International Journal of Industrial Engineering, 32(6), 1433-1462,2025

AN ADAPTIVE MULTI-SWARM GRAY WOLF OPTIMIZER FOR FLEXIBLE JOB SHOP SCHEDULING PROBLEM WITH ASSEMBLY/DISASSEMBLY, AND LOT STREAMING

Liezheng Shen[1], Haiping Zhu[1, 2, *], and Siqi Liu[1]

[1]School of Mechanical Science and Engineering
Huazhong University of Science and Technology
Wuhan, China
[*]Corresponding author's e-mail: haipzhu@hust.edu.cn

[2]National Centre of Technology Innovation for
Intelligent Design and Numerical Control
Wuhan, China

In the manufacturing/remanufacturing systems of complex products, assembly/disassembly constraints introduce challenges of synchronization and precedence constraints. And large-scale orders have rendered mass production models obsolete. Although existing research on the flexible job shop scheduling problem (FJSP) and its integration with lot streaming (LS) has been extensive, it is still unable to solve these problems simultaneously. To bridge this gap, this paper proposes a novel FJSP with assembly/disassembly and LS (FJSP-AD-LS) to minimize makespan. A MILP model is developed to describe the problem, and an adaptive multi-swarm grey wolf optimizer (AMGWO) is proposed to solve the NP-hard problem. The AMGWO employs a multi-swarm framework with dynamic regrouping to balance exploration and exploitation, and incorporates multiple encoding-decoding methods, evolutionary operators, and knowledge-based neighborhood search methods. A total of 49 new instances are conducted by extending the classical benchmarks. Computational results demonstrate that AMGWO achieves over 10% improvement in makespan compared to the basic algorithm through ablation studies. While commercial MILP solvers could only obtain feasible solutions for 4 small-scale instances within time limits, the proposed algorithm consistently generates high-quality solutions for all test instances. Comparative studies against state-of-the-art methods confirm the superior performance of AMGWO across all 49 instances.

Keywords: Flexible Job Shop Scheduling; Assembly/Disassembly Scheduling; Lot Streaming; Metaheuristics.

(Received on May 8, 2025; Accepted on August 10, 2025)

1. INTRODUCTION

In modern manufacturing, production scheduling is a crucial decision that directly affects productivity and resource utilization. With the growing demand for product customization, effectively arranging production sequences and allocating resources has become a critical issue for boosting productivity.

Research on flexible job shop scheduling problems (FJSP) and lot streaming (LS) technologies can better address these demands, offering more efficient and flexible scheduling solutions for the manufacturing industry (Gao & Pan, 2016; Singh *et al*., 2016; Xu & Nagi, 2013). FJSP is an extension of the classic NP-hard job shop scheduling problem (JSP) (Garey *et al*., 1976), which addresses these needs by allowing each operation of a job to be processed on multiple candidate machines (Brucker & Schlie, 1990; Hurink *et al*., 1994). This flexibility enhances adaptability but significantly increases the solution complexity due to the combinatorial explosion of feasible schedules (Xie *et al*., 2023). In FJSP, the basic scheduling unit is a single workpiece, which cannot be divided.

The LS technology allows jobs to be processed in multiple sublots, which greatly reduces machine idle time and improves the overall efficiency of production. Reiter (1965) first introduced the concept of LS. The fundamental idea is to divide a large batch into multiple smaller sublots. Once a sublots completes a specific operation, it can be transferred to the next operation. It means that multiple sublots of the same job can be processed in parallel. This approach effectively reduces machine idle time, decreases work-in-progress (WIP), and shortens the overall production cycle. However, LS introduces more complex decision-making, involving the determination of optimal sub-lot sizes, transfer strategies, and sequencing both within and between jobs, thereby transforming the scheduling problem into an integrated issue combining batch processing, lot sizing, and sequencing (Kumar *et al*., 2000; Trietsch & Baker, 1993).

Although research on the integration of FJSP and LS has been extensive (Li, 2022; Novas, 2019), in practical production, these approaches have failed to adequately address the inherent priority constraints and synchronization challenges in complex production processes involving multi-level assembly and disassembly operations. Such processes are prevalent in industries like manufacturing/remanufacturing, heavy equipment refurbishment, and complex machinery maintenance (Guo *et al*., 2021; Kalayci & Gupta, 2013; Li & Feng, 2021). For instance, in sectors such as construction machinery, railways, automotive, and electronics (Ehm, 2024; Ghandi & Masehian, 2015), there are many instances where structural parts are machined, then welded and assembled (disassembled) into small parts, and then machined and assembled (disassembled) into components. These complex products, containing assembly and disassembly, have special BOMs, which implicitly limit the prioritization constraints of operations. In FJSP, it is still rare to consider assembly and disassembly constraints together, and integrating lot streaming scheduling into such scenarios represents a broader and yet unexplored area.

The previous scheduling methods are difficult to apply directly to production with assembly/disassembly processes. Therefore, a flexible job shop scheduling problem with assembly/disassembly and lot-streaming (FJSP-AD-LS) is studied, and an adaptive multi-swarm gray wolf optimizer (AMGWO) is proposed for solving it. The main contributions are as follows:

(1) In terms of the problem, the FJSP-AD-LS to minimize makespan is first proposed and studied in this paper. A mathematical model of the problem is developed and validated.

(2) In terms of the algorithmic framework, the AMGWO is proposed based on the grey wolf optimizer (GWO) to solve FJSP-AD-LS. The following steps to improve the performance: 1) a multi-swarm framework with dynamic regrouping to balance exploration and exploitation; 2) three encoding-decoding methods and eleven evolutionary operators are designed to accomplish efficient search; 3) three neighborhood search operators are designed for lot-sizing, sublots sequence, and machine assignment to enhance the exploration capability.

(3) A total of 49 new instances are constructed based on FJSP benchmarks. A comprehensive experimental analysis is conducted on the algorithm parameters, encoding-decoding methods, and improvement strategies. Finally, comparative experiments with other well-known algorithms demonstrate that the proposed algorithm achieves better results.

The remainder of this paper is organized as follows. Section 2 discusses the related works. Section 3 describes the general steps of the research method. Section 4 describes the FJSP-AD-LS and gives a mathematical model. In Section 5, the AMGWO is described, including the framework, encoding-decoding methods, and evolutionary operators. Section 6 describes the comprehensive experimental analysis. The conclusions are presented in Section 7.

2. LITERATURE REVIEW

2.1 Lot-streaming scheduling

The LS techniques contain three main strategies: equal sublots, consistent sublots, and variable sublots. Currently, LS techniques are widely applied to various shop scheduling problems, especially in flow shops and job shops. Table 1 summarizes the literature related to LS scheduling for flow shop and job shop. In addition to the relevant literature in Table 1, other special job shop lot-streaming scheduling problems are also being studied.

Table 1. Related literature on lot-streaming scheduling problems in flow shop and job shop

Sublots type	Workshop	Literature
Equal sublots	Flow shop	Pan and Ruiz (2012), Zhang *et al*. (2017), Huang and Yu (2017), Sang *et al*. (2018) Meng *et al*. (2018), Han, Li, *et al*. (2019), Gong, *et al*. (2019), Gürsoy Yılmaz and Faruk Yılmaz (2022)
	Job shop	Z. Tian *et al*. (2024)
Consistent sublots	Flow shop	Pan *et al*. (2011), Mortezaei and Zulkifli (2013), Ferraro *et al*. (2019), Fang *et al*. (2021), Zhang *et al*. (2021), Zhang *et al*. (2022), Zhu *et al*. (2024)
	Job shop	Defersha and Chen (2012), Lei and Guo (2013), Defersha and Bayat Movahed (2018), Novas (2019), Yunusoglu and Topaloglu Yildiz (2023), Fan *et al*. (2024)
Variable sublots	Flow shop	Defersha and Chen (2010), Mortezaei and Zulkifli (2014), Wang *et al*. (2019), Li *et al*. (2020)
	Job shop	Bożek and Werner (2018), Xiuli *et al*. (2021), Li (2022)

For example, Zhang *et al*. (2020) studied a flexible manufacturing system with two areas (processing and assembly), where the processing area generates generic products, and the assembly area assembles them into different products. In the scheduling of the processing area, they considered lot-streaming and designed a distributed ant colony system to explore the Pareto front of makespan, total tardiness, and total workload. Fan *et al*. (2022) proposed an FJSP that considers reconfigurable machine tools with limited auxiliary modules (FJSP-LSMR). To minimize total tardiness, they developed a mathematical heuristic method with a VNS component. Xie *et al*. (2023) addressed the variable sublots and intermingling setting in the job shop lot-streaming scheduling problem. The intermingling setting allows different sublots of the same job to be processed discontinuously and interspersed with sublots from other jobs (Novas, 2019). They proposed a novel multi-objective Jaya algorithm based on decomposition to optimize total tardiness and the total number of transferred sublots.

2.2 Constrained FJSP

Previous studies on FJSP have expanded to address complex real-world constraints (Piroozfard *et al*., 2018). Transportation time impacts were investigated by Dai *et al*. (2019) and Hidri and Tlija (2024), who developed a GA for energy-efficient multi-objective solutions. Concurrently, Sun *et al*. (2021) proposed a hybrid algorithm integrating transportation and setup time considerations, while Ren *et al*. (2022) created a hybrid PSO-GA for dynamic FJSP with resource constraints. Maintenance-related constraints were explored through preventive maintenance scheduling by Baykasoğlu and Madenoğlu (2021) and Wocker *et al*. (2024). Table 1 shows more literature on FJSPs with additional constraints.

Table 1. Related literature on constrained FJSP

References	Description	Objectives
Dai *et al*. (2019)	FJSP with transportation constraints	makespan, energy consumption
Baykasoğlu and Madenoğlu (2021)	dynamic FJSP considering preventive maintenance and transportation	makespan, energy consumption
Sun *et al*. (2021)	FJSP with transportation and setup times	makespan, workload
Ren *et al*. (2022)	dynamic FJSP with transportation and resource constraints	makespan
Wocker *et al*. (2024)	FJSP with preventive maintenance	makespan
Kasapidis *et al*. (2025)	FJSP with multiple resource constraints	makespan
Daneshamooz *et al*. (2022)	processing stage is FJSP with LS, assembly stage is identical assembly stations	makespan
S. Zhang *et al*. (2020)	processing stage is FJSP, assembly stage is identical assembly robots	makespan, tardiness, workload
X. Li *et al*. (2022)	processing stage is FJSP with LS, no assembly time in the assembly stage	flow time
Fattahi *et al*. (2020)	processing stage is FJSP, assembly stage is a single machine	makespan
Fan *et al*. (2022)	FJSP with limited auxiliary modules	tardiness
Gong *et al*. (2024)	FJSP with discrete operation sequence flexibility	makespan, energy consumption
Tang *et al*. (2024)	distributed FJSP considering integrated sequencing flexibility	makespan, energy consumption
Hu *et al*. (2024)	processing stage is flexible job shop with limited auxiliary modules, assembly stage is reconfigurable machines with limited auxiliary modules	makespan
Z. Zhang *et al*. (2024)	DFJSP with preventive maintenance and transportation	makespan, workload
This paper	FJSP with LS and multi-level assembly/disassembly constraints	makespan

The FJSP with assembly constraints is an important extension. Traditional methods separate the scheduling of processing and assembly, which disrupts the production process of individual products and undermines the inherent parallelism between processing and assembly tasks, thereby affecting the synchronization and integrity of product manufacturing (Qiao *et al*., 2021). The assembly job shop scheduling problem can be described in multiple stages, such as:

(1) Two-stage (Fattahi *et al*., 2020): where components are produced in the processing stage and then assembled into products in the assembly stage; (2) Three-stage (Cheng *et al*., 2024): where an additional transfer stage is introduced to move components completed in the processing stage to the assembly stage; (3) Multi-stage (Z. Zhu *et al*., 2022): which may describe a multi-level assembly process, and can also include multiple processing stages.

2.3 Research motivation and gap

In the manufacturing/remanufacturing systems of complex products (e.g., electronic devices, mechanical equipment, repairable systems), assembly/disassembly operations introduce dual challenges of subcomponent synchronization and strict precedence constraints: all subcomponents must complete processing before their parent component's assembly begins, while disassembly-generated subcomponents can only undergo subsequent processing after the parent component's disassembly is completed. Since jobs are no longer independent, blocking of key components will cause all subsequent associated processes to wait. Compared to general job shops, unreasonable production plans will lead to more serious waste of resources.

When handling large-scale orders, traditional single-piece scheduling methods are inefficient, manifesting as follows: 1) Production delays in critical subcomponents create bottlenecks that stall all dependent assembly/disassembly operations, even when other machines/jobs are ready, resulting in prolonged waiting times that extend the overall makespan. 2) If sub-lots are large, they monopolize critical equipment for extended periods, causing other operations to wait for long durations and resulting in unbalanced machine utilization. 3) Reduced flexibility: It becomes difficult to respond to small-batch, high-priority orders inserted on short notice.

The integration of lot streaming with assembly/disassembly constraints demonstrates significant practical value for makespan reduction, resource utilization improvement, and system flexibility enhancement. While current research has extensively investigated conventional FJSP, FJSP with LS, and FJSP with assembly/disassembly as separate problems, a critical research gap persists in their combined consideration.

To study the tree-structured precedence constraints arising from assembly/disassembly constraints and the integrated requirements of batch processing, lot sizing, and sequencing introduced by lot streaming, this paper presents the first mathematical formulation and verification of FJSP-AD-LS to minimize makespan. To solve large-scale FJSP-AD-LS problems, this paper proposes an enhanced AMGWO based on the basic GWO. The AMGWO incorporates a multi-swarm framework with dynamic regrouping and variable neighborhood search (VNS) mechanisms to improve and balance the exploration and exploitation. To address precedence constraints from assembly/disassembly operations, we develop a novel pop-up coding method based on multi-sublot trees that effectively encodes sublot processing sequences while preventing infeasible solutions. For lot streaming scheduling decisions, the algorithm implements 3 different encoding-decoding schemes for lot sizing, supported by 11 specialized evolutionary operators and three neighborhood search operators. Extensive experimental evaluation on 49 newly generated benchmark instances derived from classical FJSP problems demonstrates the algorithm's effectiveness, with comprehensive validation of all proposed components.

3. RESEARCH METHOD

The primary objectives of this study are: 1) To investigate the FJSP-AD-LS problem and formulate a mathematical model; 2) To propose an effective swarm intelligence optimization algorithm for solving this problem; 3) To validate the proposed method through experimental analysis. The research methodology consists of the following steps:

Step 1: Problem Description and Modeling

A detailed description of the FJSP-AD-LS problem is provided, along with a concrete example for clarity. Through reasonable assumptions and simplifications, a MILP model is established. The MILP formulation is chosen because it can precisely describe the constraints of FJSP-AD-LS and facilitate comparison with solver-based solutions.

Step 2: Algorithm Design (AMGWO)

Based on the Grey Wolf Optimizer (GWO), this study proposes an improved AMGWO. The standard GWO suffers from premature convergence and local optima trapping. To address these issues, AMGWO introduces two key framework improvements: 1) Multi-swarm architecture with dynamic regrouping to balance exploration and exploitation; 2) VNS integration to enhance global exploration.

At the operational level, AMGWO has proposed: 1) 3 encoding-decoding schemes and 11 evolutionary operators to ensure efficient search; 2) 3 neighborhood search operators specifically designed for sub-lots sizing, sub-lots sequencing, and machine assignment, further improving exploration capability.

Step 3: Experimental Design and Analysis

A series of experiments is conducted to validate the effectiveness of the proposed method, including:

Step 3.1: Test Case Design

Multiple test cases are randomly generated, covering different problem scales, to serve as a unified benchmark for subsequent experiments.

Step 3.2: Parameter Tuning via Taguchi Method

The Taguchi experimental design is employed to optimize key algorithm parameters and analyze their impact on performance.

Step 3.3: Ablation Study

An ablation analysis is performed to evaluate the contribution of each improvement strategy in AMGWO.

Step 3.4: Model Validation

The Gurobi solver is applied to solve the MILP model for all test cases. This serves two purposes:

1) Verifying the correctness of the proposed model;
2) Comparing the efficiency of the solver versus the proposed AMGWO algorithm.

Step 3.5: Comparative Experiments

AMGWO is tested against several well-known optimization algorithms on all benchmark cases to validate its solution quality and computational efficiency.

4. PROBLEM DESCRIPTION AND FORMULATION

To simplify the problem description, the following notation is given:

Index:

i, i'	Indices of product types, $i, i' = 1,2, \dots, gn$.
j, j'	Indices of job types, $j, j' = 1,2, \dots, jn_i$.
o, o'	Indices of operations, $o, o' = 1,2, \dots, on_{i,j}$.
b, b'	Indices of sublots, $b, b' = 1,2, \dots, b_i$.
m, m'	Indices of machines, $m, m' = 1,2, \dots, mn$.

Symbols:

G_i	the i-th product type
$J_{i,j}$	the j-th job of the i-th product
$O_{i,j,o}$	the o-th operation of job $J_{i,j}$
$\tau_{i,j,o,b}$	the b-th sublots of operation $O_{i,j,o}$

Parameters:

gn	total number of product types
jn_i	number of jobs included in product G_i
$on_{i,j}$	number of processes contained in job jn_i
e_i	the demand quantity for product G_i
r_i	the release time for product G_i
$pt_{i,j,o,m}$	the processing time of $O_{i,j,o}$ on machine m
$u_{i,j,o,m}$	$u_{i,j,o,m} = 1$, if the $O_{i,j,o}$ can be processed on machine m; $u_{i,j,o,m} = 0$, otherwise.
$js_{i,j,j'}$	$js_{i,j,j'} = 1$, if $J_{i,j}$ is successor job of $J_{i,j'}$; $js_{i,j,j'} = 0$, otherwise
Ω	a big enough number

Decision variables:

c_{max}	the completion time of the schedule
$c_{i,j,o,b}$	completion time of $\tau_{i,j,o,b}$
$\sigma_{i,b}$	size of the b-th sublot of jobs of type G_i.
$\delta_{i,b}$	$\delta_{i,b} = 1$, if the $\sigma_{i,b}$ is non-zero; $\delta_{i,b} = 0$, otherwise.
$x_{i,j,o,b,m}$	$x_{i,j,o,b,m} = 1$, if $\tau_{i,j,o,b}$ is processed on machine m; $x_{i,j,o,b,m} = 0$, otherwise.
$y_{i,j,o,b,i',j',o',b'}$	$y_{i,j,o,b,i',j',o',b'} = 1$, if $\tau_{i,j,o,b}$ is processed after $\tau_{i',j',o',b'}$ on the same machine; $y_{i,j,o,b,i',j',o',b'} = 0$, otherwise.

4.1 Problem description

The FJSP-AD-LS can be described as follows: there are gn product types $\{G_1, G_2, \dots, G_{gn}\}$ to be processed on mn machines in the shop. The products' demands are $\{e_1, e_2, \dots, e_{gn}\}$, and the release time are $\{r_1, r_2, \dots, r_{gn}\}$. A product G_i contains jn_i jobs $\{J_{i,1}, J_{i,2}, \dots, J_{i,jn_i}\}$. According to the operation tree of product G_i, these jobs are processed, assembled, or disassembled in multi-stage to form the final product G_i. The job $J_{i,j}$ has $on_{i,j}$ operations $\{O_{i,j,1}, O_{i,j,2}, \dots, O_{i,j,on_{i,j}}\}$, and the $O_{i,j,o}$ can be processed by one machine in the optional machines set, and the processing time is $pt_{i,j,o,m}$. The batching information for the products are $\{b_1, b_2, \dots, b_{gn}\}$. For the job $J_{i,j}$ of product G_i, it can be divided into at most b_i sublots. The objective of this problem is to minimize the makespan.

Table 2. An example of FJSP-AD-LS

Product	B_i	Job	Precedent jobs	Operation	ID	Optional Machine	Process Time	r_i	e_i	b_i
1	2	$J_{1,1}$		$O_{1,1,1}$	1	$[M_2, M_3]$	[1,5]	0	10	3
				$O_{1,1,2}$	2	$[M_3, M_4]$	[2,8]			
		$J_{1,2}$	$J_{1,1}$	$O_{1,2,1}$	3	$[M_1, M_2]$	[7,3]			
		$J_{1,3}$	$J_{1,1}$	$O_{1,3,1}$	4	$[M_3, M_4]$	[6,4]			
		$J_{1,4}$	$J_{1,2}$,$J_{1,3}$	$O_{1,4,1}$	5	$[M_3, M_4]$	[8,3]			
2	3	$J_{2,1}$		$O_{2,1,1}$	6	$[M_1, M_2]$	[5,1]	0	20	3
		$J_{2,2}$	$J_{2,1}$	$O_{2,2,1}$	7	$[M_3, M_4]$	[6,7]			
		$J_{2,3}$	$J_{2,1}$	$O_{2,3,1}$	8	$[M_1, M_2]$	[9,4]			
		$J_{2,4}$	$J_{2,2}$	$O_{2,4,1}$	9	$[M_3, M_4]$	[7,5]			

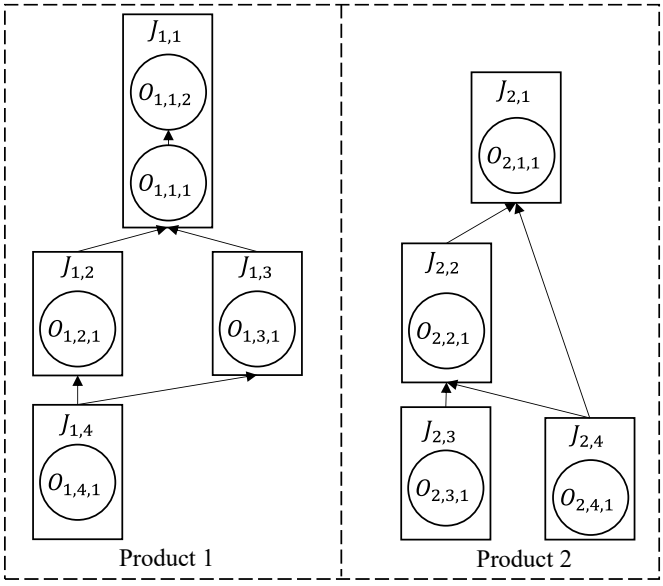


Figure 1. Product structure of the FJSP-AD-LS example

To facilitate understanding, an example is provided in Table 2 and Figure 1. Each node represents an operation, with "solid frames" indicating its corresponding job and "dashed frames" indicating its associated product. Both products include assembly and disassembly operations.

The LS problem studied in this paper is the consistent sublots problem, i.e., the sizes of different sublots of a product can be unequal, but the number and sizes of sublots remain consistent across all operations. In addition, the batching of jobs within the same product is the same. The following assumptions are made: 1) each operation can be processed only once on a single machine; 2) there is no transportation time or machine setup time; 3) operation cannot be interrupted during processing; 4) only one operation can be processed on each machine at the same time; and 5) the job and the machine are ready at the initial moment.

4.2 Mathematical model

Objective function:

$$Minimize{:}\ Objective = c_{max} \tag{1}$$

Subject to:

$$c_{max} \geq c_{i,j,o,b}, \qquad \forall i,j,o,b \tag{2}$$

Constraint **Error! Reference source not found.** calculates the maximal completion time of a schedule.

$$\sum_{b=1}^{b_i} \sigma_{i,b} = e_i, \qquad \forall i \tag{3}$$

$$\delta_{i,b} \cdot \Omega \geq \sigma_{i,b}, \qquad \forall i,b \tag{4}$$

$$\delta_{i,b} \leq \sigma_{i,b}, \qquad \forall i,b \tag{5}$$

Constraint **Error! Reference source not found.** identifies the products' demands that need to be satisfied. Constraints **Error! Reference source not found.**, **Error! Reference source not found.** are combined to identify non-zero sublots. If the size of sublots $\sigma_{i,b} = 0$, constraint **Error! Reference source not found.** will force $\delta_{i,b} = 0$. If the size of sublots $\sigma_{i,b} \neq 0$, constraint **Error! Reference source not found.** holds only if $\delta_{i,b} = 1$, which indicates that the sublots are non-empty and must be scheduled.

$$\sum_{m=1}^{mn} x_{i,j,o,b,m} = \delta_{i,b}, \qquad \forall i,j,o,b \tag{6}$$

$$x_{i,j,o,b,m} \leq u_{i,j,o,m}, \qquad \forall i,j,o,b,m \tag{7}$$

Constraint **Error! Reference source not found.** ensures that a non-empty sublots is scheduled only once. Constraint **Error! Reference source not found.** ensures that sublots can only be processed on optional machines.

$$c_{i,j,o,b} - \sigma_{i,b} \cdot pt_{i,j,o,m} + \left(1 - x_{i,j,o,b,m}\right) \cdot \Omega \geq r_i, \qquad \forall (i,j,b,m), o = 1 \tag{8}$$

$$c_{i,j,o,b} - \sigma_{i,b} \cdot pt_{i,j,o,m} + \left(1 - x_{i,j,o,b,m}\right) \cdot \Omega \geq c_{i,j,o-1,b}, \qquad \forall (i,j,b,m), o > 1 \tag{9}$$

Constraint **Error! Reference source not found.** restricts the first operation of a sublot cannot starting before the release time. Constraint **Error! Reference source not found.** means all the sublots should be processed according to the processing routes.

$$c_{i,j,o,b} - \sigma_{i,b} \cdot pt_{i,j,o,m} + \left(2 - js_{i,j,j'} - x_{i,j,o,b,m}\right) \cdot \Omega \geq c_{i,j',o',b}, \qquad \forall (i,j,b,m,j'), o = 1, o' = on_{i,j'} \tag{10}$$

Constraint **Error! Reference source not found.** defines assembly/disassembly constraints to ensure that the job cannot be started before its precedent jobs are finished.

$$c_{i,j,o,b} - \sigma_{i,b} \cdot pt_{i,j,o,m} + \left(3 - x_{i,j,o,b,m} - x_{i',j',o',b',m} - y_{i,j,o,b,i',j',o',b'}\right) \cdot \Omega \geq c_{i',j',o',b'}, \quad \forall (i,j,o,b,m,i',j',o',b'), \tau_{i,j,o,b} \neq \tau_{i',j',o',b'} \tag{11}$$

$$c_{i',j',o',b'} - \sigma_{i',b'} \cdot pt_{i',j',o',m} + \left(2 - x_{i,j,o,b,m} - x_{i',j',o',b',m} + y_{i,j,o,b,i',j',o',b'}\right) \cdot \Omega \geq c_{i,j,o,b}, \quad \forall (i,j,o,b,m,i',j',o',b'), \tau_{i,j,o,b} \neq \tau_{i',j',o',b'} \tag{12}$$

Constraints **Error! Reference source not found.** and **Error! Reference source not found.** define the processing sequence of sublots on the same machine and prevent the processing of sublots on the same machine from overlapping.

5. AMGWO ALGORITHM

5.1 Framework

The GWO is a swarm intelligence optimization algorithm that mimics the hunting behavior of grey wolves (Mirjalili *et al*., 2014). However, the basic GWO suffers from the drawback of premature convergence, as all individuals in the wolves tend to converge to the same state. The AMGWO framework is shown in Figure 2, incorporating the following improvements: 1) a multi-swarm framework with dynamic regrouping to balance exploration and exploitation; 2) A VNS mechanism is integrated to enhance the exploration capability of the algorithm.

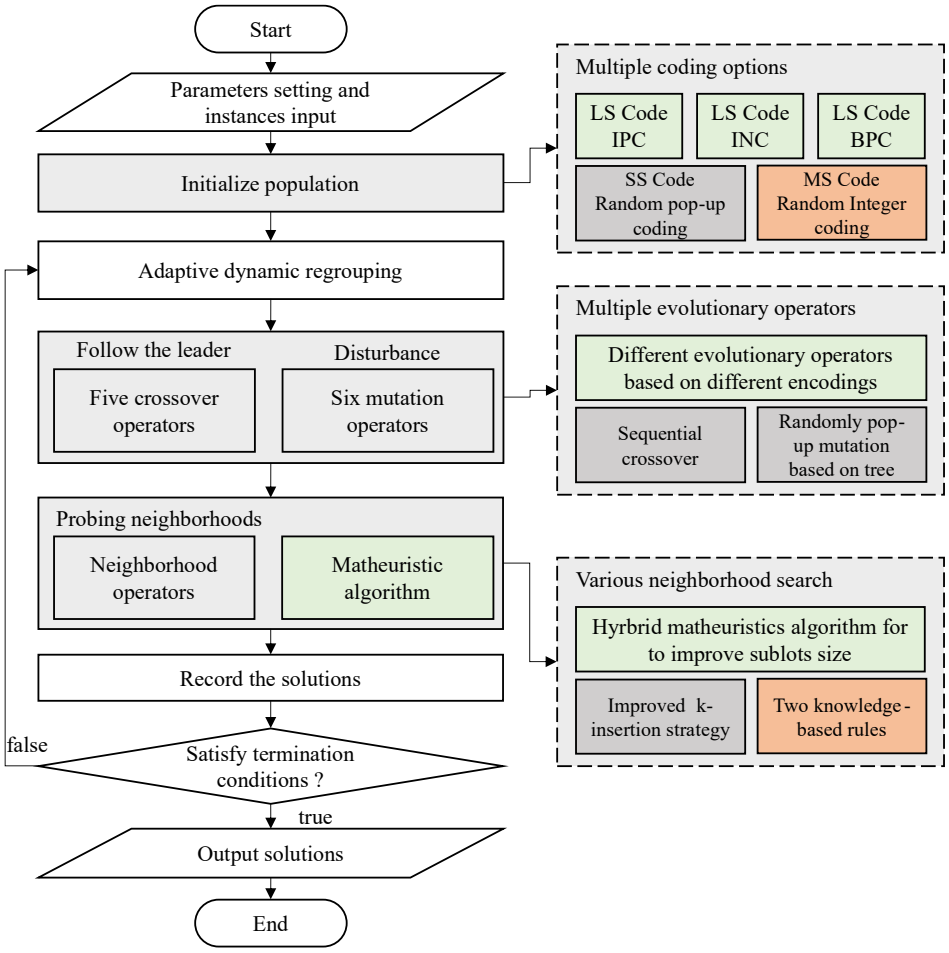


Figure 2. Framework of the AMGWO

5.2 Encoding

The FJSP-AD-LS requires decision-making on lot-sizing, machine selection, and sublots sequences. In this study, a three-segment encoding approach is adopted, consisting of: lot-sizing encoding (LS), machine selection encoding (MS), and sublots sequence encoding (SS).

5.2.1 Encoding for LS

For the LS part, three coding methods are proposed to compare in the later experiments, as shown in Figure 3.

(1) Integer-Percentage Code (IPC)

The IPC consists of a set of integers. To prevent the extreme values in the decoding process, this set of integers can be set in some interval (e.g., 0 to 500). Following the sequence of product IDs, the adjacent b_i values correspond to the sublots information of the product G_i. The specific size for each sublots can be determined based on the proportion of each element's value to the total sum of the b_i values.

(2) Binary-Percentage Code (BPC)

In BPC, the quantity information of a sublots is represented by a binary encoding of length L. For instance, as shown in Figure 3, if $L = 5$, each set of 5 binary digits determines the quantity information for one sublots. During decoding, the binary code is first converted into an integer, which is then decoded using the IPC method to obtain the quantity of each sublots.

(3) Integer-Number Code (INC)

In the order of product IDs, the neighboring b_i values each directly determine the number of sublots of G_i. This encoding method is simple and straightforward, but requires specific evolutionary operators.

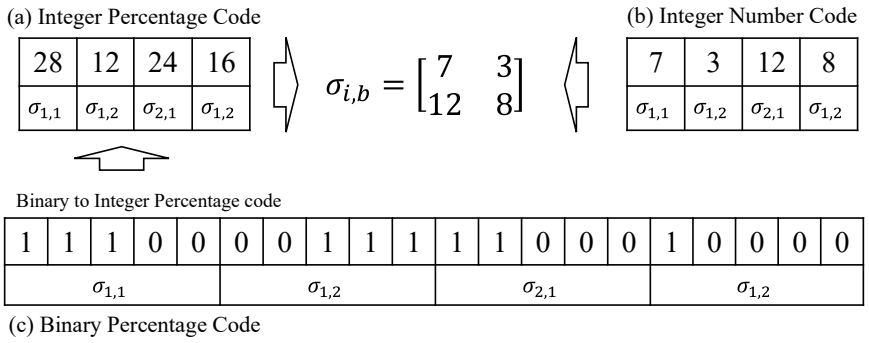


Figure 3. Three types of lot-sizing coding

5.2.2 Encoding for MS

In MS, the integer represents the index of the machine selected for the sublots. The i-th values $MS[i]$, indicates the machine index chosen for the sublots. As shown in Figure 4, the MS is prioritized by a specific index ($i \to j \to o \to b$). The machine encoding is implemented using the following rules:

(1) Random rule: randomly assign machines.

(2) Minimum processing time rule: the machine with the shortest processing time is selected preferentially.

(3) Minimum workload rule: the machine with the least accumulated workload is selected preferentially.

5.2.3 Encoding for SS

The SS is represented by the operation ID (oid) and batch ID (bid) of a sublots. As shown in Figure 4 and Table 1, in the encoding "5-1," $oid = 5, bid = 1$, where "5" represents the $O_{1,4,1}$, "1" represents the 1st sublot of $O_{1,4,1}$.

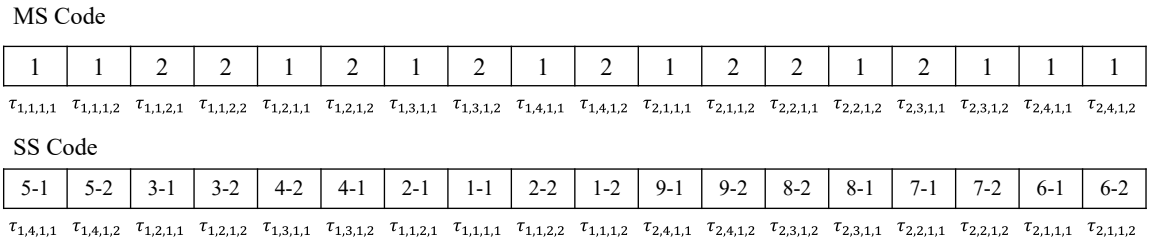


Figure 4. Machine selection coding and sublots sequence coding

This paper proposes a tree data structure that facilitates the encoding of such problems. For the tree data structure of a product, the following descriptions and definitions are provided:

For a sublots τ, it has a set of preceding sublots $JP[\tau]$ and succeeding sublots $JS[\tau]$, as well as preceding operations on the same machine $MP[\tau]$ and succeeding operations on the same machine $MS[\tau]$.

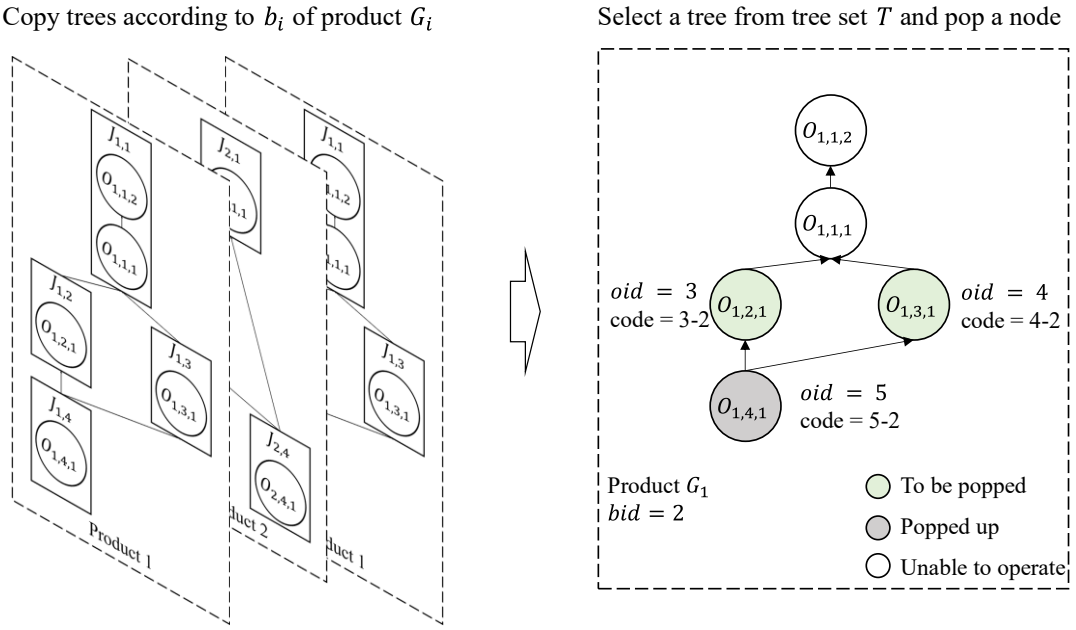


Figure 5. Pop-up Coding based on operation trees

Algorithm 1: Pop-up coding based on multi-sublots trees

Input: operation tree and number of sublots of products
Output: sublots sequence code

generate the replication set $T_i = \{T_{i,b} | b = 1 \dots b_i\}$ of the operation tree of G_i
add all replication set to $T = \{T_i | i = 1 \dots gn\}$
$n = 0$, initialize coding SS as an empty array
while (true) **do**
  if $T = \emptyset$ **then**
    break
  end if
  generate a random number $f \in \{0,1,2\}$
  if $(f = 0)$ **then**
    $T_{r,bid} \leftarrow$ randomly select a tree from T
  else if $(f = 1)$ **then**
    $T_{r,bid} \leftarrow$ select the tree with the most remaining operations
  else if $(f = 2)$ **then**
    $T_{r,bid} \leftarrow$ select the tree with the most remaining workloads
  end if
  $LN \leftarrow$ randomly pop a leaf node of $T_{r,bid}$
  $oid \leftarrow$ the ID of the operation corresponding to node LN
  $SS[n++] = oid + "-" + bid$ // String Type
  if $T_{r,bid} = \emptyset$
    remove $T_{r,bid}$ from T
  end if
end while

(1) Operation tree: A tree data structure containing nodes with the same structure as the process route. A product corresponds to a tree data structure. As shown in Figure 5, the process routes of product 1 and product 2 correspond to two kinds of operation trees.
(2) Nodes: Each node corresponds to an operation.
(3) Root node: The node corresponding to the final operation of a product. For example, as shown in Figure 1, the node for operation $O_{1,1,2}$ is the root node for product 1.
(4) Parent node: For a given node, the nodes corresponding to the operations in its successor set $JS[u]$ are its parent nodes. For example, as shown in Figure 1, the parent nodes of the node corresponding to operation $O_{1,4,1}$ are the nodes corresponding to operations $O_{1,2,1}$ and $O_{1,3,1}$.
(5) Child node: For a given node, the nodes corresponding to the operations in its predecessor set $JP[u]$ are its child nodes. For example, as shown in Figure 1, the child nodes of the node corresponding to operation $O_{1,1,1}$ are the nodes corresponding to operations $O_{1,2,1}$ and $O_{1,3,1}$.
(6) Leaf node: A node with no child nodes is a leaf node.

The operation tree can generate the SS encoding by popping the leaf nodes. It is important to note that the structure of the operation tree dynamically changes as leaf nodes are popped during the encoding process. For example, as shown in Figure 1 for G_1, the initial set of leaf nodes is $\{O_{1,4,1}\}$. After popping the leaf node $O_{1,4,1}$, the set of leaf nodes updates to $\{O_{1,2,1}, O_{1,3,1}\}$.

A pop-up coding method based on multi-sublots trees is proposed. In this method, three rules are included: (1) random pop-up, (2) priority by maximum remaining operations, and (3) priority by maximum remaining load.

5.3 Decoding

The algorithm employs an active decoding method known as the "Greedy Insertion Strategy". The main idea is to minimize the makespan during the decoding process by enabling the sublots to insert the idle time of the machine. The pseudo-code is shown in Algorithm 2.

Algorithm 2: Greedy Insertion Strategy

Input: code x
Output: decoding result
$LS \leftarrow$ lot-sizing code of x
$MS \leftarrow$ machine selection code of x
$SS \leftarrow$ sublots sequence code of x
decode LS to obtain the lot-sizing ($\sigma_{i,b}$) of products
decode MS to obtain the machine selected for each sublots
for ($int\ n = 0;\ n \leq\ SS.length;\ n++$) **then**
  $\tau_{i,j,o,b} \leftarrow$ decode $SS[n]$ to obtain the n-th sublots
  obtained the machine m selected of $\tau_{i,j,o,b}$
  if $\sigma_{i,b} = 0$ **then**
    continue; // if lot-sizing of sublots is 0, decoding for sequence is not required
  end if
  $c' \leftarrow$ the maximum end time of the preceding operations $JP[\tau_{i,j,o,b}]$
  calculate the start time $st_{i,j,o,b} = \max\{c', r_i\}$, where r_i is release time.
  calculate the idle time intervals $[I^m_{start}, I^m_{end}]$ of machine m and the completion time (C^m_{end}) of the last operation on machine m; iterate over all idle time intervals in ascending time order and calculate the start time:
  if $\max\{st_{i,j,o,b}, I^m_{start}\} + pt_{i,j,o,b} \times \sigma_{i,b} \leq I^m_{end}$ **then**
    $st_{i,j,k,b} = \max\{st_{i,j,o,b}, I^m_{start}\}$, and end the iteration.
  end if
  if no idle time interval is successfully inserted, let $st_{i,j,o,b} = C^m_{end}$
end for

5.4 Social hierarchy

The social hierarchy and leader selection are performed within each sub-swarm. For each individual in a sub-swarm, the social hierarchy and leader selection can be divided into the following cases:

1) if $f(X) \le f(X_\alpha)$, then $X_{leader} = X$;
2) if $f(X_\alpha) < f(X) \le f(X_\beta)$, then $X_{leader} = X_\alpha$;
3) if $f(X_\beta) < f(X) \le f(X_\delta)$, then X_{leader} is a random selection from $\{X_\alpha, X_\beta\}$;
4) if $f(X_\delta) < f(X)$, then X_{leader} is a random selection from $\{X_\alpha, X_\beta, X_\delta\}$;

where X denotes the current individual, $X_\alpha, X_\beta, X_\delta$ denotes the α, β, δ wolves, X_{leader} denotes the leader selected by X.

5.5 Adaptive dynamic regrouping

The main idea of dynamic regrouping is that when the diversity of a sub-swarm decreases, the entire wolves are regrouped to increase the diversity. The timing of regrouping affects the convergence accuracy and efficiency of the entire population. This paper adopts a dynamic regrouping mechanism that gives each sub-swarm more search time in the early stages of the search, while aiming to increase the diversity of the entire population in the later stages. The calculation formula for p_r is as follows:

$$p_r = \begin{cases} floor\left(p_r^{max} \times \left(1 - \frac{2 \times T_{curr}}{T_{max}}\right)\right), & T_{curr} \le \frac{T_{max}}{2} \\ 0, & T_{curr} > \frac{T_{max}}{2} \end{cases}, \tag{13}$$

where p_r is dynamic regrouping interval, p_r^{max} denotes the maximum regrouping interval, T_{curr} denotes the current algorithm runtime, T_{max} denotes the maximum runtime limit of the algorithm. The main steps of the regrouping mechanism are 1) randomly disrupting the position of the population, and 2) dividing the population equally into p_n sub-swarms according to the number of each sub-swarm as p_s.

5.6 Follow the leader

During this stage, the crossover operators will be used to guide the common wolf to search prey.

5.6.1 LS crossover

(1) IPC and BPC crossover

For the IPC and BPC, two crossover methods are proposed in this section: two-segments exchange and random points exchange, both of which will not produce infeasible solutions, as shown in Figure 6.

1) Two-segment exchange: randomly selected segments at the same position on the code to swap.
2) Random points exchange: randomly select multiple different points on the code to exchange.

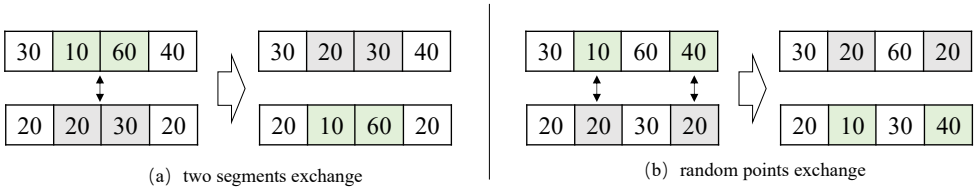


Figure 6. IPC and BPC crossover operators

(2) INC crossover

A new crossover operator is designed for INC, as shown in Figure 7. The pseudo-code is shown in Algorithm 3.

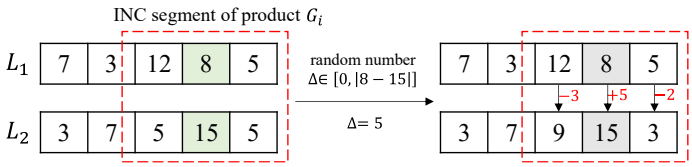


Figure 7. INC crossover operator

Algorithm 3: INC crossover operator

Input: integer-number code L_1 and L_2
Output: L_1

randomly select a product G_i with $\sigma_i \geq 2$
randomly select an index p on L_1 that belongs to G_i
$r = \min\{|L_1[p] - L_2[p]|, L_1[p], e_i - L_1[p]\}$
generate a random number $\Delta \in [0, r]$
generate a random number $f \in \{-1, 1\}$
$L_1[p] = L_1[p] - \Delta \times f$
while $\Delta > 0$ **then**
  randomly select an index j on L_1 that belongs to G_i
  if $j \neq p$ **then**
    generate a random number $d \in [0, \Delta]$
    $L_1[j] = L_1[j] + d \times f$
    $\Delta = \Delta - d$;
  end if
end while

5.6.2 SS crossover

The crossover for SS adopts a multi-segment sequence crossover method, as shown in Figure 7. Multiple non-overlapping segments are randomly selected from parent P1, and the segments from P1 are directly retained in the offspring C1. Then, after removing the selected points in P1 from P2, the remaining encoding is sequentially filled into C1. Since the encoding in the selected segment must satisfy the sequence constraints, the generated offspring encoding must be valid.

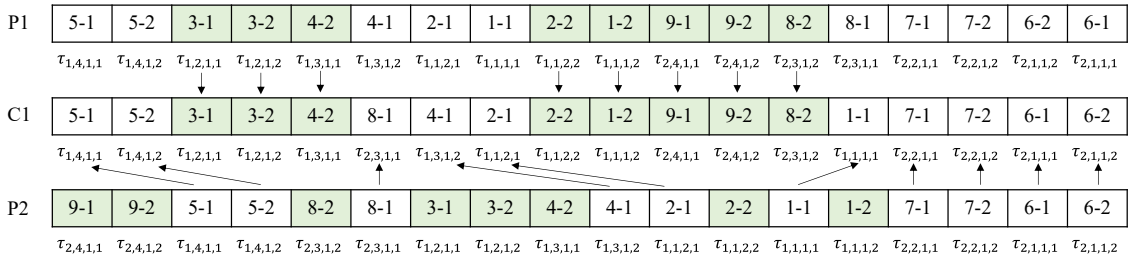


Figure 7. Multi-segment sequence crossover for SS

5.6.3 MS crossover

The multi-point random crossover is adopted for the MS. As shown in Figure 8, randomly select multiple points and exchange the corresponding values in P1 and P2, the other positions in P1 and P2 remain unchanged.

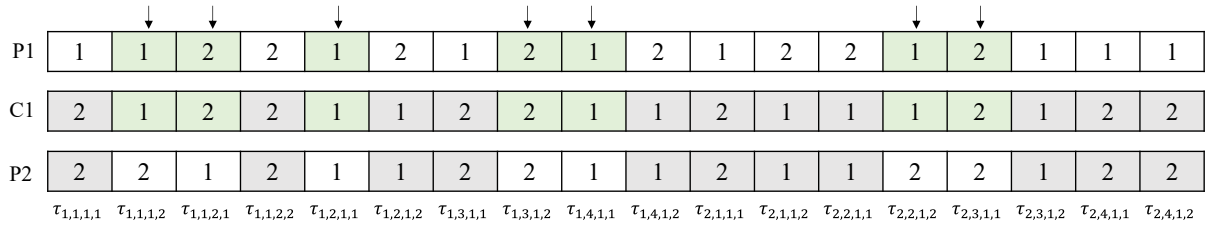


Figure 8. Multi-point random crossover for MS

5.7 Disturbance

Enforce the disturbance strategy by using the mutation operator for the new wolf.

5.7.1 LS mutation

(1) IPC mutation

The IPC mutation is performed in two ways:

1) Inversion mutation: randomly select a segment, and fill in the reverse order, as shown in Figure 9 (a);
2) Swap mutation: randomly select two different positions, and swap their values, as shown in Figure 9 (b).

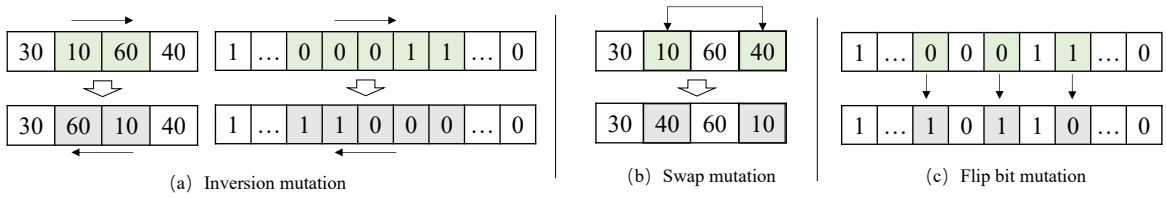


Figure 9. Mutation operators for LS

(2) INC mutation

A transition operator is proposed for INC mutation, the steps are shown in **Algorithm 4**, and an example is shown in Figure 11.

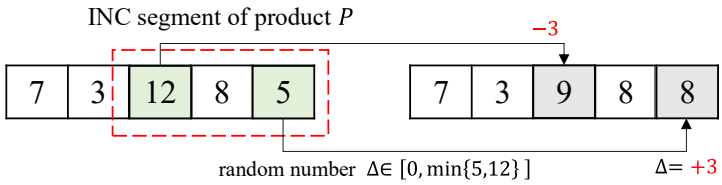


Figure 11. Transition mutation operator

Algorithm 4: INC crossover operator

Input: integer-number code L
Output: L
1: randomly select a product (G_i) with $\sigma_i \geq 2$
2: randomly select two points (r_1, r_2) on L that belongs to G_i.
3: generate a random step Δ, and $\Delta \in [0, min\{L[r_1], L[r_2]\}]$
4: generate a random number $f \in \{-1,1\}$
5: $L[r_1] = L[r_2] - \Delta \times f, L[r_2] = L[r_1] + \Delta \times f$

(3) BPC mutation

The BPC mutation is performed in three ways:

(1) Inversion mutation: randomly select a segment, and fill in the reverse order;
(2) Swap mutation: randomly select two different positions, and swap their values;
(3) Flip bit mutation: Randomly select several positions, if the code is 0, then set 1, otherwise, set 0, as shown in Figure 9 (c).

5.7.2 SS mutation

In this section, a randomly pop-up mutation method based on operation tree is proposed. Similar to the SS encoding, generate the replication set $T_i = \{T_{i,b} | b = 1 \dots b_i\}$ of the operation tree of G_i, add all replication set to $T = \{T_i | i = 1 \dots gn\}$, and randomly select a tree from T. Randomly select a node from the tree, and obtain the subtree of the node for random pop-up mutation.

As shown in Figure 10, operation $O_{1,1,1}$ is selected for mutation. Firstly, obtain the subtree corresponding to the node, then perform random pop-ups to obtain the new encoding C1. Note: For $\forall v \in JP[u]$, if the size of $JS[v]$ is greater than 1, the node v and its descendant nodes are disregarded in the pop-up, otherwise, the illegal code is generated.

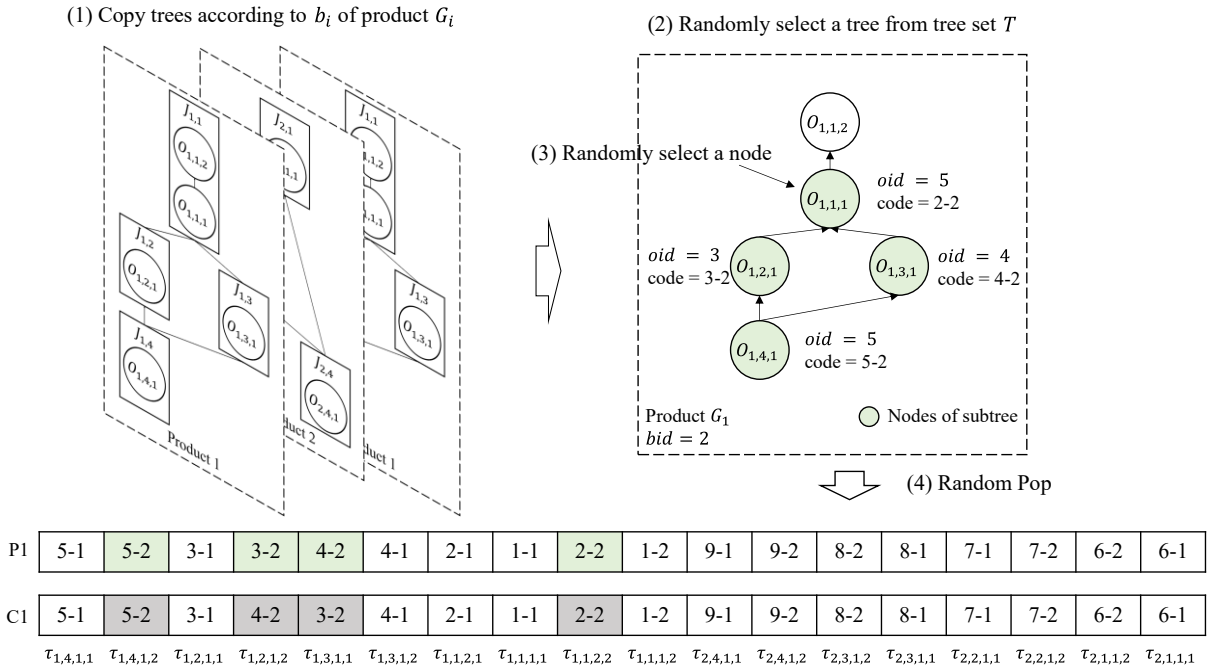


Figure 10. Randomly pop-up mutation

5.7.3 MS mutation

A random rule is adopted for MS mutation: randomly select an operation, and then randomly select another machine from the optional machines.

5.8 Probing neighborhoods

In this section, the VNS mechanism will be used to enhance wolf exploration.

5.8.1 Hybrid math-heuristic for LS

In recent years, researchers have begun to explore “math-heuristic”, which refers to optimization algorithms that combine (meta)heuristics and mathematical programming (MP) techniques. Such techniques have already achieved research results in lot-streaming scheduling (Fan *et al*., 2024). The mathematical heuristics need to be integrated into the framework of the algorithm. The main steps are:

(1) Input an individual, decode SS and MS, and transform them into model-recognizable decision variables;

(2) add sub-problem constraints and solve the sub-model with the OR solver;

(3) obtain the values of the lot-sizing decision variables ($\sigma_{i,b}$), transform them into LS in the AMGWO.

The sub-problem is modeled under the condition that the partial decision variables $x_{i,j,o,b,m}, y_{i,j,k,b,i',j',k',b'}$ ($\forall\, i,j,o,b,i',j',o',b'$) are known. The sub-problem is modeled as follows:

Objective function:

$$Minimize{:}\ Objective = c_{max} \tag{14}$$

Subject to:

$$c_{i,j,o,b} - \sigma_{i,b} \cdot pt_{i,j,o,m} + \left(1 - \delta_{i,b}\right) \cdot \Omega \geq r_i, \qquad \forall \left(x_{i,j,o,b,m} = 1\right) \cap (o = 1) \tag{15}$$

$$c_{i,j,o,b} - \sigma_{i,b} \cdot pt_{i,j,o,m} + \left(1 - \delta_{i,b}\right) \cdot \Omega \geq c_{i,j,o-1,b}, \qquad \forall \left(x_{i,j,o,b,m} = 1\right) \cap (o > 1) \tag{16}$$

$$c_{i,j,o,b} - \sigma_{i,b} \cdot pt_{i,j,o,m} + \left(1 - \delta_{i,b}\right) \cdot \Omega \geq c_{i,j',o',b}, \qquad \forall \left(x_{i,j,o,b,m} = 1\right) \cap \left(js_{i,j,j'} = 1\right) \cap \left(o = 1, o' = on_{i,j'}\right) \tag{17}$$

$$c_{i,j,o,b} - \sigma_{i,b} \cdot pt_{i,j,o,m} + \left(2 - \delta_{i,b} - \delta_{i',b'}\right) \cdot \Omega \geq c_{i',j',o',b'}\ ,\ \forall \left(x_{i,j,o,b,m} = 1\right) \cap \left(x_{i',j',o',b',m} = 1\right) \cap \left(y_{i,j,o,b,i',j',o',b'} = 1\right) \tag{18}$$

(2), (3), (4), (5) (19)

5.8.2 SS neighborhood structures

In this section, an improved k-insertion neighborhood structure on the critical path is presented, which can reduce the invalid insert operation in the neighborhood search process. The main steps are shown in **Algorithm 5**. An example from Table 2 is shown in Figure 11.

Assuming that the sublots $\tau_{1,3,1,1}$ is on the critical path and needs to neighborhood search. It is processed by the M_3. It can be seen that $JP\left(\tau_{1,3,1,1}\right) = \left\{\tau_{1,4,1,2}\right\}, JS\left(\tau_{1,3,1,1}\right) = \left\{\tau_{1,1,2,2}\right\}$, and thus the limits of moving forward and backward are $I_{min} = 2$ and $I_{max} = 9$, respectively.The sequence on M_3 is shown in orange in Figure 11. If a valid insertion is to be performed, the relative position of the sublots on the M_3 must be changed. If a valid forward or backward needs to be made to the $\tau_{1,3,1,1}$, it must be moved in front of sublots $MP\left(\tau_{1,3,1,1}\right) = \tau_{1,4,1,2}$, or behind sublots $MS\left(\tau_{1,3,1,1}\right) = \tau_{1,1,2,1}$, so $I'_{min} = 2, I'_{max} = 6$. Finally, the valid insertion positions set for $\tau_{1,3,1,1}$ is $S = S_1 \cap S_2 = \{7,8\}$.

Algorithm 5: improved k-insertion neighborhood structure

Input: chromosome x
Output: neighborhood set NS
1: $SS \leftarrow$ sublots sequence code of x, $NS \leftarrow \emptyset$
2: $I_{min} = 1, I_{max} = SS.length, I'_{min} = 1, I'_{max} = SS.length$
3: decode x and get the critical path p
4: $\tau \leftarrow$ randomly select a sublots from p
5: $I_\tau \leftarrow$ index of sublots τ in SS
6: **for** $\forall jp \in JP[\tau]$ and $\forall js \in JS[\tau]$ **do**
7: $I_{jp} \leftarrow$ index of jp in SS, $I_{js} \leftarrow$ index of js in SS
8: $I_{min} = \max\left(I_{min}, I_{jp}\right), I_{max} = \min\left(I_{max}, I_{js}\right)$
9: **end for**
10: **if** $I_{min} + 1 \geq I_{max} - 1$ **then**
11: it means there is no place to insert, and the algorithm ends
12: **end if**
13: **if** $MP[\tau] \neq \emptyset$ **then**
14: $I'_{min} \leftarrow$ index of $MP[\tau]$ in SS
15: **end if**
16: **if** $MS[\tau] \neq \emptyset$ **then**
17: $I'_{max} \leftarrow$ index of $MS[\tau]$ in SS
18: **end if**

19: $S_1 \leftarrow \{s_1 | s_1 \in (I_{min}, I_{max})\}, S_2 \leftarrow \{s_2 | s_2 \in [1, I'_{min}], or\ s_2 \in [I'_{max}, on]\}$
20: $S = S_1 \cap S_2$
21: **for** each I_s in S **do**
22: $y \leftarrow$ in SS, insert the code of position I_τ into position I_s
23: add y to set NS
24: **end for**

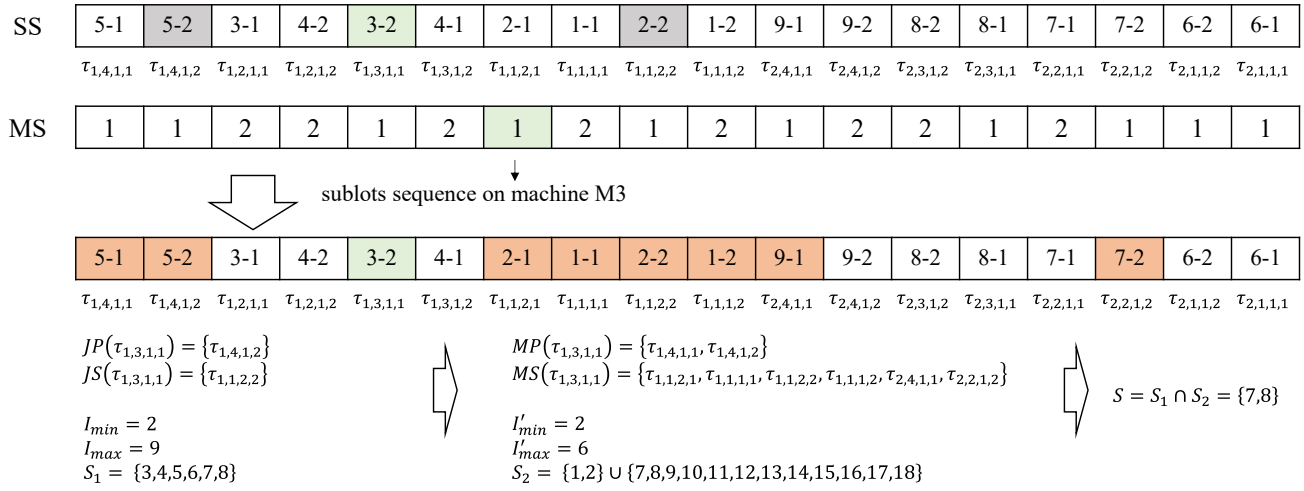


Figure 11. Example of the improved k-insertion neighborhood structure

5.8.3 MS neighborhood structures

(1) Minimum workload rule

Randomly select a sublots and choose the machine with the minimum accumulated workload (other than the current machine).

(2) Minimum processing time rule

Randomly select a sublots and choose the machine with the minimum processing time (other than the current machine).

Since the solution time of the mathematical model is highly sensitive to the problem scale, frequently activating the math-heuristic algorithm makes it difficult to balance solution quality and computational efficiency. Defersha and Bayat Movahed (2018) suggested that the model should be invoked periodically. The neighborhood operators will be executed after the algorithm has run for half of its total time, and will then be repeated at intervals of p_t iterations. Moreover, the search will be conducted exclusively on the top 1% elite individuals in terms of objective function value.

6. NUMERICAL EXPERIMENTS

6.1 Constructing benchmarks

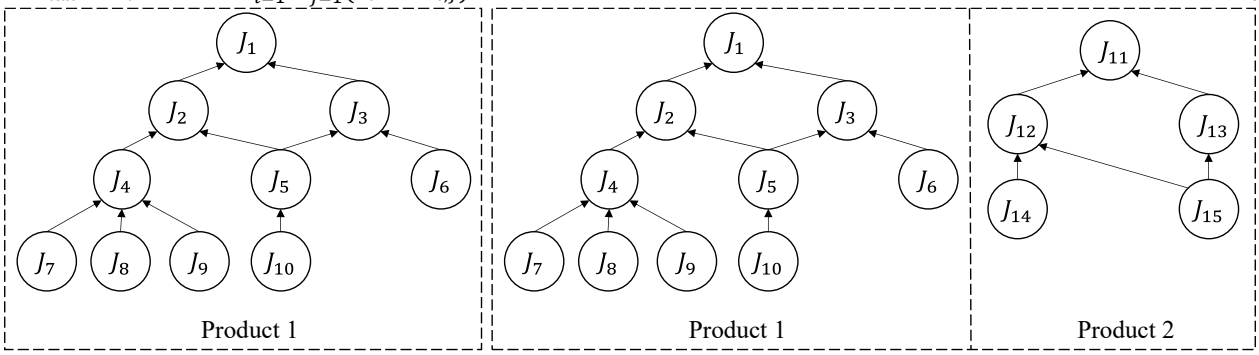
This section constructs three sets of instances for the FJSP-AD-LS based on classic benchmarks of FJSP, including: 10 BRdata instances (Brandimarte, 1993), 21 BCdata instances (Chambers & Barnes, 1996), and 18 DPdata instances (Dauzère-Pérès & Paulli, 1997). The additional data required includes product BOMs, release time (R), and batch information (E and B). As shown in Table 4, the three sets of instances are described. The parameter r_{max} represents the average processing time of all operations in the instance.

The study selected three benchmark datasets (BRdata, BCdata, and DPdata) containing only three job scales (10, 15, and 20). For these scales, three types of product structures are constructed as shown in Figure 14, where circular nodes represent jobs and arrows indicate assembly/disassembly relationships between them. Specifically, instances with 10 jobs contain only product structure 1, those with 15 jobs include product structures 1-2, and those with 20 jobs incorporate product structures 1-3. Taking the Mk03 instance with 15 jobs as an example, jobs 1-10 constitute product 1 while jobs 11-15 form product 2. In product 1's structure diagram, the arrows from jobs 7-9 pointing to job 4 indicate these three jobs will be

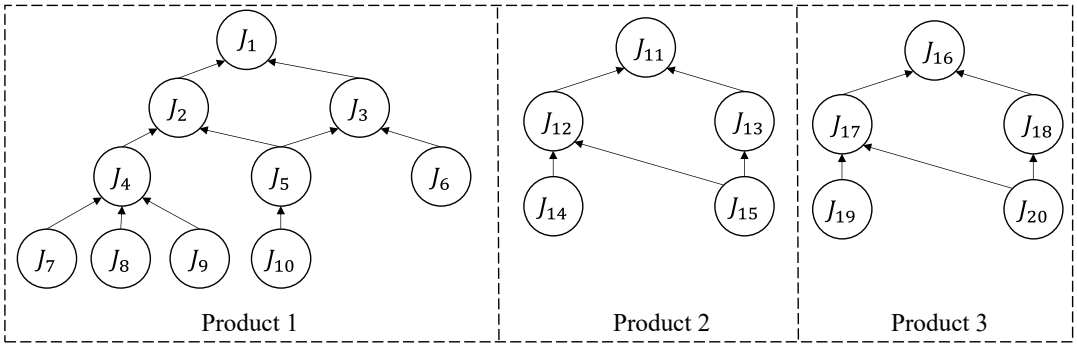
assembled into job 4 after completing all operations, whereas job 5, pointing to jobs 2-3, signifies it will be disassembled into two parts upon completion for use by jobs 2 and 3, respectively.

All of the proposed and comparative algorithms were programmed in Java with IDEA. The computer configuration is: i7-8700 CPU 4.6GHz and 16G RAM. All algorithms were tested with a time limit of T_{max} milliseconds, as shown in equation **Error! Reference source not found.**, where the T_l is the time limit factor.

$$T_{max} = T_l \times mn \times \sum_{i=1}^{gn} \sum_{j=1}^{jn_i} (b_i \times on_{i,j}) \tag{20}$$



(a) product structure of 10 jobs (b) product structures of 15 jobs



(c) product structures of 20 jobs

Figure 14. Product structure for different instances

Table 4. Parameters of constructing benchmarks

Data set	Product category			r_i	e_i	b_i
	10 jobs	15 jobs	20 jobs			
BRdata	1	1,2	1,2,3	$[0, r_{max}]$	[50,100]	[2, 4]
BCdata	1	1,2	1,2,3	$[0, r_{max}]$	[50,100]	[2, 4]
DPdata	1	1,2	1,2,3	$[0, r_{max}]$	[50,100]	[2, 4]

6.2 Parameter settings

In this section, the Taguchi method is used to determine the values of important parameters in AMGWO. To compare the effectiveness of encoding methods, experiments are conducted on three different lot-sizing coding methods.

The main parameters of AMGWO are sub-swarm size (p_s), sub-swarm number (p_n), maximum regrouping interval (p_r^{max}), and neighborhood search interval (p_t). The number of elite individuals is set to $0.01 \times p_s$. The time limit factor is set to $T_l = 100$.

As shown in equation **Error! Reference source not found.**, the relative percent deviation (RPD) is used in this chapter to measure the results of AMGWO with different parameter configurations.

$$RPD = \frac{F - F_{best}}{F_{best}} \times 100, \tag{21}$$

where F denotes the makespan value obtained in a single run, and F_{best} denotes the best result of the algorithm for all parameter configurations.

Table 3. Levels of parameters

Parameters	Level			
	1	2	3	4
p_s	10	15	20	25
p_n	10	15	20	25
p_r^{max}	10	20	30	40
p_t	25	50	75	100

Table 4. Parameter combinations

No	Parameters			
	p_s	p_n	p_r^{max}	p_t
1	10	10	10	25
2	10	15	20	50
3	10	20	30	75
4	10	25	40	100
5	15	10	20	75
6	15	15	10	100
7	15	20	40	25
8	15	25	30	50
9	20	10	30	100
10	20	15	40	75
11	20	20	10	50
12	20	25	20	25
13	25	10	40	50
14	25	15	30	25
15	25	20	20	100
16	25	25	10	75

The specific parameter levels are shown in Table 5, and the orthogonal experimental table and results are shown in Table 6. Each parameter combination is run independently 10 times to reduce the impact of random factors on the experimental results. All instances are run, and averages are taken.

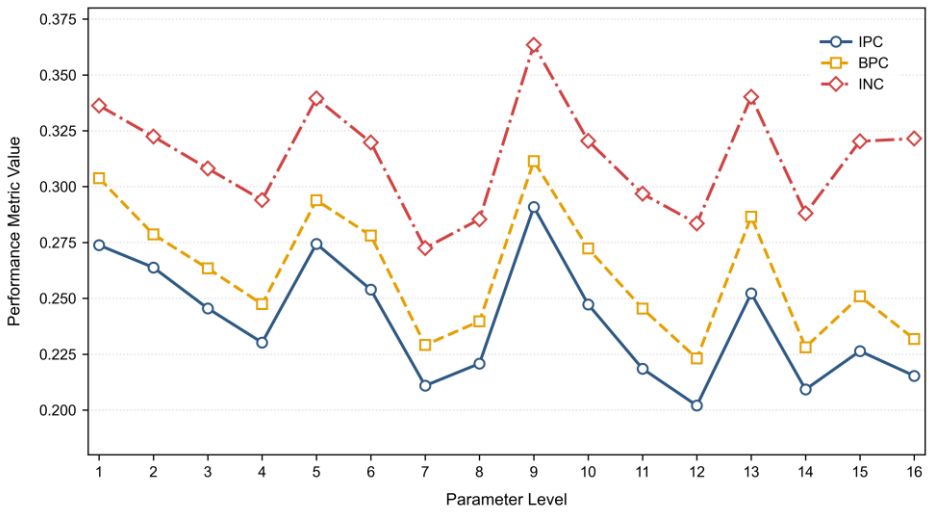


Figure 12. Experimental results of the Taguchi method under different encoding methods

Figure 15 shows the Average RPD (ARPD) results of the different encoding methods under 16 parameter combinations. In the figure, the horizontal axis represents parameter combination indices (corresponding to Table 6), while the vertical axis indicates ARPD values. The blue, orange, and yellow lines, respectively, denote three lot sizing encoding methods: IPC, BPC, and INC. Lower ARPD values indicate better algorithmic performance. It can be observed that the influence trends of different levels are consistent across all encoding methods. The blue line representing IPC consistently remains below other lines, demonstrating that the IPC method delivers optimal performance. According to Figure 15, when the parameter combination is set to $p_s = 20, p_n = 25, p_r^{max} = 20, p_t = 25$, the algorithm's performance is significantly better than other parameter levels. This parameter configuration will be adopted for all subsequent numerical experiments.

Figure 16 presents the main effects plot for key parameters of the AMGWO under the IPC method. In the figure, the horizontal axis represents the four-level values of four parameters, respectively, while the vertical axis indicates the ARPD values. For the line chart of the same parameter, a larger range indicates that the parameter has a greater influence on algorithm performance. It can be observed that p_s has the most significant impact on performance.

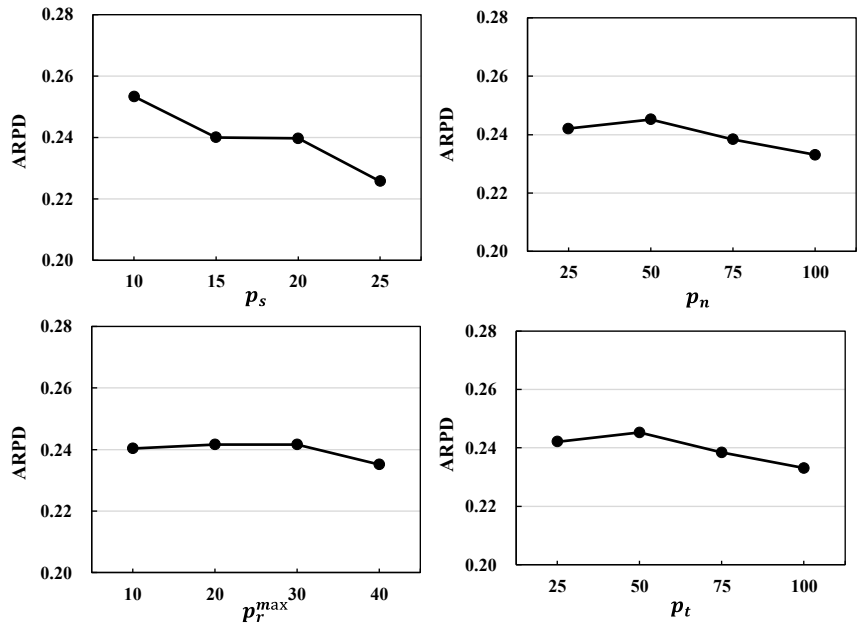


Figure 13. Main effects plot for the key parameters of the IPC method

6.3 Improvement analysis

To demonstrate the effectiveness of the improvement strategies, this section conducts comparative experiments using three algorithms with different configurations, as shown in Table 5. All algorithms employ the same encoding-decoding method, evolutionary operators.

The algorithm parameters are set as follows: lot sizing coding use IPC, $p_s = 20, p_n = 25, p_r^{max} = 20, p_t = 25$. The number of elite individuals is set to $0.01 \times p_s$. The time limit factor is set to $T_l = 100$.

Table 5. The configurations of comparison algorithms

Algorithm	Improvement strategy
GWO	basic GWO framework
GWO-I	GWO + adaptive dynamic regrouping
AMGWO	GWO-I + neighborhood search operators

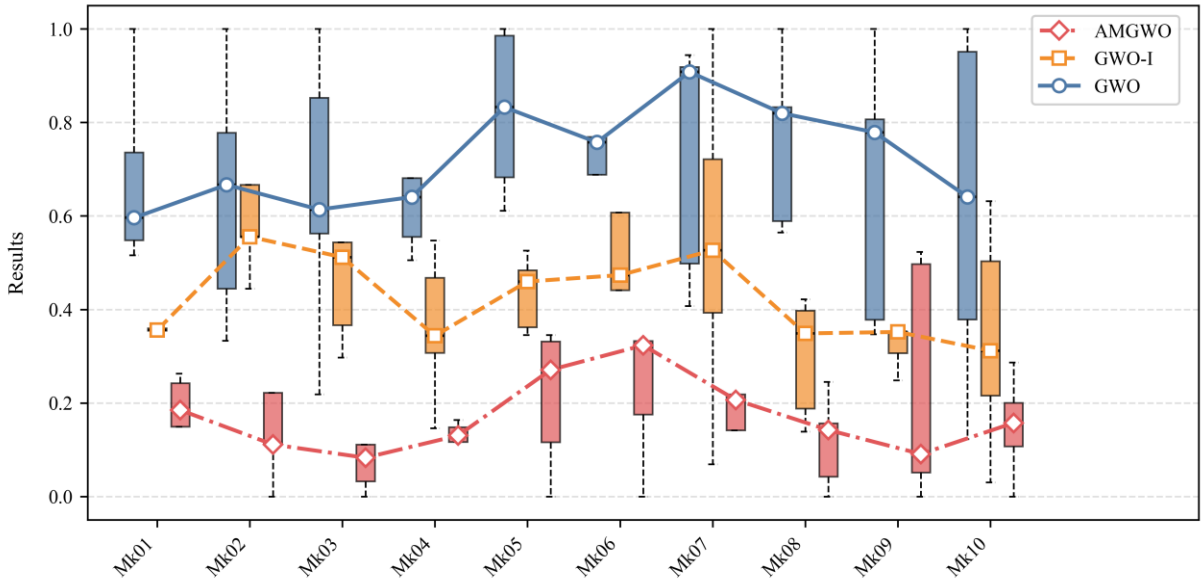


Figure 17. Box-line plot results of the improvement strategy analysis

Figure 17 presents the box-line plot results of the improvement strategy, validating the effectiveness of the proposed improvement strategies. The horizontal axis represents the 10 different cases, Mk01-Mk10, and the vertical axis represents the normalized values of the optimization objectives. The red, orange, and blue lines represent the box plots of the AMGWO, GWO-I, and GWO algorithms, respectively. Compared to GWO and GWO-I, GWO-I reduces the median of the makspan by 7.2% overall and narrows the interquartile range (IQR) by 23.5%. In comparison to GWO-I and AMGWO, AMGWO further optimizes the median by 4.8%, compresses the IQR by an additional 18.7%, and eliminates 75% of the outlier fluctuations. The median trajectory of AMGWO's box plot consistently lies below those of GWO and GWO-I (Mann-Whitney U test, $p < 0.01$), demonstrating comprehensive superiority across all 10 instances. These results confirm that the synergistic effect of adaptive dynamic regrouping and the proposed neighborhood search operators plays a critical role in suppressing random disturbances and accelerating convergence.

6.4 MILP validation

In this section, the performance of the MILP model is tested on 10 instances from the BRdata. The MILP model was solved by the Gurobi 10.0.3 software with a runtime limit of 3600s per instance. The AMGWO algorithm parameters are set as follows: lot sizing coding use IPC, $p_s = 20, p_n = 25, p_r^{max} = 20, p_t = 25$. The number of elite individuals is set to $0.01 \times p_s$. The time limit factor is set to $T_l = 100$.

As shown in Table 8, the MILP model can obtain relatively good solutions for some small-scale problems, which verifies the correctness of the proposed MILP formulation. However, the MILP exhibits suboptimal solution quality (with significant optimality gaps) or fails to find feasible solutions within the time limit as the problem scale increases. In contrast, AMGWO consistently achieves superior results compared to the MILP solver across all test instances. These findings demonstrate that the AMGWO is significantly more effective for solving large-scale problems.

Table 8. Solver validation and comparison

Instances	Size	MILP		AMGWO
		Best	Gap (%)	
Mk01	10 × 6	4889	57.52	**4573**
Mk02	10 × 6	2551	86.27	**2187**
Mk03	15 × 8	29287	97.21	**17709**
Mk04	15 × 8	8500	89.11	**5810**
Mk05	15 × 4	-	-	**13011**
Mk06	10 × 15	-	-	**6868**
Mk07	20 × 5	-	-	**15237**

Instances	Size	MILP		AMGWO
		Best	Gap (%)	
Mk08	20 × 10	-	-	**47319**
Mk09	20 × 10	-	-	**45974**
Mk10	20 × 15	-	-	**29321**

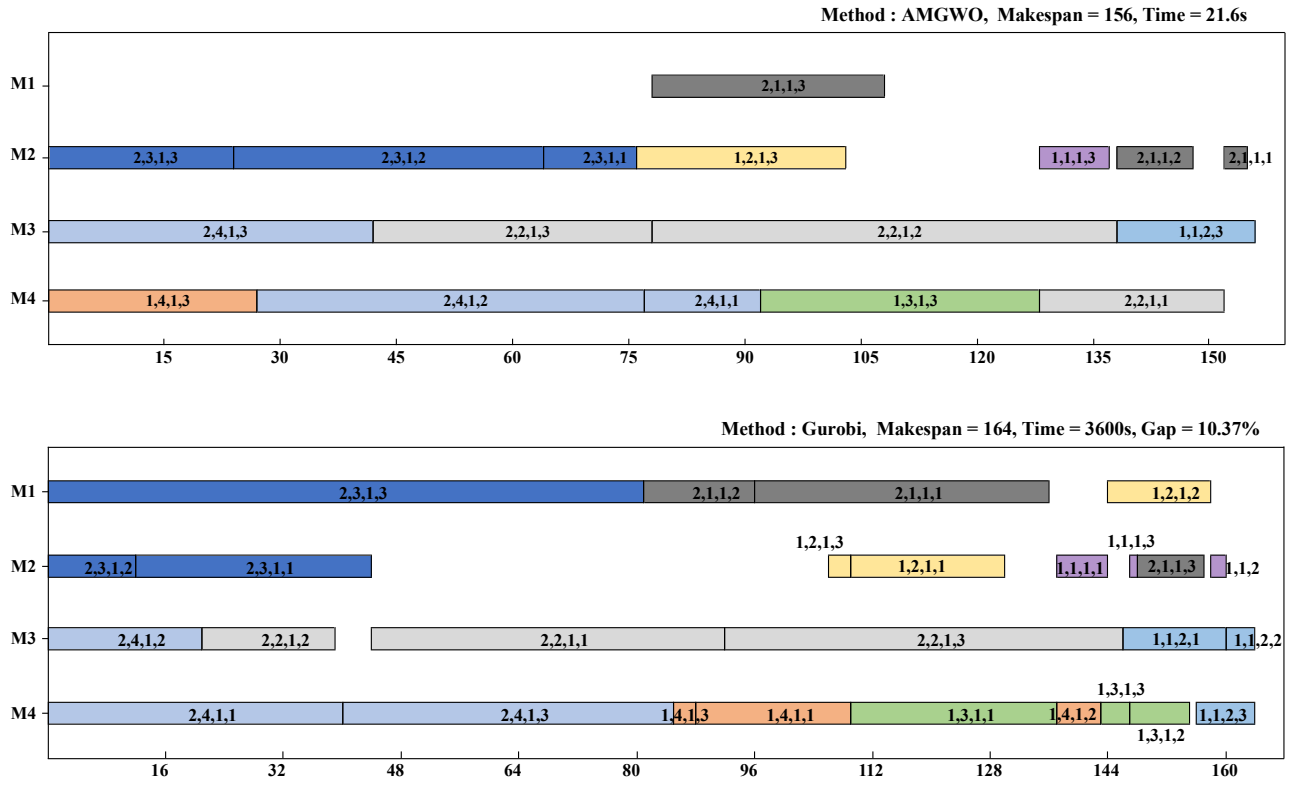


Figure 18. Gantt chart for the example

Due to the complexity of other cases, which leads to visually cluttered Gantt charts, only the Gantt chart for the example in Table 3 is presented in

Figure 18. The upper and lower parts, respectively, present the scheduling results obtained by the AMGWO and MILP models. In the Gantt chart, the notation (i, j, o, b) denotes sublots $\tau_{i,j,o,b}$. The width of the rectangle represents the processing time of the sublots on the machine. The MILP model solved by Gurobi yielded a solution of 164 (Gap = 10.37%) after 3600 seconds, while AMGWO achieved a superior result of 156 in merely 21.6 seconds.

6.5 Performance comparison

The AMGWO was tested on the generated three sets of benchmarks and compared with four other methods: (1) GA-VNS (S. Tian *et al*., 2024), which employs VNS as an improvement strategy within GA; (2) EMA (Tang *et al*., 2024), designed to solve the distributed FJSP with integrated sequencing flexibility; (3) SCEGWO (Z. Zhu *et al*., 2022), which integrates the concept of cellular automata (CA) into GWO by dividing the population into multiple sub-swarms.

Table 6 Parameter settings

Algorithm	Parameter settings
AMGWO	$p_s = 20, p_n = 25, p_r^{max} = 20, p_t = 25$
GA-VNS	population size: 500, crossover probability: 0.9, mutation probability: 0.2
SCEGWO	Grid (population) size: 20 × 25, cellular neighborhood structure: C21
EMA	population size: 500, crossover probability: 0.9, mutation probability: 0.2, local search probability: 0.5

The algorithm parameters are set as shown in Table 6. All algorithms use IPC for lot sizing coding. For the BRData and BCData, the time limit factor is set to $T_l = 100$. For the DPData, the time constraint factor was set to $T_l = 100$ for the small-scale instances (01a-12a) and $T_l = 200$ for the large-scale instances (13a-18a).

Table 7. Experimental results for BRData.

Instance	Size	GA-VNS		EMA		SCEGWO		AMGWO	
		Best	Agv	Best	Agv	Best	Agv	Best	Agv
Mk01	10 × 6	4863	5127.4	4894	5091.2	4600	4740.4	**4346**	**4503.2**
Mk02	10 × 6	2444	2546.8	2473	2554.6	2239	2267.0	**2161**	**2192.2**
Mk03	15 × 8	19372	20069.2	20470	21307.4	18673	19114.6	**17709**	**17909.6**
Mk04	15 × 8	6723	7063.8	6855	7043.6	6059	6306.2	**5810**	**5916.4**
Mk05	15 × 4	14140	14444.0	13938	14416.4	13089	13447.4	**12590**	**12850.8**
Mk06	10 × 15	7366	7872.0	7756	8014.4	6754	6929.0	**6588**	**6850.6**
Mk07	20 × 5	16551	17224.2	16275	17395.2	15497	16282.2	**14907**	**15231.8**
Mk08	20 × 10	51380	53767.8	48836	52217.0	47707	49204.8	**45435**	**46336.2**
Mk09	20 × 10	47598	49527.2	49520	51947.2	44412	45956.2	**43327**	**44565.4**
Mk10	20 × 15	29726	31356.6	29823	31678.4	28989	29599.4	**28545**	**29286.0**

Table 7, Table 8, and Table 9 present the comparative results for the BRData, BCData, and DPData benchmark sets, respectively. The results demonstrate that the proposed AMGWO achieves the best outcomes across all instances.

For both the GA-VNS and EMA, they adopt the same encoding-decoding methods, evolutionary operators, and neighborhood search operators. The primary distinction lies in their algorithmic frameworks. Comparative analysis of GA-VNS/EMA and AMGWO reveals that the proposed AMGWO framework demonstrates superiority in solving the FJSP-AD-LS.

The SCEGWO incorporates the adapted discrete GWO and CA, with its core mechanism designed to disrupt population structures and prevent convergence to local optima. While the proposed AMGWO shares similarities with SCEGWO in terms of algorithmic framework and conceptual foundation, a critical distinction lies in the integration of neighborhood search operators and math-heuristic algorithms within AMGWO. Specifically, AMGWO enhances search capability through these neighborhood search operators and a math-heuristic algorithm, whereas SCEGWO does not employ explicit neighborhood search mechanisms. This comparative analysis between SCEGWO and AMGWO further validates the efficacy of the proposed various search methods.

Figure 19 illustrates the convergence curves of representative instances (sampled at equal time intervals). In the figure, the text at the top indicates the instance name, the horizontal axis represents the sampling time, and the vertical axis represents the optimization objective value (makespan). The yellow, orange, gray, and blue lines represent the convergence curves of the AMGWO, EMA, SCEGWO, and GA-VNS algorithms, respectively. The comparison graph of algorithm convergence curves visually demonstrates the optimization performance of different algorithms, including convergence speed (the steepness of the curve decline), solution quality (the final objective value), and stability (the smoothness of the curves). Through comparison, the strengths and weaknesses of the algorithms can be evaluated. The results demonstrate that AMGWO and SCEGWO exhibit significantly superior performance compared to GA-VNS and EMA. During the initial optimization phase, GA-VNS, EMA, and SCEGWO all display rapid convergence characteristics. However, GA-VNS and EMA frequently fail due to premature convergence to local optima. While SCEGWO achieves fast early convergence through its efficient iteration mechanism, it lacks neighborhood search operators. In contrast, AMGWO employs an adapted dynamic regrouping strategy (instead of generation-by-generation regrouping). Although this design slows initial convergence due to its multi-population collaborative exploration mechanism, it substantially enhances search breadth. Notably, SCEGWO struggles to further optimize solutions in later stages due to the absence of local search capabilities. Conversely, AMGWO activates neighborhood search operators and matheuristic algorithms at the intermediate phase, enabling its convergence curve to continue descending in later stages, thereby demonstrating superior sustained optimization capability.

Table 8. Experimental results for BCData

Instance	**Size**	**GA-VNS**		**EMA**		**SCEGWO**		**AMGWO**	
		Best	**Agv**	**Best**	**Agv**	**Best**	**Agv**	**Best**	**Agv**
mt10c1	10 × 11	87413	92304.4	86035	92341.6	84135	86926.8	**80076**	**81887.6**
mt10cc	10 × 12	76123	79740.6	81447	82503.4	71759	73697.4	**69081**	**72932.4**
mt10x	10 × 11	80855	85819.6	84849	87303.4	75463	79354.6	**69023**	**71572.8**
mt10xx	10 × 12	134461	142742.8	133329	138451.6	133259	135908.0	**121211**	**122292.8**

Instance	Size	GA-VNS		EMA		SCEGWO		AMGWO	
		Best	**Agv**	**Best**	**Agv**	**Best**	**Agv**	**Best**	**Agv**
mt10xxx	10 × 13	98592	105103.0	102787	106997.0	92776	94852.2	**82176**	**85590.8**
mt10xy	10 × 12	95076	102123.0	94652	97821.4	87355	89948.4	**79916**	**83464.6**
mt10xyz	10 × 13	101775	111659.0	110385	116861.8	97795	103478.4	**91797**	**94333.4**
setb4c9	15 × 11	132457	142315.4	144260	145929.4	124716	131001.0	**121544**	**122887.4**
setb4cc	15 × 12	163081	166587.6	162254	167536.0	149322	153749.0	**139477**	**144576.8**
setb4x	15 × 11	118756	132891.0	131969	135312.6	123545	126595.4	**114147**	**117664.0**
setb4xx	15 × 12	147338	156454.4	152908	160115.8	145281	147735.4	**130126**	**133852.0**
setb4xxx	15 × 13	138641	142151.4	130416	141074.6	122451	128067.4	**110625**	**116512.0**
setb4xy	15 × 12	131208	138521.0	131196	138330.6	125932	130074.6	**116466**	**118923.6**
setb4xyz	15 × 13	170824	176812.0	176277	179947.2	156096	163032.2	**150082**	**152550.8**
seti5c12	15 × 16	156974	163595.2	155204	162053.8	142112	149367.8	**133389**	**136949.4**
seti5cc	15 × 17	154937	169082.8	154913	164009.0	149881	154204.0	**139122**	**143624.4**
seti5x	15 × 16	195770	201305.8	201185	204839.0	175940	183273.4	**168645**	**171241.0**
seti5xx	15 × 17	193481	200215.0	203468	206992.6	179772	184877.6	**165323**	**169531.4**
seti5xxx	15 × 18	202887	209793.8	200971	204934.6	180241	189997.6	**170357**	**175496.8**
seti5xy	15 × 17	149203	150294.6	143953	150455.0	128222	133506.4	**118574**	**123672.2**
seti5xyz	15 × 18	149544	153479.4	150564	156144.8	137775	141489.0	**127373**	**129447.2**

During the experimental analysis, we investigated the iterative behavior of the AMGWO and observed that its iteration count is significantly lower than that of comparative algorithms. This observation suggests that for large-scale cases, premature activation of neighborhood search operators may lead to excessive computational overhead in early stages, thereby reducing the effective number of iterations and ultimately degrading solution quality. To address this limitation, we propose three enhancement strategies: 1) Extending the algorithm's run time to allow deeper exploration; 2) Delaying the initiation of neighborhood search to prioritize global exploration in early phases; 3) Providing high-quality initial solutions for the MILP sub-models to accelerate convergence. These adjustments are expected to synergistically improve the algorithm’s optimization efficacy, particularly for complex instances.

Table 9. Experimental results for DPData

Instance	Size	GA-VNS		EMA		SCEGWO		AMGWO	
		Best	Agv	Best	Agv	Best	Agv	Best	Agv
01a	10 × 5	406999	423831.0	417942	429375.8	386915	397137.4	**368844**	**373353.8**
02a	10 × 5	364126	374240.4	367654	382968.6	357364	364720.2	**340825**	**349822.8**
03a	10 × 5	231276	237077.8	226688	234033.0	221158	227280.2	**212709**	**221785.8**
04a	10 × 5	304436	312681.8	305408	314764.8	288840	294628.0	**273800**	**282470.2**
05a	10 × 5	370104	377246.6	356041	378639.8	349508	356989.6	**332230**	**343487.4**
06a	10 × 5	299576	315478.2	311864	319831.8	299720	312884.4	**296600**	**301464.8**
07a	15 × 8	414763	432274.2	411092	422455.6	389139	394751.4	**348428**	**367224.6**
08a	15 × 8	253183	262026.6	253537	265901.6	237107	249392.4	**231756**	**242976.6**
09a	15 × 8	198154	203113.6	182147	192753.4	175621	183602.2	**167004**	**172501.6**
10a	15 × 8	243504	248341.2	237723	249719.8	220222	230597.4	**213896**	**220570.0**
11a	15 × 8	308179	320288.8	312085	322794.4	287287	301102.2	**280325**	**287810.4**
12a	15 × 8	321629	331285.4	321966	330132.6	301302	309725.4	**288844**	**295723.8**
13a	20 × 10	376820	398937.8	387744	397990.6	372746	378432.0	**359555**	**366873.4**
14a	20 × 10	316293	333604.6	318635	324130.2	283394	298007.8	**281182**	**288377.0**
15a	20 × 10	280856	288454.6	277579	286960.8	270310	278491.0	**265819**	**278782.8**
16a	20 × 10	350759	369899.8	366878	374714.8	350280	356658.2	**332768**	**337590.0**
17a	20 × 10	225075	242962.4	226183	239385.2	220071	223343.4	**219388**	**224752.6**
18a	20 × 10	208752	212259.0	206882	213324.2	188334	196124.6	**187102**	**191650.2**

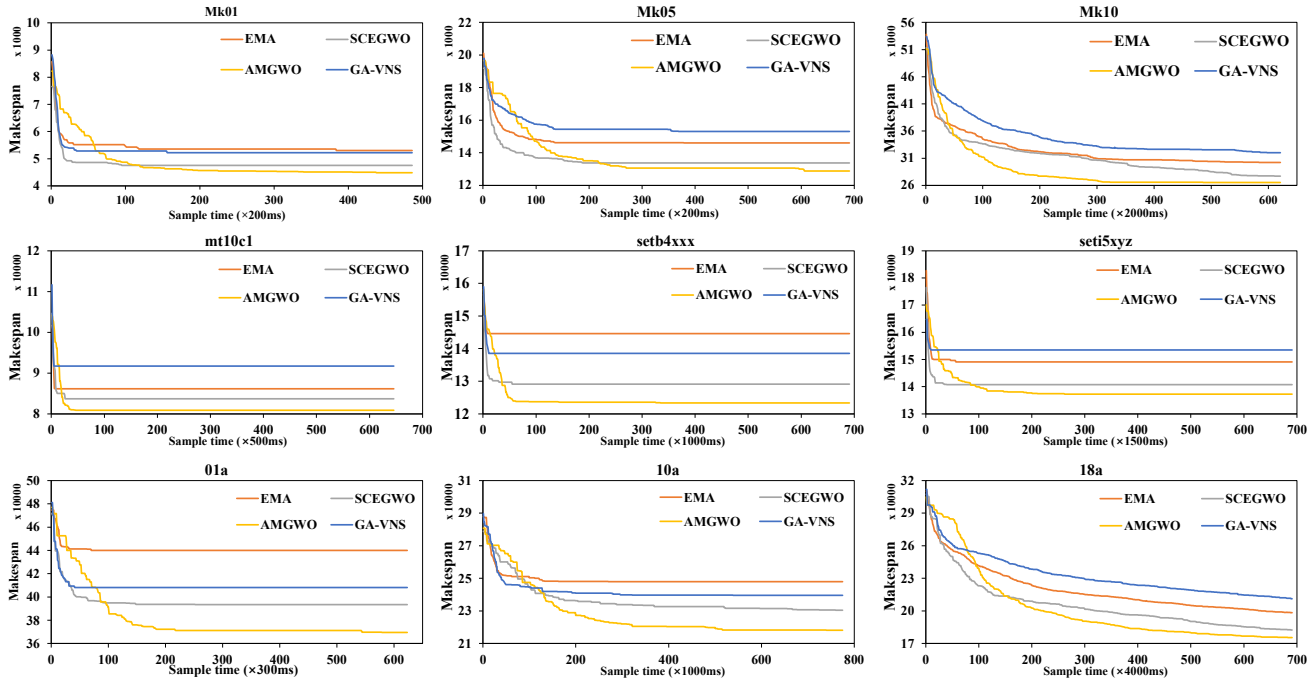


Figure 19. Convergence plots for partial instances of the comparison algorithms

7. CONCLUSIONS

This paper proposes a novel FJSP-AD-LS to minimize makespan. An AMGWO is proposed to solve this problem. In terms of the algorithmic framework, the specific contributions of the AMGWO are as follows: 1) A multi-swarm framework with dynamic regrouping to balance global exploration and local exploitation; 2) A variable neighborhood search (VNS) mechanism is integrated to enhance the exploration capability of the algorithm. In terms of FJSP-AD-LS, the specific contributions of the algorithm are as follows: 1) Three lot-sizing coding methods are proposed, and the effects will be subsequently compared in experiments. 2) A random pop-up coding based on multi-sublots trees is proposed. 3) Five crossover operators and six mutation operators are proposed for different coding. 4) In VNS, the new knowledge-based neighborhood search operators are proposed, and a hybrid math-heuristic algorithm is used to optimize lot-sizing.

We constructed 49 new instances based on the FJSP benchmarks and conducted a comprehensive evaluation of AMGWO through experiments. First, the algorithm parameters were optimized using the Taguchi method, and the effectiveness of the three proposed lot-sizing encoding methods was compared. The validity of each improvement strategy was further confirmed by modifying the algorithm's strategy configuration. Additionally, the validity of the proposed MILP model and the proposed algorithm was verified using an OR solver. Finally, comparisons with three other well-known algorithms revealed that AMGWO outperforms them.

In future research, the proposed model and methods can be extended to other types of manufacturing systems. Additionally, further studies can be conducted to improve the efficiency of the algorithm in solving large-scale problems of this nature.

In future research, more practical factors could be incorporated into FJSP-AD-LS to enhance its applicability in real-world production environments, such as worker skill heterogeneity, transportation resource constraints, and machine breakdowns. Furthermore, emerging techniques such as deep learning, reinforcement learning, and federated learning could be integrated into the optimization algorithm design to improve search efficiency. Additionally, for large-scale scheduling problems, distributed and parallel computing algorithms should be investigated, along with the development of novel optimization approaches to further enhance computational performance.

REFERENCES

Baykasoğlu, A. & Madenoğlu, F.S. (2021). Greedy randomized adaptive search procedure for simultaneous scheduling of production and preventive maintenance activities in dynamic flexible job shops. Soft Computing, 25(23), 14893-14932.

Bożek, A. & Werner, F. (2018). Flexible job shop scheduling with lot streaming and sublot size optimization. International Journal of Production Research, 56(19), 6391-6411.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. Annals of Operations Research, 41(3), 157-183.

Brucker, P. & Schlie, R. (1990). Job-shop scheduling with multi-purpose machines. Computing, 45(4), 369-375.

Chambers, J.B. & Barnes, J.W. (1996). Tabu search for the flexible-routing job shop problem. The University of Texas, Austin, TX, Technical Report Series ORP96-10, Graduate Program in Operations Research and Industrial Engineering, 1-11

Cheng, L., Tang, Q. & Zhang, L. (2024). Production costs and total completion time minimization for three-stage mixed-model assembly job shop scheduling with lot streaming and batch transfer. Engineering Applications of Artificial Intelligence, 130, 107729.

Dai, M., Tang, D., Giret, A. & Salido, M.A. (2019). Multi-objective optimization for energy-efficient flexible job shop scheduling problem with transportation constraints. Robotics and Computer-Integrated Manufacturing, 59, 143-157.

Daneshamooz, F., Fattahi, P. & Hosseini, S.M.H. (2022). Scheduling in a flexible job shop, followed by some parallel assembly stations considering lot streaming. Engineering Optimization, 54(4), 614-633.

Dauzère-Pérès, S. & Paulli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. Annals of Operations Research, 70(0), 281-306.

Defersha, F.M. & Bayat Movahed, S. (2018). Linear programming assisted (not embedded) genetic algorithm for flexible jobshop scheduling with lot streaming. Computers & Industrial Engineering, 117, 319-335.

Defersha, F.M. & Chen, M. (2010). A hybrid genetic algorithm for flowshop lot streaming with setups and variable sublots. International Journal of Production Research, 48(6), 1705-1726.

Defersha, F.M. & Chen, M. (2012). Jobshop lot streaming with routing flexibility, sequence-dependent setups, machine release dates, and lag time. International Journal of Production Research, 50(8), 2331-2352.

Ehm, F. (2024). Scheduling and process planning for the dismantling shop with flexible disassembly mode and recovery level. Computers & Industrial Engineering, 189, 109927.

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

Fan, J., Zhang, C., Tian, S., Shen, W. & Gao, L. (2024). Flexible job-shop scheduling problem with variable lot-sizing: An early release policy-based matheuristic. Computers & Industrial Engineering, 193, 110290.

Fang, K., Luo, W. & Che, A. (2021). Speed scaling in two-machine lot-streaming flow shops with consistent sublots. Journal of the Operational Research Society, 72(11), 2429-2441.

Fattahi, P., Rad, N., Daneshamooz, F. & Ahmadi, S. (2020). A new hybrid particle swarm optimization and parallel variable neighborhood search algorithm for flexible job shop scheduling with assembly process. Assembly Automation, ahead-of-print.

Ferraro, A., Rossit, D., Toncovich, A. & Frutos, M. (2019). Lot Streaming Flow Shop with a Heterogeneous Machine. Engineering Management Journal, 31(2), 113-126.

Gao, L. & Pan, Q.-K. (2016). A shuffled multi-swarm micro-migrating birds optimizer for a multi-resource-constrained flexible job shop scheduling problem. Information Sciences, 372, 655-676.

Garey, M.R., Johnson, D.S. & Sethi, R. (1976). The Complexity of Flowshop and Jobshop Scheduling. Mathematics of Operations Research, 1(2), 117-129

Ghandi, S. & Masehian, E. (2015). Review and taxonomies of assembly and disassembly path planning problems and approaches. Computer-Aided Design, 67-68, 58-86.

Gong, G., Tang, J., Huang, D., Luo, Q., Zhu, K. & Peng, N. (2024). Energy-efficient flexible job shop scheduling problem considering discrete operation sequence flexibility. Swarm and Evolutionary Computation, 84, 101421.

Guo, X., Zhou, M., Liu, S. & Qi, L. (2021). Multiresource-Constrained Selective Disassembly With Maximal Profit and Minimal Energy Consumption. IEEE Transactions on Automation Science and Engineering, 18(2), 804-816.

Gürsoy Yılmaz, B. & Faruk Yılmaz, Ö. (2022). Lot streaming in the hybrid flowshop scheduling problem by considering equal and consistent sublots under machine capability and limited waiting time constraint. Computers & Industrial Engineering, 173, 108745.

Han, Y., Gong, D., Jin, Y. & Pan, Q. (2019). Evolutionary Multi-objective Blocking Lot-Streaming Flow Shop Scheduling With Machine Breakdowns. IEEE Transactions on Cybernetics, 49(1), 184-197.

Han, Y., Li, J.Q., Gong, D. & Sang, H. (2019). Multi-Objective Migrating Birds Optimization Algorithm for Stochastic Lot-Streaming Flow Shop Scheduling With Blocking. IEEE Access, 7, 5946-5962.

Hidri, L. & Tlija, M. (2024). Efficient Algorithms for The Multi-Stage Flexible Flow Shop Scheduling Problem with Transportation and Unloading Times. International Journal of Industrial Engineering: Theory, Applications and Practice, 31(4).

Hu, Y., Zhang, L., Zhang, Z., Li, Z., & Tang, Q. (2024). Flexible assembly job shop scheduling problem considering reconfigurable machine: A cooperative co-evolutionary matheuristic algorithm. Applied Soft Computing, 166, 112148.

Huang, R.-H. & Yu, T.-H. (2017). An effective ant colony optimization algorithm for multi-objective job-shop scheduling with equal-size lot-splitting. Applied Soft Computing, 57, 642-656.

Hurink, J., Jurisch, B. & Thole, M. (1994). Tabu search for the job-shop scheduling problem with multi-purpose machines. Operations-Research-Spektrum, 15(4), 205-215.

Kalayci, C.B. & Gupta, S.M. (2013). Artificial bee colony algorithm for solving sequence-dependent disassembly line balancing problem. Expert Systems with Applications, 40(18), 7231-7241.

Kasapidis, G.A., Paraskevopoulos, D.C., Mourtos, I. & Repoussis, P.P. (2025). A unified solution framework for flexible job shop scheduling problems with multiple resource constraints. European Journal of Operational Research, 320(3), 479-495.

Kumar, S., Bagchi, T.P. & Sriskandarajah, C. (2000). Lot streaming and scheduling heuristics for m-machine no-wait flowshops. Computers & Industrial Engineering, 38(1), 149-172.

Lei, D. & Guo, X. (2013). Scheduling a job shop with lot streaming and transportation through a modified artificial bee colony. International Journal of Production Research, 51(16), 4930-4941.

Li, J.-q., Tao, X.-r., Jia, B.-x., Han, Y.-y., Liu, C., Duan, P., Zheng, Z.-x. & Sang, H.-y. (2020). Efficient multi-objective algorithm for the lot-streaming hybrid flowshop with variable sub-lots. Swarm and Evolutionary Computation, 52, 100600.

Li, L. (2022). Research on discrete intelligent workshop lot-streaming scheduling with variable sublots under engineer-to-order. Computers & Industrial Engineering, 165, 107928.

Li, N. & Feng, C. (2021). Research on Machining Workshop Batch Scheduling Incorporating the Completion Time and Non-Processing Energy Consumption Considering Product Structure. In: Energies (Vol. 14).

Li, X., Lu, J., Yang, C. & Wang, J. (2022). Research of the Flexible Assembly Job-Shop Batch–Scheduling Problem Based on Improved Artificial Bee Colony. Frontiers in Bioengineering and Biotechnology, 10.

Meng, T., Pan, Q.-K., Li, J.-Q. & Sang, H.-Y. (2018). An improved migrating birds optimization for an integrated lot-streaming flow shop scheduling problem. Swarm and Evolutionary Computation, 38, 64-78.

Mirjalili, S., Mirjalili, S.M. & Lewis, A. (2014). Grey Wolf Optimizer. Advances in Engineering Software, 69, 46-61.

Mortezaei, N. & Zulkifli, N. (2013). Integration of Lot Sizing and Flow Shop Scheduling with Lot Streaming. Journal of Applied Mathematics, 2013(1), 216595.

Mortezaei, N. & Zulkifli, N. (2014). A Study on Integration of Lot Sizing and Flow Shop Lot Streaming Problems. Arabian Journal for Science and Engineering, 39(12), 9283-9300.

Novas, J.M. (2019). Production scheduling and lot streaming at flexible job-shops environments using constraint programming. Computers & Industrial Engineering, 136, 252-264.

Pan, Q.-K. & Ruiz, R. (2012). An estimation of distribution algorithm for lot-streaming flow shop problems with setup times. Omega, 40(2), 166-180.

Pan, Q.-K., Wang, L., Gao, L. & Li, J. (2011). An effective shuffled frog-leaping algorithm for the IoT-streaming flow shop scheduling problem. The International Journal of Advanced Manufacturing Technology, 52(5), 699-713.

Piroozfard, H., Wong, K.Y., and Wong, W.P. (2018). Minimizing total carbon footprint and total late work criterion in flexible job shop scheduling by using an improved multi-objective genetic algorithm. Resources, Conservation and Recycling, 128, 267-283.

Qiao, L., Zhang, Z. & Huang, Z. (2021). A Scheduling Algorithm for Multi-Workshop Production Based on BOM and Process Route. Applied Sciences, 11(11).

Ren, W., Yan, Y., Hu, Y. & Guan, Y. (2022). Joint optimization for the dynamic flexible job-shop scheduling problem with transportation time and resource constraints. International Journal of Production Research, 60(18), 5675-5696.

Sang, H.-Y., Pan, Q.-K., Duan, P.-Y. & Li, J.-Q. (2018). An effective discrete invasive weed optimization algorithm for lot-streaming flowshop scheduling problems. Journal of Intelligent Manufacturing, 29(6), 1337-1349.

Singh, M.R., Singh, M., Mahapatra, S.S., and Jagadev, N. (2016). Particle swarm optimization algorithm embedded with maximum deviation theory for solving the multi-objective flexible job shop scheduling problem. The International Journal of Advanced Manufacturing Technology, 85(9), 2353-2366.

Sun, J., Zhang, G., Lu, J. & Zhang, W. (2021). A hybrid many-objective evolutionary algorithm for the flexible job-shop scheduling problem with transportation and setup times. Computers & Operations Research, 132, 105263.

Tang, J., Gong, G., Peng, N., Zhu, K., Huang, D., andLuo, Q. (2024). An effective memetic algorithm for the distributed flexible job shop scheduling problem considering integrated sequencing flexibility. Expert Systems with Applications, 242, 122734.

Tian, S., Zhang, C., Fan, J., Li, X. & Gao, L. (2024). A genetic algorithm with critical path-based variable neighborhood search for the distributed assembly job shop scheduling problem. Swarm and Evolutionary Computation, 85, 101485.

Tian, Z., Jiang, X., Tian, G., Li, Z., & Liu, W. (2024). Knowledge-Based Lot-Splitting Optimization Method for Flexible Job Shops Considering Energy Consumption. IEEE Transactions on Automation Science and Engineering, 21(3), 4864-4875.

Trietsch, D. & Baker, K.R. (1993). Basic Techniques for Lot Streaming. Operations Research, 41(6), 1065-1076

Wang, H.-y., Zhao, F., Gao, H.-m. & Sutherland, J.W. (2019). A three-stage method with efficient calculation for lot streaming flow-shop scheduling. Frontiers of Information Technology & Electronic Engineering, 20(7), 1002-1020.

Wocker, M.M., Ostermeier, F.F., Wanninger, T., Zwinkau, R. & Deuse, J. (2024). Flexible job shop scheduling with preventive maintenance consideration. Journal of Intelligent Manufacturing, 35(4), 1517-1539.

Xie, F., Li, L., Li, L., Huang, Y. & He, Z. (2023). A decomposition-based multi-objective Jaya algorithm for lot-streaming job shop scheduling with variable sublots and intermingling settings. Expert Systems with Applications, 228, 120402.

Xie, J., Li, X., Gao, L. & Gui, L. (2023). A hybrid genetic tabu search algorithm for distributed flexible job shop scheduling problems. Journal of Manufacturing Systems, 71, 82-94.

Xiuli, W., Junjian, P., Zirun, X., Ning, Z., & Shaomin, W. (2021). An improved multi-objective optimization algorithm for solving flexible job shop scheduling problem with variable batches. Journal of Systems Engineering and Electronics, 32(2), 272-285.

Xu, J. & Nagi, R. (2013). Solving assembly scheduling problems with tree-structure precedence constraints: A Lagrangian relaxation approach. IEEE Transactions on Automation Science and Engineering, 10(3), 757-771.

Yunusoglu, P. & Topaloglu Yildiz, S. (2023). Solving the flexible job shop scheduling and lot streaming problem with setup and transport resource constraints. International Journal of Systems Science: Operations & Logistics, 10(1), 2221072.

Zhang, B., Pan, Q.-k., Gao, L., Zhang, X.-l., Sang, H.-y. & Li, J.-q. (2017). An effective modified migrating birds optimization for hybrid flowshop scheduling problem with IoT streaming. Applied Soft Computing, 52, 14-27.

Zhang, B., Pan, Q.-k., Meng, L.-l., Lu, C., Mou, J.-h. & Li, J.-q. (2022). An automatic multi-objective evolutionary algorithm for the hybrid flowshop scheduling problem with consistent sublots. Knowledge-Based Systems, 238, 107819.

Zhang, B., Pan, Q.-K., Meng, L.-L., Zhang, X.-L., Ren, Y.-P., Li, J.-Q. & Jiang, X.-C. (2021). A collaborative variable neighborhood descent algorithm for the hybrid flowshop scheduling problem with consistent sublots. Applied Soft Computing, 106, 107305.

Zhang, S., Li, X., Zhang, B. & Wang, S. (2020). Multi-objective optimization in flexible assembly job shop scheduling using a distributed ant colony system. European Journal of Operational Research, 283(2), 441-460.

Zhang, Z., Fu, Y., Gao, K., Pan, Q. & Huang, M. (2024). A learning-driven multi-objective cooperative artificial bee colony algorithm for distributed flexible job shop scheduling problems with preventive maintenance and transportation operations. Computers & Industrial Engineering, 196, 110484.

Zhu, Y., Tang, Q., Cheng, L., Zhao, L., Jiang, G. & Lu, Y. (2024). Solving multi-objective hybrid flowshop lot-streaming scheduling with consistent and limited sub-lots via a knowledge-based memetic algorithm. Journal of Manufacturing Systems, 73, 106-125.

Zhu, Z., Zhou, X., Cao, D. & Li, M. (2022). A shuffled cellular evolutionary grey wolf optimizer for the flexible job shop scheduling problem with tree-structured job precedence constraints. Applied Soft Computing, 125, 109235.