Complex & Intelligent Systems (2025) 11:312
https://doi.org/10.1007/s40747-025-01890-0

ORIGINAL ARTICLE

# A novel two stage neighborhood search for flexible job shop scheduling problem considering reconfigurable machine tools

**Yanjun Shi[1,2] · Chengjia Yu[1,2] · Shiduo Ning[1,2]**

Received: 9 December 2024 / Accepted: 30 March 2025 / Published online: 4 June 2025
© The Author(s) 2025

**Abstract**

The rapid changes in market demand are driving a transition from traditional mass production to high-mix, low-volume production, emphasizing the need for customization and rapid response. Reconfigurable Manufacturing Systems (RMS) are crucial in this shift, providing a flexible platform to meet diverse production requirements, and forming an essential component of next-generation manufacturing. Reconfigurable Machine Tools (RMTs), the core of RMS, enable dynamic configuration adjustments through auxiliary modules (AMs), enhancing both flexibility and efficiency. However, optimizing the allocation of limited AMs, considering non-negligible assembly and disassembly times, remains a significant challenge. This paper focuses on the flexible job shop scheduling problem with machine reconfigurations (FJSP-MR) and proposes an improved genetic algorithm with a two-stage neighborhood search (IGA-TNS) to minimize total weighted tardiness (TWT). Initially, a mixed-integer linear programming (MILP) model is formulated to comprehensively represent the problem. To enhance search efficiency, a two-stage neighborhood search strategy is developed: the first stage extends the k-insertion search to facilitate operation movement across different machine configurations, while the second stage refines operations within the same configuration. Furthermore, a trend-detection-based neighborhood search activation strategy is introduced to accelerate convergence and reduce computational costs. Experimental results on extended benchmark instances demonstrate that the proposed IGA-TNS effectively addresses the FJSP-MR, outperforming other algorithms in solution quality and computational efficiency. Finally, a industrial FJSP-MR case is studied, demonstrating that the proposed IGA-TNS is applicable to large-scale problems.

**Keywords** Reconfigurable manufacturing systems · Flexible job shop scheduling · Reconfigurable machine tools · Improved genetic algorithm · Neighbourhood search

## Introduction

In modern manufacturing, rapid changes in market demand and a growing trend towards personalized product customization are driving a shift towards multi-variety, small-batch production methods. This evolution necessitates manufacturing systems that are highly flexible and responsive. Traditional manufacturing systems, typically designed for specific products or production lines, struggle to adapt to these dynamic market demands and production requirements. Reconfigurable Manufacturing System (RMS) addresses these challenges through its complex, modular assembly of reconfigurable manufacturing units, enabling rapid adjustments in production modes to accommodate varying product demands and thereby enhancing system flexibility and responsiveness [1]. Reconfigurable Machine Tools (RMTs), critical components of RMS, incorporate various auxiliary modules (AMs).

✉ Shiduo Ning
ningsd@mail.dlut.edu.cn

Yanjun Shi
shiyj@dlut.edu.cn

Chengjia Yu
ycj3839@163.com

[1] School of Mechanical Engineering, Dalian University of Technology, Dalian, China

[2] State Key Laboratory of High-Performance Precision Manufacturing, Dalian University of Technology, Dalian, China

This modular structure design allows the machine tools to transform their configurations to meet specific manufacturing requirements [2].

Koren et al. [3] first introduced the concept of reconfigurable machine tools (RMT), which can change configurations to perform different machining operations. Building upon this foundation, many researchers have studied reconfigurable machine tools. For instance, Ersal et al. [4] proposed a modular method for modeling the servo axis of RMTs, which allows assembling the corresponding component models based on the machine tool's topology to obtain a model for any given configuration. Pérez et al. [5] designed a reconfigurable next-generation dual-axis test bed CNC micro-machine tool within an integrated product, process, and manufacturing system development environment. Padayachee et al. [6] worked on the mechanical design and control system development of reconfigurable machine tools and proposed a modular electronic system using a plug-and-play approach to control the reconfiguration of the machine tools. In research on reconfigurable machine tools, the design of efficient RMTs to reduce reconfiguration time and automate the reconfiguration process remains an urgent challenge [7]. Therefore, considering the reconfiguration time of machine tools is crucial.

This paper incorporates reconfigurable machine tools into the flexible job shop scheduling problem (FJSP) and designs a novel two stage neighborhood structure based on the problem's characteristics. Existing research largely ignores the reconfiguration time of machine tools or treats machine reconfiguration as a preparatory step before production. However, in actual production, if different auxiliary modules are used between consecutive machining operations on a machine tool, the time required for assembling and disassembling these modules cannot be ignored. The reconfiguration time caused by the assembly and disassembly of auxiliary modules may increase the completion time. Therefore, it is crucial to consider the disassembly and assembly times of auxiliary modules and allocate them properly. To better reflect actual production conditions, this paper considers a scheduling problem where auxiliary modules are limited and the disassembly and assembly times for the same module vary across different machine tools. Unlike traditional flexible job shop scheduling, this problem requires the consideration of machine selection for processing operations, operation sequencing, and auxiliary module allocation. Additionally, the allocation of auxiliary modules interacts significantly with the first two issues, leading to an exponential increase in problem complexity.

The main contributions of this paper are as follows:

(1) The constraints of the Flexible Job Shop Scheduling Problem with Machine Reconfigurations (FJSP-MR) are analyzed, and a sequence-based Mixed Integer Linear Programming (MILP) model is formulated. The model is solved and verified using CPLEX.

(2) A two-stage neighborhood search genetic algorithm is proposed, incorporating a novel neighborhood search mechanism based on the disjunctive graph model to account for machine reconfiguration time. The first stage focuses on selecting the processing configuration, while the second stage determines the operation position.

(3) To further improve the algorithm's convergence speed and reduce computational costs, a trend-detection-based neighborhood search activation strategy is introduced. This strategy activates the neighborhood search to perform efficient global and local searches when the objective value shows minimal change.

(4) A industrial large-scale FJSP-MR case is tested to further investigate the performance of IGA-TNS. The results indicate that the proposed IGA is more effective in solving practical FJSP-MR problems compared to other algorithms.

The remainder of this paper is organized as follows: Chapter 2 reviews the relevant literature. Chapter 3 defines the FJSP-MR problem and establishes the MILP model. Chapter 4 describes the proposed algorithm in detail. Chapter 5 presents the experimental results and analysis. Chapter 6 investigates a industrial FJSP-MR case for algorithm validation. Chapter 7 concludes the paper and outlines future research directions.

## Literature review

### Scheduling problem in RMS

To cope with the rapid changes in market demand, RMS have gradually attracted the attention of researchers. In the field of reconfigurable flexible job shop scheduling, Liu et al. [8] were the first to consider the scheduling problem of reconfigurable manufacturing systems. They developed a novel mathematical model for the reconfigurable process of conveyor assembly lines, and since then, researchers have begun exploring scheduling problems in reconfigurable manufacturing systems. Bensmaine et al. [9] proposed a new heuristic method for process planning and scheduling of reconfigurable machine tools. Guo et al. [10] studied a dynamic flexible job shop scheduling problem with reconfigurable manufacturing units, where completion time, delay time, and reconfiguration time were the optimization objectives. They proposed an improved genetic programming method to solve this problem and designed an individual simplification strategy to shorten the evaluation time of the heuristic algorithm. Zhang et al. [11] studied the flexible assembly job shop scheduling problem with

reconfigurable machines and proposed a cooperative evolutionary mathematical algorithm based on MILP, improving the algorithm's performance through cooperative initialization and dual cooperation strategies. Bortolini et al. [12] considered the availability of auxiliary modules and proposed an optimization linear programming model to balance the reconfiguration of reconfigurable machine tools. Mahmoodjanloo et al. [13] studied the rescheduling problem in distributed job shops with reconfigurable machines and proposed an adaptive hybrid balance optimization algorithm. This algorithm designs a rescheduling module framework to minimize total weighted delays under static conditions and extends it to dynamic problems to update the current schedule. Shen et al. [14] studied a flexible job shop scheduling problem with machine reconfiguration, established a mixed-integer linear programming (MILP) model, and proposed an improved genetic algorithm with neighborhood search. Gao et al. [15] expanded on this by studying a flexible job shop scheduling problem with batch flow and machine reconfiguration. They proposed a mathematical method with a variable neighborhood search component within a genetic algorithm framework.

Current research on reconfigurable manufacturing systems (RMS) primarily focuses on scheduling and encompasses various domains, including process planning, dynamic scheduling, and cooperative evolutionary algorithms, with notable advancements achieved in these areas. These studies highlight the high complexity of RMS scheduling problems. However, certain aspects remain underexplored, such as the variation in reconfiguration times of reconfigurable machine tools under different modes. This paper delves deeper into the impact of these reconfiguration time variations in flexible job shops by rationally allocating limited auxiliary modules.

## Related work on neighborhood search

The Flexible Job Shop Scheduling Problem (FJSP) and its extensions are well-known NP-hard problems. To address the complexity of these problems, various optimization algorithms have emerged as primary approaches for finding effective solutions. Nevertheless, when tackling complex scenarios, challenges such as convergence to local optima and limited computational efficiency persist. The incorporation of neighbor-hood structures has been shown to effectively enhance solution diversity and search efficiency. Consequently, numerous researchers have developed neighborhood structures and integrated them into algorithms to explore the solution space more comprehensively. The design of efficient neighbor-hood operators has therefore emerged as a key focus in the research on solving job shop scheduling problems and its extensions.

Xie et al. [16] proposed a new N8 neighborhood structure for job shop scheduling problems, considering the movement of critical operations both inside and outside of critical blocks. They also designed a neighborhood pruning method to avoid ineffective move operations. Xu et al. [17] introduced a hybrid algorithm combining quantum particle swarm optimization and variable neighborhood search, designing nine new neighborhood structures and proposing a neighborhood transformation rule based on the encoding sequence length to efficiently solve the FJSP. Cao et al. [18] proposed a collaborative optimizer based on inverse models and adaptive neighborhood search for energy-efficient distributed flexible job shop scheduling problems. First, the inverse model was applied to the job shop scheduling problem. Then, adaptive local search operators were designed to collaboratively search for offspring. On this basis, they proposed an adaptive strategy for the local search operators. Huang et al. [19] focused on the green II-type fuzzy flexible job shop problem and designed a collaborative heuristic initialization algorithm to generate high-quality solutions. They also proposed a layered heuristic neighborhood search strategy to better allocate computational resources according to the quality of solutions, thereby improving the utilization of solutions. Zhang et al. [20] studied a distributed heterogeneous flexible job shop scheduling problem with variable processing times and proposed a deep Q-network-assisted variable neighborhood search algorithm. They introduced seven efficient neighborhood structures based on the critical path and used a three-vector encoding scheme to improve local search efficiency. Ding et al. [21] proposed a two-population evolutionary algorithm for the flexible job shop scheduling problem under time-of-use electricity pricing. They designed a new neighborhood structure and an approximate neighborhood evaluation method for this specific problem.

Numerous studies have demonstrated that well-designed neighborhood structures can significantly enhance solution diversity and optimization efficiency, particularly in complex job shop scheduling problems. Various neighborhood search strategies have substantially improved the efficiency of local search and the speed of convergence. These studies provide a solid foundation for the advancements in neighborhood structures proposed in this paper. However, when addressing the unique challenges posed by reconfigurable manufacturing systems, further refinement remains necessary. In this work, the neighborhood search is based on the K-insertion method [22] and has been enhanced to better address the scheduling requirements of reconfigurable manufacturing systems by minimizing the reconfiguration time of machine tools, thereby reducing the total processing time.

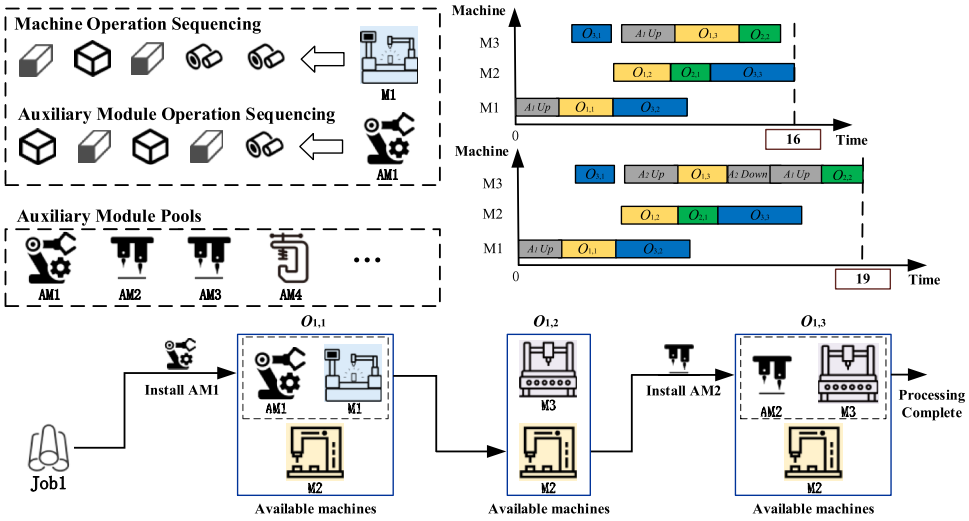


**Fig. 1** The flow chart of the flexible job shop considering reconfigurable machine tools

## Problem formulation

### Problem description

The Flexible Job Shop Scheduling Problem with Machine Reconfiguration (FJSP-MR) studied in this paper is illustrated in Fig. 1 and is described as follows: The workshop consists of $M$ machine tools, all of which are reconfigurable, denoted as $M = \{M_1, M_2, ..., M_m\}$, and $L$ auxiliary modules, denoted as $A = \{A_1, A_2, ..., A_l\}$. All machine tools are reconfigurable and can choose whether to install specific auxiliary modules to reduce processing time or to process different operations. As shown in Fig. 1, the installation of auxiliary modules $A_2$ can shorten the processing time of operation $O_{1,3}$; however, the reconfiguration time involved in assembling and disassembling the auxiliary modules may increase the completion time. Therefore, the FJSP-MR not only requires selecting the appropriate machine for each operation but also involves the reasonable allocation of auxiliary modules.

There are $J$ jobs to be processed, denoted as $J = \{J_1, J_2, ..., J_N\}$, each consisting of multiple operations. The operations of a job must be processed in a fixed order. Each operation $O_{i,j}$ can be processed on one or more of the $M$ machines. Different machine configurations correspond to different processing times, and the machine assembly time $\alpha$ and disassembly time $\beta$ are not negligible. The disassembly time of the same auxiliary module may vary depending on the machine it is installed on. Table 1 presents a specific instance of the FJSP-MR.

As shown in Table 2, after the completion of a process operation, the detachable auxiliary module can be used for processing on other machines. The configuration time for auxiliary module $A_q$ on machine $M_k$ is denoted as $\alpha_{k,q}, \beta_{k,q}$, representing the assembly and disassembly time, respectively. The problem discussed in this paper can be divided into three subproblems: machine allocation, auxiliary module selection, and operation sequencing. The objective is to minimize the total weighted tardiness, which can be expressed as: $TWT = \sum_{i=1}^{n} w_i \cdot t_i$, where $t_i$ is the tardiness of job $i$. The relevant information for the jobs is defined as

**Table 1** FJSP-MR example

| Job | Operation | Processing time | | |
|---|---|---|---|---|
| | | $M_1$ | $M_2$ | $M_3$ |
| $J_1$ | $O_{1,1}$ | – | 5/3($A_1$) | 3 |
| | $O_{1,2}$ | 6 | 4($A_2$) | – |
| | $O_{1,3}$ | 4 | – | 3($A_1$) |
| $J_2$ | $O_{2,1}$ | – | 3($A_1$) | 5/4($A_2$) |
| | $O_{2,2}$ | 2($A_1$) | 4 | 3($A_2$) |
| | $O_{2,3}$ | – | 5 | 3($A_2$) |
| $J_3$ | $O_{3,1}$ | 4 | 3($A_1$)/2($A_2$) | – |
| | $O_{3,2}$ | – | – | 3 |
| | $O_{3,3}$ | 5 | 4($A_1$) | – |

**Table 2** Configuration time

| AM | Reconfiguration time | | |
|---|---|---|---|
| | $M_1$ | $M_2$ | $M_3$ |
| $A_1$ | | | |
| Assembly time | 2 | 3 | 3 |
| Disassembly time | 3 | 4 | 2 |
| $A_2$ | | | |
| Assembly time | 2 | 2 | 3 |
| Disassembly time | 3 | 5 | 4 |

follows: release time $R = \{r_1, r_2, ..., r_n\}$, due dates $D = \{d_1, d_2, ..., d_n\}$ and tardy weights $W = \{w_1, w_2, ..., w_n\}$.

In addition, the following assumptions are made:

- A machine can only install one auxiliary module at a time.
- A machine can only process one operation at a time.
- An auxiliary module can only be installed on one machine during a processing task.
- There are no processing priority requirements between jobs.
- All machines and auxiliary modules are available starting from time 0.
- Once a processing task starts, it cannot be interrupted.

## Problem model

For the FJSP and its extended problems, most researchers have established MILP models for mathematical formulation and used Fdata (MFJS01-10) [23] and BRdata (MK01-10) [24] for validation. This section refers to the mathematical models by reference [14,25] , and builds upon reference [14] model by setting different reconfiguration times for different machines to better align with real-world production scenarios. In this section, a sequence-based MILP model is developed for the FJSP-MR. The parameters and notations used in the model are defined as follows (see Table 3):

The calculation of $\gamma_{q,q',k}$, $\tau_{k,k',q}$ is based on reference [14]. Specifically:$\gamma_{q,q',k}$ represents the reconfiguration time when different auxiliary modules are installed sequentially on the same machine, $\gamma_{q,q',k} = \beta_{k,q'} + \alpha_{k,q}$. If the AMs are identical, the value is 0. $\tau_{k,k',q}$ represents the reconfiguration time when auxiliary modules are installed on different machines, $\tau_{k,k',q} = \beta_{k,q} + \alpha_{k,q}$. If the auxiliary modules are installed on the same machine, the value is 0. To ensure consistency in calculations, when a machine does not use any auxiliary modules for processing, it is assumed that the installation and removal times for auxiliary module $A_0$, and the assembly and disassembly time of $A_0$ are all set to 0. The MILP model is constructed with the following constraints, aiming to minimize the total weighted tardiness.

$$Min\,TWT = \sum_{i=1}^{n} w_i \cdot t_i \tag{1}$$

s.t.

$$t_i \geq C_{i,N_i} - d_i, \forall i \tag{2}$$

$$t_i \geq 0, \forall i \tag{3}$$

**Table 3** Summary of the notation

| Notations | Explanations |
|---|---|
| Indices | |
| $i$, $i'$ | Indices of jobs, $i$, $i' = 1, 2, ..., n$ |
| $j$, $j'$ | Indices of operations, $j$, $j' = 1, 2, ..., N_i$ |
| $k$, $k'$ | Indices of machines, $k$, $k' = 1, 2, ..., \|m\|$ |
| $q$, $q'$ | Indices of AMs, $q$, $q' = 0, 1, 2, ..., \|l\|$ |
| Parameters | |
| $n$ | Set of total jobs |
| $m$ | Set of total machines |
| $l$ | Set of total AMs |
| $N_i$ | Total operation number of job $i$ |
| $O_{i,j}$ | Operation $j$ of job $i$ |
| $r_i$ | Release date of $J_i$ |
| $d_i$ | Due date of $J_i$ |
| $\omega_i$ | Tardiness weight of $J_i$ |
| $p_{i,j,k,q}$ | Processing time of $O_{i,j}$ on $M_k$ with $A_q$ |
| $\gamma_{q,q',k}$ | Reconfiguration time for $M_k$ to disassembly $A_{q'}$ and assembly $A_q$ |
| $\tau_{k,k',q}$ | Reconfiguration time for $A_q$ to be detached from $M_{k'}$ and attached to $M_k$ |
| $\alpha_{k,q}$ | Assembly time of $A_q$ on $M_k$ |
| $\beta_{k,q}$ | Disassembly time of $A_q$ on $M_k$ |
| $x_{i,j,k}$ | $x_{i,j,k} = 1$, if $O_{i,j}$ can be processed by $M_k$; |
| | $x_{i,j,k} = 0$, otherwise |
| $y_{i,j,k,q}$ | $y_{i,j,k,q} = 1$, if $O_{i,j}$ can be processed by $M_k$ with $A_q$; |
| | $y_{i,j,k,q} = 0$, otherwise |
| Decision variables | |
| $t_i$ | Tardiness of $J_i$ |
| $S_{i,j}$ | Start time of processing $O_{i,j}$ |
| $C_{i,j}$ | Completion time of processing $O_{i,j}$ |
| $X_{i,j,k}$ | $X_{i,j,k} = 1$, if $O_{i,j}$ is processed by $M_k$; |
| | $X_{i,j,k} = 0$, otherwise |
| $Y_{i,j,k,q}$ | $Y_{i,j,k,q} = 1$, if $O_{i,j}$ is processed by $M_k$ with $A_q$; |
| | $Y_{i,j,k,q} = 0$, otherwise |
| $U_{i,j,i',j'}$ | $U_{i,j,i',j'} = 1$, if $O_{i,j}$ is processed before $O_{i',j'}$ on a machine; |
| | $U_{i,j,i',j'} = 0$, otherwise |
| $V_{i,j,i',j'}$ | $V_{i,j,i',j'} = 1$, if $O_{i,j}$ is processed before $O_{i',j'}$ on an AM; |
| | $V_{i,j,i',j'} = 0$, otherwise |

Constraints (2) and (3) ensure that tardiness is only incurred when the completion time of a job exceeds its due date.

$$c_{i,j} = s_{i,j} + \sum_{k=1}^{m}\sum_{q=0}^{l} p_{i,j,k,q} \cdot Y_{i,j,k,q}, \forall i,\ j \tag{4}$$

Constraint (4) determines the completion time of each operation.

$$\sum_{k=1}^{m} X_{i,j,k} = 1, \forall i,\ j \tag{5}$$

$$X_{i,j,k} \le x_{i,j,k}, \forall i,\ j,\ k \tag{6}$$

Constraints (5) and (6) ensure that each operation is processed only once and that every operation is assigned to a machine for processing.

$$\sum_{q=0}^{l} Y_{i,j,k,q} = X_{i,j,k}, \forall i,\ j,\ k \tag{7}$$

$$Y_{i,j,k,q} \le y_{i,j,k,q}, \forall i,\ j,\ k,\ q \tag{8}$$

Constraints (7) and (8) restrict each machine to install at most one auxiliary module during processing.

$$s_{i,j} \ge r_i, \forall i,\ j = 1 \tag{9}$$

Constraint (9) ensures that the first operation of each job starts processing only after its release time.

$$s_{i,j} \ge s_{i,j-1} + \sum_{k=1}^{m}\sum_{q=0}^{l} \left(p_{i,j,k,q} \cdot Y_{i,j,k,q}\right), \forall i,\ j > 1 \tag{10}$$

Constraint (10) enforces the precedence constraints between operations of the same job, requiring the current operation to start only after the completion of the previous one.

$$s_{i,j} \ge \sum_{k=1}^{m}\sum_{q=0}^{l} (\alpha_{k,q} \cdot Y_{i,j,k,q}), \forall i,\ j \tag{11}$$

Constraint (11) ensures that a job can only begin processing after the auxiliary module is fully installed.

$$\begin{aligned} &s_{i,j} + p_{i,j,k,q} Y_{i,j,k,q} + \gamma_{q,q',k} \\ &\le s_{i',j'} + L\left(3 - U_{i,j,i',j'}\right. \\ &\quad \left. - Y_{i,j,k,q} - Y_{i',j',k,q'}\right), \forall i,\ i',\ j,\ j',\ k,\ q,\ q',\ o_{i,j} \\ &\ne o_{i',j'} \end{aligned} \tag{12}$$

$$\begin{aligned} &s_{i',j'} + p_{i',j',k,q'} Y_{i',j',k,q'} + \gamma_{q',q,k} \\ &\le s_{i,j} + L(2 + U_{i,j,i',j'} \\ &\quad - Y_{i,j,k,q} - Y_{i',j',k,q'}), \forall i,\ i',\ j,\ j',\ k,\ q,\ q',\ o_{i,j} \\ &\ne o_{i',j'} \end{aligned} \tag{13}$$

The dual constraints (12) and (13) determine the processing sequence of different operations on the same machine.

$$\begin{aligned} &s_{i,j} + p_{i,j,k,q} Y_{i,j,k,q} + \tau_{k,k',q} \\ &\le s_{i',j'} + L(3 - V_{i,j,i',j'} \\ &\quad - Y_{i,j,k,q} - Y_{i',j',k',q}), \forall i,\ i',\ j,\ j',\ k,\ k'q,\ o_{i,j} \\ &\ne o_{i',j'} \end{aligned} \tag{14}$$

$$\begin{aligned} &s_{i',j'} + p_{i',j',k',q} Y_{i',j',k',q} + \tau_{k,k',q} \\ &\le s_{i,j} + L(2 + V_{i,j,i',j'} \\ &- Y_{i,j,k,q} - Y_{i',j',k',q}), \forall i,\ i',\ j,\ j',\ k,\ k',\ q,\ o_{i,j} \\ &\ne o_{i',j'} \end{aligned} \tag{15}$$

The dual constraints (14) and (15) establish the processing order of different operations requiring the same auxiliary module.

## The proposed IGA-TNS

### Framework of IGA-TNS

The proposed IGA-TNS is an improved genetic algorithm based on the classical genetic algorithm first introduced by John Holland [26]. In the later stages of iterations, a two-stage neighborhood search based on a disjunctive graph model is incorporated for local search. The designed neighborhood structure consists of: First Stage: Selecting an appropriate processing configuration for jobs. Second Stage: Sorting the processing sequence of operations on machines. This two-stage neighborhood structure ensures the convergence efficiency of the algorithm. The algorithm flowchart is shown in Fig. 2. The following sections provide a detailed explanation of the encoding and decoding methods for the problem, as well as the design of the critical path selection and neighborhood structure tailored to the proposed FJSP-MR.

### Encoding and decoding

#### Encoding

In scheduling problems, individuals in a genetic algorithm are typically represented by chromosomes, which can be decoded into feasible scheduling solutions and modified through crossover and mutation operators. The quality of the encoding method directly impacts the search space and efficiency of the genetic algorithm. For the FJSP-MR problem, this paper employs a two-layer encoding scheme, encoding elements as integers. The two layers of encoding include

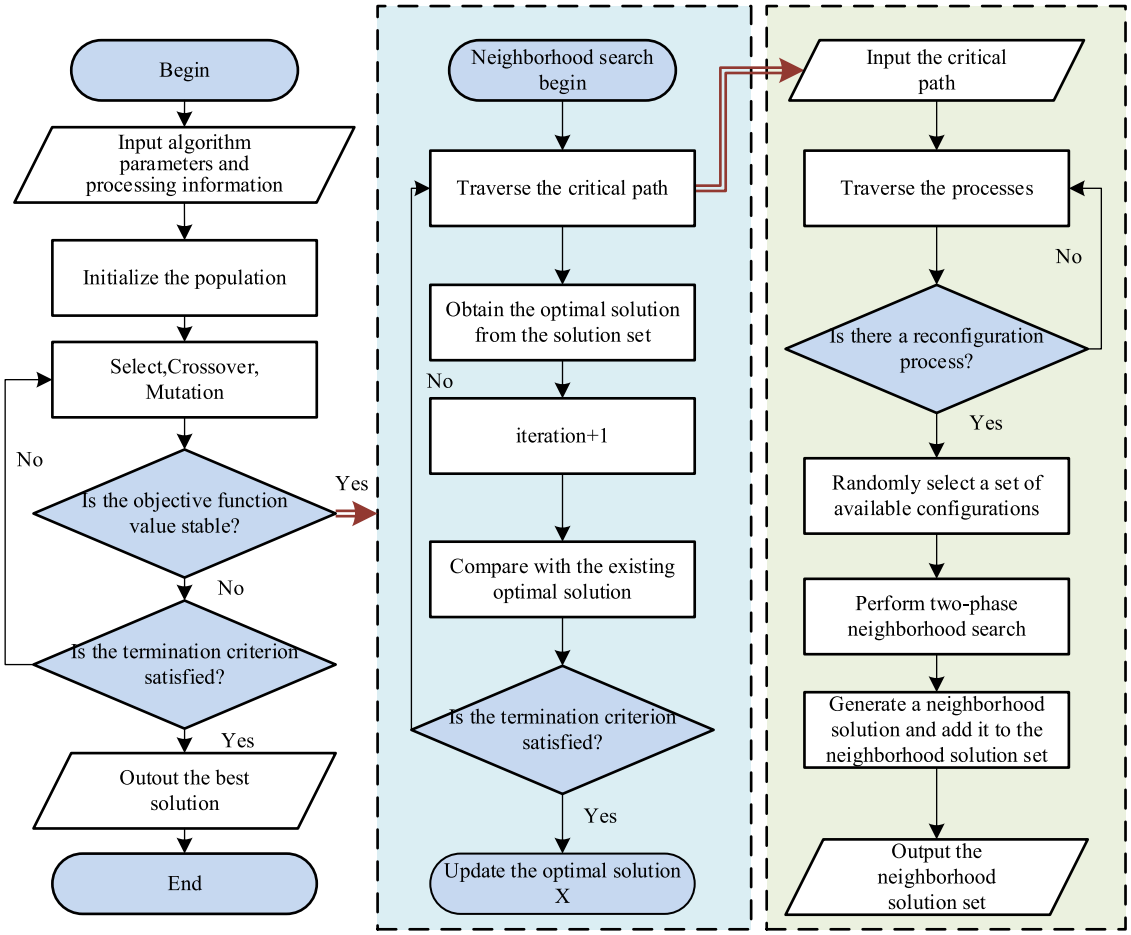


**Fig. 2** The flow chart of IGA-TNS

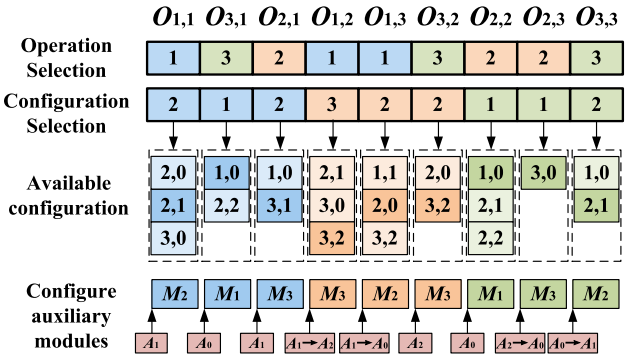


**Fig. 3** The encoding chromosome of a feasible solution

Operation Selection (OS) and Configuration Selection (CS), as shown in Fig. 3, which represents a feasible solution encoding for nine operations.

As depicted in Fig. 3, the encoding length corresponds to the total number of operations. The first layer, OS encoding, represents the operation selection. For example, the first digit "1" represents the first operation $O_{1,1}$ of workpiece 1, the second digit "1" represents the second operation $O_{1,2}$ of workpiece 1, and the first digit "3" represents the first operation $O_{3,1}$ of workpiece 3. The second layer, CS encoding, represents the selection of the processing configuration for $\{O_{1,1}, O_{1,2}, O_{1,3}, O_{2,1}, O_{2,2}, O_{2,3}, O_{3,1}, O_{3,2}, O_{3,3}\}$. For example, $O_{1,1}$ can be processed on one of three configurations. The number "2" indicates the selection of the second processing configuration $< M_2, A_1 >$ for processing.

## Decoding

The decoding method used in this paper is semi-active decoding, which ensures that no operation can be processed ahead of schedule without changing the order of operations on the machine. The Gantt chart shown in Fig. 4 illustrates the decoding of two encoding sequences from the previous example using a semi-active scheduling strategy, where gray blocks represent the installation and dismantling of auxiliary modules. "$A_1$ UP" indicates the installation of $A_1$ on the machine, while " $A_1$ Down" indicates the removal of $A_1$ on the machine, $A_2$ similarly. Arrows represent the transfer of the auxiliary module between machines.

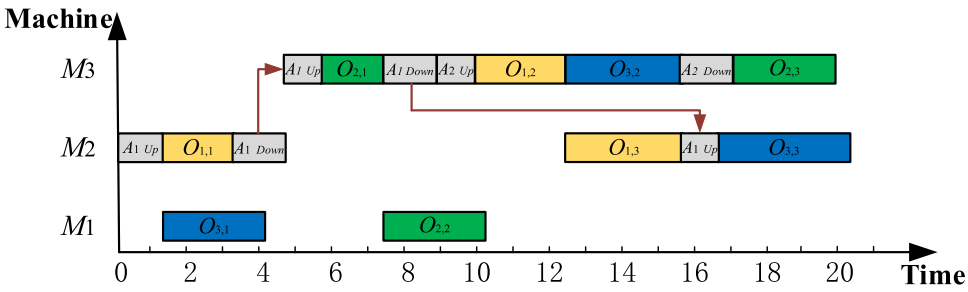


**Fig. 4** The decoded Gantt chart for the example

## Two stage neighborhood search

### Framework of neighborhood search

The K-insertion process [22] first removes the machine arc of the critical operation, then checks if another available machine has a feasible position to insert the operation. By evaluating the objective values of all feasible solutions, the operation is assigned to the best position, generating a neighborhood solution. The neighborhood structure designed by Shen et al. [14] is based on the K-insertion process and constructs a disjunctive graph model considering the auxiliary module arcs. The movement of operations on the critical path is carried out through the disjunctive graph model. The two-stage neighborhood structure proposed in this paper improves upon Shen's work by performing a two-stage neighborhood search across configurations and within the same configuration. To enhance the search efficiency, this paper designs a neighborhood search activation strategy based on the Mann–Kendall (MK) trend test [27] and uses the neighborhood search in the latter half of the population iterations. As shown in Algorithm 1, the pseudocode of the neighborhood search proposed in this paper is presented, where *iterMax* represents the maximum number of iterations for the search.

### Mann–Kendall (MK) trend test

The MK test, as part of statistical analysis, has been widely adopted as a non-parametric testing method originally proposed by Kendall [27]. This method does not require sample data to follow a specific distribution, making it suitable for detecting trends in time series data, such as consistent increases, decreases, or stabilization. In this paper, the MK test is utilized as a trigger for neighborhood search. When the objective value shows no significant variation during the algorithm's iteration process, a two-stage neighborhood search is performed. The MK test analyzes the sign differences between data points to identify trends. If a trend exists, the sign values will show a consistent increase or decrease. The process is as follows: starting from the current iteration, $N$ previous iterations are selected, and the Total Weighted Tardiness ($TWT$) of each iteration is compared with that of its preceding iterations in the time series. This generates a total of $N(N-1)/2$ data pairs. As shown in (16), a sign function is first defined, where $x_i$ represents the $TWT$ at iteration $i$, with $i > j \geq 300$. The sign functions of all data pairs are then summed to obtain $S$, as shown in (17). $S$ indicates the trend between the latter and earlier observations. A negative $S$ suggests a decreasing trend in $TWT$ during iterations. The variance $VAR(S)$ of$S$ can then be computed, and the sample data are converted into a standard normal distribution. In this study, the significance level $\alpha$ is set to 0.05. If the sample contains tied values, adjustments can be made according to the correction methods described in [28]. When the hypothesis test results indicate no significant monotonic trend in the objective function values, the neighborhood search is activated. If too few or too many generations are selected for the trend test, the evaluation may lack statistical significance, leading to unreliable conclusions. Therefore, it is necessary to determine an appropriate value for $N$ to balance computational efficiency and accuracy.

$$sgn(x_i - x_j) = \begin{cases} 1 & , x_i - x_j > 0 \\ 0 & , x_i - x_j = 0 \\ -1 & , x_i - x_j < 0 \end{cases} \tag{16}$$

$$S = \sum_{j=1}^{N-1} \sum_{i=j+1}^{N} sgn(x_i - x_j) \tag{17}$$

**Algorithm 1** Pseudocode of neighborhood search

**Input**: An initial solution $X$
**Output**: A solution $X'$ obtained by applying neiborhood search to $X$

```
X' = X
while iter ≤ iterMax do
    Find the critical paths in X' and sort them by weight(Algorithm 2)
    flag = False
    foreach critical path in X' do
        Apply the first stage neighborhood search(Algorithm 3)
        Obtain the best solution X_local in first stage neighborhood search
        if X_local is better than X' then
            Replace X' with X_local
            Apply the second stage neighborhood search for the best solution(Algorithm 4)
            Obtain the best solution X_local in second stage neighborhood search
        Iter++
        if X_local is better than X' then
            Replace X' with X_local
            flag = True
            break
        end if
    end for
    if flag = = False then
        break
    end if
end while
return solution X'
```

### FJSP-MR disjunctive graph model

The disjunctive graph model, as proposed in [29] , provides an intuitive framework for analyzing shop scheduling problems. In this model, each node represents a process, while arcs connect two adjacent processes that either belong to the same workpiece or are processed by the same machine. In this paper, the disjunctive graph model is employed to represent the Flexible Job Shop Scheduling Problem with Machine Reconfiguration (FJSP-MR). Building on the work in [30] , additional arc types are introduced for multi-layer encoding problems to incorporate other process constraints. Figure 5 illustrates the disjunctive graph $G = \{V, C_J \cup C_M \cup C_A\}$ of a feasible solution decoded from the previous section. In this graph, $V$ represents all nodes, with each node corresponding to a specific operation. Additionally, two virtual nodes, $O_{begin}$ and $O_{end}$, denote the start and end of the scheduling process, respectively. $C_J$, $C_M$, $C_A$ represent three types of directed edges: job arcs, machine arcs, and auxiliary module (AM) arcs. $C_J$ imposes precedence constraints between operations within the same job, where the preceding operation points to the subsequent one. The path length between adjacent nodes corresponds to the processing time of the preceding operation. For the first operation, it is determined by the release time of the job. $C_M$ represents the precedence between operations on the same machine, where the previous operation points to the next operation. The path length between adjacent nodes equals the processing time of the previous operation plus the necessary reconfiguration time, $C_A$ similarly. For an operation node $O_{i,j}$, the following symbols are defined: $JP_{O_{i,j}}$: the preceding operation of $O_{i,j}$ in the same job, $JS_{O_{i,j}}$: the succeeding operation of $O_{i,j}$ in the same job. $MP_{O_{i,j}}$: the preceding operation of $O_{i,j}$ on the same machine, $MS_{O_{i,j}}$: the succeeding operation of $O_{i,j}$ on the same machine, $AP_{O_{i,j}}$: the preceding operation of $O_{i,j}$ in terms of the AM, $AS_{O_{i,j}}$: the succeeding operation of $O_{i,j}$ in terms of the AM. The head and tail lengths of $O_{i,j}$ are defined as $R_{O_{i,j}}$ and $Q_{O_{i,j}}$, respectively, where: $R_{O_{i,j}}$: the longest path from $O_{begin}$ to $O_{i,j}$. $Q_{O_{i,j}}$: the longest path from $O_{i,j}$ to $O_{end}$.

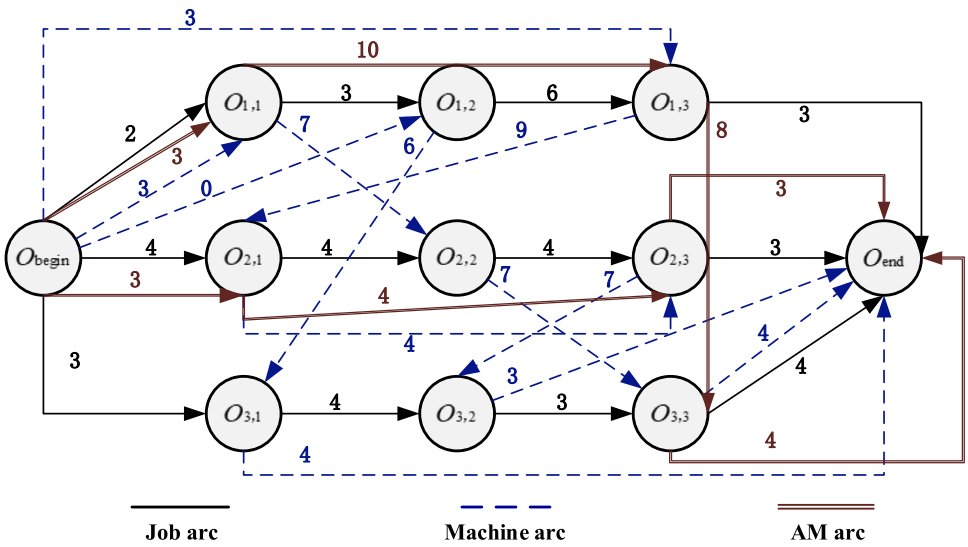


**Fig. 5** FJSP-MR Disjunctive Graph Model

**Algorithm 2** Identifying the critical path

**Input:** An solution $X$
**Output:** A set of critical paths in $X$
Decode the solution $X \Rightarrow$ obtain tasks list
**foreach** task in tasks **do**
  Check if the task is the last operation and is delayed
  $\Rightarrow O_{i,j}$ is the last operation and is delayed
  $\Rightarrow$Obtain a critical path
  critical paths'weight = $\omega i \times (Ci\text{-}di)$
  **while** True **do**
    $\Rightarrow O_{i,j}$ is the current critical operation
    $\Rightarrow$Find the last operation on the same machine $MP_{Oi,j}$
    $\Rightarrow$Find the last operation $JP_{Oi,j}$
      **if** $R_{JPOi,j} + P_{JPOi,j} > R_{MPOi,j} + P_{MPOi,j}$ **then**
        critical operation = $JP_{Oi,j}$
      **else**
        critical operation = $MP_{Oi,j}$
      **if** find the first operation of a job
        **break**
    **end while**
  **end for**
  return critical paths set $X$

## Identifying the critical path

Critical paths are the longest paths from $O_{begin}$ to $O_{end}$ in the disjunctive graph. By moving operations along these paths, the makespan can be reduced. However, since this study aims to minimize total weighted tardiness, every delayed job can impact the results, leading to multiple critical paths—one for each delayed job. By checking whether the completion time of the last operation of each job exceeds its due date, critical operations can be identified by backtracking from the delayed operations. The specific implementation is shown in Algorithm 2. Given the presence of multiple critical paths, searching all of them would reduce the algorithm's efficiency, so prioritization based on their significance is necessary. According to formula $TWT = \sum_{i=1}^{n} w_i \cdot t_i$, $w_i$ represents the importance of job $i$, and $t_i$ represents the tardiness of job $i$. The product $w_i \times t_i$ is used as the weight of the critical path, and the higher-priority critical paths are improved accordingly.

## Two stage neighborhood search

According to Reference [31] , which analyzes the neighborhood structures in existing literature for the JSP, the essence of a neighborhood structure lies in guiding critical operations to utilize machine idle time, thereby reducing the makespan. The neighborhood search for the JSP only involves moving operations forward or backward on the current processing machine. Compared to the JSP, the FJSP accounts for flexible processing in the workshop, so its neighborhood search can be divided into cross-machine operation movement and same-machine operation movement. Since the FJSP-MR simultaneously uses machines and auxiliary modules for processing, this study moves critical operations across configurations and within configurations. The movement process involves deleting and reconnecting the arcs connected to the critical operation nodes in the disjunctive graph [14]. The insertion positions are further restricted, considering whether conflicts exist between machine arcs and auxiliary module arcs. If no reconfiguration process is involved, critical operations will not undergo neighborhood search to improve efficiency and retain some good initial solutions. Since machine selection is a prerequisite

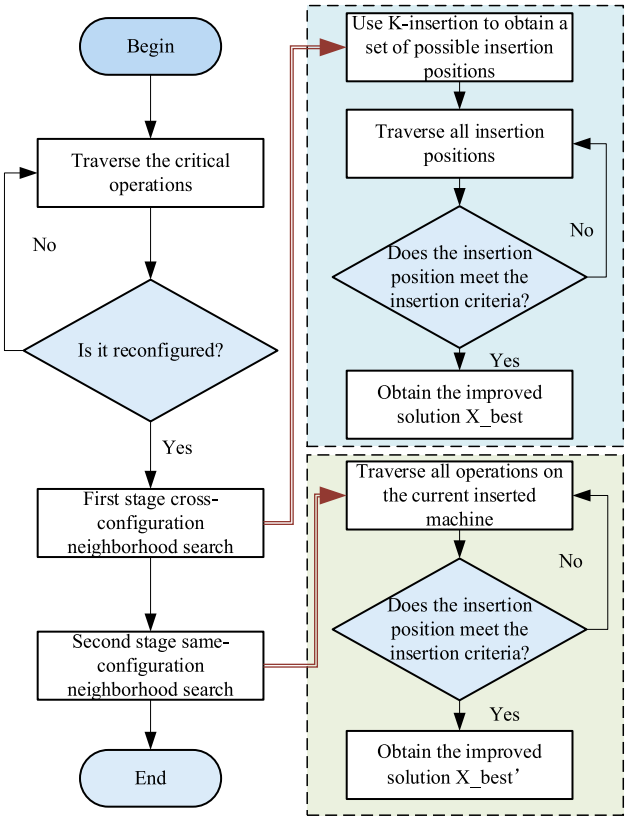


**Fig. 6** The flow chart of proposed neighborhood search

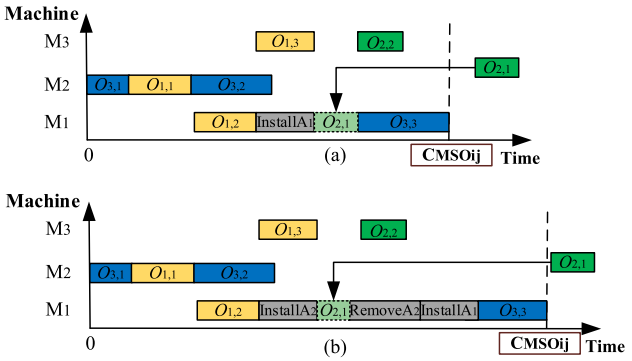


**Fig. 7** Cross-configuration neighborhood search diagram

and foundation for optimizing operation sequencing, the first stage of neighborhood search-cross configuration operation movement is conducted first. When the current solution can no longer be improved, the second stage of neighborhood search-same configuration operation movement is performed to optimize the operation sequence of the current configuration and make full use of its idle time, continuing to improve the current solution.

The two-stage neighborhood structure framework is shown in Fig. 6, and the specific implementation method is as follows:

1. Cross-configuration Neighborhood Search

The first phase of the neighborhood search in this paper is based on the original K-insertion process [22] , which was initially developed to solve the FJSP aimed at minimizing the maximum completion time. The K-insertion process begins by removing the machine arc of a critical operation, then examines another available machine to find a set of feasible positions for operation insertion. After evaluating the objective values of all possible solutions, the operation is assigned to the optimal position. Since this paper involves simultaneous selection of operations for machines and AM, by applying the K-insertion to machines and AM separately, two sets of feasible insertions, Øk and Øq, are determined respectively for machines and AM. These two sets are combined to obtain a candidate solution X. The two sets of feasible insertions Øk and Øq obtained by K-insertion ensure that the process arcs do not conflict with the machine arcs and AM arcs, as guaranteed by the constraints derived in literature [14]. However, this method only guarantees the feasibility of the candidate solution, and random insertion of the process arcs into the feasible positions of key processes may result in invalid solutions that cannot be improved. Therefore, it is necessary to further discuss the insertion positions, and the specific implementation process is shown in Algorithm 3.

Assuming the current operation to be moved is $O_{i,j}$, a new processing configuration $< M_k, A_q >$ is randomly selected. Since the assembly and disassembly time of AM is significant, the insertion position of the process needs to be considered along with whether there is a reconfiguration process between the preceding and succeeding processes. For simple process blocks on the machine $MP_{O_{i,j}}$, $O_{i,j}$, $MS_{O_{i,j}}$, we only consider the processing order before and after the three processes on the machine, without considering the process constraints or AM constraints. If the AM is different for $MP_{O_{i,j}}$, $O_{i,j}$, $MS_{O_{i,j}}$, assuming $A_q$ is currently available, the earliest completion time of $MS_{O_{i,j}}$ is shown in Eq. (18). If $A_q$ is currently installed on another machine and is unavailable, the completion time of $MS_{O_{i,j}}$ will be shown in the Eq. (19). If $MP_{O_{i,j}}$ and $O_{i,j}$ use the same AM, then $\beta_{k,q'}$ and $\alpha_{k,q}$ will both be 0; similarly, if $MS_{O_{i,j}}$ and $O_{i,j}$ use the same AM, $\beta_{k,q}$ and $\alpha_{k,q''}$ will both be 0. In this case, the completion time of $MS_{O_{i,j}}$ will be advanced. As shown in Fig. 7, when $O_{2,1}$ is moved to the position shown in the diagram, in (a) where $O_{2,1}$ and $O_{3,3}$ use the same processing configuration, the total completion time is ahead of (b). Therefore, further select solutions in the candidate solutions that use the same AM for the processes before and after the insertion position as neighborhood solutions, optimizing the objective by reducing reconfiguration time.

2. Same-configuration Neighborhood Search

After completing the first stage neighborhood search, in order to further improve the algorithm's search performance

and make full use of the current configuration's idle time, a second-stage neighborhood search is performed by optimizing the sequence of operations on the same configuration. The same-configuration neighborhood search reduces the idle time of the machine or AM by moving the current operation nodes on the machine arc and AM arc in the disjunctive graph, both forwards and backwards. Therefore, the machine arc and AM arc should be modified separately. Assume $MP_\omega$ and $MS_\omega$ are two adjacent operations on the processing machine selected by $O_{i,j}$ after the first stage neighborhood search. Whether inserting operation $O_{i,j}$ between $MP_\omega$ and $MS_\omega$ can shorten the completion time depends on whether the current machine's idle time is zero. As shown in Eq. (20), when the machine idle time between $MP_\omega$ and $MS_\omega$ intersects with the completion time of the previous operation $JP_{O_{\mathrm{i,j}}}$ and the start time of the subsequent operation $JS_{O_{\mathrm{i,j}}}$, and the intersection is not zero (i.e., there is available idle time on the machine), inserting $O_{i,j}$ between $MP_\omega$ and $MS_\omega$ may shorten the completion time of the job, thereby reducing the total weighted tardiness. Similarly, the operation shift on the AM arc is shown in Eq. (21). When the AM idle time between $AP_\omega$ and $AS_\omega$ intersects with the completion time of $JP_{O_{\mathrm{i,j}}}$ and the start time of $JS_{O_{\mathrm{i,j}}}$, and the intersection is not zero, $O_{i,j}$ can be inserted between $AP_\omega$ and $AS_\omega$. The specific implementation process is shown in Algorithm 4. After obtaining the two sets of feasible insertion positions, they are combined to form the candidate solution $X$. After obtaining the candidate solution, it is necessary to further determine whether it is a feasible solution.

$$C_{MSO_{\mathrm{ij}}} = C_{MPO_{\mathrm{ij}}} + \beta'_{k,q} + \alpha_{k,q} + P_{oij} + \beta_{k,q} + \alpha_{k,q''} + P_{MSO_{\mathrm{ij}}} \tag{18}$$

$$C_{MSO_{\mathrm{ij}}} = max(C_{MPO_{\mathrm{ij}}}, C_{APO_{\mathrm{ij}}}) + \beta_{k,q'} + \alpha_{k,q} + P_{oij} + \beta_{k,q} + \alpha_{k,q''} + P_{MSO_{\mathrm{ij}}} \tag{19}$$

$$\left[C_{JPO_{ij}},\ S_{JSO_{ij}}\right] \cap \left[C_{MP_\omega},\ S_{MS_\omega}\right] \neq 0 \tag{20}$$

$$\left[C_{JPO_{ij}},\ S_{JSO_{ij}}\right] \cap \left[C_{AP_\omega},\ S_{AS_\omega}\right] \neq 0 \tag{21}$$

**Algorithm 3** Pseudocode of cross-configuration neighborhood search

**Input:** A critical path for solution $X$
**Output:** An improved solution $X_{best}$ obtained by the neighborhood structure

$X_{best} = X$
**foreach** critical operation $O_{i,j}$ in critical path **do**
  **if** $MP_{Oi,j} == AP_{Oi,j}$ or $MS_{Oi,j} == AS_{Oi,j}$ **then**
    **continue**
  **end if**
  Choose an alternative configuration <$M_k$,$A_q$> for $O_{i,j}$ randomly
  ⇒Obtain a set of feasible insertions Øk by applying K-insertion to $O_{i,j}$ and $M_k$
  ⇒Obtain a set of feasible insertions Øq by applying K-insertion to $O_{i,j}$ and $A_q$
  **foreach** insertion k in Øk **do**
    **foreach** insertion q in Øq **do**
      **if** k and q are not conflicting **then**
        **if** Using Same AM($MP_{Oi,j}$, $O_{i,j}$) or Using Same AM($MS_{Oi,j}$, $O_{i,j}$) **then**
          Generate a neighboring solution $X_{neiborhood}$
          **if** $X_{neiborhood}$ is better than $X_{best}$ **then**
            Replace $X_{best}$ with $X_{neiborhood}$
          **end if**
        **end if**
      **end for**
    **end for**
  Apply same-configuration neighborhood search for $X_{best}$
**end for**
return the best solution $X_{best}$

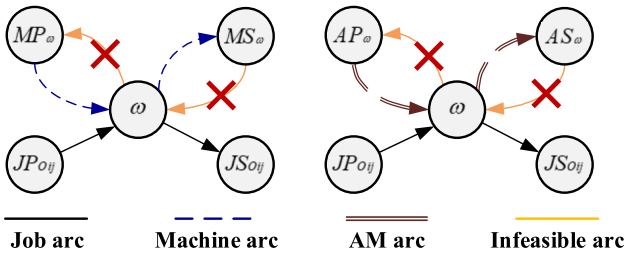


**Fig. 8** Feasible insertion position $\omega$ diagram

The proposed IGA-TNS is dominated by the genetic algorithm operations and the two-stage neighborhood search. The genetic operations, including selection, crossover, and mutation, have a complexity of $O(N_p)$, where $N_p$ is the population size. Fitness evaluation, which involves calculating the total weighted tardiness for each individual, requires $O(N_p \cdot N_j \cdot N_{op})$, where $N_j$ is the number of jobs and$N_{op}$ is the number of operations per job. The two-stage neighborhood search has a complexity of $O(M \cdot L \cdot G + M \cdot G)$, where $M$ is the number of machines, $L$ is the number of auxiliary modules, and $G$ is the number of operations. Combining the complexities of the genetic operations and the neighborhood search, the overall computational complexity of the IGA-TNS algorithm is $O(N \cdot (N_p \cdot N_j \cdot N_{op} + N_p + M \cdot L \cdot G + M \cdot G))$, where $N$ is the number of generations.

**Algorithm 4** Pseudocode of same-configuration neighborhood search

**Input:** A solution $X$
**Output:** An improved solution $X'_{best}$
$X'_{best} = X$
**foreach** $O_{i,j,k}$ processed by $M_k$ **do**
  **if** $[C_{JPOi,j}, S_{JSOi,j}] \cap [C_{MP\omega}, S_{MS\omega}] \neq 0$ **then**
    ⇒Obtain a set of feasible insertions Øk'
**end for**
**foreach** $O_{i,j,k}$ processed by $A_q$ **do**
  **if** $[C_{JPOi,j}, S_{JSOi,j}] \cap [C_{AP\omega}, S_{AS\omega}] \neq 0$ **then**
    ⇒Obtain a set of feasible insertions Øq'
**end for**
  **foreach** insertion k in Øk' **do**
    **foreach** insertion q in Øq' **do**
      **if** Using Same AM($MP_{Oi,j}$, $O_{i,j}$) or Using Same AM($MS_{Oi,j}$, $O_{i,j}$) **then**
        Generate a neighboring solution $X_{neiborhood}$
        Check $X_{neiborhood}$ feasibility
      **if** $X_{neiborhood}$ is better than $X'_{best}$ **then**
        Replace $X'_{best}$ with $X_{neiborhood}$
    **end for**
  **end for**
  return $X'_{best}$

The determination conditions from reference [14] ensure that there are no conflicts between the machine arc and the auxiliary module arc, but further checks are needed to verify whether conflicts exist between the job arc and the machine arc, as well as between the job arc and the auxiliary module arc. As shown in Fig. 8, let the current insertion position be $\omega$. To ensure that the current solution is feasible, it is necessary that $\omega$ does not have arcs pointing to $MP_\omega$ and $AP_\omega$, and that neither $MS_\omega$ nor $AS_\omega$ have arcs pointing to $\omega$. According to the proof in reference [31] , when condition 1 is satisfied, the head length of $MP_\omega$ plus the processing time is greater than the head length of $JPO_{i,j}$ plus the processing time, it can be ensured that there is no arc from $\omega$ to $MP_\omega$. When condition 2 is satisfied, it guarantees that there is no arc from $MS_\omega$ to $\omega$. The other conditions follow similarly. When all four conditions are simultaneously met, there are no conflicts between the job arc, machine arc, and AM arc. At this point, the candidate solution is feasible.

Condition 1. $R_{MS_\omega} + P_{MS_\omega} > R_{JPO_{i,j}} + P_{JPO_{i,j}}$.
Condition 2. $Q_{MP_\omega} > Q_{JSO_{i,j}}$.
Condition 3. $R_{AS_\omega} + P_{AS_\omega} > R_{JPO_{i,j}} + P_{JPO_{i,j}}$.
Condition 4. $Q_{AP_\omega} > Q_{JSO_{i,j}}$.

## Experimental results and discussion

### Test instances

In this chapter, three sets of test instances are set up to test the efficiency of the algorithm in different scenarios. All parameter generation is derived from the existing literature. To evaluate the performance of the proposed MILP model and IGA-TNS, twenty instances from the well-known FJSP benchmark sets, Fdata (MFJS01-10) [23] and BRdata (MK01–10) [24] , are expanded into FJSP-MR and conducted experiments. This paper also introduces Flex [14] to represent the impact of different shop floor flexibilities on the algorithm. The supplementary data is as follows:

- Number of AMs $l \in [m/4, m/2]$.
- Release date $r_i \in [0, pmax]$.
- Due date $d_i = 1.2 \sum_{j=1}^{N_i} p_{i,j}^{min}$ for group A and C; $d_i = 1.0 \sum_{j=1}^{N_i} p_{i,j}^{min}$ for group B.
- Assembly time $\alpha_{k,q} \in [p_{max}/\eta_k, 2 \cdot p_{max}/\eta_k]$.
- Disassembly time $\beta_{k,q} \in [0.8, 1.2] \cdot \alpha_{k,q}$.
- $Flex = 2$ for group A and B; $Flex = 3$ for group C.
- $p_{i,j,k,q,S} = \begin{cases} \infty & , S_k = 0 \\ p_{avg} \cdot x & , S_k = 1 \end{cases}$

The calculation of $pmax$ and $p_{i,j}^{min}$ refers to reference [14] , $\eta_k$ is the efficiency factor of $M_k$, and the setting of $\eta_k$ is based on reference [32]. The higher the efficiency, the shorter the time required for the installation and removal of the auxiliary modules. $p_{i,j,k,q,S}$ represents the processing time for $M_k$ installation of $A_q$ to process $O_{i,j}$. $S_k$ represents whether $M_k$ has the module $A_q$ installed, The value of $S_k$ is zero indicates that $M_k$ does not have the module $A_q$ installed and cannot process $O_{i,j}$. The value of $S_k$ is one indicates that $M_k$ has the module $A_q$ installed and cannot process $O_{i,j}$. $p_{avg}$ represents the average processing time of all processing configurations for $O_{i,j}$. Define $x$ as a random variable that takes values between 0.8 and 1.0, inclusive. The difference between groups A and B lies in the delivery dates, which are used to test the algorithm's performance under urgent delivery conditions. Group C differs in having greater flexibility.

## Parameter setting

The proposed algorithm IGA-TNS includes five key parameters that significantly impact the results: population size $N_p$, crossover probability $C_r$, mutation probability $M_r$, trend test generations $N$, and neighborhood search iteration count $iterMax$. Through preliminary experimental analysis, the offspring size is set to be equal to the population size. Parameter levels are set as: $N_p \in \{100, 200, 300, 400\}$, $C_r \in \{0.80, 0.85, 0.90, 0.95\}$, $M_r \in \{0.10, 0.15, 0.20, 0.25\}$,$iterMax \in \{10, 15, 20, 25\}$, and $N \in \{20, 30, 40, 50\}$. To determine the above five parameters, an orthogonal experiment is designed using instance MFJS01. The average values of 20 repetitions are shown in Table 4, and the trend chart is shown in Fig. 9.

It can be observed that the performance is significantly better when the $N_p = 400$, $itMax = 20$, and $M_r = 0.10$, compared to other levels. From the trend in Fig. 9, it is evident that the algorithm achieves the minimum $Mean$ when $C_r = 0.85$ and $N = 20$. This parameter configuration is applied across all numerical experiments.

## Experimental results and analyses

This paper compares the proposed MILP model and IGA-VNS with the basic genetic algorithm (GA) and the genetic algorithm with neighborhood search (GA-VNS) [33]. The only difference between GA-VNS and IGA-VNS lies in the neighborhood structure. By comparing GA and GA-VNS, the effectiveness of the proposed two-stage neighborhood search is evaluated. All test instances are generated according to the method described in Section Test instances, and the parameter settings follow the parameter selection described in Section Parameter setting. The IBM CPLEX solver is adopted for the MILP model and all algorithms are programmed with Python 3.10 by PyCharm. The algorithms are running on a computer having an Intel Core I9 14,900 KF processor running at 3.5 GHz with 32 GBytes of RAM. Twenty independent runs were performed on all the instances with a termination criterion n · m · l · 100 ms, where the best values were recorded. All MILP models are run with a termination criterion of 3600 s [34] , The symbol "-" indicates that the CPLEX solver is unable to find a feasible solution. As shown in Tables 5, 6 and 7, these are the results of the experimental runs.

Figure 10 presents the convergence graphs randomly selected from 20 independent runs of the MFJS01-10 instances under Group A parameter settings. It is observed that for most instances, after approximately 300 iterations, when the objective values stabilize, the neighborhood search is activated following the MK test, resulting in a sharp decline in objective values. This observation demonstrates that the two-stage neighborhood search, guided by trend testing, is highly effective during the latter stages of the iterations.

Specifically, for the MFJS04 and MFJS06 instances, the neighborhood search did not immediately reduce the objective values at 300 iterations, indicating a potential entrapment in local optima. However, after additional iterations, particularly at 400 and 450 iterations, a significant drop in objective values is observed, suggesting that the algorithm successfully escapes local optima and continues optimization. These results highlight the effectiveness of the proposed neighborhood structure in sustaining the search process and finding improved solutions, even when initially encountering local optima.

As presented in the three tables, the proposed MILP model successfully achieves optimal solutions for MFJS01-07 in Group A, MFJS01-06 in Group B, and MFJS01-05 in Group C. For the MFJS06 instance, the increased flexibility in Group C leads to more complex processing scenarios compared to other groups. This added complexity results in a larger problem size, rendering the MILP model unable to achieve an optimal solution within the 3600s time limit. For the MFJS07 instance, the earlier delivery deadlines in Group B compared to Group A exacerbate the dependency among processing

**Table 4** Mean value of MFJS01 different combination parameters

| Test | $N$ | $iterMax$ | $M_r$ | $C_r$ | $N_p$ | $Mean$ |
|---|---|---|---|---|---|---|
| 1 | 20 | 10 | 0.10 | 0.80 | 100 | 1046.20 |
| 2 | 20 | 15 | 0.15 | 0.85 | 200 | 997.16 |
| 3 | 20 | 20 | 0.20 | 0.90 | 300 | 956.08 |
| 4 | 20 | 25 | 0.25 | 0.95 | 400 | 960.28 |
| 5 | 30 | 10 | 0.15 | 0.90 | 400 | 978.40 |
| 6 | 30 | 15 | 0.10 | 0.95 | 300 | 964.00 |
| 7 | 30 | 20 | 0.25 | 0.80 | 200 | 973.96 |
| 8 | 30 | 25 | 0.20 | 0.85 | 100 | 1055.24 |
| 9 | 40 | 10 | 0.20 | 0.95 | 200 | 1128.08 |
| 10 | 40 | 15 | 0.25 | 0.90 | 100 | 1180.08 |
| 11 | 40 | 20 | 0.10 | 0.85 | 400 | 969.08 |
| 12 | 40 | 25 | 0.15 | 0.80 | 300 | 1047.60 |
| 13 | 50 | 10 | 0.25 | 0.85 | 300 | 1020.76 |
| 14 | 50 | 15 | 0.20 | 0.80 | 400 | 976.04 |
| 15 | 50 | 20 | 0.15 | 0.95 | 100 | 1030.76 |
| 16 | 50 | 25 | 0.10 | 0.90 | 200 | 1013.36 |

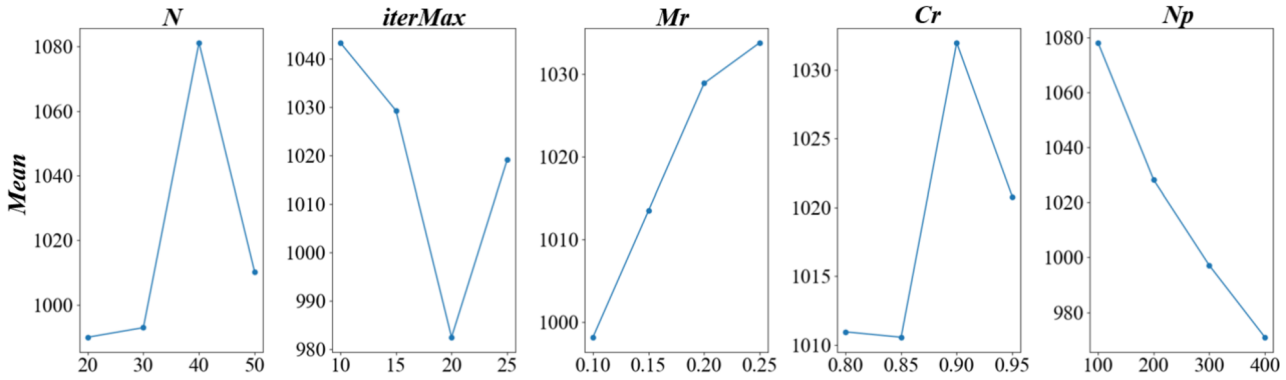


**Fig. 9** Orthogonal test trend chart

operations, potentially altering the optimization path of the objective function. This increased interdependency further complicates the problem, leading to suboptimal solutions when constrained by limited computational time. For the MK03 and MK05-10 instances, the significant increase in problem size causes an exponential expansion of the search tree. As a result, the MILP model is unable to identify a feasible solution within 3600 s. In summary, while the MILP model demonstrates efficiency in solving small-scale problems and produces high-quality results within a reasonable computational time, its performance deteriorates significantly as the problem size and model complexity increase. Consequently, the MILP model is not directly suitable for solving large-scale FJSP-MR problems due to its high computational demands.

For the MFJS01-MR-A and MFJS01-MR-C instances, the basic genetic algorithm is also capable of obtaining optimal solutions. However, on other instances, when the problem size increases, the performance of the basic algorithm falls significantly short of that achieved by GA-VNS and IGA-TNS. This indicates that the integration of genetic algorithms with neighborhood search can substantially enhance the algorithm's effectiveness. As shown in the last three rows of tables, all algorithms were individually compared with IGA-TNS. The term "better" indicates the number of instances where IGA-TNS performed superiorly compared to the current algorithm. "Even" denotes the number of instances where IGA-TNS achieved results equivalent to those of the current algorithm. "Worse" signifies the number of instances where IGA-TNS performed inferiorly to the current algorithm. Compared to IGA-TNS, GA-VNS achieved the same results in 5 instances and even outperformed IGA-TNS in 8 instances, demonstrating that the neighborhood structure

**Table 5** Experimental results for group A

| Instance | n × m × l | MILP | | | GA | | GA-VNS | | IGA-TNS | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Best | Gap (%) | CPU time | Best | Avg | Best | Avg | Best | Avg |
| MFJS01-MR-A | 5 × 6 × 2 | **943** | 0.00 | 7.06 | **943** | 994.20 | **943** | 990.20 | **943** | 983.60 |
| MFJS02-MR-A | 5 × 7 × 2 | **672** | 0.00 | 9.85 | 763 | 903.60 | **672** | 830.50 | **672** | 769.05 |
| MFJS03-MR-A | 6 × 7 × 2 | **1399** | 0.00 | 18.28 | 2367 | 2546.60 | 2296 | 2463.45 | 2178 | 2485.50 |
| MFJS04-MR-A | 7 × 7 × 2 | **1996** | 0.00 | 191.42 | 2754 | 3254.15 | 2502 | 2811.30 | 2484 | 2742.55 |
| MFJS05-MR-A | 7 × 7 × 3 | **2297** | 0.00 | 1057.61 | 3719 | 4216.65 | 3432 | 3888.65 | 3363 | 3915.85 |
| MFJS06-MR-A | 8 × 7 × 2 | **2889** | 19.89 | 3600.86 | 4149 | 5129.85 | 4902 | 5765.25 | 4396 | 5493.65 |
| MFJS07-MR-A | 8 × 7 × 3 | **3853** | 57.78 | 3600.42 | 6385 | 7368.60 | 6009 | 6883.65 | 5120 | 5857.60 |
| MFJS08-MR-A | 9 × 8 × 3 | 6408 | 48.29 | 3601.34 | 6292 | 6666.50 | 5310 | 6721.85 | **5306** | 6578.40 |
| MFJS09-MR-A | 11 × 8 × 4 | 10,232 | 69.83 | 3600.97 | 10,994 | 13,011.25 | 10,592 | 12,762.85 | **8307** | 9067.50 |
| MFJS10-MR-A | 12 × 8 × 4 | 13,528 | 80.50 | 3606.41 | 13,369 | 16,148.60 | 13,136 | 14,582.60 | **12,643** | 15,810.70 |
| MK01-MR-A | 10 × 6 × 2 | 351 | 88.79 | 3607.98 | 387 | 553.40 | 378 | 527.25 | **340** | 463.95 |
| MK02-MR-A | 10 × 6 × 3 | 348 | 85.20 | 3612.84 | 375 | 522.75 | 356 | 495.95 | **344** | 445.40 |
| MK03-MR-A | 15 × 8 × 2 | – | – | – | 4309 | 5559.05 | 4021 | 5329.25 | **3642** | 4759.00 |
| MK04-MR-A | 15 × 8 × 3 | 2515 | 99.80 | 3606.84 | 2413 | 3336.85 | 2108 | 2996.80 | **2066** | 2569.65 |
| MK05-MR-A | 15 × 4 × 2 | – | – | – | 5845 | 7244.75 | 5424 | 6752.90 | **5299** | 6297.10 |
| MK06-MR-A | 10 × 15 × 4 | – | – | – | 9016 | 11,341.60 | **7383** | 10,575.55 | 7818 | 10,575.55 |
| MK07-MR-A | 20 × 5 × 2 | – | – | – | 9175 | 11,253.15 | 8252 | 10,578.35 | **7665** | 9718.95 |
| MK08-MR-A | 20 × 10 × 3 | – | – | – | 11,768 | 13,933.95 | 11,110 | 13,449.75 | **11,056** | 12,581.45 |
| MK09-MR-A | 20 × 10 × 4 | – | – | – | 10,953 | 12,669.25 | 10,584 | 12,330.35 | **10,468** | 11,612.60 |
| MK10-MR-A | 20 × 15 × 5 | – | – | – | 10,681 | 12,137.60 | 10,563 | 11,809.20 | **10,439** | 11,241.70 |
| #better | | 12 | | | 18 | | 17 | | – | |
| #even | | 2 | | | 1 | | 2 | | – | |
| #worse | | 5 | | | 1 | | 1 | | – | |

Bold value represents the optimal value obtained from all methods under the current example

**Table 6** Experimental results for group B

| Instance | n × m × l | MILP | | | GA | | GA-VNS | | IGA-TNS | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Best | Gap (%) | CPU time | Best | Avg | Best | Avg | Best | Avg |
| MFJS01-MR-B | 5 × 6 × 2 | **1616** | 0.00 | 7.30 | 1872 | 2396.00 | 1776 | 2426.10 | **1616** | 2181.40 |
| MFJS02-MR-B | 5 × 7 × 2 | **1408** | 0.00 | 5.63 | 1602 | 2112.40 | **1408** | 2290.45 | **1408** | 2149.20 |
| MFJS03-MR-B | 6 × 7 × 2 | **1933** | 0.00 | 23.05 | 2113 | 2956.20 | 2153 | 3302.85 | **1933** | 2977.45 |
| MFJS04-MR-B | 7 × 7 × 2 | **2705** | 0.00 | 260.25 | 2815 | 4091.15 | **2705** | 4415.75 | 2716 | 3982.85 |
| MFJS05-MR-B | 7 × 7 × 3 | **3329** | 0.00 | 1514.45 | 3456 | 4167.70 | 3509 | 4785.85 | **3329** | 4231.15 |
| MFJS06-MR-B | 8 × 7 × 2 | **3745** | 16.27 | 3601.34 | 4604 | 5913.80 | 4760 | 6432.60 | 4049 | 5465.45 |
| MFJS07-MR-B | 8 × 7 × 3 | 7028 | 32.37 | 3600.42 | 7003 | 8271.55 | 6352 | 8365.35 | **5497** | 7828.45 |
| MFJS08-MR-B | 9 × 8 × 3 | 7266 | 45.11 | 3602.36 | 8266 | 9949.60 | **7158** | 9677.30 | 7406 | 8979.95 |
| MFJS09-MR-B | 11 × 8 × 4 | 14,912 | 62.91 | 3605.11 | 13,366 | 16,023.15 | 12,402 | 14,924.85 | **10,401** | 14,576.35 |
| MFJS10-MR-B | 12 × 8 × 4 | 14,129 | 81.23 | 3602.42 | 16,654 | 20,929.60 | **13,947** | 19,151.10 | 15,403 | 18,390.80 |
| MK01-MR-B | 10 × 6 × 2 | 470 | 72.27 | 3603.92 | 478 | 615.20 | 463 | 574.75 | **427** | 524.05 |
| MK02-MR-B | 10 × 6 × 3 | 466 | 68.05 | 3605.42 | 453 | 603.55 | 423 | 583.00 | **419** | 501.15 |
| MK03-MR-B | 15 × 8 × 2 | – | – | – | 4464 | 5720.60 | 4464 | 5445.20 | **4307** | 4979.20 |
| MK04-MR-B | 15 × 8 × 3 | 2911 | 91.16 | 3600.02 | 2402 | 3531.70 | 2052 | 3246.20 | **1998** | 2762.05 |
| MK05-MR-B | 15 × 4 × 2 | – | – | – | 6218 | 7337.00 | 5685 | 7057.20 | **5358** | 6474.55 |
| MK06-MR-B | 10 × 15 × 4 | – | – | – | 9158 | 11,694.75 | 8799 | 11,112.70 | **8287** | 9554.55 |
| MK07-MR-B | 20 × 5 × 2 | – | – | – | 8876 | 11,592.95 | 8566 | 10,705.45 | **8287** | 9554.55 |
| MK08-MR-B | 20 × 10 × 3 | – | – | – | 12,218 | 14,115.95 | 11,198 | 13,447.95 | **11,037** | 12,655.50 |
| MK09-MR-B | 20 × 10 × 4 | – | – | – | 11,142 | 12,670.95 | 10,935 | 12,240.25 | **10,648** | 11,524.50 |
| MK10-MR-B | 20 × 15 × 5 | – | – | – | 10,655 | 12,305.80 | 10,253 | 12,047.55 | **10,151** | 11,388.20 |
| #better | | 12 | | | 20 | | 16 | | – | |
| #even | | 4 | | | 0 | | 1 | | – | |
| #worse | | 4 | | | 0 | | 3 | | – | |

Bold value represents the optimal value obtained from all methods under the current example

**Table 7** Experimental results for group C

| Instance | n × m × l | MILP | | | GA | | GA-VNS | | IGA-TNS | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Best | Gap (%) | CPU time | Best | Avg | Best | Avg | Best | Avg |
| MFJS01-MR-C | 5 × 6 × 2 | **946** | 0.00 | 40.81 | **946** | 1454.45 | **946** | 1690.95 | **946** | 1416.30 |
| MFJS02-MR-C | 5 × 7 × 2 | **674** | 0.00 | 24.06 | 712 | 1714.25 | **674** | 1514.75 | **674** | 1443.14 |
| MFJS03-MR-C | 6 × 7 × 2 | **1323** | 0.00 | 465.59 | 1493 | 2550.60 | **1323** | 2828.70 | 1408 | 2432.55 |
| MFJS04-MR-C | 7 × 7 × 2 | **1537** | 0.00 | 922.17 | 2046 | 3193.35 | 2167.8 | 3336.05 | **1537** | 3203.12 |
| MFJS05-MR-C | 7 × 7 × 3 | **1985** | 6.70 | 3600.53 | 2263 | 3284.10 | 2076 | 3553.15 | **1985** | 3052.05 |
| MFJS06-MR-C | 8 × 7 × 2 | 3502 | 5.04 | 3600.95 | 3875 | 5106.95 | **3133** | 4636.90 | 3191 | 4281.00 |
| MFJS07-MR-C | 8 × 7 × 3 | 4330 | 52.22 | 3603.27 | 4545 | 6405.65 | 4439 | 6161.50 | **4257** | 5796.90 |
| MFJS08-MR-C | 9 × 8 × 3 | 6292 | 57.94 | 3615.13 | 5612 | 7759.45 | 5564 | 7683.25 | **4804** | 7140.60 |
| MFJS09-MR-C | 11 × 8 × 4 | 12,989 | 82.17 | 3601.73 | 9682 | 13,626.80 | **8914** | 12,637.45 | 9568 | 11,658.05 |
| MFJS10-MR-C | 12 × 8 × 4 | 15,405 | 81.22 | 3603.23 | 13,147 | 17,316.35 | 12,413 | 16,099.35 | **12,072** | 14,779.40 |
| MK01-MR-C | 10 × 6 × 2 | 453 | 92.77 | 3603.48 | 439 | 593.55 | **406** | 503.45 | 417 | 557.85 |
| MK02-MR-C | 10 × 6 × 3 | 414 | 92.66 | 3602.98 | 369 | 505.95 | 350 | 499.70 | **321** | 433.65 |
| MK03-MR-C | 15 × 8 × 2 | – | – | – | 4869 | 6636.40 | 4482 | 6177.45 | **3963** | 5415.00 |
| MK04-MR-C | 15 × 8 × 3 | – | – | – | 2316 | 3325.65 | 2279 | 2980.00 | **2193** | 2583.85 |
| MK05-MR-C | 15 × 4 × 2 | – | – | – | 6511 | 8414.15 | 6510 | 8067.90 | **6421** | 7305.80 |
| MK06-MR-C | 10 × 15 × 4 | – | – | – | 7994 | 11,220.90 | 7886 | 10,576.85 | **7724** | 9198.65 |
| MK07-MR-C | 20 × 5 × 2 | – | – | – | 10,498 | 14,780.80 | 9514 | 13,118.65 | **9291** | 11,632.35 |
| MK08-MR-C | 20 × 10 × 3 | – | – | – | 11,622 | 12,985.85 | 11,322 | 12,672.50 | **11,293** | 11,981.80 |
| MK09-MR-C | 20 × 10 × 4 | – | – | – | 11,076 | 12,753.40 | 11,052 | 12,413.80 | **10,710** | 11,598.25 |
| MK10-MR-C | 20 × 15 × 5 | – | – | – | 10,373 | 11,960.20 | 9622 | 11,555.80 | **9493** | 10,915.30 |
| #better | | 15 | | | 19 | | 14 | | – | |
| #even | | 4 | | | 1 | | 2 | | – | |
| #worse | | 1 | | | 0 | | 4 | | – | |

Bold value represents the optimal value obtained from all methods under the current example

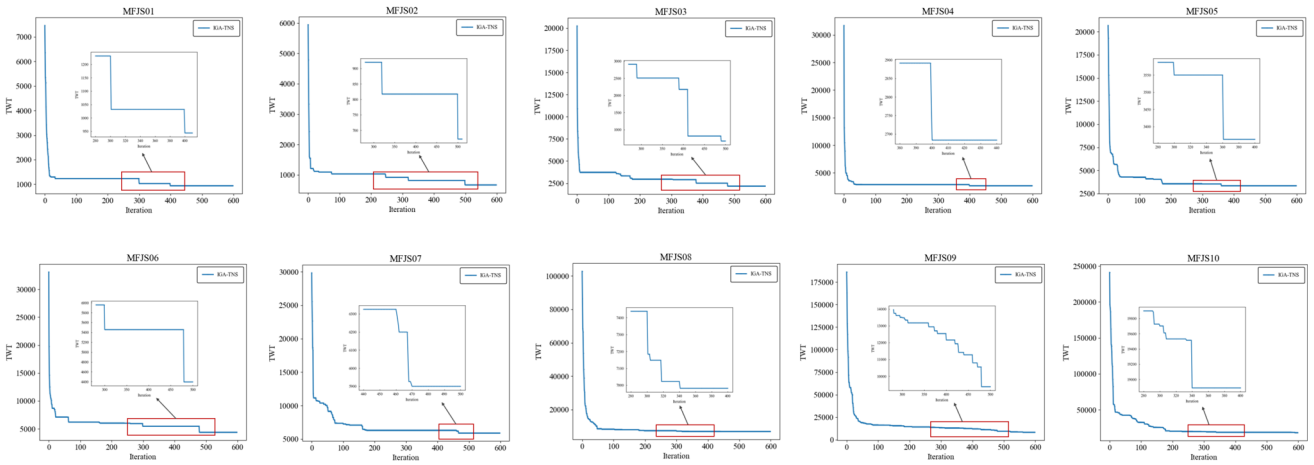


**Fig. 10** Convergence curves and an intuitive illustration of neighborhood structure for group A MFJS01-10 instances

of GA-VNS indeed significantly enhances the genetic algorithm. However, in most instances, both the best and average values obtained by GA-VNS are inferior to those achieved by IGA-TNS, particularly as the size of the instances increases. The only difference between GA-VNS and IGA-TNS lies in their approaches to neighborhood search, which suggests that the neighborhood structure proposed in this paper is more suitable for FJSP-MR.

Figures 11, 12 and 13 provide detailed violin plots comparing the performance of three algorithms—GA, GA-VNS, and IGA-TNS—across groups A, B, and C of the MK instances for the Flexible Job Shop Scheduling Problem with Machine Flexibility and Randomness (FJSP-MR). The plots highlight key statistical measures, including the mean, median, and optimal values, offering insights into the effectiveness and stability of the algorithms. As illustrated in Fig. 11, GA-VNS and IGA-TNS significantly outperform the basic GA in both optimal solutions and overall statistical metrics across most instances. For MK02-MR, MK04-MR, MK05-MR, MK08-MR, and MK10-MR, the optimal values achieved by GA-VNS and IGA-TNS are either comparable or identical, demonstrating their capability to consistently reach high-quality solutions. Interestingly, for the MK06-MR instance, GA-VNS surpasses IGA-TNS in terms of optimal performance. However, IGA-TNS maintains a clear edge in terms of stability, as evidenced by its superior mean and median values across all instances. This indicates that IGA-TNS offers more reliable performance with less variability, which is crucial for practical applications. Figure 13 focuses on group C instances, which are characterized by higher complexity and variability due to increased problem size and flex. In these scenarios, IGA-TNS demonstrates a clear advantage over GA-VNS, particularly in terms of stability and robustness. Although GA-VNS achieves competitive optimal values for some instances, such as MK02-MR, MK03-MR, and MK09-MR, it exhibits significant fluctuations in mean and median values. This suggests that as the problem complexity increases, GA-VNS becomes less reliable compared to IGA-TNS, which maintains its stability and performance even under challenging conditions. The results also emphasize the importance of the proposed two-stage neighborhood structure (TNS) in IGA-TNS. By leveraging adaptive and dynamic neighborhood mechanisms, IGA-TNS consistently outperforms traditional variable neighborhood search methods. The integration of TNS not only enhances solution quality but also ensures greater robustness against variations in problem scale and complexity. This highlights the strength of IGA-TNS in tackling medium to large-scale FJSP-MR problems, where exact methods such as MILP models become computationally infeasible. Overall, the findings underscore the effectiveness of hybrid metaheuristic approaches like IGA-TNS in solving complex scheduling problems. For small-scale instances, the MILP model remains a highly efficient tool, delivering optimal solutions with minimal computational effort. However, for medium and large-scale problems, hybrid methods, particularly those integrating genetic algorithms and neighborhood search strategies, provide a practical and high-performing alternative. These results not only validate the proposed approach but also offer valuable insights for future research and practical applications in advanced scheduling and optimization tasks.

## Industrial case study

To demonstrate that the designed IGA-TNS algorithm can solve real-world large-scale problems, a industrial case is introduced. The case comes from a cable processing workshop, where RMTs can process different operations by installing a series of AMs. Additionally, at some manual

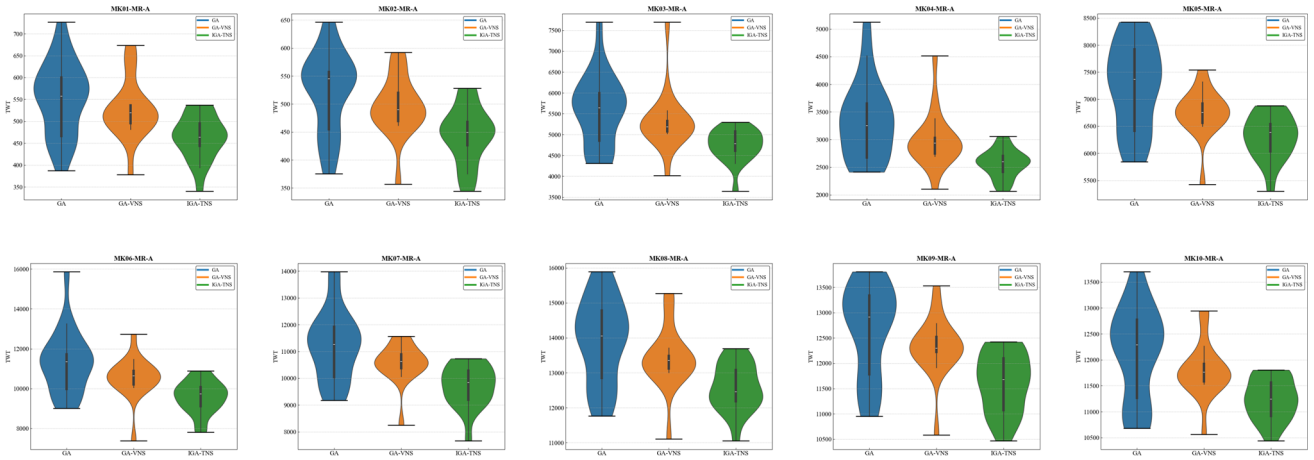


**Fig. 11** Violin plots for group A MK01-10 instances

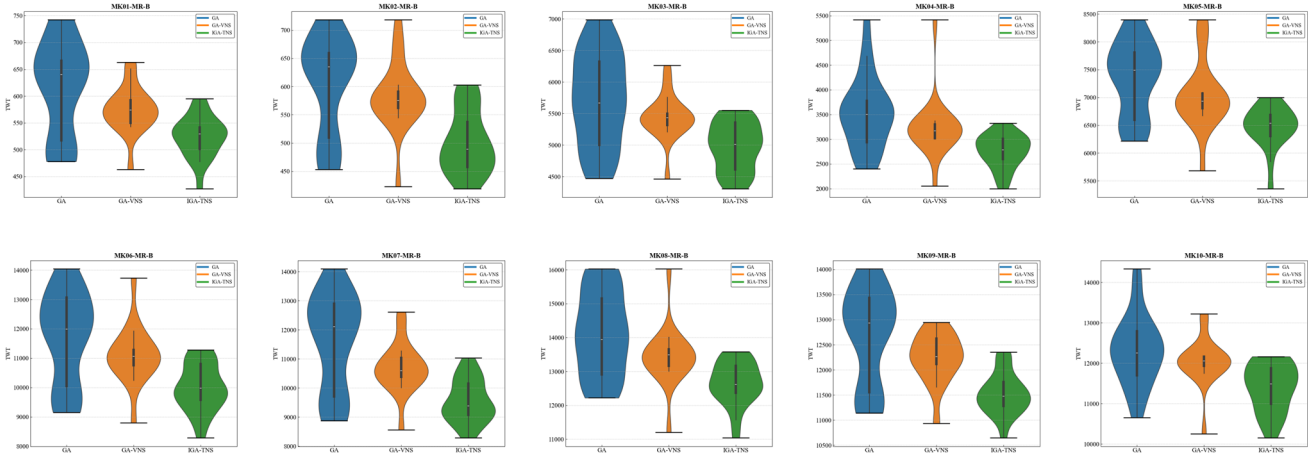


**Fig. 12** Violin plots for group B MK01-10 instances

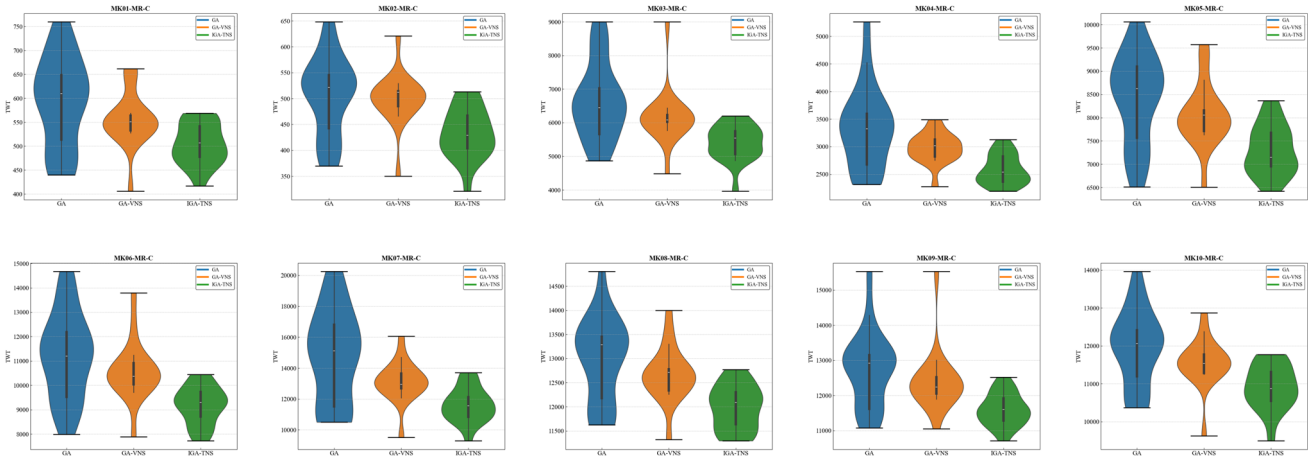


**Fig. 13** Violin plots for group C MK01-10 instances

**Table 8** Test results for comparative algorithms on the industrial FJSP-MR case

| Algorithm | TWT | |
|---|---|---|
| | Best | Avg |
| GA | 320 | 494.80 |
| GA-VNS | 165 | 313.50 |
| IGA-TNS | **122** | 220.10 |

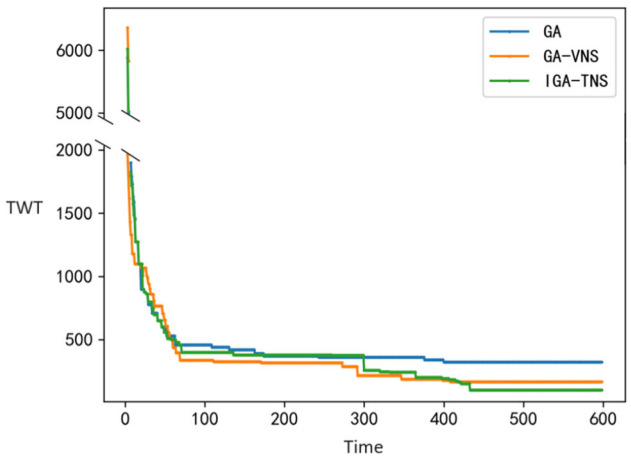


**Fig. 14** Convergence curves of comparative algorithms for the industrial FJSP-MR case

workstations, workers need to select specific tools to complete tasks, which can also be considered as using AMs to configure RMTs. Therefore, the studied industrial case can be formulated as an FJSP-MR problem. The case includes 38 jobs, 48 machines, and 20 AMs, where jobs with five priority levels have different release dates and due dates. The scale of this case far exceeds that of previous standard benchmarks. The matching relationships between RMTs and AMs, reconfiguration times, processing times, and worker scheduling are all set based on the actual production data from the company.

The industrial case was tested 20 times using basic GA, GA-VNS, and IGA-TNS, with each run having a termination condition of 10 min. The computed results, as shown in the Table 8 indicate that IGA-TNS outperforms the other two algorithms in both optimal and average values. To further investigate the real-time performance of the algorithm, the convergence curves of the algorithm comparison are shown in Fig. 14. It can be observed that at the beginning of the iterations, all algorithms converge rapidly, and after approximately 80 s, the curves exhibit minimal fluctuations. Among them, GA-VNS and IGA-TNS, due to their neighborhood search strategies, find better solutions compared to GA. Additionally, the figure shows that the proposed TNS approach performs significantly better than VNS. However, in the first half of the iterations, GA-VNS outperforms IGA-TNS, indicating that TNS is less computationally efficient than VNS for large-scale problems. When considering practical applications, adjustments to IGA-TNS may be necessary to achieve higher search efficiency.

Overall, based on the test results and analysis, IGA-TNS proves to be effective in solving real-world FJSP-MR problems.

## Conclusions and future work

This paper investigates a reconfigurable flexible job shop scheduling problem (FJSP) considering limited auxiliary modules (AMs) and varying assembly times for AMs. First, a Mixed Integer Linear Programming (MILP) model is formulated to represent the problem. Then, a two-stage neighborhood structure based on the disjunctive graph model is developed. Specifically, for the characteristics of the FJSP, the first stage focuses on selecting processing configurations, while the second stage further optimizes the idle time within these configurations. In addition, a neighborhood search activation strategy based on trend testing is proposed. When the objective value exhibits no significant trend, the neighborhood search is triggered to enhance search efficiency. Finally, experiments are conducted on both extended standard test cases and a industrial FJSP-MR cases and compared with other algorithms. The results demonstrate that the incorporation of neighborhood structures yields favorable outcomes, and the proposed two-stage neighborhood search exhibits greater stability for this problem.

In future work, green manufacturing and emission reduction will be key considerations. Factors such as reconfiguration time and the usage frequency of auxiliary modules will be introduced as new objectives, extending the problem into a multi-objective reconfigurable flexible job shop scheduling problem. In terms of the model, future research will consider the uncertainty in reconfiguration time caused by factors such as equipment aging or human intervention to enhance the model's practicality. To address this extended problem, multi-objective algorithms such as NSGA-II will be explored, along with the design of various neighborhood structures to further enhance search efficiency.

**Acknowledgements** This work is supported in part by the National Key Research and Development Program of China (Grant NO. 2024YFD2400200; 2024YFD2400204), and in part by the Science and Technology development program for "The Two Zones" (Grant No. 2023LQ02004), and in part by "the Fundamental Research Funds for the Central Universities" (DUT24BK046).

**Author contributions** Yanjun Shi: Conceptualisation, Funding Acquisition, Investigation. Chengjia Yu: Validation, Visualisation, Writing—Original Draft. Shiduo Ning: Supervision, Writing—Review &

Editing. A novel two stage neighborhood search for flexible job shop scheduling problem considering reconfigurable machine tools.

**Data availability** The data are available from the corresponding author on reasonable request.

## Declarations

**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.

**Open Access** This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by-nc-nd/4.0/.

## References

1. Yelles-Chaouche AR, Gurevsky E, Brahimi N et al (2021) Reconfigurable manufacturing systems from an optimisation perspective: a focused review of literature. Int J Prod Res 59(21):6400–6418. https://doi.org/10.1080/00207543.2020.1813913
2. Mahmoodjanloo M, Tavakkoli-Moghaddam R, Baboli A et al (2020) Flexible job shop scheduling problem with reconfigurable machine tools: An improved differential evolution algorithm. Appl Soft Comput 94:106416. https://doi.org/10.1016/j.asoc.2020.106416
3. Koren Y, Heisel U, Jovane F et al (1999) Reconfigurable manufacturing systems. CIRP Ann 48(2):527–540. https://doi.org/10.1016/S0007-8506(07)63232-6
4. Ersal T, Stein JL, Louca LS (2004) A modular modeling approach for the design of reconfigurable machine tools[C] , Anaheim, CA, United states. American Society of Mechanical Engineers. https://doi.org/10.1115/IMECE2004-59806
5. Perez R, Molina A, Ramirez-Cadena M (2014) Development of an integrated approach to the design of reconfigurable micro/mesoscale CNC machine tools. J Manuf Sci Eng-Trans Asme 136(3):10. https://doi.org/10.1115/1.4025405
6. Padayachee J, Bright G (2012) Modular machine tools: design and barriers to industrial implementation. J Manuf Syst 31(2):92–102. https://doi.org/10.1016/j.jmsy.2011.10.003
7. Hasan F, Jain PK, Kumar D (2013) Machine reconfigurability models using multi-attribute utility theory and power function approximation. Proc Eng 64:1354–1363. https://doi.org/10.1016/j.proeng.2013.09.217
8. Liu C, Li W, Lian J et al (2012) Reconfiguration of assembly systems: from conveyor assembly line to serus. J Manuf Syst 31(3):312–325. https://doi.org/10.1016/j.jmsy.2012.02.003
9. Bensmaine A, Dahane M, Benyoucef L (2014) A new heuristic for integrated process planning and scheduling in reconfigurable manufacturing systems. Int J Prod Res 52(12):3583–3594. https://doi.org/10.1080/00207543.2013.878056
10. Guo H, Liu J, Wang Y et al (2024) An improved genetic programming hyper-heuristic for the dynamic flexible job shop scheduling problem with reconfigurable manufacturing cells. J Manuf Syst 74:252–263. https://doi.org/10.1016/j.jmsy.2024.03.009
11. Hu Y, Zhang L, Zhang Z et al (2024) Flexible assembly job shop scheduling problem considering reconfigurable machine: a cooperative co-evolutionary matheuristic algorithm. Appl Soft Comput. https://doi.org/10.1016/j.asoc.2024.112148
12. Bortolini M, Ferrari E, Galizia FG et al (2021) An optimisation model for the dynamic management of cellular reconfigurable manufacturing systems under auxiliary module availability constraints. J J Manuf Syst 58:442–451. https://doi.org/10.1016/j.jmsy.2021.01.001
13. Mahmoodjanloo M, Tavakkoli-Moghaddama R, Baboli A et al (2022) Distributed job - shop rescheduling problem considering reconfigurability of machines: a self-adaptive hybrid equilibrium optimiser. Int J Prod Res 60(16):4973–4994. https://doi.org/10.1080/00207543.2021.1946193
14. Fan J, Zhang C, Liu Q et al (2022) An improved genetic algorithm for flexible job shop scheduling problem considering reconfigurable machine tools with limited auxiliary modules. J Manuf Syst 62:650–667. https://doi.org/10.1016/j.jmsy.2022.01.014
15. Fan J, Zhang C, Shen W et al (2023) A matheuristic for flexible job shop scheduling problem with lot-streaming and machine reconfigurations. Int J Prod Res 61(19):6565–6588. https://doi.org/10.1080/00207543.2022.2135629
16. Xie J, Li X, Gao L et al (2023) A new neighbourhood structure for job shop scheduling problems. Int J Prod Res 61(7):2147–2161. https://doi.org/10.1080/00207543.2022.2060772
17. Xu Y, Zhang M, Yang M et al (2024) Hybrid quantum particle swarm optimization and variable neighborhood search for flexible job-shop scheduling problem. J Manuf Syst 73:334–348. https://doi.org/10.1016/j.jmsy.2024.02.007
18. Cao S, Li R, Gong W et al (2023) Inverse model and adaptive neighborhood search based cooperative optimizer for energy-efficient distributed flexible job shop scheduling. Swarm Evol Comput. https://doi.org/10.1016/j.swevo.2023.101419
19. Huang K, Gong W, Lu C (2024) An enhanced memetic algorithm with hierarchical heuristic neighborhood search for type-2 green fuzzy flexible job shop scheduling. Eng Appl Artif Intell. https://doi.org/10.1016/j.engappai.2023.107762
20. Zhang Q, Shao W, Shao Z et al (2024) Deep reinforcement learning driven trajectory-based meta-heuristic for distributed heterogeneous flexible job shop scheduling problem. Swarm Evol Comput. https://doi.org/10.1016/j.swevo.2024.101753
21. Ding J, Dauzere-Peres S, Shen L et al (2023) A novel evolutionary algorithm for energy efficient scheduling in flexible job shops. IEEE Trans Evol Comput 27(5):1. https://doi.org/10.1109/TEVC.2022.322279
22. Mastrolilli M, Gambardella LM (2000) Effective neighbourhood functions for the flexible job shop problem. J Sched 3(1):3–20. https://doi.org/10.1002/(SICI)1099-1425(200001/02)3:1%3c3::AID-JOS32%3e3.0.CO;2-Y
23. Fattahi P, Saidi Mehrabad M, Jolai F (2007) Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. J Intell Manuf 18(3):331–342. https://doi.org/10.1007/s10845-007-0026-8
24. Brandimarte P (1993) Routing and scheduling in a flexible job shop by tabu search. Ann Oper Res 41(3):157–183. https://doi.org/10.1007/BF02023073
25. Meng L, Zhang C, Ren Y et al (2020) Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem. Comput Ind Eng 142:106347

26. Holland JH (1992) Genetic algorithms, vol 267. Scientific American, Incorporated, New York, pp 66–73
27. Kendall MG (1938) A new measure of rank correlation. Biometrika 30(1/2):81. https://doi.org/10.2307/2332226
28. Helsel DR (2005) More than obvious: better methods for interpreting nondetect data. Environ Sci Technol 39(20):419A-423A. https://doi.org/10.1021/es053368a
29. Burke E, Pinedo M (2013) Journal of Scheduling (2013). J Sched 16(1):1–2. https://doi.org/10.1007/s10951-013-0316-2
30. Müller D, Kress D (2022) Filter-and-fan approaches for scheduling flexible job shops under workforce constraints. Int J Prod Res 60(15):4743–4765. https://doi.org/10.1080/00207543.2021.1937745
31. Gröflin H, Klinkert A (2009) A new neighborhood and tabu search for the Blocking Job Shop. Discret Appl Math 157(17):3643–3655. https://doi.org/10.1016/j.dam.2009.02.020
32. Lou H, Wang X, Dong Z et al (2022) Memetic algorithm based on learning and decomposition for multiobjective flexible job shop scheduling considering human factors. Swarm Evol Comput 75:101204. https://doi.org/10.1016/j.swevo.2022.101204
33. Zhang G, Zhang L, Song X et al (2019) A variable neighborhood search based genetic algorithm for flexible job shop scheduling problem. Clust Comput 22(Suppl 5):11561–11572. https://doi.org/10.1007/s10586-017-1420-4
34. Meng L, Gao K, Ren Y et al (2022) Novel MILP and CP models for distributed hybrid flowshop scheduling problem with sequence-dependent setup times. Swarm Evol Comput 71:101058. https://doi.org/10.1016/j.swevo.2022.101058

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.