Computers, Materials & Continua

Tech Science Press

Doi:10.32604/cmc.2025.063944

ARTICLE

An Adaptive Cooperated Shuffled Frog-Leaping Algorithm for Parallel Batch Processing Machines Scheduling in Fabric Dyeing Processes

Lianqiang Wu, Deming Lei* and Yutong Cai

School of Automation, Wuhan University of Technology, Wuhan, 430070, China

*Corresponding Author: Deming Lei. Email: demingleil1@163.com

Received: 29 January 2025; Accepted: 10 March 2025; Published: 16 April 2025

ABSTRACT: Fabric dyeing is a critical production process in the clothing industry and heavily relies on batch processing machines (BPM). In this study, the parallel BPM scheduling problem with machine eligibility in fabric dyeing is considered, and an adaptive cooperated shuffled frog-leaping algorithm (ACSFLA) is proposed to minimize makespan and total tardiness simultaneously. ACSFLA determines the search times for each memeplex based on its quality, with more searches in high-quality memeplexes. An adaptive cooperated and diversified search mechanism is applied, dynamically adjusting search strategies for each memeplex based on their dominance relationships and quality. During the cooperated search, ACSFLA uses a segmented and dynamic targeted search approach, while in non-cooperated scenarios, the search focuses on local search around superior solutions to improve efficiency. Furthermore, ACSFLA employs adaptive population division and partial population shuffling strategies. Through these strategies, memeplexes with low evolutionary potential are selected for reconstruction in the next generation, while those with high evolutionary potential are retained to continue their evolution. To evaluate the performance of ACSFLA, comparative experiments were conducted using ACSFLA, SFLA, ASFLA, MOABC, and NSGA-CC in 90 instances. The computational results reveal that ACSFLA outperforms the other algorithms in 78 of the 90 test cases, highlighting its advantages in solving the parallel BPM scheduling problem with machine eligibility.

KEYWORDS: Batch processing machine; parallel machine scheduling; shuffled frog-leaping algorithm; fabric dyeing process; machine eligibility

1 Introduction

Batch processing machine (BPM) has been widely applied in many real applications such as fabric dyeing [1], chemical production [2], tire manufacturing [3], steel smelting [4], semiconductor manufacturing [5]. BPMs are capable of simultaneously processing multiple jobs within a single batch, where all jobs in the batch share the same start and end times. Scheduling problems involving BPM can be broadly divided into single BPM scheduling [6,7], parallel BPM scheduling, and other BPM problems [8,9]. Parallel BPM scheduling, as a representative type, is an extension of the parallel machine scheduling problem and has attracted extensive research attention in recent years.

Table 1 summarizes relevant research. In Scenario 1, researchers use different algorithms to solve BPM-related scheduling problems in various industries, aiming at different objectives. Scenario 2 focuses on the shuffled frog-leaping algorithm (SFLA) and its applications in scheduling problems. Scenario 3 covers other multi-objective scheduling problems solved by different algorithms. However, parallel BPM scheduling in fabric dyeing still has many unsolved problems.

Table 1: Summary of related studies

Scenario	Reference	Objectives	Approaches
1	Zhang et al. [1]	Makespan, total tardiness	MO-ABC
	Jing Wang et al. [10]	Makespan	AABC
	Abedi et al. [11]	Makespan, total tardiness	NSGA-II and MOICA
	Zhang et al. [12]	Total service completion time	IPSO
	Zhou et al. [13]	Makespan, total electricity cost	MODDE
	Xue et al. [14]	Makespan	MPGA
	Arroyo et al. [15]	Makespan	IG
2	Cai et al. [16]	TAI, makespan, TEC	CSFLA
	Kong et al. [17]	Makespan	SFLA-VNS
	Yang et al. [18]	Makespan	GSFLA
	Lei et al. [19]	Makespan, number of tardy jobs	MGSFLA
3	Li et al. [20]	Maximum lateness, TC	NSGA-CC
	Wang et al. [21]	Makespan, TEC	CMOEA/I
	Wang et al. [22]	Makespan, TEC	RLMA
	Wu et al. [23]	Makespan, energy consumption	Evolutionary algorithm

The parallel BPM scheduling problem in fabric dyeing is crucial for real-world production. Efficient BPM scheduling in fabric dyeing can shorten production time, reduce costs, and improve resource utilization. However currently, this field faces several severe challenges. For example, job family compatibility is a major hurdle. In fabric dyeing, jobs with different color-dyeing needs, such as those from different job families, can't be processed in the same batch. Also, machine eligibility matters. Each machine has its capabilities, so not all machines can handle every job. These constraints make the problem more complex than traditional models, and existing algorithms often can't handle them well. Another challenge is the difficulty of multi-objective optimization. Simultaneously minimizing makespan and total tardiness in fabric dyeing parallel BPM scheduling is very hard. In actual production, achieving both objectives is crucial for efficient operations and customer satisfaction. However, most current research only focuses on one of these goals. For example, algorithms that prioritize reducing makespan often result in numerous jobs being tardy.

Because of the limitations of existing algorithms in dealing with the complex constraints and multi-objective requirements of parallel BPM scheduling in fabric dyeing, a new algorithm is urgently needed. A well-designed meta-heuristic can comprehensively explore the large solution space, considering both practical constraints and multi-objective optimization.

The shuffled frog-leaping algorithm (SFLA), inspired by the foraging behavior of frogs, is a good choice for developing a new algorithm. SFLA has unique advantages as it combines local search and global information exchange effectively. In scheduling problems, local search helps it explore the area around promising solutions to find better-optimized ones. Global information exchange enables sharing good solutions across different parts of the solution space, preventing the algorithm from getting stuck in local optima. SFLA has been applied to similar BPM-related scheduling problems and has achieved good results through techniques like reinforcement learning, cooperation, and memeplex grouping. These successful applications show its potential for the parallel BPM scheduling problem in fabric dyeing.

This study focuses on the parallel BPM scheduling problem in fabric dyeing and aims to develop an enhanced SFLA to minimize makespan and total tardiness at the same time. The main contributions are as follows.

(1) The parallel BPM scheduling problem with machine eligibility refinement from the fabric dyeing process is solved.
(2) An Adaptive Cooperated Shuffled Frog-Leaping Algorithm (ACSFLA) is proposed, featuring two key strategies: an adaptive population division strategy, which ensures the uninterrupted evolution of high-potential memeplexes, and a diversified search based on adaptive cooperation, which balances global exploration and local exploitation according to dominance relationships.
(3) The performance of ACSFLA is evaluated through experiments, which show that the new strategies are effective and that ACSFLA offers significant advantages in solving the considered problem.

The rest of the paper is organized as follows. Section 2 describes the parallel BPM scheduling problem defined by the fabric dyeing process. Section 3 introduces the SFLA. Section 4 presents the ACSFLA for the considered problem in the fabric dyeing process. The computational experiments are shown in Section 5. In the last section, conclusions are drawn, and potential topics for future research are discussed.

2 Problem Description

Zhang et al. [1] developed a Mixed-Integer Linear Programming (MILP) model for parallel BPM scheduling based on a real fabric dyeing process, which is described as follows. There are n jobs $J_1, J_2, ..., J_n$ waiting to be processed by m parallel BPMs $M_1, M_2, ..., M_m$. For each job J_i, it has a weight of w_i, a due date of d_i, a set Θ_i of eligible machines where $\Theta_i \subseteq \{M_1, M_2, ..., M_m\}$, and belongs to the family fa_i. Meanwhile, each BPM M_i has a volume of v_k, representing its weight capacity. The notations and descriptions for MILP are listed in Table 2.

Table 2: Notations and descriptions

Notation	Description	Notation	Description
a, b, i, j, g, h	Indexes	n	The number of jobs
J	Jobs	m	The number of BPMs
M	Batch processing machine	w	The weight of job
d	The due date of job	f	The family
Θ	Eligible machine set	v	The volume of BPM
p	Processing time	s	Pre-specified setup time
C_{max}	Makespan	TT	Total tardiness
$[\pi_1, \pi_2, \ldots, \pi_n]$	Scheduling string	$[\theta_1, \theta_2, \ldots, \theta_n]$	Machine assignment string

All jobs are classified into F families according to the required dyeing color. Jobs within the same family share the same processing time, with p_g denoting the processing time of each job in family f_g. Families are not compatible with each other, meaning jobs from different families cannot be processed simultaneously in a batch. Each bach B_h is composed of jobs from the same family. When forming a batch B_h on M_k with jobs form f_g, job J_i can be added to B_h if $w_i + \sum_{j \in B_h} w_j \leq v_k$. Jobs in the same batch share start and end times, with batch processing time equal to that of the job family. When a machine M_k switches from family f_a to a different family f_b, a pre-specified setup time s_{ab} is needed for cleaning.

There are some constraints on jobs and machines:

Each BPM can only process one batch at a time.

No job can be processed in distinct batches across multiple BPMs.

The processing cannot be interrupted.

All BPMs and jobs are accessible at all times.

Solving the BPMs scheduling problem requires addressing three sub-problems: (1) batch formation determines which jobs are grouped to form each batch; (2) BPM assignment selects an M_k for each batch; and (3) batch scheduling defines the processing sequence of all batches on each M_k.

Suppose that batches $B_1, B_2, ..., B_{uk}$ are processed on M_k consecutively. The completion time C_i of job J_i within these batches can be calculated as Eq. (1).

$$C_i = p_{f_{B_1}} + \sum_{a=2}^{h} s_{f_{B_{a-1}} f_{B_a}} + p_{f_{B_a}} \tag{1}$$

where f_{B_1} represents the family of batch B_a, which equal to f_{a_i}, $i \in B_a$.

The goal of the problem is to simultaneously minimize the maximum completion time and total tardiness of all jobs, whilst satisfying all constraints. The makespan C_{max} is defined as Eq. (2). The total tardiness TT is defined as Eq. (3).

$$\text{Minimize } C_{max} = \max\{C_i \,|\, i = 1, 2, ..., n\} \tag{2}$$

$$\text{Minimize } TT = \sum_{i=1}^{n} \max\{C_i - d_i, 0\} \tag{3}$$

The MILP model proposed by Zhang et al. [1] can be directly applied once the total finishing time of all machines is replaced by the maximum finishing time of all machines, which is equal to C_{max}.

For the problem with $C_{\max}$ and TT, $x \succ y$ means that x dominates y if $C_{\max}(x) \leq C_{\max}(y)$ and $TT(x) \leq TT(y)$, with either $C_{\max}(x) < C_{\max}(y)$ or $TT(x) < TT(y)$. If neither $x \succ y$ nor $y \succ x$ holds, then x and y are non-dominated.

3 Introduction to SFLA

The SFLA [24] is a meta-heuristic based on the behavior of frogs. In SFLA, each solution represents the position of a frog, and the population of possible solutions is modeled as a set of virtual frogs. After generating the initial population P, the algorithm iteratively performs population division, memeplex search, and population shuffling until the stopping criterion is satisfied.

Population division is shown below. All solutions are sorted in the descending order of fitness, suppose that $Fit_1 \geq Fit_2 \geq ... \geq Fit_N$, then assign x_k, $k = 1, 2, ..., N$ into memeplex $k(mod\ s) + 1$, s memeplexes $\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_s$ are finally obtained, where $k(mod\ s)$ means the remainder of k/s and Fit_i is the fitness of solution x_i.

The search process in memeplex $\mathcal{M}_i$ is described below. x_w is used as an optimization object, repeat the following steps μ times: a new solution x'_w is produced by Eq. (4) with x_w and x_b, if x'_w is better than x_w, then replace x_w with x'_w; otherwise, x_w and $gbest$ are used to generate a solution x'_w becomes the new x'_w by Eq. (5); otherwise, replace x_w with a random solution directly, where x_w, x_b and $gbest$ are the worst solution and best solution and best solution in memeplex $\mathcal{M}_i$ and the best solution of P:

$$x'_w = x_w + rand \times (x_b - x_w), \tag{4}$$

$$x'_w = x_w + rand \times (gbest - x_w), \tag{5}$$

where *rand* is a random real number following uniform distribution in $[0,1]$.

A new population P is constructed by shuffling all evolved memeplexes.

4 The Proposed ACSFLA

In ACSFLA, both the quality of the memeplex and its evolution quality are considered to determine the shuffling pool for partial population shuffling and the diverse search process of the memeplex. Meanwhile cooperated search is conducted within the diversified search processes, which is determined by the degree of dominance between the two memeplexes. The detailed steps of ACSFLA are shown below.

4.1 Population Initialization and Adaptive Division

The solution representation and decoding process follow the approach [25], which is described as follows. For the BPM scheduling problem with n jobs, m BPMs, and F families, use two-string representation. A scheduling string $[\pi_1, \pi_2, \ldots, \pi_n]$ and a machine assignment string $[\theta_1, \theta_2, \ldots, \theta_n]$ for batches, where $\pi_i \in \{1, 2, \ldots, n\}$ and $\theta_i \in \{1, 2, \ldots, m\}$. The decoding process, based on the scheduling string sequence and machine capacity constraints, repeatedly assigns jobs to batches on machines until all jobs are assigned. Finally, the formed batches are processed on each machine in the first formed first processing rule.

The population initialization process generates an initial population P consists of N solutions, as follows: $N/2$ solutions are randomly generated to ensure diversity, while the remaining $N/2$ solutions are produced using a heuristic method [25].

ACSFLA introduces an adaptive population division strategy. This strategy can adjust the population division targets based on $\tilde{P}$ generated in Section 4.3. The adaptive division process is as follows:

(1) If $gen = 1$, set $\tilde{P} = P$, $\Gamma = 1, 2, \ldots, s$
(2) Let $i = 1$
(3) If $i \in \Gamma$, set $\mathcal{M}_i$ be empty, then repeat the following steps N/s times: randomly select $x, y \in \tilde{P}$, if $x \succ y$, add x to $\mathcal{M}_i$; if $y \succ x$, add y to $\mathcal{M}_i$; if both $x \nsucc y$ and $y \nsucc x$, randomly select one of x, y to add into $\mathcal{M}_i$
(4) $i = i + 1$, if $i < s$, then go to step(3).

The quality Mq of a memeplex quantifies its overall performance and is calculated as follows:

$$Mq_i = \sum_{x \in \mathcal{M}_i} |y \in P \mid x \succ y| \tag{6}$$

After calculating the quality Mq for each memeplex, the s memeplexes are sorted in descending order based on their Mq values, such that $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_s$ satisfy the condition $Mq_1 > Mq_2 > \cdots > Mq_s$.

4.2 Adaptive Cooperated and Diversified Search

Unlike existing SFLA [26,27] and MGFLA [19], which apply the same process to all memeplexes or uniformly conduct cooperated searches, ACSFLA dynamically determines whether memeplexes cooperate based on their dominance relationship and adjust search strategies accordingly. This method emphasizes the balance of exploration and exploitation, executing cooperation when there is a significant advantage, to enhance solution quality and maintain population diversity. The diversified search processes are listed as follows:

(1) For the best memeplex $\mathcal{M}_1$:
If $C(\mathcal{M}_1, \mathcal{M}_s) \geq \gamma_2$, execute the following steps:
(i) Let $l = 1$, $\tau = 0$

(ii) If $\tau = 0$, randomly choose a solution $x \in \mathcal{M}_1$ and a solution $y \in \mathcal{M}_s$
(iii) Perform $\left[10 - \delta_i^{gen}\right]$ times global search between x and y
(iv) Perform δ_i^{gen} times local search on x
(v) $l = l + 1$, if $l < \mu_1/\mu$, go to step(ii).

Else if $C(\mathcal{M}_1, \mathcal{M}_s) < \gamma_2$, execute the following steps:

(i) Let $j = 1$ and decide a non-dominated solution $x_b \in \mathcal{M}_1$
(ii) Randomly choose a solution $y \in \mathcal{M}_1$
(iii) Perform $\left[10 - \delta_i^{gen}\right]$ times global search between x_b and y
(iv) Perform δ_i^{gen} times local search on x_b
(v) Perform δ_i^{gen} times local search on y
(vi) $j = j + 1$, if $j < \mu_1$, go to step(ii).

(2) For the worst memeplex $\mathcal{M}_s$:

If $C(\mathcal{M}_1, \mathcal{M}_s) \geq \gamma_2$, execute the following steps:

(i) Let $l = 1$, $\tau = 0$ and decide a non-dominated solution $x_b \in \mathcal{M}_s$
(ii) If $\tau = 0$, randomly choose a solution $y \in \mathcal{M}_1$
(iii) Perform $\left[10 - \delta_i^{gen}\right]$ times global search between x_b and y
(iv) Perform δ_i^{gen} times local search on x_b
(v) $l = l + 1$, if $l < \mu_s/\mu$, go to step(ii).

Else if $C(\mathcal{M}_1, \mathcal{M}_s) < \gamma_2$, execute the following steps:

(i) Let $j = 1$ and decide a non-dominated solution $x_b \in \mathcal{M}_s$
(ii) Randomly choose a solution $y \in \mathcal{M}_1$
(iii) Perform $\left[10 - \delta_i^{gen}\right]$ times global search between x_b and y
(iv) Perform δ_i^{gen} times local search on x_b
(v) $j = j + 1$, if $j < \mu_s$, go to step(ii).

(3) For each memeplex $\mathcal{M}_r, 1 < r < s$, repeat the following steps μ_r times:

(i) Decide a non-dominated solution $x_b \in \mathcal{M}_r$. Randomly choose a solution $y_1 \in \mathcal{M}_1$ and a solution $y_2 \in \mathcal{M}_s$
(ii) Perform $\left[10 - \delta_i^{gen}\right]$ times global search between x_b and y_1
(iii) Perform $\left[10 - \delta_i^{gen}\right]$ times global search between x_b and y_2
(iv) Perform δ_i^{gen} times local search on x_b.

Metric C [28] is employed to describe the dominance relationship between two solution sets. Specifically, $C(A, B)$ represents the proportion of solutions in set B that are dominated by at least one solution in set A. When $C(A, B) = 0$, it means that none of the solutions in set B are dominated by any solution in set A. Conversely, $C(A, B) = 1$ indicates that set A completely dominates set B. μ_i $(1 \leq i \leq s)$ indicates the search times of the memeplex $\mathcal{M}_i$, which is decided on the quality Mq_i. δ_i^{gen} is a dynamic factor that changes with iterations. It determines whether the memeplex search in generation gen is inclined towards global search or local search, based on the quality Mq_i and evolution quality Mo_i^{gen}.

$C(A, B)$, μ_i and δ_i^{gen} are computed as follows:

$$C(A, B) = \frac{\left|\{b \in B : \exists a \in A, a \succ b\}\right|}{|B|} \tag{7}$$

$$\mu_i = \left[5 \cdot \frac{Mq_i - Mq_{\min}}{Mq_{\max} - Mq_{\min}} + 5\right] \cdot \mu \tag{8}$$

$$\delta_i^{gen} = \left[5 \cdot \left(\frac{Mq_i - Mq_{\min}}{Mq_{\max} - Mq_{\min}} - \frac{Mo_i^{gen-1} - Mo_{\min}^{gen-1}}{Mo_{\max}^{gen-1} - Mo_{\min}^{gen-1}} \right) \right] + 5 \tag{9}$$

where μ is the basic search times.

There are 3 global search operators and 6 local search operators used in the search process. GS_1 replaces the machines between positions k_1 and k_2 in the machine assignment string of x with those from y. GS_2 applies the order crossover to the scheduling strings of x and y. GS_3 executes GS_1 and GS_2 sequentially. NS_1 and NS_2 are insertion operators for the scheduling and machine assignment strings. NS_3 and NS_4 are swap operators for the scheduling and machine assignment strings. NS_5 and NS_6 are inversion operators for the scheduling and machine assignment strings.

In the global search between x and y, randomly select a global search operator GS_i ($i \in 1, 2, 3$) to generate a new solution x_{new}. If $x_{new} \succ x$, then set $x = x_{new}$ and $\tau = 1$; if $x_{new} \succ y$, set $y = x_{new}$; if both $x_{new} \nsucc x$ and $x_{new} \nsucc y$, update Ω with x_{new}.

In the local search on x, randomly choose a local search operators NS_i ($i \in 1, 2, 3, 4, 5, 6$) to generate a new solution x_{new}. If $x_{new} \succ x$, $x = x_{new}$, $\tau = 1$, otherwise update Ω with x_{new}.

Initially, Ω consists of N/s randomly chosen solutions from the initial population Pop, and it is updated with the solution x in the following way: non-dominated sorting is performed on Ω after adding x into it. Then, the solution with the largest $rank_x$ and smallest crowding distance is removed from Ω.

4.3 Partial Population Shuffling

Partial Population Shuffling is a key step in ACSFLA, aimed at dynamically constructing the division pool $\bar{P}$. The process evaluates the evolutionary quality Mo of each memeplex. Memeplexes with low evolutionary potential are selected to $\bar{P}$ for reconstruction in the next iteration. Memeplexes with high evolutionary potential are retained to continue their evolution. The partial population shuffling process is as follows:

(1) Let $i = 1$, Γ and $\bar{P}$ be empty
(2) Compute Mo_i^{gen}
(3) If $Mo_i^{gen} < \gamma_1$ and $Mo_i^{gen} - Mo_i^{gen-1} < 0$, add i to Γ and add $\mathcal{M}_i$ to $\bar{P}$
(4) $i = i + 1$, if $i < s$, then go to step(2).

where Mo_i^{gen-1} indicates the evolution quality of the memeplex $\mathcal{M}_i$ in the latest generation. The set Γ stores the indices of memeplexes added to $\bar{P}$, which are used in the next iteration to decide if memeplex need to be reconstructed. Mo_i^{gen} is calculated as follows:

$$Mo_i^{gen} = \frac{\lambda_i^{gen}}{\mu_i} \tag{10}$$

where λ_i^{gen} represents the updated times of $\mathcal{M}_i$ during the memeplex search. Each time the optimization object of $\mathcal{M}_i$ is replaced with a new solution, $\lambda_i^{gen} = \lambda_i^{gen} + 1$. μ_i is the search times of $\mathcal{M}_i$.

4.4 Algorithm Description

ACSFLA is described in Algorithm 1.

Algorithm 1: ACSFLA

1: Initialize $gen \leftarrow 1$, generate initial population P with N solutions, and initialize Ω
2: **while** termination condition is not met **do**
3: Execute adaptive population division on $\bar{P}$
4: Sort memeplexes based on their quality
5: Perform adaptive cooperated and diversified search process
6: Compute Mo_i^{gen}
7: Perform partial population shuffling to update $\bar{P}$
8: Increment generation counter $gen \leftarrow gen + 1$
9: **end while**
10: Stop the search

Unlike previous SFLA, ACSFLA incorporates several features that improve optimization performance. First, it determines the search times for each memeplex based on its quality, efficiently focusing on high-quality memeplexes and minimizing effort on low-quality ones. Second, it decides whether to perform a cooperated search by evaluating dominance relationships, maintaining population diversity, and avoiding premature convergence. Third, during the cooperated search, the algorithm uses a segmented and dynamic targeted search strategy, while in non-cooperated scenarios, it focuses on local search around higher-quality solutions to improve efficiency and explore promising regions more effectively. Finally, adaptive population division and partial shuffling select low-quality memeplexes for reconstruction in the next generation. These features balance exploration and exploitation, sustain diversity, and improve search efficiency. Fig. 1 shows the flowchart of ACSFLA.

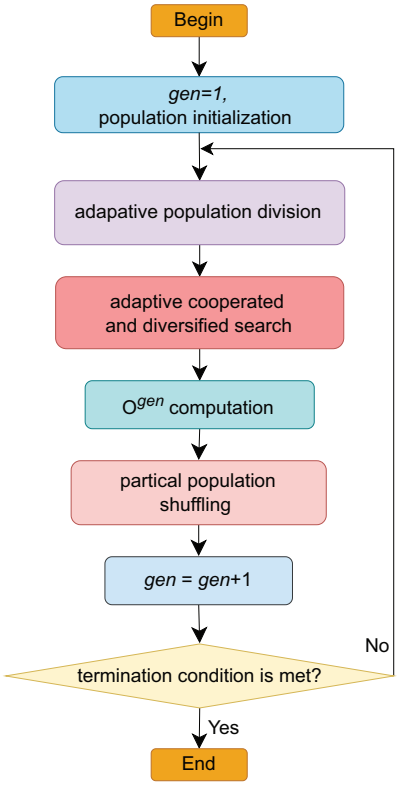


Figure 1: Flowchart of ACSFLA

5 Computational Experiments

Extensive experiments are conducted to test the performances of ACSFLA for the BPM scheduling problems in the fabric dyeing process. Experiments are implemented by using Matlab R2021a and run on 16 G RAM 3.1 GHz CPU PC.

5.1 Test Instances, Metrics and Comparative Algorithms

A total of 90 test instances are utilized, which is provided by Zhang et al. [1]. The related data are shown below. $(n, F) \in \{(100, 6), (100, 9), (200, 9), (200, 12), (300, 12), (300, 15)\}$, $m \in \{5, 7, 9\}$, $P_g \in [10, 30]$, $s_{ab} \in [2, 5]$, $s_{ab=}s_{ba} - 1$, $a > b$, $w_i \in [5, 60]$, $d_i = \varepsilon \times \frac{5n}{m}$, $\varepsilon \in [0.5, 1.5]$, $v_k = 30 + 10k$. There are 18 instance combinations, each of which is presented as $n \times F \times m$. Then randomly produces five instances for each combination.

Metric C [28] is used to compare the approximate Pareto optimal sets generated by different algorithms. The calculation formula for metric C has already been provided in Eq. (7) above.

The metric ρ [29] presents the ratio of $|\{x \in \Omega_l \,|x \in \Omega^*\}|$ to $|\Omega^*|$, where Ω_l denotes the non-dominated set obtained by Algorithm l, and Ω^* refers to the reference set, which consists of all non-dominated solutions in the union of non-dominated sets produced by all algorithms.

IGD [30] is used to calculate the distance of the non-dominated set Ω_l relative to a reference set Ω^*.

$$IGD(\Omega l, \Omega^*) = \frac{1}{|\Omega^*|} \sum_{x \varepsilon \Omega^*} \min_{y \varepsilon \Omega^*} d(x, y) \tag{11}$$

where $d(x, y)$ is the distance between a solution x and a reference solution y in the normalized objective space.

Three comparative algorithms are chosen. Zhang et al. [1] presented MOABC for parallel BPM scheduling in the fabric dyeing process and MOABC is directly used in this study. Lei et al. [25] proposed ASFLA, an adaptive SFLA-based algorithm, that integrating dynamic population division and adaptive search processes. Li et al. [20] provided an NSGA-CC, which is formed by using NSGA-II [31] framework, incorporating a hierarchical clustering-based environmental selection strategy. An SFLA is also applied to show the effect of new strategies of ACSFLA, deleting the adaptive population and diversified cooperated search.

5.2 Parameter Settings

ACSFLA has following parameters: N, s, μ, γ_1, γ_2 and stopping condition. Regarding the stopping condition, ACSFLA demonstrates good convergence when the CPU time reaches $0.05 \times n \times m$ seconds CPU time. Moreover, all comparative algorithms also converge effectively within this CPU time. Therefore, $0.05 \times n \times m$ seconds CPU time is set as the stopping condition for all algorithms.

Taguchi method [32] is used to decide the settings for other parameters by using instance 25, which belongs to combination $100 \times 9 \times 7$. Table 3 shows the levels of each parameter, in which five four levels are set for each parameter.

Table 3: Parameters and their levels

	Factor level			
Parameters	**1**	**2**	**3**	**4**
N	30	60	90	120
s	3	6	9	12

(Continued)

Table 3 (continued)

Parameters	Factor level			
	1	2	3	4
μ	3	6	9	12
γ_1	0.01	0.02	0.03	0.04
γ_2	0.6	0.7	0.8	0.9

Fig. 2 shows the results of ρ and S/N ratio, which is defined as $-10 \times \log_{10}\left(\rho^2\right)$. It can be found from Fig. 2 that ACSFLA with the following combination: $N = 120$, $s = 6$, $\mu = 6$, $\gamma_1 = 0.03$ and $\gamma_2 = 0.7$ can obtain better results than ACSFLA with other parameter combinations, so choose the above parameter settings.

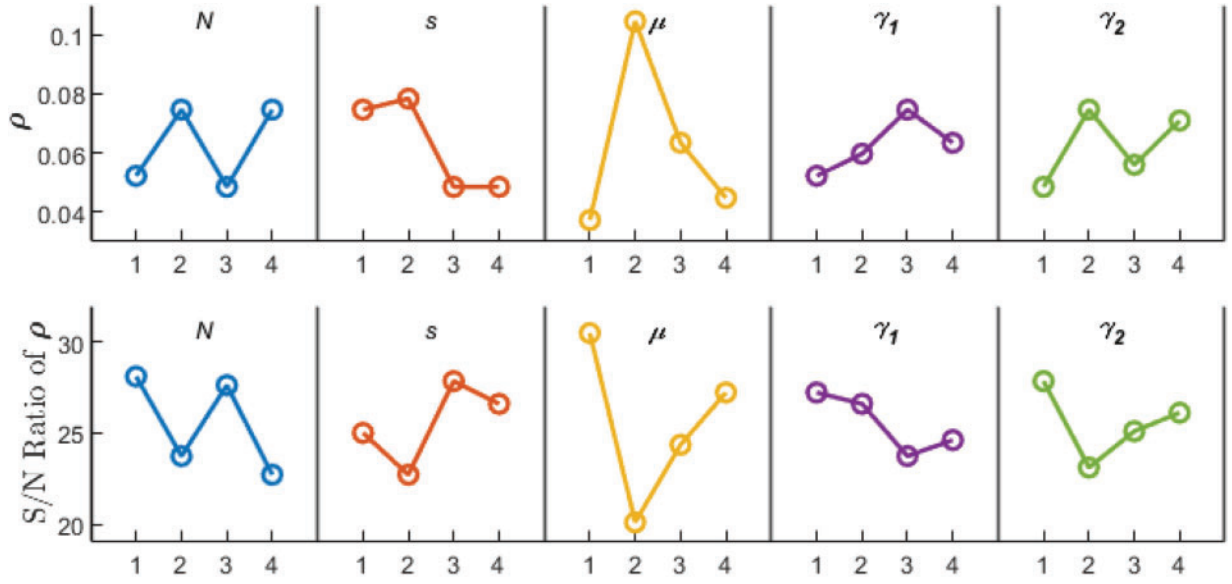


Figure 2: Main effect plot for means and S/N ratios

SFLA has $N = 150$, $s = 5$, $\mu = 60$, and the above stopping condition. With respect to ASFLA, MOABC, NSGA-CC, Lei et al. [25], Zhang et al. [1] and Li et al. [20] provided their parameter settings. These settings except the stopping condition are directly used in this study. The experimental results show that these settings of each comparative algorithm are still effective, so they are kept.

5.3 Results and Discussion

ACSFLA, ASFLA, and three comparative algorithms are used. Each algorithm randomly runs 10 times for each instance. Tables 4–7 describe the corresponding results of five algorithms, in which AC, S, A, M, N indicate ACSFLA, SFLA, ASFLA, MOABC, NSGA-CC. Fig. 3 gives a mean plot with 95% confidence interval. Fig. 4 shows distributions of non-dominated solutions produced by each algorithm.

Table 4: Computational results of five algorithms on metric ρ

Comb	No.	AC	S	A	M	N	Comb	No.	AC	S	A	M	N
100 × 6 × 5	1	**1.000**	0.000	0.000	0.000	0.000	200 × 12 × 5	46	**1.000**	0.000	0.000	0.000	0.000
	2	**1.000**	0.000	0.000	0.000	0.000		47	**1.000**	0.000	0.000	0.000	0.000
	3	**1.000**	0.000	0.000	0.000	0.000		48	**1.000**	0.000	0.000	0.000	0.000
	4	**1.000**	0.000	0.000	0.000	0.000		49	**1.000**	0.000	0.000	0.000	0.000
	5	**1.000**	0.000	0.000	0.000	0.000		50	**1.000**	0.000	0.000	0.000	0.000
100 × 6 × 7	6	**1.000**	0.000	0.000	0.000	0.000	200 × 12 × 7	51	**1.000**	0.000	0.000	0.000	0.000
	7	0.667	0.000	0.333	0.000	0.000		52	0.625	0.000	0.000	0.375	0.000
	8	**1.000**	0.000	0.000	0.000	0.000		53	**1.000**	0.000	0.000	0.000	0.000
	9	0.700	0.000	0.300	0.000	0.000		54	**1.000**	0.000	0.000	0.000	0.000
	10	**1.000**	0.000	0.000	0.000	0.000		55	**1.000**	0.000	0.000	0.000	0.000
100 × 6 × 9	11	**1.000**	0.000	0.000	0.000	0.000	200 × 12 × 9	56	0.750	0.000	0.000	0.250	0.000
	12	**1.000**	0.000	0.000	0.000	0.000		57	0.714	0.000	0.000	0.286	0.000
	13	**1.000**	0.000	0.000	0.000	0.000		58	0.714	0.000	0.286	0.000	0.000
	14	**1.000**	0.000	0.000	0.000	0.000		59	**1.000**	0.000	0.000	0.000	0.000
	15	0.357	0.000	0.429	0.000	0.214		60	**1.000**	0.000	0.000	0.000	0.000
100 × 9 × 5	16	**1.000**	0.000	0.000	0.000	0.000	300 × 12 × 5	61	0.000	0.200	0.800	0.000	0.000
	17	0.500	0.500	0.000	0.000	0.000		62	0.167	0.000	0.333	0.167	0.333
	18	**1.000**	0.000	0.000	0.000	0.000		63	**1.000**	0.000	0.000	0.000	0.000
	19	**1.000**	0.000	0.000	0.000	0.000		64	0.000	0.000	0.500	0.000	0.500
	20	**1.000**	0.000	0.000	0.000	0.000		65	0.143	0.000	0.857	0.000	0.000
100 × 9 × 7	21	**1.000**	0.000	0.000	0.000	0.000	300 × 12 × 7	66	0.000	0.000	0.571	0.000	0.429
	22	**1.000**	0.000	0.000	0.000	0.000		67	**1.000**	0.000	0.000	0.000	0.000
	23	0.600	0.000	0.300	0.000	0.100		68	0.000	0.000	0.000	0.667	0.333
	24	**1.000**	0.000	0.000	0.000	0.000		69	**1.000**	0.000	0.000	0.000	0.000
	25	0.750	0.000	0.000	0.250	0.000		70	0.700	0.000	0.000	0.300	0.000
100 × 9 × 9	26	0.933	0.000	0.000	0.000	0.067	300 × 12 × 9	71	0.600	0.000	0.000	0.400	0.000
	27	**1.000**	0.000	0.000	0.000	0.000		72	0.455	0.000	0.182	0.364	0.000
	28	**1.000**	0.000	0.000	0.000	0.000		73	**1.000**	0.000	0.000	0.000	0.000
	29	**1.000**	0.000	0.000	0.0	0.000		74	0.444	0.000	0.000	0.444	0.111
	30	**1.000**	0.000	0.000	0.000	0.000		75	**1.000**	0.000	0.000	0.000	0.000
200 × 9 × 5	31	**1.000**	0.000	0.000	0.000	0.000	300 × 15 × 5	76	0.625	0.000	0.375	0.000	0.000
	32	0.667	0.000	0.333	0.000	0.000		77	**1.000**	0.000	0.000	0.000	0.000
	33	**1.000**	0.000	0.000	0.000	0.000		78	**1.000**	0.000	0.000	0.000	0.000
	34	**1.000**	0.000	0.000	0.000	0.000		79	**1.000**	0.000	0.000	0.000	0.000
	35	**1.000**	0.000	0.000	0.000	0.000		80	**1.000**	0.000	0.000	0.000	0.000
200 × 9 × 7	36	**1.000**	0.000	0.000	0.000	0.000	300 × 15 × 7	81	**1.000**	0.000	0.000	0.000	0.000
	37	**1.000**	0.000	0.000	0.000	0.000		82	**1.000**	0.000	0.000	0.000	0.000
	38	**1.000**	0.000	0.000	0.000	0.000		83	**1.000**	0.000	0.000	0.000	0.000
	39	**1.000**	0.000	0.000	0.000	0.000		84	**1.000**	0.000	0.000	0.000	0.000
	40	**1.000**	0.000	0.000	0.000	0.000		85	**1.000**	0.000	0.000	0.000	0.000
200 × 9 × 9	41	0.846	0.000	0.000	0.000	0.154	300 × 15 × 9	86	0.857	0.000	0.143	0.000	0.000
	42	**1.000**	0.000	0.000	0.000	0.000		87	**1.000**	0.000	0.000	0.000	0.000
	43	**1.000**	0.000	0.000	0.000	0.000		88	0.667	0.000	0.167	0.167	0.000
	44	0.500	0.000	0.000	0.000	0.500		89	0.200	0.000	0.000	0.000	0.800
	45	0.250	0.000	0.375	0.375	0.000		90	0.714	0.000	0.000	0.000	0.286

Note: The bold entries represent the data where ACSFLA outperforms other algorithms.

Table 5: Computational results of five algorithms on metric IGD

Comb	No.	AC	S	A	M	N	Comb	No.	AC	S	A	M	N
100 × 6 × 5	1	**0.000**	1.204	0.848	0.865	1.026	200 × 12 × 5	46	**0.000**	0.723	0.424	0.566	0.312
	2	**0.000**	0.730	0.452	0.783	0.680		47	**0.000**	0.683	0.350	0.487	0.346
	3	**0.000**	0.611	0.761	1.095	0.643		48	**0.000**	0.922	0.452	0.901	0.601
	4	**0.000**	0.582	0.469	0.762	0.500		49	**0.000**	0.706	0.302	0.786	0.559
	5	**0.000**	1.072	0.668	0.837	0.776		50	**0.000**	1.034	0.246	0.514	0.387
100 × 6 × 7	6	**0.000**	0.816	0.208	0.646	0.476	200 × 12 × 7	51	**0.000**	0.605	0.738	0.405	0.426
	7	0.018	0.833	0.082	0.519	0.177		52	0.070	0.436	0.314	0.117	0.396
	8	**0.000**	0.838	0.124	0.478	0.268		53	**0.000**	0.701	0.168	0.338	0.516
	9	0.065	0.374	0.076	0.496	0.151		54	**0.000**	0.408	0.243	0.489	0.547
	10	**0.000**	1.048	0.350	0.903	0.515		55	**0.000**	0.734	0.487	0.454	0.789
100 × 6 × 9	11	**0.000**	0.437	0.653	0.424	0.304	200 × 12 × 9	56	0.115	0.746	0.150	0.154	0.304
	12	**0.000**	0.766	0.716	0.597	0.356		57	0.128	1.132	0.634	0.254	0.447
	13	**0.000**	0.538	0.261	0.695	0.302		58	0.439	0.400	0.195	0.026	0.217
	14	**0.000**	0.603	0.799	0.520	0.510		59	0.724	0.236	0.208	0.132	0.626
	15	**0.210**	0.387	0.189	0.593	0.230		60	0.554	0.842	0.210	0.000	0.391
100 × 9 × 5	16	**0.000**	0.635	0.266	0.408	0.424	300 × 12 × 5	61	0.201	0.292	0.068	0.288	0.206
	17	0.218	0.184	0.421	0.610	0.378		62	0.203	0.284	0.229	0.237	0.148
	18	**0.000**	0.696	0.607	0.767	0.582		63	0.344	0.540	0.000	0.662	0.502
	19	**0.000**	0.738	0.682	0.880	0.547		64	0.269	0.536	0.196	0.513	0.196
	20	**0.000**	1.080	0.955	1.161	0.889		65	0.221	0.916	0.052	0.396	0.178
100 × 9 × 7	21	**0.000**	0.453	0.275	0.353	0.504	300 × 12 × 7	66	0.157	0.174	0.120	0.117	0.094
	22	**0.000**	0.637	0.068	0.594	0.532		67	**0.000**	0.382	0.447	0.656	0.189
	23	0.206	0.286	0.210	0.460	0.296		68	0.317	0.425	0.596	0.122	0.227
	24	**0.000**	0.447	0.224	0.548	0.462		69	**0.000**	0.888	0.185	0.476	0.319
	25	0.055	0.344	0.462	0.184	0.451		70	0.141	0.492	0.271	0.032	0.342
100 × 9 × 9	26	0.008	0.074	0.198	0.348	0.122	300 × 12 × 9	71	0.240	0.424	0.373	0.168	0.229
	27	**0.000**	0.339	0.225	0.805	0.498		72	0.312	0.319	0.275	0.044	0.247
	28	**0.000**	0.682	0.552	0.657	0.256		73	**0.000**	0.851	0.553	0.539	0.294
	29	**0.000**	0.389	0.569	0.682	0.620		74	0.083	0.534	0.122	0.109	0.175
	30	**0.000**	0.566	0.401	0.789	0.420		75	**0.000**	0.780	0.467	0.593	0.416
200 × 9 × 5	31	**0.000**	0.576	0.417	0.812	0.690	300 × 15 × 5	76	0.072	0.347	0.217	0.276	0.245
	32	0.053	0.461	0.096	0.428	0.578		77	**0.000**	0.793	0.361	0.702	0.741
	33	**0.000**	0.792	0.469	0.507	0.533		78	**0.000**	1.098	0.643	0.762	0.742
	34	**0.000**	0.501	0.518	0.407	0.542		79	**0.000**	0.577	0.337	0.491	0.482
	35	**0.000**	0.515	0.530	0.824	0.550		80	**0.000**	0.700	0.198	0.712	0.603
200 × 9 × 7	36	**0.000**	0.652	0.302	0.512	0.557	300 × 15 × 7	81	**0.000**	0.737	0.648	0.679	0.559
	37	**0.000**	0.641	0.522	0.590	0.681		82	**0.000**	0.900	0.560	0.569	0.359
	38	**0.000**	0.681	0.242	0.171	0.420		83	**0.000**	0.748	0.383	0.669	0.515
	39	**0.000**	0.816	0.474	0.842	0.349		84	**0.000**	0.614	0.433	0.521	0.510
	40	**0.000**	0.542	0.561	0.706	0.554		85	**0.000**	0.405	0.482	0.512	0.405
200 × 9 × 9	41	0.073	0.583	0.474	0.509	0.272	300 × 15 × 9	86	0.030	0.472	0.228	0.313	0.344
	42	**0.000**	0.734	0.407	0.973	0.607		87	**0.000**	0.692	0.693	0.647	0.425
	43	**0.000**	0.739	0.260	0.806	0.428		88	0.094	0.390	0.248	0.227	0.133
	44	**0.365**	0.722	0.366	0.557	0.161		89	0.434	0.393	0.445	0.324	0.035
	45	0.249	0.333	0.185	0.098	0.256		90	0.131	0.497	0.264	0.293	0.111

Note: The bold entries represent the data where ACSFLA outperforms other algorithms.

Table 6: Computational results of five algorithms on metric *C* part1

Comb	No.	*C* (AC, S)	*C* (S, AC)	*C* (AC, A)	*C* (A, AC)	*C* (AC, M)	*C* (M, AC)	*C* (AC, N)	*C* (N, AC)
100 × 6 × 5	1	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	2	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	3	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	4	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	5	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
100 × 6 × 7	6	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	7	**1.000**	0.000	0.000	0.200	**1.000**	0.000	**1.000**	0.000
	8	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	9	**1.000**	0.000	0.250	0.000	**1.000**	0.000	0.750	0.000
	10	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
100 × 6 × 9	11	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	12	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	13	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	14	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	15	**1.000**	0.000	0.000	0.000	**1.000**	0.000	0.556	0.000
100 × 9 × 5	16	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	17	0.500	0.833	0.000	0.000	**1.000**	0.000	**1.000**	0.000
	18	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	19	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	20	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
100 × 9 × 7	21	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	22	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	23	**1.000**	0.000	0.250	0.000	**1.000**	0.000	0.000	0.000
	24	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	25	**1.000**	0.000	**1.000**	0.000	0.950	0.000	**1.000**	0.000
100 × 9 × 9	26	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	0.500	0.000
	27	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	28	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	29	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	30	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
200 × 9 × 5	31	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	32	**1.000**	0.000	0.500	0.000	**1.000**	0.000	**1.000**	0.000
	33	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	34	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	35	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
200 × 9 × 7	36	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	37	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	38	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	39	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	40	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000

(Continued)

Table 6 (continued)

Comb	No.	*C* (AC, S)	*C* (S, AC)	*C* (AC, A)	*C* (A, AC)	*C* (AC, M)	*C* (M, AC)	*C* (AC, N)	*C* (N, AC)
200 × 9 × 9	41	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	0.600	0.000
	42	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	43	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	44	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	0.833	0.000
	45	**1.000**	0.000	0.571	0.000	0.733	0.000	**1.000**	0.000

Note: The bold entries represent the data where ACSFLA outperforms other algorithms.

Table 7: Computational results of five algorithms on metric *C* part2

Comb	No.	*C* (AC, S)	*C* (S, AC)	*C* (AC, A)	*C* (A, AC)	*C* (AC, M)	*C* (M, AC)	*C* (AC, N)	*C* (N, AC)
200 × 12 × 5	46	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	47	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	48	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	49	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	50	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
200 × 12 × 7	51	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	52	**1.000**	0.000	**1.000**	0.000	0.750	0.000	**1.000**	0.000
	53	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	54	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	55	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
200 × 12 × 9	56	**1.000**	0.000	**1.000**	0.000	0.500	0.000	**1.000**	0.000
	57	**1.000**	0.000	**1.000**	0.000	0.111	0.000	**1.000**	0.000
	58	0.667	0.833	0.000	1.000	0.000	1.000	0.000	1.000
	59	0.667	1.000	0.000	1.000	0.000	1.000	0.000	0.667
	60	**1.000**	0.000	0.000	1.000	0.000	1.000	0.000	1.000
300 × 12 × 5	61	0.750	0.000	0.000	1.000	0.583	0.000	0.333	0.333
	62	**1.000**	0.000	0.000	0.667	0.778	0.000	0.333	0.000
	63	**1.000**	0.000	0.000	1.000	**1.000**	0.000	0.600	0.000
	64	**1.000**	0.000	0.000	1.000	0.917	0.000	0.000	1.000
	65	**1.000**	0.000	0.000	0.667	**1.000**	0.000	0.500	0.333
300 × 12 × 7	66	0.667	0.667	0.000	1.000	0.375	0.333	0.250	0.333
	67	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	68	**1.000**	0.000	**1.000**	0.000	0.250	1.000	0.600	0.000
	69	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	70	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
300 × 12 × 9	71	**1.000**	0.000	**1.000**	0.000	0.882	0.000	**1.000**	0.000
	72	**1.000**	0.000	0.500	0.000	0.533	0.000	0.750	0.000
	73	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	74	**1.000**	0.000	0.167	0.000	0.474	0.200	0.800	0.000

(Continued)

Table 7 (continued)

Comb	No.	C (AC, S)	C (S, AC)	C (AC, A)	C (A, AC)	C (AC, M)	C (M, AC)	C (AC, N)	C (N, AC)
	75	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
300 × 15 × 5	76	**1.000**	0.000	0.000	0.167	0.556	0.000	**1.000**	0.000
	77	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	78	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	79	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	80	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
300 × 15 × 7	81	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	82	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	83	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	84	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	85	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
300 × 15 × 9	86	**1.000**	0.000	0.000	0.000	0.714	0.000	0.909	0.000
	87	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	88	**1.000**	0.000	0.667	0.000	0.889	0.000	**1.000**	0.000
	89	**1.000**	0.000	**1.000**	0.000	0.929	0.000	0.556	0.000
	90	**1.000**	0.000	**1.000**	0.000	0.889	0.000	0.667	0.000

Note: The bold entries represent the data where ACSFLA outperforms other algorithms.

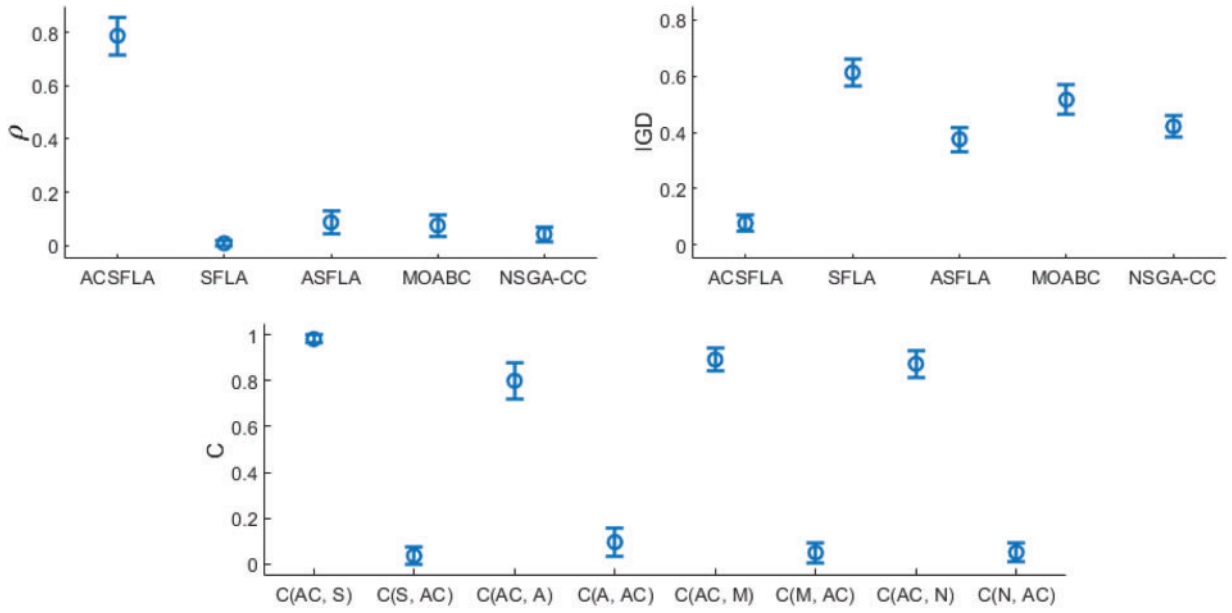
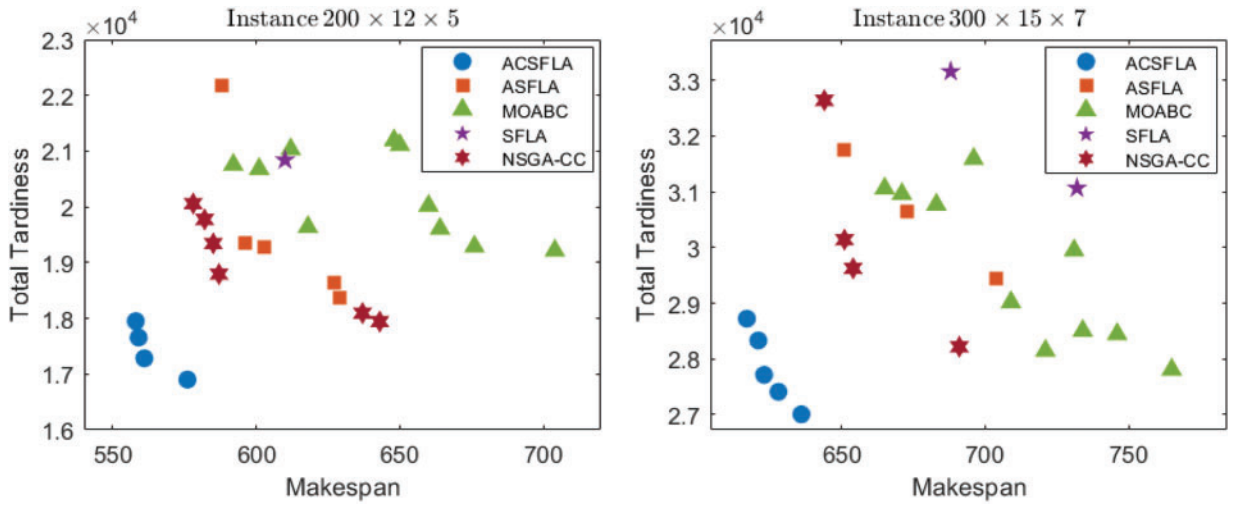


Figure 3: Mean plot with 95% confidence interval

ACSFLA demonstrates clear superiority in terms of the ρ metric. As shown in Table 4, ACSFLA achieves higher ρ values than the other algorithms in 79 of 90 instances, achieving a ρ value of 1 in 60 instances. This indicates that all the non-dominated solutions produced by ACSFLA are also non-dominated in the

combined reference set Ω, demonstrating that ACSFLA not only generates high-quality solutions but also outperforms the other algorithms. The average ρ value for ACSFLA is 0.82, with a standard deviation of 0.29, reflecting consistent high performance. Furthermore, when ACSFLA is compared with other algorithms using a t-test, the p-values are all far smaller than 0.05, highlighting its significant advantage.



Figurer 4: Distribution of non-dominated solutions

ACSFLA also demonstrates superior performance in terms of the *IGD* metric. Table 5 shows that ACSFLA achieves smaller *IGD* values than SFLA in 88 instances, ASFLA in 78 instances, MOABC in 83 instances, and NSGA-CC in 82 instances. These results highlight ACSFLA's superior ability to approximate the ideal reference set Ω and its better convergence ability toward the Pareto front. The average *IGD* value for ACSFLA is 0.076, with a standard deviation of 0.14, indicating consistent performance. The maximum *IGD* value achieved is 0.724, demonstrating that ACSFLA maintains a competitive edge in most cases. Furthermore, when ACSFLA is compared with other algorithms using a t-test, the p-values are all far smaller than 0.05, confirming its significant advantage in terms of convergence and solution quality.

In terms of the C metric, ACSFLA shows a significant advantage over other algorithms. As presented in Tables 6 and 7, ACSFLA achieves smaller $C(S, AC)$ values than $C(AC, S)$ in 89 of 90 instances, smaller $C(A, AC)$ than $C(AC, A)$ in 79 instances, smaller $C(M, AC)$ than $C(AC, M)$ in 86 instances, and smaller $C(N, AC)$ than $C(AC, N)$ in 85 instances. Specifically, ACSFLA reaches $C(AC, S) = 1$ in 85 instances, $C(A, AC) = 1$ in 69 instances, $C(AC, M) = 1$ in 68 instances, and $C(AC, N) = 1$ in 69 instances. The average values for $C(AC, S)$, $C(AC, A)$, $C(AC, M)$, and $C(AC, N)$ are 0.98, 0.79, 0.89, and 0.87, respectively, with standard deviations of 0.08, 0.38, 0.24, and 0.27. This indicates that the non-dominated solutions generated by ACSFLA are stronger than other solutions from other algorithms.

The above result analyses show that ACSFLA obtains better results than its three comparative algorithms. Initialization is done by the heuristic method, Mo and Mq are used for adaptive population division. Executing adaptive cooperated and diversified memeplex search based on the dominance relationship between memeplexes. As a result, the exploration capability is intensified and high diversity is maintained. Thus, ASFLA is a very competitive method for solving parallel BPM scheduling in the fabric dyeing process.

6 Conclusion and Future Topics

In this paper, we have effectively resolved the parallel BPM scheduling problem with machine eligibility in fabric dyeing processes using the newly developed adaptive cooperated shuffled frog-leaping algorithm (ACSFLA). ACSFLA incorporates innovative strategies based on the quality M_q and evolutionary quality M_o of memeplexes, along with the dominance relationships C between them. By eliminating global population shuffling and conducting targeted shuffling within the division pool for selected memeplexes, ACSFLA significantly enhances algorithm efficiency. Extensive experiments have demonstrated the effectiveness of ACSFLA's new strategies, with the algorithm outperforming comparative algorithms from the literature in minimizing makespan and total tardiness, thus highlighting its substantial advantages in solving this specific scheduling problem.

For future research directions in parallel BPM scheduling for fabric dyeing, several promising paths could be explored. Incorporating energy consumption as an additional objective might be a key step toward achieving more energy-efficient processes. This could involve devising algorithms considering the power consumption patterns of machines and job sequences to minimize energy use during dyeing. Developing new neighborhood search methods could be an effective way to enhance search efficiency, enabling the algorithm to find better solutions in a shorter time. Applying reinforcement learning techniques may also hold great potential in making the scheduling more intelligent and adaptable to dynamic production situations. The key features of the algorithm include adaptive population division, as well as an adaptive cooperated and diversified search strategy.

Acknowledgement: We sincerely appreciate the perceptive feedback from the anonymous reviewers. It has substantially contributed to improving the overall quality of this paper.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: Study conception and design: Lianqiang Wu; analysis and interpretation of results: Yutong Cai; draft manuscript preparation: Lianqiang Wu; review and editing: Deming Lei. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The data supporting this study are described in the first paragraph of Section 5.1.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

1. Zhang R, Chang PC, Song S, Wu C. A multi-objective artificial bee colony algorithm for parallel batch-processing machine scheduling in fabric dyeing processes. Knowl-Based Syst. 2017;116(7):114–29. doi:10.1016/j.knosys.2016.10.026.
2. Burkard RE, Hatzl J. A complex time based construction heuristic for batch scheduling problems in the chemical industry. Eur J Oper Res. 2006;174(2):1162–83. doi:10.1016/j.ejor.2005.03.011.
3. Bellanger A, Oulamara A. Scheduling hybrid flowshop with parallel batching machines and compatibilities. Comput Oper Res. 2009;36(6):1982–92. doi:10.1016/j.cor.2008.06.011.
4. Tang L, Wang G, Chen ZL. Integrated charge batching and casting width selection at Baosteel. Oper Res. 2014;62(4):772–87. doi:10.1287/opre.2014.1278.
5. Hulett M, Damodaran P, Amouie M. Scheduling non-identical parallel batch processing machines to minimize total weighted tardiness using particle swarm optimization. Comput Ind Eng. 2017;113(8):425–36. doi:10.1016/j.cie.2017.09.037.

6. Trindade RS, De Araujo OCB, Fampa MHC, Muller FM. Modelling and symmetry breaking in scheduling problems on batch processing machines. Int J Prod Res. 2018;56(22):7031–48. doi:10.1080/00207543.2018.1424371.
7. Kong M, Wang W, Deveci M, Zhang Y, Wu X, Coffman D. A novel carbon reduction engineering method-based deep Q-learning algorithm for energy-efficient scheduling on a single batch-processing machine in semiconductor manufacturing. Int J Prod Res. 2024;62(18):6449–72. doi:10.1080/00207543.2023.2252932.
8. Jia W, Jiang Z, Li Y. Combined scheduling algorithm for re-entrant batch-processing machines in semiconductor wafer manufacturing. Int J Prod Res. 2015;53(6):1866–79. doi:10.1080/00207543.2014.965355.
9. Zhao A, Bard JF. Batch scheduling in a multi-purpose system with machine downtime and a multi-skilled workforce. Int J Prod Res. 2024;62(12):4470–93. doi:10.1080/00207543.2023.2265508.
10. Jing Wang DL, Tang H. An adaptive artificial bee colony for hybrid flow shop scheduling with batch processing machines in casting process. Int J Prod Res. 2024;62(13):4793–808. doi:10.1080/00207543.2023.2279145.
11. Abedi M, Seidgar H, Fazlollahtabar H, Bijani R. Bi-objective optimisation for scheduling the identical parallel batch-processing machines with arbitrary job sizes, unequal job release times and capacity limits. Int J Prod Res. 2015;53(6):1680–711. doi:10.1080/00207543.2014.952795.
12. Zhang H, Li K, Chu C, Zh Jia. Parallel batch processing machines scheduling in cloud manufacturing for minimizing total service completion time. Comp Oper Res. 2022;146(2):105899. doi:10.1016/j.cor.2022.105899.
13. Zhou S, Li X, Du N, Pang Y, Chen H. A multi-objective differential evolution algorithm for parallel batch processing machine scheduling considering electricity consumption cost. Comput Oper Res. 2018;96(1):55–68. doi:10.1016/j.cor.2018.04.009.
14. Xue L, Zhao S, Mahmoudi A, Feylizadeh MR. Flexible job-shop scheduling problem with parallel batch machines based on an enhanced multi-population genetic algorithm. Complex Intell Syst. 2024;10(3):4083–101. doi:10.1007/s40747-024-01374-7.
15. Arroyo JEC, Leung JYT. An effective iterated greedy algorithm for scheduling unrelated parallel batch machines with non-identical capacities and unequal ready times. Comput Ind Eng. 2017;105(6):84–100. doi:10.1016/j.cie.2016.12.038.
16. Cai J, Lei D. A cooperated shuffled frog-leaping algorithm for distributed energy-efficient hybrid flow shop scheduling with fuzzy processing time. Complex Intell Syst. 2021;7(5):2235–53. doi:10.1007/s40747-021-00400-2.
17. Kong M, Liu X, Pei J, Pardalos PM, Mladenovic N. Parallel-batching scheduling with nonlinear processing times on a single and unrelated parallel machines. J Global Optim. 2020;78(4):693–715. doi:10.1007/s10898-018-0705-3.
18. Yang Y, Song Y, Guo W, Lei Q, Sun A, Fan L. Guided shuffled frog-leaping algorithm for flexible job shop scheduling problem with variable sublots and overlapping in operations. Comput Ind Eng. 2023;180(19):109209. doi:10.1016/j.cie.2023.109209.
19. Lei D, Wang T. Solving distributed two-stage hybrid flowshop scheduling using a shuffled frog-leaping algorithm with memeplex grouping. Eng Optim. 2020;52(9):1461–74. doi:10.1080/0305215X.2019.1674295.
20. Li K, Zhang H, Chu C, Zh Jia, Wang Y. A bi-objective evolutionary algorithm for minimizing maximum lateness and total pollution cost on non-identical parallel batch processing machines. Comput Ind Eng. 2022;172(3):108608. doi:10.1016/j.cie.2022.108608.
21. Wang Y, Han Y, Wang Y, Pan QK, Wang L. Sustainable scheduling of distributed flow shop group: a collaborative multi-objective evolutionary algorithm driven by indicators. Trans Evol Comp. 2024;28(6):1794–808. doi:10.1109/TEVC.2023.3339558.
22. Wang Y, Han Y, Wang Y, Wang X, Liu Y, Gao K. Reinforcement learning-assisted memetic algorithm for sustainability-oriented multiobjective distributed flow shop group scheduling. IEEE Trans Syst Man Cybern: Syst. 2025;1–15. doi:10.1109/TSMC.2025.3541795.
23. Wu X, Cao Z. An improved multi-objective evolutionary algorithm based on decomposition for solving re-entrant hybrid flow shop scheduling problem with batch processing machines. Comput Ind Eng. 2022;169(1):108236. doi:10.1016/j.cie.2022.108236.
24. Eusuff M, Lansey K, Pasha F. Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization. Eng Optim. 2006;38(2):129–54. doi:10.1080/03052150500384759.

25. Lei D, Dai T. An adaptive shuffled frog-leaping algorithm for parallel batch processing machines scheduling with machine eligibility in fabric dyeing process. Int J Prod Res. 2024;62(21):7704–21. doi:10.1080/00207543.2024.2324452.
26. Cai J, Zhou R, Lei D. Dynamic shuffled frog-leaping algorithm for distributed hybrid flow shop scheduling with multiprocessor tasks. Eng Appl Artif Intell. 2020;90(1):103540. doi:10.1016/j.engappai.2020.103540.
27. Xu Y, Wang L, Wang S, Liu M. An effective shuffled frog-leaping algorithm for solving the hybrid flow-shop scheduling problem with identical parallel machines. Eng Optim. 2013;45(12):1409–30. doi:10.1080/0305215X.2012.737784.
28. Zitzler E, Thiele L. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE Trans Evol Comput. 1999;3(4):257–71. doi:10.1109/4235.797969.
29. Lei D. A pareto archive particle swarm optimization for multi-objective job shop scheduling. Comput Ind Eng. 2008;54(4):960–71. doi:10.1016/j.cie.2007.11.007.
30. Bosman PAN, Thierens D. The balance between proximity and diversity in multiobjective evolutionary algorithms. IEEE Trans Evol Comput. 2003;7(2):174–88. doi:10.1109/TEVC.2003.810761.
31. Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput. 2002;6(2):182–97. doi:10.1109/4235.996017.
32. Taguchi G. Introduction to quality engineering. Tokyo, Japan: Asian Productivity Organization; 1986.