, depending on user priorities, several configurations offer better performance than the baseline. For instance, inserting only Job_{12} (cfg-5) or both Job_{12} and Job_{13} (cfg-6) are viable strategies that balance profitability and resource efficiency. The framework thus provides flexible decision-support for selecting the most suitable expedited order insertion plan under varying job importance settings.



(a) C_{min}, M_{min}, E_{max} of all cfg's



(b) TPG values



(c) New job-machine schedule

Figure 12. Decision-making for re-scheduling with $\alpha_1 = 0.3$, $\alpha_2 = 0.1$, and $\alpha_3 = 0.6$, and $\omega_{11} = 0.1$, $\omega_{12} = 0.5$, and $\omega_{13} = 1.0$. The best result is the cfg-6 model-insert Job_{12} and Job_{13}.

6. Conclusion and future work

6.1. Conclusion

This study presents a GA-based scheduling framework that simultaneously addresses two critical and often overlooked challenges in flexible job shop scheduling: expedited order insertion and tool change constraints. By incorporating a job priority (JP) template and a tool constraint (TC) template into the

GA processing flow, the proposed method enables dynamic adjustment of schedules to accommodate expedited orders while ensuring tooling feasibility and minimizing disruptions to in-progress operations. Few studies have simultaneously considered both expedited orders and tool constraints in the context of the FJSSP. The value of our study lies in integrating these two practical factors into a unified GA-based framework and demonstrating its applicability through validation with real industrial data. The approach was evaluated using real-world scheduling data from a precision screw manufacturing plant, rather than synthetic benchmarks, thereby demonstrating its practical relevance and robustness under realistic production conditions. The results confirm that the method can flexibly integrate dynamic job requirements while maintaining schedule efficiency, highlighting its potential as a reliable decision-support tool for real-world production environments.

In practical manufacturing settings, job, machine, and tool requirements are often complex and interrelated. Due to the absence of publicly available benchmark datasets, the test data used in this study were simplified and de-identified for confidentiality. However, the proposed method is inherently adaptable, allowing practitioners to adjust input parameters and formats without modifying the core algorithm. Moreover, the combination of job priority weights (ω) and objective weights (α_1, α_2, α_3) effectively guides re-scheduling decisions. The ω values influence the selection of expedited jobs during chromosome evolution and are embedded in the fitness function, while the α weights control trade-offs among makespan, machine wear, and profit. This integration enables adaptive determination of both the timing and number of inserted jobs, providing a flexible and extensible framework for real-time scheduling adjustments.

6.2. Discussions and future work

Discussions

This study contributes not only by proposing a scheduling algorithm with strong optimization performance, but also by addressing two practical challenges often overlooked in existing literature: dynamic expedited job handling and tool changeover coordination. This study adopts a single-objective fitness function that linearly combines multiple objectives using user-defined weights. While this approach is straightforward and computationally efficient, making it suitable for responsive scheduling environments, it inherently simplifies the trade-offs among objectives. Although multi-objective optimization methods [36] are well-suited for capturing Pareto-optimal trade-offs, our tests revealed that their longer computational time is less compatible with the rapid decision-making needs of real-world production. Additionally, hybrid tabu search (TS) and simulated annealing (SA) have been applied to the FJSSP; Kacem et al. [37] showed that GA outperforms both in multi-objective contexts. Recent evidence confirms that GA-based methods remain among the most effective and practical approaches for solving the FJSSP [37–39]. Compared with traditional scheduling methods and advanced optimization approaches, GA-based frameworks are easier to implement while still delivering high-quality solutions. Based on this established result and the scope of this work, we omit redundant comparisons and focus instead on real-time, tool-aware scheduling under dynamic conditions.

Although optimization-based methods such as MILP and CP [40] can provide optimal solutions for small and static problems, their applicability is limited in the present context. The NP-hard nature of FJSSP makes exact methods impractical for producing timely solutions for the underlying applications. The objective function used here combines multiple dimensions, including makespan, tool-related costs,

and profit, with dynamic priority adjustments for expedited orders. Translating these non-linear and conditional components into a linear MILP or CP model would be prohibitively complex. In contrast, exact optimization often requires long computation times that are not acceptable in practice. Besides, it is easier to expand the objective function when new scheduling factors arise. For these reasons, GA offers a practical balance between solution quality and computational efficiency in real factory settings.

Table 8. Qualitative comparison of different scheduling approaches in the context of the FJSSP with expedited orders and tooling constraints

Aspect	Manual	Typical	MOO	DL/RL	Ours
Decision basis	Experience	Exact models or simple rules	Evolutionary Pareto search	Trained policies	Evolutionary search with priorities
Expedited orders	Manual insert, delays others	Fast but static and shortsighted	Possible but slow	Retraining often needed	Dynamic insertion, minimal disruption
Tool changes	By judgment, frequent	Modelable but complex, ignore	Modelable but costly	Rarely modeled	Explicitly minimized in fitness
Reaction time	About 8H reschedule	Minutes to hours, for small jobs, low quality	Hours+	Long training time, fast inference	Minutes, revised schedule
Stability	Large rework	Disrupts plan, local only	Pareto front selection	Stability unclear	Preserves ongoing jobs
Scalability	Poor	Exponential time	Grows rapidly with size	Heavy compute	Effective for medium/large jobs
Adoption	Common, inconsistent	MILP rare in SMEs; Rules used simply	Mostly academic	Few pilot cases	Lightweight, practical in SMEs

SME: Small and Medium-sized Enterprises

The tool-aware genetic encoding and dynamic insertion mechanism together allow the system to preserve in-process schedules, avoid unnecessary tool switches, and accommodate urgent orders without complete re-scheduling. These characteristics are crucial for deployment in real-world factories, where stability, reactivity, and transparency are equally important as solution quality. Our industrial partners also indicate that methods that are simple, fast, and responsive are preferable in practical applications. Feedback from field engineers further confirms the model's usability and adaptability in production scenarios with fluctuating priorities and tooling limitations. Table 8 provides a qualitative comparison of different scheduling approaches to clarify their relative strengths and limitations.

Future work

The crossover and mutation rates in this study were chosen based on preliminary tests (e.g., Table 4) to ensure stable results within short computation times. Although effective for the industrial case, fixed settings may not be optimal. Future work will explore adaptive or learning-based tuning, such as reinforcement learning or Bayesian optimization [41], to enhance robustness in larger or more dynamic environments. As scheduling is an NP-hard problem, its combinatorial complexity renders exact methods impractical for industrial-scale instances, making metaheuristic approaches an effective alternative for efficiently navigating the vast solution space. As stated in Section 2, DRL is a state-of-the-art framework showing significant promise in dynamic scheduling by learning complex, adaptive policies from data [17, 19, 34]. While powerful, its practical application in many manufacturing settings, especially small and medium-sized enterprises, is hindered by substantial requirements for large-scale data, long training times, and high computational cost.

However, recent developments in DRL-based metaheuristics have shown promising results for complex scheduling problems. In particular, proximal policy optimization (PPO) [42, 43] has been effectively combined with variable neighborhood search (VNS) [44] in population-based frameworks to improve search efficiency. PPO serves as a learning agent that guides the neighborhood selection or switching process in VNS, using feedback from the search trajectory to balance exploration and exploitation dynamically. Such hybrid DRL-VNS mechanisms have demonstrated faster convergence and better adaptability in stochastic and dynamic environments. A potential direction for future research is to explore hybrid DRL-heuristic strategies that reduce these burdens while retaining adaptability, making them more practical for resource-constrained manufacturing environments. Similar learning-guided strategies could be integrated into the proposed GA framework to adjust search parameters or insertion weights in real time, improving responsiveness for expedited orders and tool change scenarios in flexible job-shop scheduling.

In scheduling, "nervousness" refers to situations where small changes can lead to large disruptions. Our GA mitigates this effect by incorporating tool-change costs into the fitness function and restricting changes to jobs that have not yet commenced. This design lowers unnecessary re-scheduling and improves stability. A full treatment of nervousness would require additional factors, such as labor and material costs and organizational aspects. However, our proposed GA framework can flexibly integrate these factors if they can be quantitatively defined. These will be addressed in our future research.

Author contributions

Conceptualization: Chih-Hung Wu, Yi-Han Chen; Data curation: Yi-Han Chen, Hsi-Wen Wang, Chiung-Hui Tsai; Formal analysis: Chih-Hung Wu, Yi-Han Chen; Funding acquisition: Chih-Hung Wu, Yi-Han Chen; Investigation: Yi-Han Chen, Chih-Hung Wu; Methodology: Yi-Han Chen, Hsi-Wen Wang; Project administration: Chih-Hung Wu, Yi-Han Chen; Resources: Chih-Hung Wu, Yi-Han Chen; Software: Hsi-Wen Wang, Yi-Han Chen; Supervision: Chih-Hung Wu; Validation: Yi-Han Chen, Chiung-Hui Tsai, Chih-Hung Wu; Visualization: Yi-Han Chen, Chiung-Hui Tsai, Hsi-Wen Wang; Writing – original draft: Chih-Hung Wu, Hsi-Wen Wang; Writing – review & editing: Chih-Hung Wu, Yi-Han Chen, Chiung-Hui Tsai

All authors have read and agreed to the published version of the manuscript.

Use of Generative-AI tools declaration

The authors acknowledge the use of generative artificial intelligence (AI) tools (ChatGPT, Gemini, and Grammarly) strictly for technical support during manuscript preparation, limited to language refinement (grammar, clarity) and LaTeX formatting corrections. All intellectual content, including study design, data analysis, and scientific conclusions, originated solely from the authors, who reviewed and verified all AI-assisted technical suggestions for accuracy.

Conflict of interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. X. Xu, Y. Lu, B. Vogel-Heuser, L. Wang, Industry 4.0 and industry 5.0-inception, conception and perception, *J. Manuf. Syst.*, **61** (2021), 530–535. https://doi.org/10.1016/j.jmsy.2021.10.006
2. M. M. A. P. Mick, J. L. Kovaleski, R. T. Yoshino, D. M. de Genaro Chiroli, The influence between industry 4.0 and technology transfer: A framework based on systematic literature review, *SAGE Open*, **14** (2024), 21582440241295580. https://doi.org/10.1177/21582440241295580
3. M. D. Sinanis, P. Adhikari, T. R. Jones, M. Abdelfattah, D. Peroulis, High-Q high power tunable filters manufactured with injection molding technology, *IEEE Access*, **10** (2022), 19643–19653. https://doi.org/10.1109/ACCESS.2022.3151849
4. H. T. Nguyen, R. Olsson, Ø. Haugen, Automatic synthesis of recurrent neurons for imitation learning from CNC machine operators, *IEEE Open J. Ind. Electron. Soc.*, **5** (2024), 91–108. https://doi.org/10.1109/OJIES.2024.3363500
5. A. Nouri, A. Soroudi, A. Keane, Strategic scheduling of discrete control devices in active distribution systems, *IEEE Trans. Power Delivery*, **35** (2020), 2285–2299. https://doi.org/10.1109/TPWRD.2020.2965110
6. B. She, F. Li, J. Wang, H. Cui, X. Wang, R. Bo, Virtual inertia scheduling (VIS) for microgrids with static and dynamic security constraints, *IEEE Trans. Sustainable Energy*, **16** (2025), 785–796. https://doi.org/10.1109/TSTE.2024.3481239
7. S. C. Graves, A review of production scheduling, *Oper. Res.*, **29** (1981), 646–675. https://doi.org/10.1287/opre.29.4.646
8. R. Buddala, S. S. Mahapatra, An integrated approach for scheduling flexible job-shop using teaching–learning-based optimization method, *J. Ind. Eng. Int.*, **15** (2019), 181–192. https://doi.org/10.1007/s40092-018-0280-8
9. S. Jun, S. Lee, H. Chun, Learning dispatching rules using random forest in flexible job shop scheduling problems, *Int. J. Prod. Res.*, **57** (2019), 3290–3310. https://doi.org/10.1080/00207543.2019.1581954

10. G. Gong, Q. Deng, R. Chiong, X. Gong, H. Huang, An effective memetic algorithm for multi-objective job-shop scheduling, *Knowl.-Based Syst.*, **182** (2019), 104840. https://doi.org/10.1016/j.knosys.2019.07.011
11. M. Samouilidou, G. Georgiadis, M. Georgiadis, A multi-bucket time representation framework for optimal scheduling in beverage production facilities, *Comput. Chem. Eng.*, **183** (2024), 108611. https://doi.org/10.1016/j.compchemeng.2024.108611
12. Y. J. Yao, Q. H. Liu, X. Y. Li, L. Gao, A novel MILP model for job shop scheduling problem with mobile robots, *Rob. Comput.-Integr. Manuf.*, **81** (2023), 102506. https://doi.org/10.1016/j.rcim.2022.102506
13. H. El Hafdaoui, A. Khallaayoun, S. Al-Majeed, Controlled non-dominated sorting genetic algorithms for multi-objective optimal design of standalone and grid-connected renewable energy systems in integrated energy sectors, *IEEE Access*, **13** (2025), 14658–14685. https://doi.org/10.1109/ACCESS.2025.3530084
14. S. C. Burmeister, D. Guericke, G. Schryen, A memetic NSGA-II for the multi-objective flexible job shop scheduling problem with real-time energy tariffs, *Flex. Serv. Manuf. J.*, **36** (2024), 1530–1570. https://doi.org/10.1007/s10696-023-09517-7
15. T. L. Chen, C. Y. Cheng, Y. H. Chou, Multi-objective genetic algorithm for energy-efficient hybrid flow shop scheduling with lot streaming, *Ann. Oper. Res.*, **290** (2020), 813–836. https://doi.org/10.1007/s10479-018-2969-x
16. K. Lei, P. Guo, W. Zhao, Y. Wang, L. Qian, X. Meng, et al., A multi-action deep reinforcement learning framework for flexible job-shop scheduling problem, *Expert Syst. Appl.*, **205** (2022), 117796. https://doi.org/10.1016/j.eswa.2022.117796
17. H. Wang, J. Cheng, C. Liu, Y. Zhang, S. Hu, L. Chen, Multi-objective reinforcement learning framework for dynamic flexible job shop scheduling problem with uncertain events, *Appl. Soft Comput.*, **131** (2022), 109717. https://doi.org/10.1016/j.asoc.2022.109717
18. K. Lei, P. Guo, Y. Wang, J. Zhang, X. Meng, L. Qian, Large-scale dynamic scheduling for flexible job-shop with random arrivals of new jobs by hierarchical reinforcement learning, *IEEE Trans. Ind. Inf.*, **20** (2024), 1007–1018. https://doi.org/10.1109/TII.2023.3272661
19. C. Ngwu, Y. Liu, R. Wu, Reinforcement learning in dynamic job shop scheduling: a comprehensive review of ai-driven approaches in modern manufacturing, *J. Intell. Manuf.*. https://doi.org/10.1007/s10845-025-02585-6
20. J. Pei, B. Cheng, X. Liu, P. M. Pardalos, M. Kong, Single-machine and parallel-machine serial-batching scheduling problems with position-based learning effect and linear setup time, *Ann. Oper. Res.*, **272** (2019), 217–241. https://doi.org/10.1007/s10479-017-2481-8
21. N. Mladenović, P. Hansen, Variable neighborhood search, *Comput. Oper. Res.*, **24** (1997), 1097–1100. https://doi.org/10.1016/S0305-0548(97)00031-2
22. A. Hashemi, M. B. Dowlatshahi, H. Nezamabadi-Pour, Gravitational search algorithm: Theory, literature review, and applications, *Handbook of AI-based Metaheuristics*, 119–150.
23. D. Lei, Y. Yuan, J. Cai, An improved artificial bee colony for multi-objective distributed unrelated parallel machine scheduling, *Int. J. Prod. Res.*, **59** (2020), 5259–5271. https://doi.org/10.1080/00207543.2020.1775911

24. L. Fanjul-Peyro, Models and an exact method for the unrelated parallel machine scheduling problem with setups and resources, *Expert Syst. Appl.: X*, **5** (2020), 100022. https://doi.org/10.1016/j.eswax.2020.100022

25. M. S. Umam, M. Mustafid, S. Suryono, A hybrid genetic algorithm and tabu search for minimizing makespan in flow shop scheduling problem, *J. King Saud Univ.-Comput. Inf. Sci.*, **34** (2022), 7459–7467. https://doi.org/10.1016/j.jksuci.2021.08.025

26. V. K. Prajapati, M. Jain, L. Chouhan, Tabu search algorithm TSA: A comprehensive survey, in *Proceedings of the 3th International Conference on Emerging Technologies in Computer Engineering: Machine Learning and Internet of Things (ICETCE)*. IEEE, 2020, 1–8. https://doi.org/10.1109/ICETCE48199.2020.9091743

27. D. Gao, G. G. Wang, W. Pedrycz, Solving fuzzy job-shop scheduling problem using DE algorithm improved by a selection mechanism, *IEEE Trans. Fuzzy Syst.*, **28** (2020), 3265–3275. https://doi.org/10.1109/TFUZZ.2020.3003506

28. S. Afsar, C. R. Vela, J. J. Palacios, I. González-Rodríguez, Mathematical models and benchmarking for the fuzzy job shop scheduling problem, *Comput. Ind. Eng.*, **183** (2023), 109454. https://doi.org/10.1016/j.cie.2023.109454

29. Bilal, M. Pant, H. Zaheer, L. Garcia-Hernandez, A. Abraham, Differential evolution: A review of more than two decades of research, *Eng. Appl. Artif. Intell.*, **90** (2020), 103479. https://doi.org/10.1016/j.engappai.2020.103479

30. S. Luo, Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning, *Appl. Soft Comput.*, **91** (2020), 106208. https://doi.org/10.1016/j.asoc.2020.106208

31. J. S. Neufeld, S. Schulz, U. Buscher, A systematic review of multi-objective hybrid flow shop scheduling, *Eur. J. Oper. Res.*, **309** (2023), 1–23. https://doi.org/10.1016/j.ejor.2022.08.009

32. T. J. Ikonen, K. Heljanko, I. Harjunkoski, Reinforcement learning of adaptive online rescheduling timing and computing time allocation, *Comput. Chem. Eng.*, **141** (2020), 106994. https://doi.org/10.1016/j.compchemeng.2020.106994

33. M. E. Samouilidou, N. Passalis, G. P. Georgiadis, M. C. Georgiadis, Enhancing industrial scheduling through machine learning: A synergistic approach with predictive modeling and clustering, *Comput. Chem. Eng.*, **200** (2025), 109174. https://doi.org/10.1016/j.compchemeng.2025.109174

34. C. Y. Lee, Y. T. Huang, P. J. Chen, Robust-optimization-guiding deep reinforcement learning for chemical material production scheduling, *Comput. Chem. Eng.*, **187** (2024), 108745. https://doi.org/10.1016/j.compchemeng.2024.108745

35. I. Kacem, S. Hammadi, P. Borne, Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems, *IEEE Trans. Syst. Man Cybern. Part C*, **32** (2002), 1–13. https://doi.org/10.1109/TSMCC.2002.1009117

36. G. Adabbo, A. Andreozzi, M. Iasiello, G. Napoli, G. P. Vanoli, A multi-objective optimization framework through genetic algorithm for hyperthermia-mediated drug delivery, *Comput. Biol. Med.*, **189** (2025), 109895. https://doi.org/10.1016/j.compbiomed.2025.109895

37. S. Dauzère-Pérès, J. Ding, L. Shen, K. Tamssaouet, The flexible job shop scheduling problem: A review, *Eur. J. Oper. Res.*, **314** (2024), 409–432. https://doi.org/10.1016/j.ejor.2023.05.017

38. V. Boyer, J. Vallikavungal, X. Cantú Rodríguez, M. A. Salazar-Aguilar, The generalized flexible job shop scheduling problem, *Comput. Ind. Eng.*, **160** (2021), 107542. https://doi.org/10.1016/j.cie.2021.107542

39. L. Meng, W. Cheng, B. Zhang, W. Zou, W. Fang, P. Duan, An improved genetic algorithm for solving the multi-agv flexible job shop scheduling problem, *Sensors*, **23** (2023), 3815. https://doi.org/10.3390/s23083815

40. G. Da Col, E. C. Teppan, Industrial-size job shop scheduling with constraint programming, *Oper. Res. Perspect.*, **9** (2022), 100249. https://doi.org/10.1016/j.orp.2022.100249

41. M. Garouani, M. Bouneffa, Automated machine learning hyperparameters tuning through meta-guided Bayesian optimization, *Prog. Artif. Intell.*, (2024). https://doi.org/10.1007/s13748-023-00311-y

42. W. Zhang, M. Kong, Y. Zhang, A. M. Fathollahi-Fard, Proximal policy optimization with population-based variable neighborhood search algorithm for coordinating photo-etching and acid-etching processes in sustainable storage chip manufacturing, *J. Ind. Inf. Integr.*, **42** (2024), 100727. https://doi.org/10.1016/j.jii.2024.100727

43. F. Zhao, T. Yang, L. Song, J. Zhang, T. Xu, Jonrinaldi, An online learning metaheuristic algorithm with proximal policy optimization mechanism, *Expert Syst. Appl.*, **289** (2025), 128403. https://doi.org/10.1016/j.eswa.2025.128403

44. F. Grumbach, N. E. A. Badr, P. Reusch, S. Trojahn, A memetic algorithm with reinforcement learning for sociotechnical production scheduling, *IEEE Access*, **11** (2023), 68760–68775. https://doi.org/10.1109/ACCESS.2023.3292548

AIMS Press