Computers & Industrial Engineering 188 (2024) 109903

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Multi-resource constrained flexible job shop scheduling problem with fixture-pallet combinatorial optimisation

Molin Liu[a], Jun Lv[b], Shichang Du[a,*], Yafei Deng[a], Xiaoxiao Shen[a], Yulu Zhou[a]

[a] Department of Industrial Engineering and Management, Shanghai Jiao Tong University, Shanghai 200240, People's Republic of China
[b] Faculty of Economics and Management, East China Normal University, Shanghai 200241, People's Republic of China

A R T I C L E I N F O

Keywords:
Flexible job shop
Mixed integer programming
Fixture-pallet constraint
Feasibility correction strategy
Genetic algorithm
self-learning VNS

A B S T R A C T

There is a lack of research on the flexible job shop scheduling problem (FJSP) considering limited fixture-pallet resources in multi-product mixed manufacturing workshops. However, field research in a leading engine manufacturer in China has revealed that fixture-pallet resources are a key factor limiting capacity breakthroughs although they play an auxiliary role in the production process. Thus, in this paper, we propose a methodology for the multi-resource constrained flexible job shop scheduling problem with fixture-pallet combinatorial optimisation. First, a mixed integer programming model with machine-fixture-pallet constraints is constructed aiming to minimize makespan. Then, a novel genetic algorithm integrated with feasibility correction strategy and self-learning variable neighbourhood search (VNS) is proposed to address the complicated scheduling problem, where the feasibility correction strategy is designed to solve potential conflict between machine selection and fixture selection chromosomes and self-learning VNS is executed to further improve the optimisation capability. Moreover, the effectiveness and efficiency of proposed algorithm are demonstrated by computational experiments with real data from cooperated engine manufacturing plant, which would provide convincing support for real production scheduling under complex scenarios.

1. Introduction

Smart factory, probably the most significant concept within Industry 4.0, draws a blueprint with a fully connected manufacturing system, where preeminent cyber technology and physical technology are applied instead of human force (Cañas, Mula, Díaz-Madroñero, & Campuzano-Bolarín, 2021; Osterrieder, Budde, & Friedli, 2020). As a critical component of manufacturing system, scheduling involves determining the sequence of production operations, allocating resources, and setting timelines for completion. To achieve the objective of minimizing costs and improving overall productivity, scheduling holds an urgent need for intelligent upgrades.

With augmenting customer demand for personalization and intense market competition, the diversification of product types with different process routes and multiple possibilities of machine selection become the notable features of production mode in manufacturing industry. Scheduling with the characteristics above is defined as the flexible job shop scheduling problem (FJSP), which consists of two subproblems: machine selection and operation sequencing (Li et al., 2017). It is a combinatorial optimisation problem that seeks to determine an optimal schedule for a set of jobs with multiple operations to be processed on a set of machines. And there is a well-established methodological system for the study of typical FJSP. However, in real-world production scenarios, the influence of auxiliary resources on scheduling should not be underestimated, and representatives of such critical resources are fixture and pallet.

Fixtures, normally held by pallets, play a pivotal role in the manufacturing process by serving as essential tools for fixating, positioning and supporting workpieces (Gothwal & Raj, 2017). The limited availability of fixture-pallet resources often acts as a bottleneck, restricting further improvement in production capacity, and can even result in unfavorable outcomes such as production stagnation or delay. Moreover, suboptimal fixture-pallet combinations can exacerbate the impact of this bottleneck. Considering there is rare research on this issue, studying FJSP with fixture-pallet constraints holds great significance in the intelligent transformation of the manufacturing industry.

In this paper, we focus on the multi-resource constrained flexible job shop scheduling problem with fixture-pallet combinatorial optimisation

* Corresponding author.
E-mail addresses: toujours.molin@sjtu.edu.cn (M. Liu), jlv@dbm.ecnu.edu.cn (J. Lv), lovbin@sjtu.edu.cn (S. Du), phoenixdyf@sjtu.edu.cn (Y. Deng), sjtusxx98@sjtu.edu.cn (X. Shen), yuluzhou@sjtu.edu.cn (Y. Zhou).

https://doi.org/10.1016/j.cie.2024.109903
Received 7 April 2023; Received in revised form 4 October 2023; Accepted 10 January 2024
Available online 13 January 2024

(MRFJSP-FPCO). A mixed integer programming (MIP) model with machine-fixture-pallet constraints is constructed to minimize the maximal completion time of production tasks as well as provide an optimal strategy for fixture-pallet combination. To handle with the complexity of solving under large-scale scheduling scenarios, we design an improved genetic algorithm, in which a feasibility correction strategy targeting on fixture-pallet relationship and a self-learning variable neighbourhood search (VNS) are involved. And first-hand data from a leading Chinese engine manufacturer is employed to evaluate the performance of the proposed algorithm, which demonstrates remarkable advantages of our approach.

The remainder of this paper is organised as follows. Section 2 reviews previous studies on FJSP. Problem description and mathematical formulation are presented in Section 3. In Section 4, an advanced and innovative algorithm is proposed to solve the problem. And Section 5 conducts the numerical experiments and analyses the corresponding results. Finally, Section 6 summarises the conclusions and prospects.

2. Literature review

As an extension of typical job shop scheduling problem (JSP), FJSP was first proposed in 1990 (Brucker & Schlie, 1990) and it has been proven to be a NP-hard problem (Fattahi, Saidi Mehrabad, & Jolai, 2007). Over the past 33 years, many researchers have applied various methods and techniques to solve JFSP. And the optimisation algorithms for FJSP could be divided into two categories: exact optimisation methods and approximate methods (Zhang, Ding, Zou, Qin, & Fu, 2019).

Representatives of exact optimisation algorithms are mathematical programming approach, branch and bound method (B&B), and Benders decomposition. Meng, Zhang, Ren, Zhang, and Lv (2020) formualted four mixed integer programming models which are sequence-based, position-based, time-indexed and adjacent sequence-based respectively and designed an efficient constraint programming model to solve the problem. Soto et al. (2020) presented a novel parallel branch and bound algorithm where shared-memory architectures were implemented to solve the multi-objective FJSP. Naderi and Roshanaei (2022) integrated Benders decomposition with constraint programming and demonstrated that its performance was better than other well-known methods. Although exact algorithms can theoretically lead to optimal solutions, research on approximate methods cannot be ignored, considering FJSP's NP-hard nature and the large size of problems in reality.

As computer technology and intelligent algorithms progress incessantly, various metaheuristics like particle swarm optimisation (PSO), genetic algorithms (GA), simulated annealing (SA) and even some machine learning tools like neural networks (NN) and reinforcement learning (RL) are utilized in FJSP. Pezzella, Morganti, and Ciaschetti (2008) designed a GA for FJSP involving various strategies for initialization, selection and reproduction, which had a profound influence for subsequent scholars. Li, Gong, and Lu (2022) focused on FJSP with fuzzy processing time to achieve a multi objective of minimizing the makespan and total workload, and proposed a self-adaptive evolutionary algorithm with decomposition, which proved to be more effective than other well-known algorithms. Ding and Gu (2020) designed an improved PSO algorithm for FJSP, which was achieved by innovative encoding/decoding scheme, information communication between particles and enhancement of machine selection rule. Ren et al. (2021) considered energy consumption in FJSP and presented an integrated heuristic algorithm combining PSO and GA to solve the established multi-objective problem. Fan, Shen, Gao, Zhang, and Zhang (2021) proposed a Jaya algorithm hybrid with tabu search (TS) to handle with FJSP and considered the multiple critical paths as the main bottleneck which lacked a formal discussion in previous literature. Hajibabaei and Behnamian (2021) considered unrelated parallel machines as well as sequence-dependent setup time in a multi-objective FJSP, where a TS algorithm was designed targeting on large-size instances. Defersha, Obimuyiwa, and Yimer (2022) introduced the assumption of machine tenders in FJSP and formulated a mathematical model for FJSP with setup operator constraints, which was solved by a SA algorithm. Chen, Yang, Li, and Wang (2020) addressed FJSP by a self-learning GA whose key parameters could be adjusted based on reinforcement learning method, which outperformed common approximate algortihsm with fixed pamaters. Müller, Müller, Kress, and Pesch (2022) trained a prediction model with the aid of decision trees and deep neural networks, which aimed to select most suitable constraint programming solvers for specific FJSP instances. Lei et al. (2022) constructed an end-to-end deep reinforcement learning (DRL) framework to solve FJSP by automatically learn policies consisting of two sub-policies called operation policy and machine policy from a large number of instances.

However, auxiliary resources are not supposed to be ignored in FJSP research as they are usually the key constraints to production scheduling. Common extra resources in FJSP include manpower, cutters, fixtures, pallets and so on. Gong, Chiong, Deng, and Gong (2020) introduced a mathematical model for FJSP with worker flexibility, incorporating a hybrid ABC algorithm with local search strategy, which effectively mitigated production costs and address manpower-related bottleneck issues. Fan et al. (2022) considered reconfigurable machine reconfigurations with auxiliary modules in FJSP and designed an improved GA, where a disjunctive graph and a modified k-insertion were applied to analyze the bottlenecks. Tian, Gao, Zhang, Chen, and Wang (2023) studied the impact of cutting-tool degradation in manufacturing process and formulated a FJSP model hybrid with tool life prediction as well as machining power prediction. Vallikavungal Devassia, Salazar-Aguilar, and Boyer (2018) got inspired by a real situation of a brewing company and proposed a FJSP with resource recovery constraints, which supposed that auxiliary resources required a recovery time between each batch. As for fixtures resources, Chan, Wong, and Chan (2006) considered fixture and tool constraints in FJSP, but every operation had only one specific fixture in their hypothesis. Thörnblad, Strömberg, Patriksson, and Almgren (2015) focused on a real-life production scenario in Sweden and formulated a FJSP model with fixture availability as well as preventive maintenance requirements, which was solved by a fast iterative approach with great advantage in computation time. Wu, Peng, Xiao, and Wu (2021) presented research on FJSP with the loading and unloading time of fixtures, where a multi-objective mathematical model was formulated and an improved non-dominated sorting genetic algorithm II (NSGA- II) with setup-time reduction strategy was proposed to solve the problem.

The brief literature above shows that many studies around FJSP have noted the importance of ancillary resources in production scheduling. However, limited works have focused on fixtures, especially as flexible resources. Besides, there is no similar research like ours to solve FJSP and optimize fixture-pallet combination solutions at the same time. Therefore, it is necessary and significant to describe MRFJSP-FPCO in a mathematical form and develop an efficient and superior algorithm to solve it.

3. Problem formulation

3.1. Problem description

The proposed MRFJSP-FPCO is defined as follows. The product set, machine set, fixture set, and pallet set are noted as I, M, F, and P respectively. J_i refers to the process route of product $i(i \in I)$ and $O_{i,j}$ represents the j th operation of product $i(i \in I, O_{i,j} \in J_i)$. Every machine $m(m \in M)$ is equipped with a pallet station $p(p \in P)$ to hold necessary fixtures, which are used to support products. Matching of fixtures and pallets is done before production starts, which indicates that every fixture will stay at the same pallet till all tasks are completed. Prior information about all process routes $J_i(i \in I)$ is fixed and the manufacturing will be executed in strict accordance with it. Besides, every operation $O_{i,j}$ has an eligible machine set $M(O_{i,j})$ and an eligible fixture set $F(O_{i,j})$, and the processing time might vary depending on which machine is chosen to operate. MRFJSP-FPCO is concerned with arranging appropriate machine and fixture for every operation, sequencing all operations, and providing a reasonable fixture-pallet matching plan to minimize the makespan, while meeting the various resources constraints. Assumptions below should also be satisfied in MRFJSP-FPCO:

(1) Every product $i(i \in I)$ can be processed from time 0.
(2) The processing is supposed to be continuous, break-down is not allowed in principle once started.
(3) Every machine $m(m \in M)$ is only available for one single operation $O_{i,j}$ of one product at the same time.
(4) Every fixture $f(f \in F)$ can hold at most one single operation $O_{i,j}$ of product at the same time.
(5) Setup time of machines and fixtures is ignored during the process.
(6) Fixture-pallet resources can satisfy the production demand although they are sparse, which means that extreme cases where operations executed on different machines rely on a single selectable fixture are not considered.

To provide a more intuitive comprehension, an example of three products' process information is shown in Table 1. There are four operations for product 1 and three operations for both product 2 and product 3. It is clear to find that one single operation could have several eligible machines and fixtures, which may require different processing time. For instance, the first operation of product $1 O_{1,1}$ could be assigned to either $M1$ or $M2$ with processing time $12min$ and $10min$ respectively.

Without fixture-pallet constraints, a typical FJSP only focuses on operation sequencing and machine selection, and several feasible scheduling schemes are shown in Fig. 1(a), (b) and (c). While in MRFJSP-FPCO, limited fixture resources and multi-possible combinatorial approaches suddenly arouse the complexity of scheduling. As shown in Fig. 1(d), $O_{1,1}$, $O_{1,2}$, and $O_{2,1}$ must be done in $M1$ as $F2$ is not available and $F1$ is held by $P1$ of $M1$. However, similar condition still exists in Fig. 1(e) because $F1$ and $F2$ are put in $P1$ of $M1$ although both are available, which proves the necessity of fixture-pallet combinatorial optimisation. And Fig. 1(f) presents a good example of scheduling with an appropriate fixture arrangement. It is worth noting that the makespan in FJSP-FPCO is longer than typical FJSP in this example, which is reasonable because fixture resource F_3 is a bottleneck requirement for both product 1 and product 2.

3.2. Mathematical formulation

To provide an optimal scheduling plan as well as an ideal fixture-pallet combination strategy under complicated trio-resource constraints, we formulate a mixed integer programming model with the target of minimizing the makespan based on the assumptions above. And the notation involved in the model is defined in Table 2. The pallet's entity is not necessary to add in this model, given that each machine is equipped with a pallet station.

The MIP model of MRFJSP-FPCO is shown below.

$$\min C_{max}$$

$$s.t.$$

$$\sum_{f \in F_{i,j}} \sum_{m \in M_{i,j}} X_{i,j,m,f} = 1, \forall i \in I, \forall j \in J_i \tag{1}$$

$$U_{m,f} \geq 1 - L \cdot \left(1 - X_{i,j,m,f}\right), \forall i \in I, \forall j \in J_i, \forall m \in M_{i,j}, \forall f \in F_{i,j} \tag{2}$$

$$\sum_{m' \in M_{i,j} \setminus m} U_{m',f} \leq 0 + L \cdot \left(1 - X_{i,j,m,f}\right), \forall i \in I, \forall j \in J_i, \forall m \in M_{i,j}, \forall f \in F_{i,j} \tag{3}$$

$$X_{i,j,m,f} \leq 0 + L \cdot U_{m,f}, \forall i \in I, \forall j \in J_i, \forall m \in M_{i,j}, \forall f \in F_{i,j} \tag{4}$$

$$S_{i,j} + \sum_{m \in M_{i,j}} \left(p_{i,j,m} \cdot \sum_{f \in F_{i,j}} X_{i,j,m,f} \right) \leq C_{i,j}, \forall i \in I, \forall j \in J_i \tag{5}$$

$$S_{i,j+1} \geq C_{i,j}, \forall i \in I, \forall j \in J_i \tag{6}$$

$$S_{i,j} \geq C_{i',j'} - L \cdot \left(2 - \sum_{f \in F_{i,j}} X_{i,j,m,f} - \sum_{f \in F_{i',j'}} X_{i',j',m,f}\right) - L \cdot Y_{i,j,i',j',m}, \forall i \in I, \forall j \in J_i, \forall i' \in I \setminus i, \forall j' \in J_{i'}, \forall m \in M_{i,j} \cap M_{i',j'} \tag{7}$$

$$S_{i',j'} \geq C_{i,j} - L \cdot \left(2 - \sum_{f \in F_{i,j}} X_{i,j,m,f} - \sum_{f \in F_{i',j'}} X_{i',j',m,f}\right) - L \cdot \left(1 - Y_{i,j,i',j',m}\right), \forall i \in I, \forall j \in J_i, \forall i' \in I \setminus i, \forall j' \in J_{i'}, \forall m \in M_{i,j} \cap M_{i',j'} \tag{8}$$

$$S_{i,j} \geq C_{i',j'} - L \cdot \left(2 - \sum_{m \in M_{i,j}} X_{i,j,m,f} - \sum_{m \in M_{i',j'}} X_{i',j',m,f}\right) - L \cdot Z_{i,j,i',j',f}, \forall i \in I, \forall j \in J_i, \forall i' \in I \setminus i, \forall j' \in J_{i'}, \forall f \in F_{i,j} \cap F_{i',j'} \tag{9}$$

$$S_{i',j'} \geq C_{i,j} - L \cdot \left(2 - \sum_{m \in M_{i,j}} X_{i,j,m,f} - \sum_{m \in M_{i',j'}} X_{i',j',m,f}\right) - L \cdot (1 - Z_{i,j,i',j',f}), \forall i \in I, \forall j \in J_i, \forall i' \in I \setminus i, \forall j' \in J_{i'}, \forall f \in F_{i,j} \cap F_{i',j'} \tag{10}$$

$$Y_{i,j,i',j',m} \leq \sum_{f \in F_{i,j}} X_{i,j,m,f}, \forall i \in I, \forall j \in J_i, \forall m \in M_{i,j} \tag{11}$$

$$Y_{i,j,i',j',m} \leq \sum_{f \in F_{i',j'}} X_{i',j',m,f}, \forall i' \in I, \forall j' \in J_{i'}, \forall m \in M_{i',j'} \tag{12}$$

Table 1
Schematic table of products' process information.

Product	Operation	Machine Set				Fixture Set				
		$M1$	$M2$	$M3$	$M4$	$F1$	$F2$	$F3$	$F4$	$F5$
product1	$O_{1,1}$	12*min*	10*min*	—	—	✓	✓	—	—	—
	$O_{1,2}$	5*min*	5*min*	—	—	✓	✓	—	—	—
	$O_{1,3}$	—	—	8*min*	8*min*	—	—	✓	—	—
	$O_{1,4}$	—	—	10*min*	11*min*	—	—	✓	—	—
product2	$O_{2,1}$	15*min*	15*min*	—	—	✓	✓	—	—	—
	$O_{2,2}$	—	—	8*min*	9*min*	—	—	✓	—	—
	$O_{2,3}$	—	—	12*min*	10*min*	—	—	✓	—	—
product3	$O_{3,1}$	6*min*	—	6*min*	—	—	—	—	✓	—
	$O_{3,2}$	8*min*	—	8*min*	—	—	—	—	✓	—
	$O_{3,3}$	—	10*min*	—	9*min*	—	—	—	—	✓

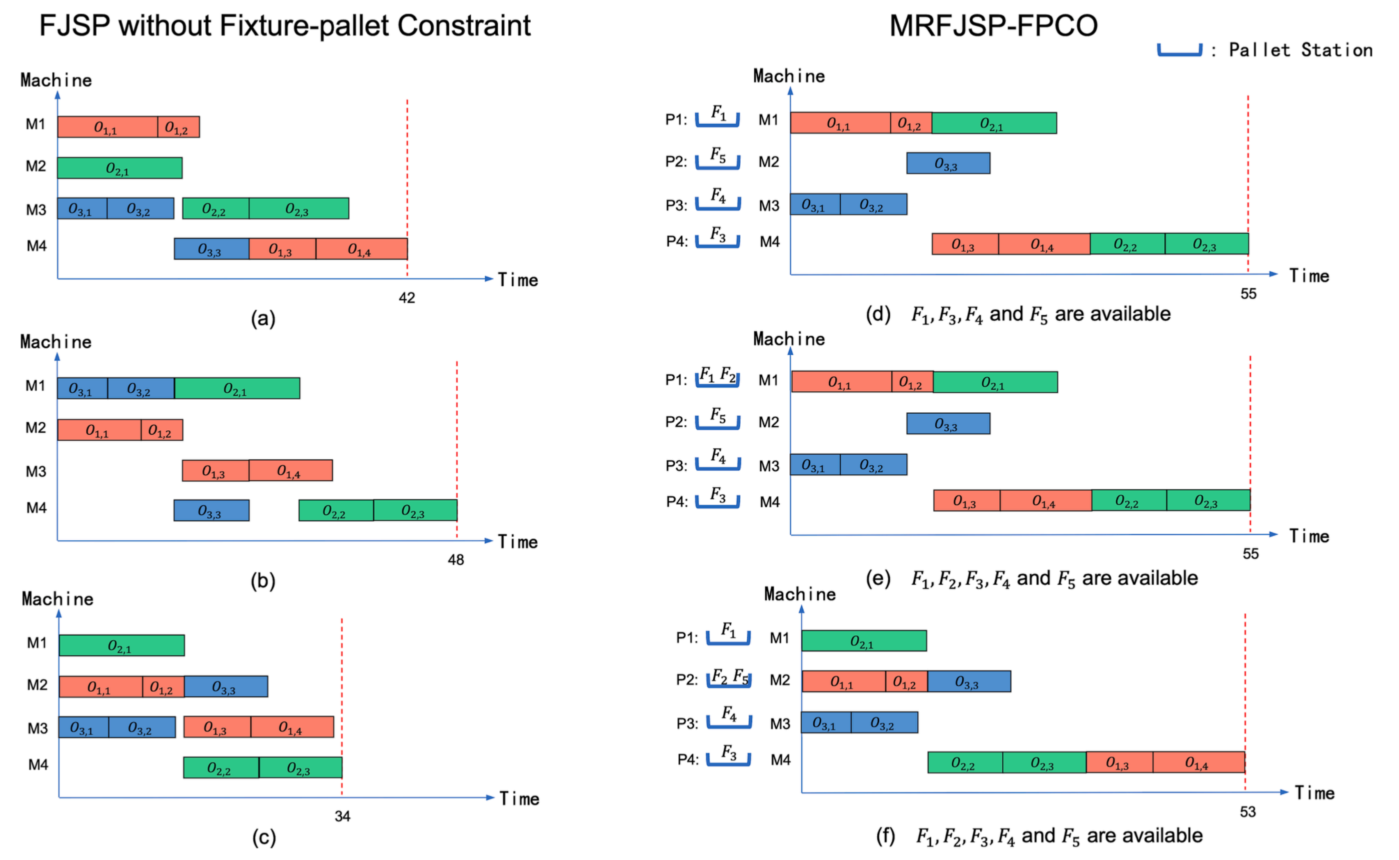


Fig. 1. Gannt charts under different scenarios.

Table 2
Notation of proposed MIP model.

Notation	Definition
I	Product set, $i = 1, 2, \cdots, \lvert I\rvert$
M	Machine set, $m = 1, 2, \cdots, \lvert M\rvert$
F	Fixture set, $f = 1, 2, \cdots, \lvert F\rvert$
J_i	Operation set of product $i, i \in I$
O_{ij}	j th operation of product $i, i \in I, j \in J_i$
M_{ij}	Eligible machine set of product i's j th operation, $i \in I, j \in J_i$
F_{ij}	Eligible fixture set of product i's j th operation, $i \in I, j \in J_i$
$pt_{i,j,m}$	Processing time of product i's j th operation on machine m, $i \in I, j \in J_i$, $m \in M_{ij}$
L	A large number, $L > 0$
S_{ij}	Continuous decision variable: Start time of product i's j th operation, $i \in I, j \in J_i$
C_{ij}	Continuous decision variable: Completion time of product i's j th operation, $i \in I, j \in J_i$
C_{max}	Continuous decision variable: Makespan, $C_{max} > 0$
$X_{i,j,m,f}$	Binary decision variable: It is equal to 1 if j th operation of product i is assigned to machine m and positioned by fixture f ; otherwise, it is equal to 0. $i \in I, j \in J_i, m \in M_{i,j}, f \in F_{i,j}$,
$Y_{i,j,i',j',m}$	Binary decision variable: It is equal to 1 if j th operation of product i precedes j' th operation of product i' on machine m ; otherwise, it is equal to 0. $i \in I, j \in J_i, i' \in I\backslash i, j' \in J_{i'}, m \in M_{ij} \cap M_{i'j'}$
$Z_{i,j,i',j',f}$	Binary decision variable: It is equal to 1 if j th operation of product i precedes j' th operation of product i' fixated by fixture f ; otherwise, it is equal to 0. $i \in I, j \in J_i, i' \in I\backslash i, j' \in J_{i'}, f \in F_{ij} \cap F_{i'j'}$
$U_{m,f}$	Binary decision variable: It is equal to 1 if fixture f is put in the pallet station of machine m ; otherwise, it is equal to 0. $m \in M, f \in F$,

$$Z_{i,j,i',j',f} \leq \sum_{m \in M_{i,j}} X_{i,j,m,f}, \forall i \in I, \forall j \in J_i, \forall f \in F_{i,j} \tag{13}$$

$$Z_{i,j,i',j',f} \leq \sum_{m \in M_{i',j'}} X_{i',j',m,f}, \forall i' \in I, \forall j' \in J_{i'}, \forall f \in F_{i',j'} \tag{14}$$

$$C_{max} \geq C_{i,j}, \forall i \in I, \forall j \in J_i \tag{15}$$

$$X_{i,j,m,f} \in \{0,1\}, \forall i \in I, \forall j \in J_i, \forall m \in M_{i,j}, \forall f \in F_{i,j} \tag{16}$$

$$Y_{i,j,i',j',m}, Z_{i,j,i',j',f} \in \{0,1\}, \forall i \in I, \forall j \in J_i, \forall i' \in I\backslash i, \forall j' \in J_{i'}, \forall m \in M_{i,j} \cap M_{i',j'}, \forall f \in F_{i,j} \cap F_{i',j'} \tag{17}$$

$$U_{m,f} \in \{0,1\}, \forall m \in M, \forall f \in F \tag{18}$$

$$S_{i,j}, C_{i,j}, C_{max} \geq 0, \forall i \in I, \forall j \in J_i \tag{19}$$

The objective function aims to minimize the makespan. Constraints (1) guarantee that every operation of every product could be processed by only one machine and positioned by only one fixture. Constraints (2), (3) and (4) ensure that the matching relation between fixture and pallet cannot change during the manufacturing once they are fixed. The sum of start time and processing time is no more than completion time for each operation is realized by constraints (5). And constraints (6) describes the strict precedence of process route for every product. Otherwise, constraints (7) and constraints (8) make sure that one machine could be occupied with at most one operation at the same time, similarly, constraints (9) and constraints (10) ensure that one fixture could be utilized with at most one operation at the same time. Constraints (11), (12), (13) and (14) describe the mutual relations among decision variables $Y_{i,j,i',j',m}$, $Z_{i,j,i',j',f}$, $X_{i,j,m,f}$ and $X_{i',j',m,f}$ and constraints (15) limit the makespan no less than any completion time of all operations. In the end, constraints (16), (17), (18) and (19) determine the basic variable types of all decision variables.

4. Proposed methodology

4.1. Framework of proposed algorithm

In MRFJSP-FPCO, varying fixtures are required for different operations along the process routes, and even a single operation may have multiple eligible fixtures. Additionally, uncertain fixture-pallet combinations have a huge effect on the generation of scheduling plans. Evidently, FJSP involving fixture-pallet constraint is significantly more intricate to address compared with the conventional FJSP, which is already difficult to be solved by an exact algorithm in polynomial time.

Therefore, we propose an improved genetic algorithm hybrid with feasibility correction strategy and self-learning variable neighbourhood search (IGA-FCSSVNS). As machine-fixture-pallet resources are all involved in the problem, a three-string chromosome structure is designed to encode. An accompanying feasibility correction strategy is designed to solve potential conflict of fixture-pallet shifting in the initialization and evolution of the population. In the metaphase, we introduce a self-learning VNS procedure, which gathers useful information from the elite pool and constructs a search strategy repository, leading the elite solutions to further optimize in the local search. The framework of proposed IGA-FCSSVSN is shown in Fig. 2.

4.2. Chromosome encoding and decoding

With extra fixture-pallet constraint, TRFJSP-FPCO could be divided into three subproblems: operation sequencing, machine selection and fixture selection. And we design a three-string chromosome encoding method, as shown in Fig. 3, which are noted as OS, MS, and FS respectively corresponding to the three subproblems. It is not necessary to introduce a fourth string to represent pallet information because the fixture-pallet relation could be obtained directly from MS and FS considering that every machine has an exclusive pallet station.

The length of every string is equal to the total amount of operations of all products and every gene position filled with a number represents a specific operation with extra information. In OS, number means product index and the order it appears determines which operation this gene is. For example, the fifth gene of OS in Fig. 3 refers to $O_{3,1}$ because it is encoded as 3 for the first time. While in MS and FS, the operation every gene stands for is arranged in order of operation sequence within product index. Besides, number represents machine index and fixture index in MS and FS. For instance, the first gene of MS and FS in Fig. 3 contains the following information: first operation of first product $O_{1,1}$ is processed in machine M2 and positioned by fixture F1.

The decoding procedure of MRFJSP-FPCO is equivalent to that of typical FJSP because the fixture-pallet combination is fixed, which has no influence on decoding. Similar with the process in previous research (Demir & İşleyen, 2014; Li & Gao, 2016), decoding is achieved by iterating from left to right to find the corresponding operation in OS as well as obtaining the allocated machine in MS and fixture in FS. And the processing time of each operation is also determined by current OS and MS gene. Fig. 4 shows an example of decoding mechanism.

4.3. Feasibility correction strategy

In three-string chromosome, the initialization and evolution of MS and FS are independent, which may arouse conflict against the assumption that fixture-pallet relationship should not be changed in the manufacturing. An example is given in Fig. 5 to strengthen comprehension. MS and FS imply that machine M2 and machine M1 are responsible for $O_{2,1}$ and $O_{3,1}$ respectively. However, both $O_{2,1}$ and $O_{3,1}$ require fixture F3 to hold, which is contrary to the preconditions of MRFJSP-FPCO. Consequently, a feasibility correction strategy is offered to solve this dilemma.

Assuming that the operation set where fixture F is required in string FS is defined as $\mathscr{O}_F$, $\mathscr{O}_F = \{\cdots, O_{i,j}, \cdots\}$, and the machine set corresponding to operations in $\mathscr{O}_F$ is defined as $M_{\mathscr{O}_F}$, $M_{\mathscr{O}_F} = \{\cdots, M_F, \cdots\}$, which is easy to obtain from string MS. For instance, $\mathscr{O}_{F_2} = \{O_{1,2}, O_{2,2}\}$ and $M_{\mathscr{O}_{F_2}} = \{M3\}$ in Fig. 5. In $\mathscr{O}_F$, the operation set where operations have common optional machine is defined as $\mathscr{O}_{COM}$, $\mathscr{O}_{COM} = \{\cdots, O_{k,l}, \cdots\}$. For example, $\mathscr{O}_{COM} = \{O_{1,2}, O_{1,3}, O_{2,2}\}$ if $O_{1,2}$, $O_{1,3}$ and $O_{2,2}$ can be processed by the same machine M_{COM}. Besides, the operation set where operations have only one optional fixture F is defined as $\mathscr{O}_{Fonly}$, $\mathscr{O}_{Fonly} = \{\cdots, O_{m,n}, \cdots\}$. The strict definition of $\mathscr{O}_{COM}$ and $\mathscr{O}_{Fonly}$ is:

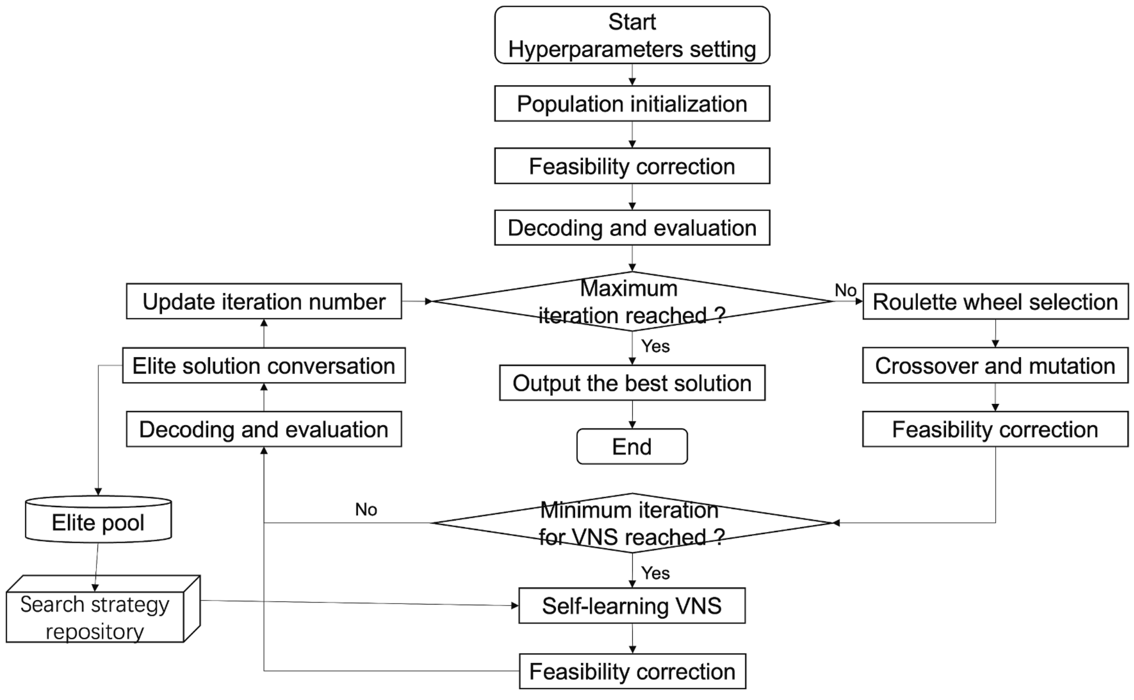


Fig. 2. Framework of IGA-FCSSVNS.

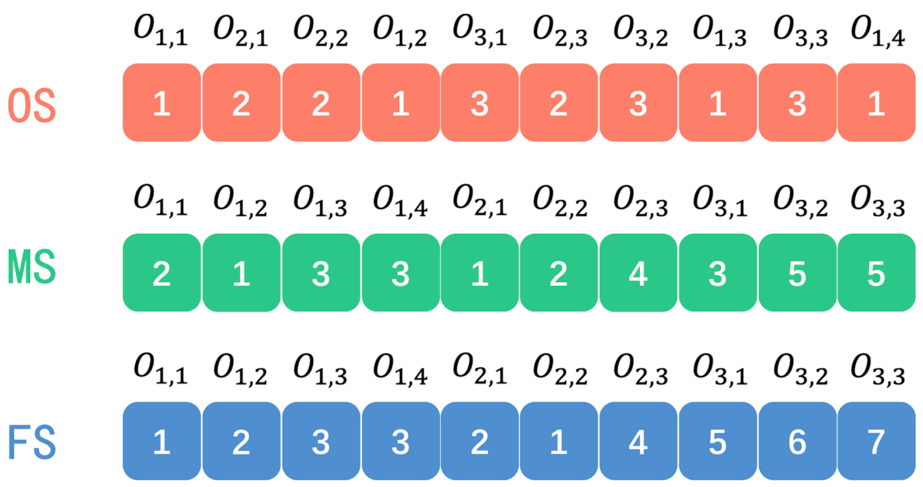

OS
$O_{1,1}$ $O_{2,1}$ $O_{2,2}$ $O_{1,2}$ $O_{3,1}$ $O_{2,3}$ $O_{3,2}$ $O_{1,3}$ $O_{3,3}$ $O_{1,4}$
1 2 2 1 3 2 3 1 3 1
MS
$O_{1,1}$ $O_{1,2}$ $O_{1,3}$ $O_{1,4}$ $O_{2,1}$ $O_{2,2}$ $O_{2,3}$ $O_{3,1}$ $O_{3,2}$ $O_{3,3}$
2 1 3 3 1 2 4 3 5 5
FS
$O_{1,1}$ $O_{1,2}$ $O_{1,3}$ $O_{1,4}$ $O_{2,1}$ $O_{2,2}$ $O_{2,3}$ $O_{3,1}$ $O_{3,2}$ $O_{3,3}$
1 2 3 3 2 1 4 5 6 7

Fig. 3. Three-string chromosome encoding example.

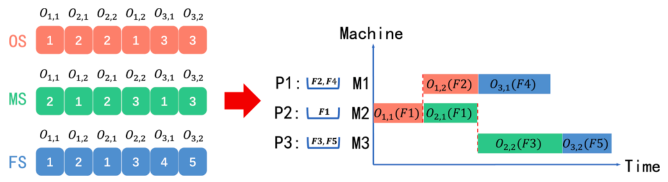

OS
$O_{1,1}$ $O_{2,1}$ $O_{2,2}$ $O_{1,2}$ $O_{3,1}$ $O_{3,2}$
1 2 2 1 3 3
MS
$O_{1,1}$ $O_{1,2}$ $O_{2,1}$ $O_{2,2}$ $O_{3,1}$ $O_{3,2}$
2 1 2 3 1 3
FS
$O_{1,1}$ $O_{1,2}$ $O_{2,1}$ $O_{2,2}$ $O_{3,1}$ $O_{3,2}$
1 2 1 3 4 5
Machine
P1 : F2, F4 M1
P2 : F1 M2
P3 : F3, F5 M3
$O_{1,2}(F2)$
$O_{3,1}(F4)$
$O_{1,1}(F1)$
$O_{2,1}(F1)$
$O_{2,2}(F3)$
$O_{3,2}(F5)$
Time

Fig. 4. Three-string chromosome decoding example.

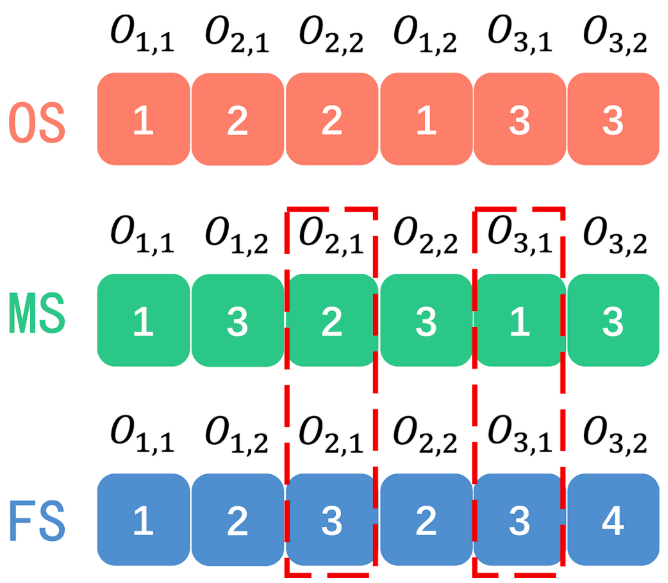

OS
$O_{1,1}$ $O_{2,1}$ $O_{2,2}$ $O_{1,2}$ $O_{3,1}$ $O_{3,2}$
1 2 2 1 3 3
MS
$O_{1,1}$ $O_{1,2}$ $O_{2,1}$ $O_{2,2}$ $O_{3,1}$ $O_{3,2}$
1 3 2 3 1 3
FS
$O_{1,1}$ $O_{1,2}$ $O_{2,1}$ $O_{2,2}$ $O_{3,1}$ $O_{3,2}$
1 2 3 2 3 4

Fig. 5. Conflict example of MS and FS.

$\forall O_{i,j}, O_{k,l} \in \mathscr{O}_{COM}, O_{i,j} \neq O_{k,l}, M_{i,j} \cap M_{k,l} \neq \varnothing;$

$\forall O_{i,j} \in \mathscr{O}_{Fonly}, F_{i,j} = \{F\}$

$\mathscr{O}_F$, $\mathscr{O}_{COM}$, $\mathscr{O}_{Fonly}$, and $M_{\mathscr{O}F}$ have the following properties:

$Property1 : Feasiblitycorrectionisnotnecessaryif |M_{\mathscr{O}_F}| = 1.$

$Property2 : \ \mathscr{O}_{COM} \subseteq \mathscr{O}_F, \ \mathscr{O}_{Fonly} \subseteq \mathscr{O}_F.$

$Property3 : if \left|\mathscr{O}_{Fonly}\right| > 1, \mathscr{O}_{Fonly} \subseteq \mathscr{O}_{COM}.$

$Property4 : if \mathscr{O}_{COM} = \varnothing, \left|\mathscr{O}_{Fonly}\right| \leq 1.$

Based on the properties above, the feasibility correction algorithm is shown below.

Step 1: for every fixture F appeared in chromosome FS, judge if $|M_{\mathscr{O}_F}| = 1$. If yes, algorithm ends; else, move to step 2.

Step 2: judge if $\mathscr{O}_{COM} = \varnothing$. If yes, move to step3; else, calculate $|\mathscr{O}_{Fonly}|$: if $|\mathscr{O}_{Fonly}| \geq 1$, move to step 4; else, move to step5.

Step 3: for operation who holds the least optional fixtures in $\mathscr{O}_F$, corresponding MS and FS genes stay the same; for other operations in $\mathscr{O}_F$, MS gene doesn't need to change while FS gene should be modified based on its eligible fixture set.

Step 4: operations in $\mathscr{O}_{Fonly}$ keep the original FS gene, and change MS gene into the common optional machine m_{Fonly}; for other operations, if m_{Fonly} is not included in eligible machine set, MS gene stays unchanged and FS gene should be substituted from the eligible fixture set; else, correct MS gene into m_{Fonly}.

Step 5: operations in $\mathscr{O}_{COM}$ keep the original FS gene, and change MS gene into the common optional machine m_{COM}; for other operations, if m_{COM} is not included in eligible machine set, MS gene stays unchanged and FS gene should be substituted from the eligible fixture set; else, correct MS gene into m_{COM}.

In conclusion, the feasibility correction is designed for three different scenarios determined by $|M_{\mathscr{O}_F}|$, $\mathscr{O}_{COM}$, and $|\mathscr{O}_{Fonly}|$. As shown in Fig. 6, the restoration details are presented in a specific example.

4.4. Population initialization

The convergence speed and algorithm accuracy of proposed IGA-FCSSVNS can be vitally impacted by the quality of initial chromosomes. In order to achieve effective optimisation, the initial population must serve two purposes. Firstly, it should encompass a broad coverage of the solution space, allowing for optimisation in all directions. Secondly, this population should also include high-quality or near-optimal solutions, potentially accelerating the convergence process.

Therefore, a combination of random generation and multi heuristic rules is applied in the initialization. Stochastically generated solutions make up 75 % of the original population, which strengthen the variety of community. And the rest is engendered by global search (GS) rule (Singh & Mahapatra, 2016), shortest processing time (SPT) rule (Hamzadayi & Yildiz, 2017), and most total work remaining (MTWR) rule (Dominic, Kaliyamoorthy, & Kumar, 2004) with a percentage distribution of 23 %, 1 % and 1 % respectively. Feasibility correction strategy shown in section 4.3. is already required at this stage.

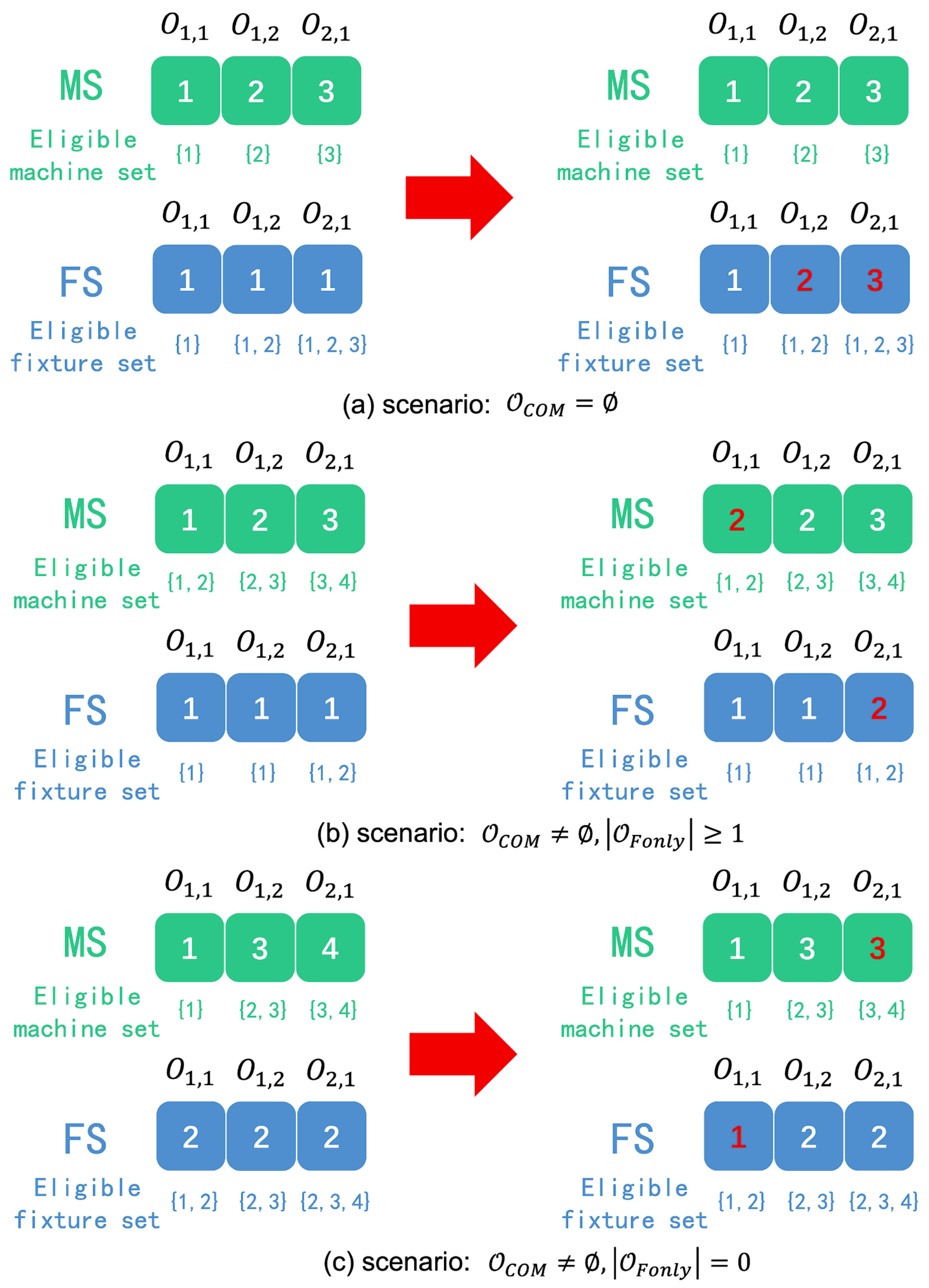


Fig. 6. Schematic diagram of feasibility correction.

4.5. Selection and reproduction

Similar with most metaheuristic algorithms, iterative optimisation is mainly achieved by selection, crossover, and mutation. The fitness whose value is the reciprocal of makespan after decoding is supposed to be calculated and evaluated for every individual. And solutions with bigger fitness value will be stored in the elite pool. After that, current population as the parent will take a roulette wheel strategy (Teekeng & Thammano, 2012) to select solutions and the selected ones will operate the crossover and mutation with a certain probability to generate offspring, which become the new parent. The process is repeated until the maximum number of iterations is reached.

Precedence preservative crossover operator (Bierwirth & Mattfeld, 1999) and multi-times swap mutation operator (Tian et al., 2022) are applied for string OS. While string MS chooses the multi-point crossover operator (Zhang, Hu, Sun, & Zhang, 2020) and string FS takes the order crossover operator (Zhang, Gao, & Shi, 2011), and both of them utilize random mutation operator (Mahmudy, Marian, & Luong, 2013).

4.6. Self-learning VNS

In the mid-to-late stage of population iteration, a self-learning VNS strategy is introduced to further improve the solution quality and enhance the optimisation capacity of algorithm, and the iteration when it starts is noted as I_{VNS_Start} in this work. As an effective local search method, VNS has been proved to be quite useful in existing research work (Lei & Guo, 2014). On this basis, SVNS constructs a search repository of three neighbourhood structures from the elite solution pool, and iteratively implements VNS strategy preferences autonomously through continuous learning, feedback and updating to optimize the elite individuals.

4.6.1. Neighbourhood structure

Three neighbourhood structures are designed and implemented in IGA-FCSSVNS.

- VNS1: VNS for string MS and string FS

Select $k, k \in [1, |MS|]$ gene positions randomly from MS, substitute original machine by machine with the minimum processing time for the current operation. Afterwards, execute the feasibility correction strategy for MS and FS.

- VNS2: VNS for string OS

Select $k, k \in [1, |OS|]$ gene positions from string OS and put the corresponding operations in an inverse order.

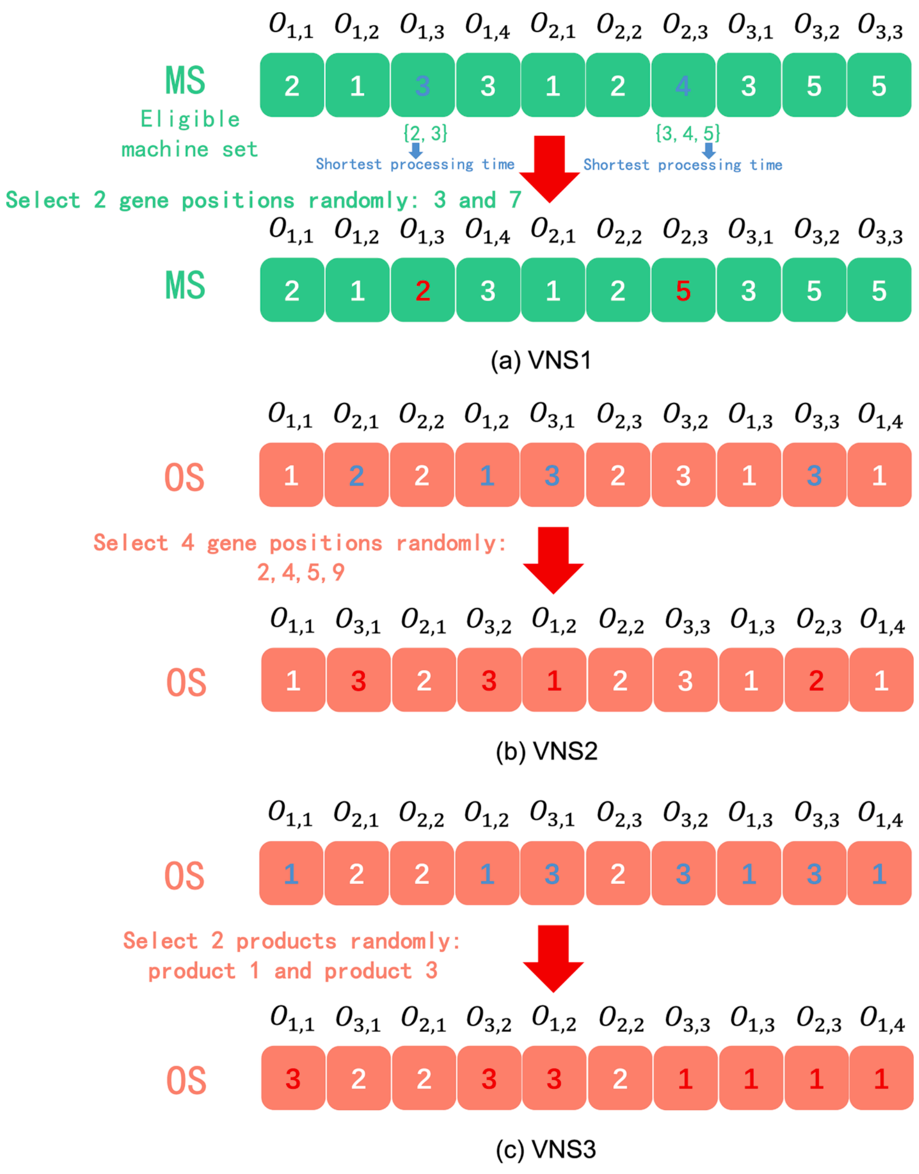
- VNS3: VNS for string OS

Select two products $i, i \in I$ and $j, j \neq i, j \in I$ randomly and put the product in prior order if it has a smaller total operations number.

The schematic diagrams of VNS1, VNS2, and VNS3 are shown in Fig. 7.

4.6.2. Search strategy repository

Search strategy repository is established with the aid of success and failure knowledge matrixes filled with knowledge from previous VNS iterations, which records the effect on elite solutions of VNS1, VNS2 and VNS3 independently. General form of elements in the success knowledge matrix is $n_{ij}^{S}, i \in \{1, 2, \cdots, m\}, j \in \{1, 2, 3\}$, which represents how many times structure VNS j improved the elite solution i in the current VNS iteration. Similarly, $n_{ij}^{f}, i \in \{1, 2, \cdots, m\}, j \in \{1, 2, 3\}$ in the failure knowledge matrix refers to the total times that structure VNS j failed to ameliorate the elite solution i in the current VNS iteration. And the following relationship holds:



$O_{1,1}$ $O_{1,2}$ $O_{1,3}$ $O_{1,4}$ $O_{2,1}$ $O_{2,2}$ $O_{2,3}$ $O_{3,1}$ $O_{3,2}$ $O_{3,3}$
MS
2 1 3 3 1 2 4 3 5 5
Eligible machine set
{2, 3}
Shortest processing time
{3, 4, 5}
Shortest processing time
Select 2 gene positions randomly: 3 and 7
MS
2 1 2 3 1 2 5 3 5 5
(a) VNS1
$O_{1,1}$ $O_{2,1}$ $O_{2,2}$ $O_{1,2}$ $O_{3,1}$ $O_{2,3}$ $O_{3,2}$ $O_{1,3}$ $O_{3,3}$ $O_{1,4}$
OS
1 2 2 1 3 2 3 1 3 1
Select 4 gene positions randomly: 2, 4, 5, 9
$O_{1,1}$ $O_{3,1}$ $O_{2,1}$ $O_{3,2}$ $O_{1,2}$ $O_{2,2}$ $O_{3,3}$ $O_{1,3}$ $O_{2,3}$ $O_{1,4}$
OS
1 3 2 3 1 2 3 1 2 1
(b) VNS2
$O_{1,1}$ $O_{2,1}$ $O_{2,2}$ $O_{1,2}$ $O_{3,1}$ $O_{2,3}$ $O_{3,2}$ $O_{1,3}$ $O_{3,3}$ $O_{1,4}$
OS
1 2 2 1 3 2 3 1 3 1
Select 2 products randomly: product 1 and product 3
$O_{1,1}$ $O_{3,1}$ $O_{2,1}$ $O_{3,2}$ $O_{1,2}$ $O_{2,2}$ $O_{3,3}$ $O_{1,3}$ $O_{2,3}$ $O_{1,4}$
OS
3 2 2 3 3 2 1 1 1 1
(c) VNS3

Fig. 7. Schematic diagrams of three neighbourhood structures.

$$\sum_{j=1}^{3}\left(n_{ij}^{s}+n_{ij}^{f}\right)=ITER_{VNS},\forall i\in\{1,2,\cdots,m\} \tag{20}$$

In equation (20), $ITER_{VNS}$ refers to the number of VNS performed in one iteration of IGA-FCSSVNS (Fig. 8).

4.6.3. Local search based on self-learning VNS

When VNS is first executed in the population, three VNS structures are selected randomly to function on the elite solution during the early stages, whose results are gathered to fill in the knowledge matrixes. After a certain number of iterations, the algorithm reaches the iteration I_{Random}, and the selection of VNS starts to be based on the statistical probability knowledge extracted from the repository. Furthermore, the effects are also fed back to the repository to ensure that dynamically updated knowledge base always provides the most reliable policy recommendations. The detailed procedure of self-learning VNS algorithm is shown below.

Step 1: parameter initialization. S_c^* represents the best elite solution of the current generation i, i_{VNS} refers to the time of executed VNS in current generation i and $ITER_{VNS}$ is the same meaning as in equation (20). Initialize the i^{th} row of knowledge matrixes by 0 and set i_{VNS} to be 1.

Step 2: judge if $i_{VNS} > ITER_{VNS}$. If yes, output the latest best solution S_c^* and the process ends; else, continue.

Step 3: judge if $i \le I_{Random}$. If yes, VNS in this iteration will be done by selecting randomly from $VNS_j, j \in \{1,2,3\}$. Otherwise, a self-learning VNS draft will be done with respect to equations (21), (22), and (23).

Step 4: perform VNS based on the chosen structure $VNS_j, j \in \{1,2,3\}$, obtain a new solution S'.

Step 5: if $fitness(S') > fitness(S_c^*)$, $S_c^* = S', n_{ij}^s = n_{ij}^s + 1$; else, S_c^* stays the same and $n_{ij}^f = n_{ij}^f + 1$.

Step 6: update $\mathscr{P}_j, j \in \{1,2,3\}$ in equation (23) based on the latest repository, $i_{VNS} = i_{VNS} + 1$, move to step 2.

$$VNS = Random(VNS_1, VNS_2, VNS_3|\mathscr{P}_1,\mathscr{P}_2,\mathscr{P}_3) \tag{21}$$

$$P_j=\frac{\sum_{k=1}^{i}n_{kj}^{s}}{\sum_{k=1}^{i}n_{kj}^{s}+\sum_{k=1}^{i}n_{kj}^{f}},j\in\{1,2,3\} \tag{22}$$

$$\mathscr{P}_j=\frac{P_j}{\sum_{n=1}^{3}P_n},j\in\{1,2,3\} \tag{23}$$

The mechanism of the procedure above is shown in Fig. 9.

5. Numerical experiments

To verify the correctness of MIP model formulated in section 3.2, we chose CPLEX, a high-powered mathematical programming solver commonly used in academia and industry, solving the model of MRFJSP-FPCO and comparing the results with that of IGA-FCSSVNS. Besides, two additional sets of algorithms were designed as control groups to demonstrate the superiority of the proposed feasibility correction strategy and self-learning VNS mechanism. Every test example was repeated 5 times given the stochastic attribute of metaheuristic algorithms and all experiments in this research were coded in Python3.8 and conducted on a personal computer configured with an AMD Ryzen 7 4800HS CPU @ 2.90 GHz + 16 GB RAM.

5.1. Case introduction

We investigated the current situation of scheduling in a leading engine manufacturing industry in northern China and found that fixture-pallet resources had become the most prominent constraints in the production planning process of the new product trial center of the company. Different products may require various kinds of fixtures, which could be met in different pallets of corresponding machines, as shown in Fig. 10. Therefore, we utilized first-hand data from this workshop as the basis for generating various scales of test cases covering process routes of 15 different kinds of products. There are 107 operations in total within all categories, and every product route consists of 5 to 10 operations, where 25 machines and 61 fixtures are required during the whole process.

The partition of test cases is based on the scale of the production system, which encompasses the number of product types, products, machines, and fixtures. Three distinct categories include small, medium, and large scale, whose detailed data composition is shown in Table 3. For instance, the small-scale case includes 5 to 10 pieces of products belonging to 5 categories, where 16 machines and 25 fixtures are sufficient and available for scheduling.

Furthermore, every category is composed of several different subcases, which are named in the format as $P-M-F$, where P, M, and F refer to the total volume of production, total amount of machines and total number of fixtures respectively. And every subcase also includes various examples with different order information. For instance, subcase $5-16-25$ represents that there are 5 products in this test subcase with 16 machines and 25 fixtures to choose. However, 5 products may refer to 2 pieces of product 1, 2 pieces of product 2 and 1 piece of product 3 (example 1) or 1 piece from product 1 to product 5 respectively (example 2). In our experiments, two or three examples are designed in every subcase to test the applicability of proposed algorithm under different scenarios.

5.2. Parameter settings

We set the time limit to 3600 s, 14400 s, and 14400 s for small, medium, and large-scale cases respectively in CPLEX solver. If CPLEX is unable to find the theoretical global optimum within time limit, the current best solution is used as the result.

As mentioned above, two extra groups of algorithms are tested in comparison with IGA-FCSSVNS. The first is the classical genetic algorithm (GA) with penalty in fitness function to promote the legalization of MS and FS wherever possible, which aims to demonstrates the

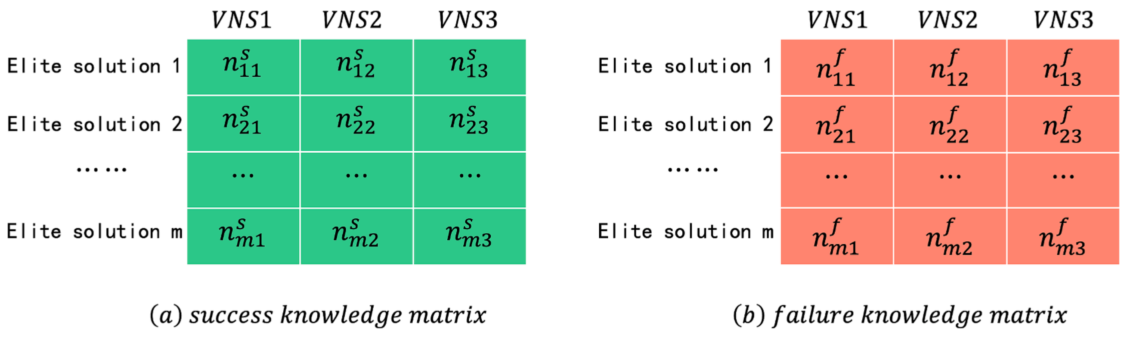

VNS1
VNS2
VNS3
Elite solution 1
Elite solution 2
……
Elite solution m
n_{11}^s n_{12}^s n_{13}^s
n_{21}^s n_{22}^s n_{23}^s
n_{m1}^s n_{m2}^s n_{m3}^s
n_{11}^f n_{12}^f n_{13}^f
n_{21}^f n_{22}^f n_{23}^f
n_{m1}^f n_{m2}^f n_{m3}^f
(a) success knowledge matrix
(b) failure knowledge matrix

Fig. 8. Schematic diagram of search strategy repository.

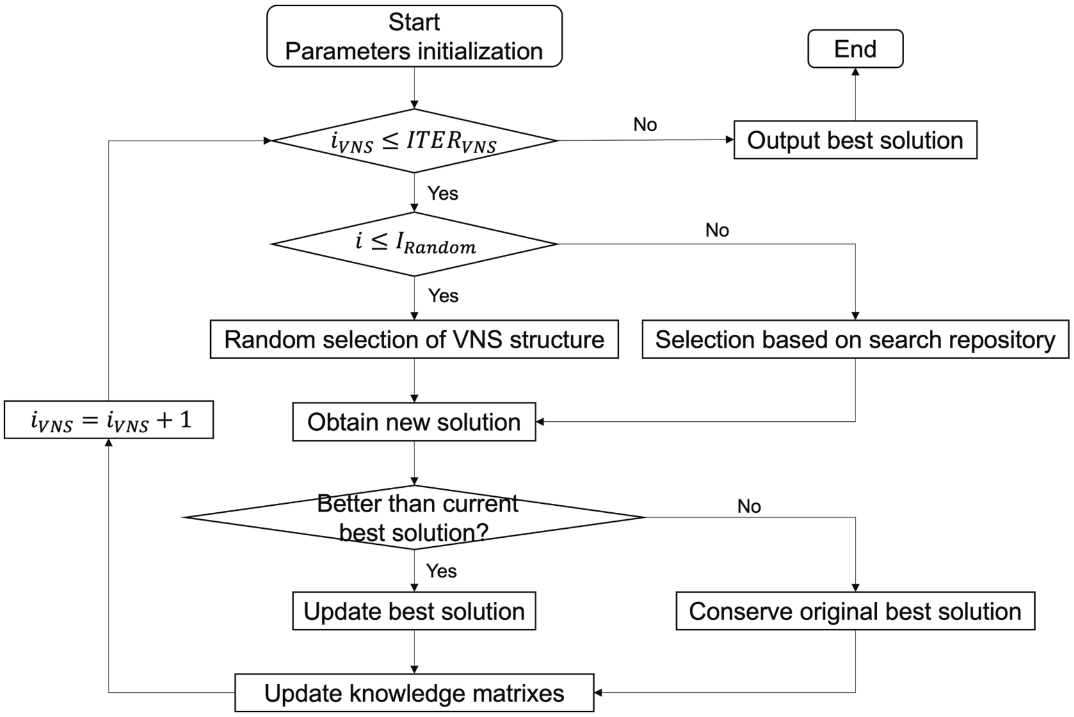

Start
Parameters initialization
$i_{VNS} \leq ITER_{VNS}$
No
Output best solution
End
Yes
$i \leq I_{Random}$
No
Yes
Random selection of VNS structure
Selection based on search repository
$i_{VNS} = i_{VNS} + 1$
Obtain new solution
Better than current best solution?
No
Yes
Update best solution
Conserve original best solution
Update knowledge matrixes

Fig. 9. Structure of self-learning VNS algorithm.

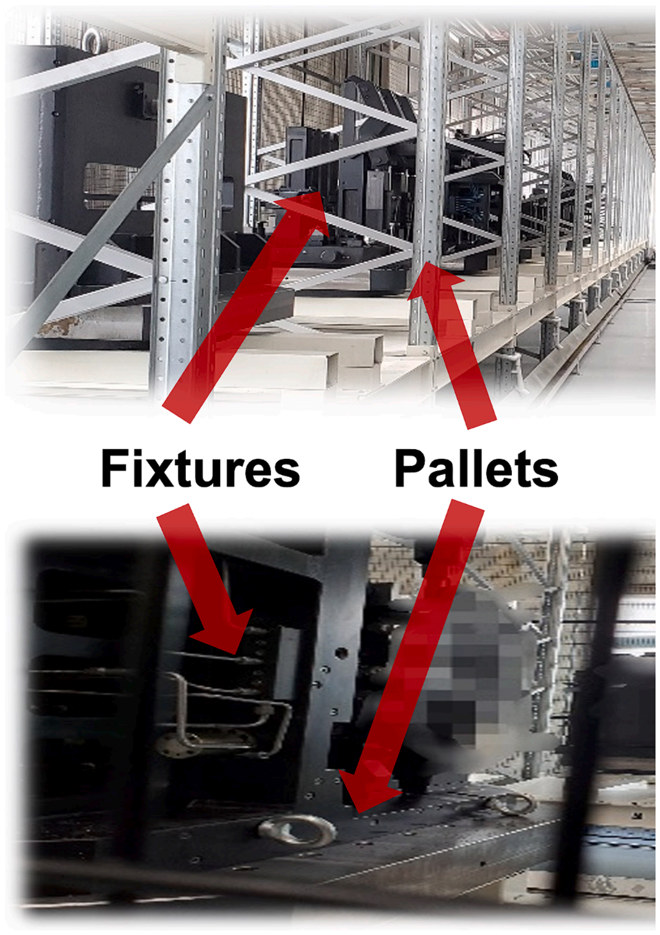


Fig. 10. Fixture-pallet resources in engine manufacturing workshop.

Table 3
Classification of test cases.

Case classification	Product type	Production volume	Machine number	Fixture number
Small-scale	5	5–10	16	25
Medium-scale	5–10	30–40	16–20	25–52
Large-scale	10–15	50–60	20–25	52–61

effectiveness of feasibility correction strategy in our work. On this basis, the second is the genetic algorithm hybrid with feasibility correction strategy (GA-FCS), which further proves the advantage of our self-learning VNS method. Like Pezzella et al. (2008) and Fan et al. (2022), we conducted experiments with various values for the key parameters and introduced an adaptive mechanism characterized by linearly decreasing crossover and mutation ratios. The selected parameter values are shown in Table 4.

5.3. Performance analysis

Following the parameter settings in section 5.2, we tested 18 groups of examples under 8 subcases with the comparison among CPLEX, GA, GA-FCS and IGA-FCSSVNS. The experimental results are shown in

Table 4
Parameter settings.

Notation	Meaning	Value
population	Population scale in the algorithm.	300
totaliteration	Total iteration times in the algorithm.	150
P_{cMAX}	Crossover probability in the beginning of the iteration process as it is designed to be linearly decreasing.	0.7
P_{cMIN}	Crossover probability in the end of the iteration process.	0.5
P_{mMAX}	Mutation probability in the beginning of the iteration process as it is designed to be linearly decreasing.	0.35
P_{mMIN}	Mutation probability in the end of iteration process.	0.2
I_{VNS_Start}	Iteration number when the self-learning VNS begins.	80
I_{Random}	Upper bound of iteration when VNS is chosen randomly.	110
$ITER_{VNS}$	Total amount of VNS performed in one iteration.	30

Table 5
Experiment results.

Scale	Subcase	Example	CPLEX		GA			GA-FCS			IGA-FCSSVNS		
			t_{CPU}	$\overline{Opt_1}$	t_{CPU}	Opt_2	$\overline{Opt_2}$	t_{CPU}	Opt_3	$\overline{Opt_3}$	t_{CPU}	Opt_4	$\overline{Opt_4}$
Small	5–16-25	1	0.28	218	111.00	226	232.20	195.60	219	219.00	191.20	218	218.00
		2	1.59	234	113.20	239	244.20	137.00	236	237.60	124.00	234	234.00
		3	18.17	284	127.60	302	313.80	144.00	289	293.20	148.00	285	287.20
	10–16-25	1	7200.00	277	259.80	344	356.00	245.20	284	287.20	251.00	277	281.00
		2	7200.00	302	259.20	369	403.40	251.20	318	322.40	251.00	303	304.20
		3	7200.00	252	254.40	308	319.20	439.00	258	269.00	444.00	253	254.80
Medium	30–20-52	1	14400.00	568	995.60	820	886.00	919.80	555	560.40	962.00	532	545.20
		2	14400.00	758	1007.80	822	863.80	1065.40	586	599.00	1038.00	566	571.80
	35–16-25	1	14400.00	986	1560.60	1345	1419.40	1658.00	843	852.00	1683.80	804	815.60
		2	14400.00	798	1592.40	1254	1403.00	1700.00	818	836.40	1697.00	773	792.80
	40–20-52	1	14400.00	758	1490.60	1097	1232.80	1461.00	739	763.40	1528.60	708	718.60
		2	14400.00	848	1559.00	1157	1291.80	1543.20	753	774.00	1617.40	729	748.60
Large	50–25-61	1	14400.00	690	1928.20	1135	1163.00	1874.60	692	705.00	2037.00	649	677.60
		2	14400.00	833	1970.20	1163	1221.20	1959.80	683	730.60	2127.00	669	685.40
	55–20-52	1	14400.00	1394	2566.80	1663	1696.60	2426.60	1017	1047.40	2553.00	926	954.20
		2	14400.00	1231	2354.80	1554	1584.60	2285.00	1003	1023.40	2474.40	956	1001.00
	60–25-61	1	14400.00	992	2669.00	1327	1425.40	2546.40	815	837.00	2615.60	791	804.40
		2	14400.00	1319	2935.60	1553	1583.20	2920.00	915	943.80	3030.40	869	892.60

Table 5, where $Opt_i, i \in \{2,3,4\}$ refers to the makespan (in minutes) recorded in repeated experiments for every test example, $\overline{Opt_i}, i \in \{1,2,3,4\}$ represents the mean value of optimal solutions (in minutes), and t_{CPU} stands for the average computation time (in seconds).

We calculate the following evaluation indicators based on data in Table 5. Q_1, Q_2, and Q_3 are introduced to measure the relative deviation of solutions obtained by different methods, which are defined in equations (24). And it is evident that a negative Q value indicates that proposed IGA-FCSSVNS performs better than method in comparison. The detailed values of indicators are shown in Table 6.

$$Q_i = \frac{\overline{Opt_4} - \overline{Opt_i}}{\overline{Opt_i}} \times 100\%, i \in \{1,2,3\} \tag{24}$$

In subcase 5 –16 –25, both CPLEX and IGA-FCSSVNS can find the global optimal solution of example 1 and example 2 in a short time, verifying the correctness of proposed MIP model, but GA and GA-FCS can only obtain the average optimal solution within 4.18 % and 1.52 % relative deviation from the global optimal solution. In subcase 10 –16 –25, CPLEX fails to find the global optimal solution within 7200 s, while IGA-FCSSVNS achieves an approximate optimal solution no more than 1.44 % different from CPLEX solution within 450 s, which also outperforms the other two.

Table 6
Relative deviation calculation.

Scale	Subcase	Example	Q1	Q2	Q3
Small	5-16-25	1	0.00 %	−6.12 %	−0.46 %
		2	0.00 %	−4.18 %	−1.52 %
		3	1.13 %	−8.48 %	−2.05 %
	10-16-25	1	1.44 %	−21.07 %	−2.16 %
		2	0.73 %	−24.59 %	−5.65 %
		3	1.11 %	−20.18 %	−5.28 %
Average relative deviation			0.74 %	−14.10 %	−2.85 %
Medium	30–20-52	1	−4.01 %	−38.47 %	−2.71 %
		2	−24.56 %	−33.80 %	−4.54 %
	35-16-25	1	−17.28 %	−42.54 %	−4.27 %
		2	−0.65 %	−43.49 %	−5.21 %
	40-20-52	1	−5.20 %	−41.71 %	−5.87 %
		2	−11.72 %	−42.05 %	−3.28 %
Average relative deviation			−10.57 %	−40.34 %	−4.31 %
Large	50-25-61	1	−1.80 %	−41.74 %	−3.89 %
		2	−17.72 %	−43.87 %	−6.19 %
	55-20-52	1	−31.55 %	−43.76 %	−8.90 %
		2	−18.68 %	−36.83 %	−2.19 %
	60-25-61	1	−18.91 %	−43.57 %	−3.89 %
		2	−32.33 %	−43.62 %	−5.42 %
Average relative deviation			−20.17 %	−42.23 %	−5.08 %

The advantage of IGA-FCSSVNS begins to emerge as the test scale expands. From subcase 30 –20 –52 to subcase 60 –25 –61, CPLEX could not obtain theoretical optimal solution within 14,400 s. However, IGA-FCSSVNS outperforms CPLEX in all examples, where Q_1 can even reach –24.56 % and –32.34 % in example 2 of subcase 30 –20 –52 and subcase 60 –25 –61. In contrast, GA-FCS obtains a better solution than CPLEX in certain examples, while in other cases, it exhibits inferior performance. And GA always provides the worst result in medium and large-scale cases.

Table 5 and Table 6 show evidently that GA method always provides the worst solution in all kinds of cases, especially in medium and large-scale cases, where Q_2 is even as high as –30 % to –50 %. This phenomenon is reasonable as the penalty strategy in GA method can only passively punish infeasible MS-FS strings, not actively promote correction of chromosome. Comparing GA and GA-FCS individually, we can find that Q_3 is always bigger than Q_2 although GA-FCS is not as good as GA-FCSVNS. And this advantage becomes more and more obvious as the size of test case increases, which demonstrates the effectiveness of feasibility correction strategy. Besides, iteration curves achieved by CPLEX, GA, GA-FCS and IGA-FCSSVNS of representative examples are shown in Fig. 11 to observe the performances of algorithms in iteration process. GA method always converge to an unsatisfactory solution at an early stage while GA-FCS method converges during the mid-term with an acceptable solution, which further improves the value of feasibility correction.

Given the large gap between GA and other methods, iterations curves without GA are redrawn as shown in Fig. 12, which aims to provide a better view of the local features of GA-FCS and IGA-FCSSVNS. Consistent with information reflected in Table 6, IGA-FCSSVNS and CPLEX both obtain the global optimal solution in small-scale cases (except for the subcase 10 –16 –25); in medium and large-scale cases, IGA-FCSSVNS is far superior to CPLEX. Most importantly, green curve (IGA-FCSSVNS) shows an evident downward trend compared with the blue one (GA-FCS) after the introduction of self-learning VNS mechanism starting from the 80th generation, and eventually obtains a better solution than any other method. This characteristic reveals the local breakthrough

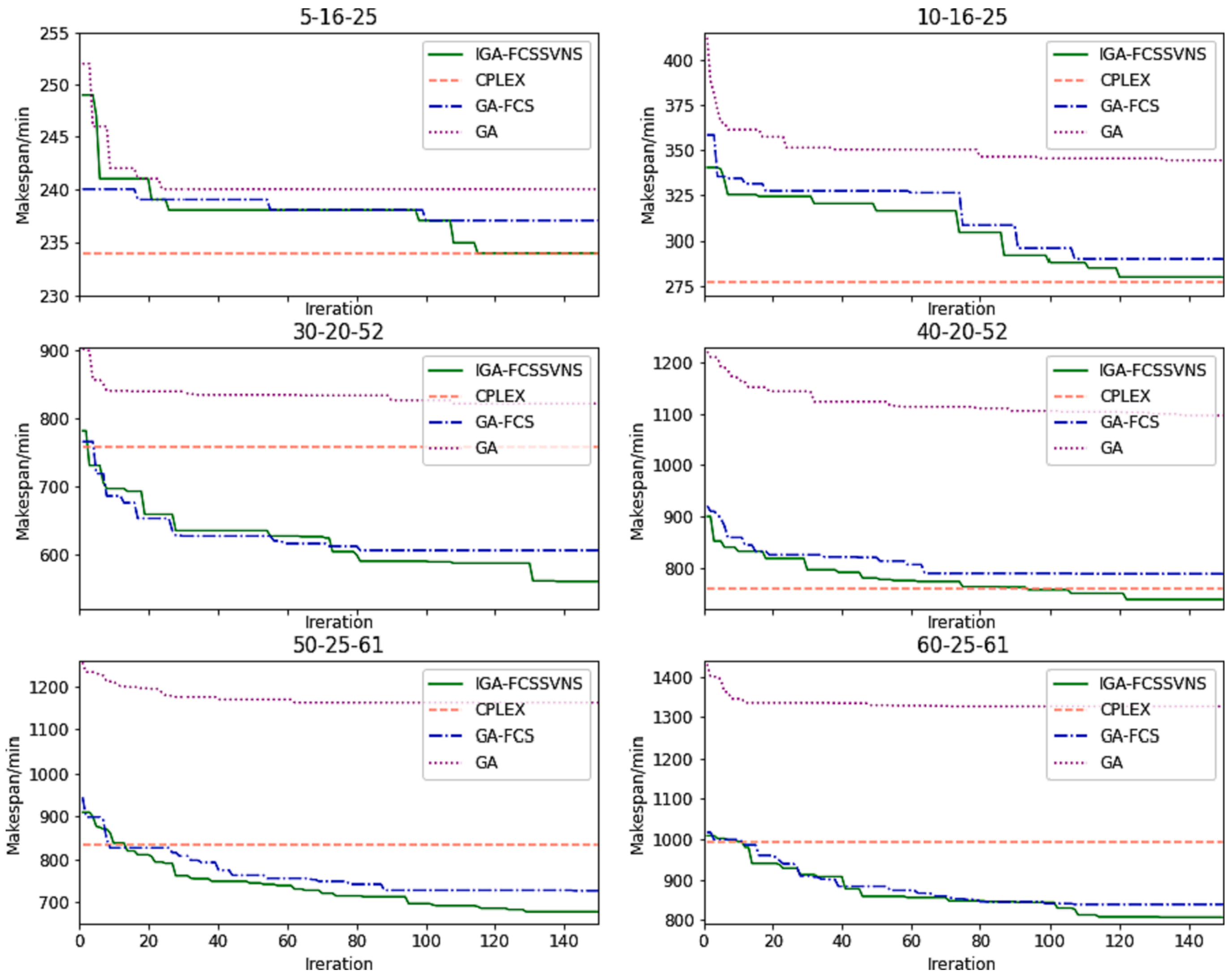


Fig. 11. Iteration curves of representative cases (GA method included).

capability of self-learning VNS.

In general, MIP model targeting on MRFJSP-FPCO is unable to produce a high-quality solution within the ideal time in a complex scenario with complicated resources constraints as well as fixture-pallet combinatorial optimisation. Whereas IGA-FCSSVNS is able to find an excellent solution within a relatively short time, with the aid of feasibility correction strategy and self-learning VNS.

6. Conclusion

In this paper, we concentrate on a brand-new flexible job shop scheduling problem considering tri-resource constraints of fixture-pallet-machine, in which a MIP model is formulated, and an algorithm named IGA-FCSSVNS is proposed and tested. To ensure the legality of individuals during the entire iteration process, feasibility repair strategies targeted at chromosome strings MS and FS are introduced. Furthermore, a self-learning variable neighbourhood search is achieved in the later stages of iteration through the establishment of knowledge matrixes and VNS repository. The effectiveness and superiority of IGA-FCSSVNS are verified through solving various examples under different scales based on real production scenarios from a leading engine manufacturer in China.

Unlike common production scheduling, IGA-FCSSVNS not only executes optimal machine selection and sequence arrangement for operations, but also optimizes the combination of fixture-pallet resources. It closely adheres to the actual production needs of workshop and has high potential for application in the era of intelligent transformation.

In our study, there are several potential avenues to explore for future improvement. First, the set-up time of fixtures could be taken into consideration as it satisfies the real demand in factory, and it may vary with different types of products. In addition, further research is needed to address timely and reasonable dynamic response to abnormal conditions in the flexible workshops. Finally, as the current research boom in the field of intelligent decision making, how to apply latest deep reinforcement learning methodology in combinatorial optimisation also deserves our attention.

Funding

This work was funded by National Natural Science Foundation of China (Grant No. 52275499), and the National Key Research and Development Program of China (No. 2022YFF0605700).

CRediT authorship contribution statement

Molin Liu: Conceptualization, Methodology, Software, Writing – original draft. **Jun Lv:** Writing – review & editing. **Shichang Du:**

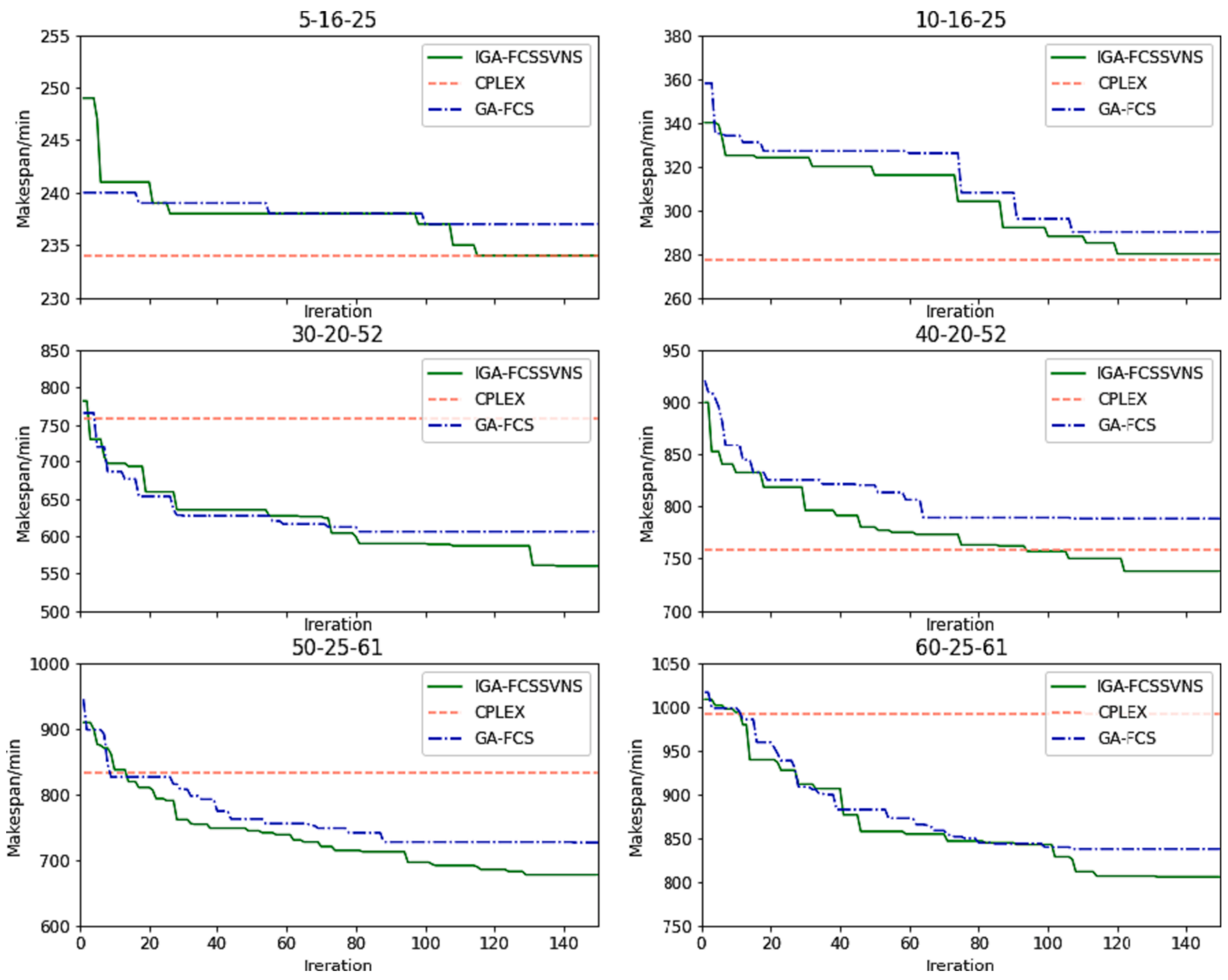


Fig. 12. Iteration curves of representative cases (without GA method).

Supervision. **Yafei Deng:** Validation, Visualization. **Xiaoxiao Shen:** Investigation. **Yulu Zhou:** Data curation.

Declaration of competing 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.

Data availability

The data that has been used is confidential.

References

Bierwirth, C., & Mattfeld, D. C. (1999). Production scheduling and rescheduling with genetic algorithms. *Evolutionary Computation, 7*(1), 1–17.

Brucker, P., & Schlie, R. (1990). Job-shop scheduling with multipurpose machines. *Computing*.

Cañas, H., Mula, J., Díaz-Madroñero, M., & Campuzano-Bolarín, F. (2021). Implementing industry 4.0 principles. *Computers & Industrial Engineering, 158*, Article 107379.

Chan, F., Wong, T., & Chan, L. (2006). Flexible job-shop scheduling problem under resource constraints. *International Journal of Production Research, 44*(11), 2071–2089.

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

Defersha, F. M., Obimuyiwa, D., & Yimer, A. D. (2022). Mathematical model and simulated annealing algorithm for setup operator constrained flexible job shop scheduling problem. *Computers & Industrial Engineering, 171*, Article 108487.

Demir, Y., & İşleyen, S. K. (2014). An effective genetic algorithm for flexible job-shop scheduling with overlapping in operations. *International Journal of Production Research, 52*(13), 3905–3921.

Ding, H., & Gu, X. (2020). Improved particle swarm optimization algorithm based novel encoding and decoding schemes for flexible job shop scheduling problem. *Computers & Operations Research, 121*, Article 104951.

Dominic, P. D., Kaliyamoorthy, S., & Kumar, M. S. (2004). Efficient dispatching rules for dynamic job shop scheduling. *The International Journal of Advanced Manufacturing Technology, 24*, 70–75.

Fan, J., Shen, W., Gao, L., Zhang, C., & Zhang, Z. (2021). A hybrid Jaya algorithm for solving flexible job shop scheduling problem considering multiple critical paths. *Journal of Manufacturing Systems, 60*, 298–311.

Fan, J., Zhang, C., Liu, Q., Shen, W., & Gao, L. (2022). An improved genetic algorithm for flexible job shop scheduling problem considering reconfigurable machine tools with limited auxiliary modules. *Journal of Manufacturing Systems, 62*, 650–667.

Fattahi, P., Saidi Mehrabad, M., & Jolai, F. (2007). Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *Journal of Intelligent Manufacturing, 18*, 331–342.

Gong, G., Chiong, R., Deng, Q., & Gong, X. (2020). A hybrid artificial bee colony algorithm for flexible job shop scheduling with worker flexibility. *International Journal of Production Research, 58*(14), 4406–4420.

Gothwal, S., & Raj, T. (2017). Different aspects in design and development of flexible fixtures: Review and future directions. *International Journal of Services and Operations Management, 26*(3), 386–410.

Hajibabaei, M., & Behnamian, J. (2021). Flexible job-shop scheduling problem with unrelated parallel machines and resources-dependent processing times: A tabu search algorithm. *International Journal of Management Science and Engineering Management, 16*(4), 242–253.

Hamzadayi, A., & Yildiz, G. (2017). Modeling and solving static m identical parallel machines scheduling problem with a common server and sequence dependent setup times. *Computers & Industrial Engineering, 106*, 287–298.

Lei, D., & Guo, X. (2014). Variable neighbourhood search for dual-resource constrained flexible job shop scheduling. *International Journal of Production Research, 52*(9), 2519–2529.

Lei, K., Guo, P., Zhao, W., Wang, Y., Qian, L., Meng, X., & Tang, L. (2022). A multi-action deep reinforcement learning framework for flexible Job-shop scheduling problem. *Expert Systems with Applications, 205*, Article 117796.

Li, R., Gong, W., & Lu, C. (2022). Self-adaptive multi-objective evolutionary algorithm for flexible job shop scheduling with fuzzy processing time. *Computers & Industrial Engineering, 168*, Article 108099.

Li, X., & Gao, L. (2016). An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. *International Journal of Production Economics, 174*, 93–110.

Li, X., Peng, Z., Du, B., Guo, J., Xu, W., & Zhuang, K. (2017). Hybrid artificial bee colony algorithm with a rescheduling strategy for solving flexible job shop scheduling problems. *Computers & Industrial Engineering, 113*, 10–26.

Mahmudy, W. F., Marian, R. M., & Luong, L. H. (2013). *Real coded genetic algorithms for solving flexible job-shop scheduling problem-Part II: Optimization* (Vol. 701). Trans Tech Publ.

Meng, L., Zhang, C., Ren, Y., Zhang, B., & Lv, C. (2020). Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem. *Computers & Industrial Engineering, 142*, Article 106347.

Müller, D., Müller, M. G., Kress, D., & Pesch, E. (2022). An algorithm selection approach for the flexible job shop scheduling problem: Choosing constraint programming solvers through machine learning. *European Journal of Operational Research, 302*(3), 874–891.

Naderi, B., & Roshanaei, V. (2022). Critical-path-search logic-based benders decomposition approaches for flexible job shop scheduling. *INFORMS Journal on Optimization, 4*(1), 1–28.

Osterrieder, P., Budde, L., & Friedli, T. (2020). The smart factory as a key construct of industry 4.0: A systematic literature review. *International Journal of Production Economics, 221*, Article 107476.

Pezzella, F., Morganti, G., & Ciaschetti, G. (2008). A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research, 35*(10), 3202–3212.

Ren, W., Wen, J., Yan, Y., Hu, Y., Guan, Y., & Li, J. (2021). Multi-objective optimisation for energy-aware flexible job-shop scheduling problem with assembly operations. *International Journal of Production Research, 59*(23), 7216–7231.

Singh, M. R., & Mahapatra, S. S. (2016). A quantum behaved particle swarm optimization for flexible job shop scheduling. *Computers & Industrial Engineering, 93*, 36–44.

Soto, C., Dorronsoro, B., Fraire, H., Cruz-Reyes, L., Gomez-Santillan, C., & Rangel, N. (2020). Solving the multi-objective flexible job shop scheduling problem with a novel parallel branch and bound algorithm. *Swarm and Evolutionary Computation, 53*, Article 100632.

Teekeng, W., & Thammano, A. (2012). Modified genetic algorithm for flexible job-shop scheduling problems. *Procedia Computer Science, 12*, 122–128.

Thörnblad, K., Strömberg, A.-B., Patriksson, M., & Almgren, T. (2015). Scheduling optimisation of a real flexible job shop including fixture availability and preventive maintenance. *European Journal of Industrial Engineering, 9*(1), 126–145.

Tian, Y., Gao, Z., Zhang, L., Chen, Y., & Wang, T. (2023). A multi-objective optimization method for flexible job shop scheduling considering cutting-tool degradation with energy-saving measures. *Mathematics, 11*(2), 324.

Tian, Y., Xiong, T., Liu, Z., Mei, Y., & Wan, L. (2022). Multi-objective multi-skill resource-constrained project scheduling problem with skill switches: Model and evolutionary approaches. *Computers & Industrial Engineering, 167*, Article 107897.

Vallikavungal Devassia, J., Salazar-Aguilar, M. A., & Boyer, V. (2018). Flexible job-shop scheduling problem with resource recovery constraints. *International Journal of Production Research, 56*(9), 3326–3343.

Wu, X., Peng, J., Xiao, X., & Wu, S. (2021). An effective approach for the dual-resource flexible job shop scheduling problem considering loading and unloading. *Journal of Intelligent Manufacturing, 32*, 707–728.

Zhang, G., Gao, L., & Shi, Y. (2011). An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications, 38*(4), 3563–3573.

Zhang, G., Hu, Y., Sun, J., & Zhang, W. (2020). An improved genetic algorithm for the flexible job shop scheduling problem with multiple time constraints. *Swarm and Evolutionary Computation, 54*, Article 100664.

Zhang, J., Ding, G., Zou, Y., Qin, S., & Fu, J. (2019). Review of job shop scheduling research and its new perspectives under Industry 4.0. *Journal of Intelligent Manufacturing, 30*, 1809–1830.