Journal Européen des Systèmes Automatisés
Vol. 58, No. 7, July, 2025, pp. 1403-1416
Journal homepage: http://iieta.org/journals/jesa

Multi-Objective Identical Parallel Flow Shop Scheduling Using NSGA-II and MOPSO with a Novel Load Balancing Procedure

Milad Mansouri*, Hacene Smadi, Younes Bahmani

Laboratory of Automation and Manufacturing, Department of Industrial Engineering, Faculty of Technology, University of Batna 2 – Mostefa Ben Boulaïd, Batna 05000, Algeria

Corresponding Author Email: milad.mansouri@univ-batna2.dz

https://doi.org/10.18280/jesa.580709

Received: 20 May 2025
Revised: 28 June 2025
Accepted: 6 July 2025
Available online: 31 July 2025

Keywords:
enhanced heuristics, Identical Parallel Flow Shop Scheduling (IPFSSP), Load Balancing Procedure (LBP), multi-objective optimization, waiting time constraint

ABSTRACT

Although flow shop scheduling has been widely investigated, the Identical Parallel Flow Shop Scheduling Problem (IPFSSP) remains largely overlooked, particularly in multi-objective optimization contexts. This study addresses this gap by formulating a bi-objective mathematical model that minimizes makespan and earliness–tardiness under strict waiting time constraints. To solve it, a dynamic Load Balancing Procedure (LBP) is embedded within two established metaheuristics: NSGA-II and MOPSO. The proposed algorithms are evaluated across six benchmark instance scales with varying job–machine configurations. Results show that NSGA-II–LBP and MOPSO–LBP achieve average reductions of 13.7%–19.2% in makespan and 18.4%–22.8% in earliness–tardiness compared to their baseline counterparts. Statistical analyses using ANOVA and paired t-tests confirm the significance of these improvements. NSGA-II–LBP delivers superior convergence, solution diversity, and scalability, while MOPSO–LBP offers higher computational efficiency, making it particularly well-suited for real-time scheduling in complex manufacturing systems.

1. INTRODUCTION

Parallel flow shop scheduling is a cornerstone of modern manufacturing systems, where the ability to execute jobs concurrently across multiple production lines is critical for meeting stringent efficiency and flexibility demands. While classical scheduling models such as the Flow Shop (FS) [1], Job Shop (JS) [2], and Hybrid Flow Shop (HFS) [3], have been extensively investigated, the Identical Parallel Flow Shop Scheduling Problem (IPFSSP) remains relatively understudied. This limited attention is primarily attributed to the structural complexity of the IPFSSP, which integrates elements of FS, HFS, and parallel machine configuration [4]. The problem entails two interrelated sub-tasks: Assigning jobs to lines and sequencing them within each line.

Originally conceptualized by Graham (1969) in the context of parallel computing with identical processors [5], the IPFSSP has more recently attracted attention through the work of Ribas et al. [6-8], who proposed constructive heuristics for variants such as the Parallel Blocking Flow Shop Problem (PBFSP). Nonetheless, the bulk of existing research on IPFSSP has focused on single-objective formulations, predominantly aimed at minimizing makespan.

However, real-world manufacturing environments, require multi-objective optimization (MOO) frameworks that address multiple, often conflicting, performance criteria. In such domains, quality-critical constraints, notably mandatory waiting times between consecutive operations, are indispensable. These waiting times are inherent to processes such as curing, cooling, or chemical stabilization and are essential to ensure product integrity and compliance with safety standards. Neglecting these constraints may compromise product quality and lead to operational failures.

Although metaheuristics such as NSGA-II and MOPSO have demonstrated promising results in solving multi-objective scheduling problems [4, 9], their direct application to the IPFSSP reveals two critical shortcomings. First, static job assignments can lead to load imbalances across production lines, reducing overall throughput. Second, naïve sequencing methods often fail to coordinate job flows effectively, resulting in idle periods, violations of wait-time constraints, and increased earliness–tardiness penalties.

While dynamic load balancing strategies exist for parallel and distributed systems, such as workload-based dispatching heuristics that aim to optimize task-to-resource allocation [10], queue-length threshold-based resource configuration mechanisms that adjust allocations dynamically based on job queue backlogs [11], and decentralized agent-based techniques involving autonomous agents for local decision-making and cooperative resource management [12]. These approaches typically do not model inter-stage temporal dependencies and lack strict enforcement of wait-time constraints. Moreover, they often rely on reactive mechanisms, which may limit their ability to handle highly constrained or time-sensitive jobs proactively.

To address these limits, we propose a novel Load Balancing Procedure (LBP) that introduces three key advancements over conventional methods. First, LBP incorporates a proactive, criticality-driven assignment mechanism, which calculates job criticality scores based on processing time and slack analysis. This enables the prioritized placement of constrained jobs into feasible wait-time windows, unlike traditional reactive methods that intervene only after imbalances arise. Second, LBP adopts a two-phase adaptive balancing approach, where critical jobs are pre-assigned, followed by the distribution of non-critical jobs using load variance minimization strategies. This contrasts with single-phase strategies that ignore job criticality and treat all jobs uniformly. Third, LBP includes a dedicated feasibility repair phase, which dynamically resequences jobs that violate wait-time constraints. This constraint-aware adjustment mechanism is notably absent in most classical load balancing algorithms. Through the structured integration of prioritization, adaptive load balancing, and constraint repair, LBP achieves synchronized scheduling across production lines while ensuring strict compliance with wait-time constraints and promoting equitable load distribution.

To evaluate the effectiveness of this approach, the proposed LBP is integrated into both NSGA-II and MOPSO frameworks to enhance their performance in solving the Identical Parallel Flow Shop Scheduling Problem (IPFSSP). A bi-objective mathematical model is formulated to simultaneously minimize makespan and total earliness–tardiness, in conjunction with the LBP. Experimental results on multiple benchmark instance groups show that NSGA-II–LBP and MOPSO–LBP achieve average reductions of 13.7%–19.2% in makespan and 18.4%–22.8% in total earliness–tardiness. These improvements are statistically validated using ANOVA and paired t-tests, with strong performance observed on large-scale instances.

The paper is structured as follows: Section 2 reviews related literature on the IPFSSP and identifies existing research gaps. Section 3 details the proposed bi-objective mathematical formulation. Section 4 introduces the Load Balancing Procedure and its integration with MOO algorithms. Section 5 outlines the experimental design, and Section 6 discusses the computational results. Finally, Section 7 concludes the study.

2. RELATED WORK AND RESEARCH GAPS

2.1 Existing methods and models

Most existing studies in flow shop and parallel flow shop scheduling adopt single-objective formulations, primarily focusing on makespan minimization due to its mathematical tractability and long-standing use as a benchmark in scheduling research. A comparative simulation study was conducted to evaluate production scheduling strategies in hybrid and parallel flow shop environments using the General Shifting Bottleneck Routine (SBR). The study assessed the effectiveness of several dispatching rules, including First Come First Served (FCFS), Longest Processing Time (LPT), and Shortest Processing Time (SPT), in managing workflow and improving throughput [13]. Complementing this line of research, a multi-phase heuristic algorithm was developed, comprising a constructive heuristic for initial job sequencing and an improvement heuristic aimed at balancing workloads by exchanging jobs between production lines [14].

Building on prior work, subsequent studies introduced methodological advancements centered on approximation techniques. One study proposed a 3/2-approximation algorithm for scheduling jobs in a two-stage parallel flow shop with multiple identical machines. A 12/7-approximation algorithm was also developed for a three-stage configuration [15]. Subsequently, a pseudo-polynomial time dynamic programming algorithm was introduced to generate exact solutions. This algorithm served as a core subroutine in constructing a Fully Polynomial-Time Approximation Scheme (FPTAS) [16]. In a related study, the FPTAS was further refined by classifying jobs into large and small categories. Schedules for large jobs were enumerated, while small jobs were allocated using a linear programming model combined with a sliding window technique [17]. In a later enhancement, the FPTAS was extended by scaling arbitrary instances into restricted forms and solving them using a Mixed-Integer Linear Programming (MILP) model, resulting in near-optimal solutions with bounded approximation errors [18].

Recently, Ribas et al. [7] conducted several studies focusing on the Parallel Flow Shop Scheduling Problem (PFSSP) under blocking constraints. The initial work introduced the use of Iterated Local Search (ILS) and Iterated Greedy Algorithm (IGA), integrated with two types of Variable Neighborhood Search (VNS), in conjunction with constructive and improvement heuristics. A MILP model was also proposed for solving small-scale instances. In subsequent work, an Iterated Greedy (IG) algorithm was developed, beginning with a high-quality initial solution and refining it through perturbation and local search, using a simulated annealing-based acceptance criterion to escape local optima [8]. More recently, 36 constructive heuristics were evaluated, combining seven sequencing rules with five lines allocation methods. From this set, the RCP0 heuristic was proposed, prioritizing the line with the earliest available machine and the job that minimizes machine idle time [6].

Some studies have addressed other alternative objectives using diverse heuristic and optimization techniques. In another study, total tardiness was minimized using an Iterated Greedy Algorithm (IGA), which enhanced initial solutions through Variable Neighborhood Search (VNS) and job reassignment mechanisms [19]. Additionally, total flow time was targeted through the application of priority rule-based heuristics, the NEH algorithm, and IGA to obtain near-optimal schedules [20].

Multi-objective optimization aligns more closely with practical manufacturing needs by simultaneously optimizing conflicting performance criteria. A simheuristic algorithm was proposed for stochastic, non-Identical Parallel Flow Shop Scheduling, integrating a biased-randomized NEH heuristic, local search, and Monte Carlo simulation to evaluate deterministic and expected makespan values [21]. Another study addressed the bicriteria objective of minimizing total earliness and tardiness using a hybrid approach that combines a Greedy Randomized Adaptive Search Procedure (GRASP) with Genetic Algorithm (GA) and Particle Swarm Optimization (PSO), demonstrating superior performance over individual strategies [22]. A further contribution focused on minimizing total flow time and the number of tardy jobs in a two-machine, non-identical parallel flow shop. The proposed Multi-Objective Evolutionary Algorithm (MOEA) incorporated local search into the NSGA-II framework, proving more effective and efficient than conventional MOEAs [23].

Task sequencing has long been a cornerstone of flow shop scheduling research, with foundational works such as Johnson's (1954) rule for two-machine optimization [24], the

Nawaz-Enscore-Ham (NEH) algorithm (1983) for permutation flow shops [25], and the Campbell-Dudek-Smith (CDS) method (1970) for multi-machine scheduling establishing key heuristics [26]. Over time, substantial progress has been made in addressing sequencing under complex constraints, particularly in dynamic and data-driven manufacturing environments. Recent studies reflect a shift toward intelligent, adaptive methods. One approach applies deep reinforcement learning (DRL) to dynamic parallel machine scheduling, using a modified Transformer model and a variable-length state matrix to capture job and machine data. The DRL agent autonomously extracts features and selects jobs for idle machines to improve scheduling performance [27]. Another study builds on the NEH algorithm, adapting it for the satellite industry by sorting jobs based on processing times and enhancing it with a discrete-event simulation model that reflects real workshop conditions, improving both adaptability and practicality [28]. Another study introduces a dynamic knowledge graph to represent distributed manufacturing resources, combined with an AI scheduler trained through DRL and optimized using a dueling Deep Q-Network (DQN). Semantic matching aligns resources with subtasks, while meta-heuristic methods support near-optimal scheduling decisions [29].

Despite advances in task sequencing, load balancing is still underdeveloped in manufacturing. Most current approaches rely on static or predefined job-to-line assignments, leading to workload imbalances. In contrast, domains like cloud computing have advanced adaptive load balancing techniques to address similar challenges. Recent studies have explored intelligent optimization methods to enhance task scheduling and resource allocation. One approach uses deep learning models, specifically Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), to cluster virtual machines (VMs) into overloaded and underloaded groups. It combines Reinforcement Learning with a Hybrid Lyrebird Falcon Optimization (HLFO) algorithm, which merges Lyrebird and Falcon Optimization strategies to improve system performance [30]. Another hybrid model, QMPSO, integrates Modified Particle Swarm Optimization (MPSO) with improved Q-learning, adjusting MPSO's velocity based on Q-learning policies to balance VM loads and optimize resource use [31]. Similarly, Load Balancing Modified PSO (LBMPSO), a tailored variant of PSO, updates fitness values and tracks particle positions while continuously monitoring VM loads for efficient task assignment [32].

2.2 Research gaps and study motivations

A review of parallel flow shop scheduling literature reveals several key research gaps:

Current research remains heavily focused on single-objective formulations, with a predominant emphasis on makespan minimization. While this objective offers mathematical tractability and benchmarking consistency, it fails to capture the multi-dimensional trade-offs encountered in real-world manufacturing. Multi-objective models are significantly underexplored in IPFSSP contexts.

The prevailing focus on proportional parallel flow shop configurations limits the generalizability of many existing approaches. In these systems, job-to-line assignment is often simplified by performance asymmetries across lines. However, this assumption does not apply in identical configurations, where all lines have uniform capabilities. Effective job allocation in such systems requires more sophisticated load balancing strategies.

Challenges related to heterogeneous job priorities in identical parallel flow shops remain insufficiently addressed. The absence of line hierarchies and the mix of critical and non-critical tasks increase scheduling complexity. These conditions highlight the need for adaptive, real-time load balancing mechanisms.

Existing load balancing strategies in IPFSSP are typically static or heuristic-based and lack the flexibility to respond to system dynamics. This rigidity often results in inefficient resource utilization and the emergence of operational bottlenecks.

The scheduling literature continues to prioritize task sequencing, often treating it independently from job-to-line assignment. This separation creates fragmented decision-making, despite growing evidence that integrated approaches yield more balanced and efficient schedules in identical parallel environments.

Cross-disciplinary knowledge transfer between cloud computing and manufacturing scheduling remains limited. Proven concepts from computing—such as decentralized load balancing and adaptive resource allocation—are rarely applied in production scheduling, despite their relevance.

In response to these gaps, this study proposes a novel dynamic load balancing strategy embedded within NSGA-II and MOPSO frameworks, specifically designed for multi stage identical parallel flow shop systems. The approach jointly optimizes job-to-line assignment and sequencing while minimizing makespan and total earliness–tardiness, under inter-machine waiting time constraints.

3. MATHEMATICAL MODEL

The IPFSS problem is a variant of the classical flow shop problem, which is known to be NP-hard. To address this, we propose a Mixed Integer Linear Programming (MILP) model that incorporates job-specific constraints, such as release times, waiting time limits, and due dates, alongside machine-level precedence constraints. The notation and symbol definitions used in the proposed model are summarized in Table 1. The objective is to minimize both makespan and total earliness/tardiness.

Table 1. Notation and symbol definitions

Symbol	Description
Sets and indices	
$J = \{1,2,\dots,n\}$	Set of jobs
$L = \{1,2,3\}$	Set of identical parallel production lines
$M = \{1,2,\dots,m\}$	Set of machines per line
$j, i \in J$	Job indices
$l \in L$	Production line index
$k \in M$	Machine index

Parameters	
$p_{jk} \geq 0$	Processing time of job *j* on machine *k*
$p_{ik} \geq 0$	Processing time of job *i* on machine *k*
$p_{j1} \geq 0$	Processing time of job *j* on the first machine
$d_j \geq 0$	Due date of job *j*
$\alpha_j, \beta_j \geq 0$	Weight coefficient for earliness and tardiness of job *j*
$r_j \geq 0$	Release time of job *j*
W_{max}	Maximum allowable wait time before job *j* starts
$G \gg 0$	Large positive constant
Decision variables	
$x_{jl} \in \{0,1\}$	Binary variable equal to 1 if job *j* is assigned to line *l*; 0 otherwise
$x_{il} \in \{0,1\}$	Binary variable equal to 1 if job *i* is assigned to line *l*; 0 otherwise
$y_{jikl} \in \{0,1\}$	Binary variable equal to 1 if job *j* precedes job *i* on machine *k* in line *l*; 0 otherwise
$y_{ijkl} \in \{0,1\}$	Binary variable equal to *1* if job *i* precedes job *j* on machine *k* in line *l*; 0 otherwise
$c_{jkl} \geq 0$	Completion time of job *j* on machine *k* in line *l*
$c_{j1l} \geq 0$	Completion time of job *j* on the first machine in line *l*
$c_{jml} \geq 0$	Completion time of job *j* on the last machine in line *l*
$s_{jl} \geq 0$	Start time of job *j* on the first machine in line *l*
$E_j, T_j \geq 0$	Earliness and tardiness of job *j*
C_{max}	Makespan (latest job completion time across all lines)

Objective functions

(1) Minimize Makespan.

$$\min \mathrm{C}_{\max} \tag{1}$$

(2) Minimize total earliness and tardiness.

$$\min \sum_{j\in J}\left(\alpha_j E_j + \beta_j T_j\right) \tag{2}$$

Constraints

(3) Job assignment: Each job is assigned to exactly one line.

$$\sum_{l\in L} x_{jl} = 1 \quad \forall j \in J \tag{3}$$

(4) Makespan definition: Makespan is at least the latest completion time on the last machine.

$$c_{\max} \geq c_{jml} - G\left(1 - x_{jl}\right) \quad \forall j \in J, l \in L \tag{4}$$

(5) Release time and wait time constraint.

Completion time on the first machine is linked to job start time and processing duration.

$$c_{j1l} \geq s_{jl} + p_{j1} - G\left(1 - x_{jl}\right) \quad \forall j \in J, l \in L \tag{5}$$

Start time is bounded by job release time and maximum allowable wait time.

$$r_j \leq s_{jl} \leq r_j + W_{\max} \quad \forall j \in J, l \in L$$

(6) Sequential machine processing: job j must be completed on machine $k-1$ before starting on machine k on the same line.

$$c_{jkl} \geq c_{j,k-1,l} + p_{jk} - G\left(1 - x_{jl}\right) \quad \forall j \in J, l \in L, k \in M \setminus \{1\} \tag{6}$$

(7) Machine precedence:

If job *j* precedes i, if job *j* precedes job i, job *j* must finish before i starts on the same machine and line.

$$c_{ikl} \geq c_{jkl} + p_{ik} - G\left(1 - y_{jikl}\right) \quad \forall i \neq j \in J, l \in L, k \in M \tag{7}$$

Precedence enforcement only if both jobs are assigned to the same line.

$$y_{jikl} + y_{ijkl} \leq x_{jl} \times x_{il} \quad \forall i \neq j \in J, l \in L, k \in M$$

(8) Earliness and tardiness calculation: based on completion time on the last machine and due date.

$$\begin{aligned} E_j &\geq d_j - \sum_{l\in L} c_{jml} \times x_{jl} \quad \forall j \in J \\ T_j &\geq \sum_{l\in L} c_{jml} \times x_{jl} - d_j \quad \forall j \in J \end{aligned} \tag{8}$$

(9) Non-negativity for all time-related variables.

$$c_{jkl}, s_{jl}, E_j, T_j, \mathrm{C}_{\max} \geq 0 \quad \forall j \in J, l \in L, k \in M \tag{9}$$

4. METHODOLOGY

This section presents the methodological framework developed to address the proposed scheduling problem.

4.1 Load Balancing Procedure (LBP)

The Load Balancing Procedure (LBP) is a multi-phase scheduling heuristic specifically designed to address the load imbalance problem in parallel flow shop environments with multiple machines per line. It aims to minimize the makespan and total earliness/tardiness while strictly enforcing a maximum wait time constraint before job processing begins. The procedure starts by evaluating each job's criticality score, derived from processing time and slack, to prioritize urgent and resource-intensive jobs. This phase has a time complexity of $O(n \log n + n.L.m)$, accounting for criticality scoring, sorting, and constraint-aware assignment. The remaining non-critical jobs are then distributed to balance the load across all lines, with a complexity of $O(n.L)$, based on comparative workload evaluation across lines. A final feasibility repair phase reorders and adjusts job start times to ensure full compliance with the wait time constraint; this step involves resequencing within lines and has a worst-case complexity of $O(L.n^2.m)$. Overall, the LBP achieves efficient and constraint-aware scheduling with an upper-bound complexity of $O(L.n^2.m)$, ensuring computational scalability for large, constrained production environments and demonstrating strong practical applicability in real-world industrial systems characterized by high complexity and scale. The steps of the proposed procedure are detailed in the pseudo code shown in Table 2.

Table 2. Pseudo-code for proposed LBP

```
Algorithm: Load Balancing Procedure (LBP)
Input: Job set J, production lines L, machines M,
maximum wait time $W_{max}$
Output: Job sequences S, makespan $C_{max}$, total
earliness/tardiness E/T

// Phase 1: Job Classification
for all job $j \in J$ do
    $P_j \leftarrow \sum_{k=1}^{|M|} p_{j,k}$
$P_{max} = \max_{j \in J} P_j$
for all job $j \in J$ do
    $slack_j \leftarrow \max(0, d_j - (r_j + P_j))$
    $crit_j \leftarrow (P_j / P_{jmax}) + (1 / (slack_j + 1))$
$J_{sorted} \leftarrow$ jobs sorted by $crit_j$ (descending)
$J_{crit} \leftarrow$ top $\lceil 0.4 . |J| \rceil$ jobs from $J_{sorted}$
for all line $\ell \in L$ do
    $CT_\ell \leftarrow 0$, $TL_\ell \leftarrow 0$, $S[\ell] \leftarrow \emptyset$
// Phase 2: Critical Job Assignment
for all job $j \in J_{crit}$ do
    $F \leftarrow \{ \ell \in L \mid \max(CT_\ell, r_j) - r_j \le W_{max} \}$
    if $F \neq \emptyset$ then
        $\ell^* \leftarrow \arg\min_{l \in F} (\max(CT_\ell, r_j) + P_j)$
    else
        $\ell^* \leftarrow \arg\min_{l \in L} (CT_\ell + P_j)$
    $S[\ell^*] \leftarrow S[\ell^*] \cup \{j\}$
    $CT_{\ell^*} \leftarrow \max(CT_{\ell^*}, r_j) + P_j$
    $TL_{\ell^*} \leftarrow TL_{\ell^*} + P_j$
// Phase 3: Non-Critical Job Assignment
for all job $j \in J \setminus J_{crit}$ do
    $F \leftarrow \{ \ell \in L \mid \max(CT_{\ell^*}, r_j) - r_j \le W_{max} \}$
    if $F \neq \emptyset$ then
        $\ell^* \leftarrow \arg\min_{l \in F} TL_\ell$
    else
        $\ell^* \leftarrow \arg\min_{l \in L} TL_\ell$
    $S[\ell^*] \leftarrow S[\ell^*] \cup \{j\}$
    $CT_{\ell^*} \leftarrow \max(CT_l, r_j) + P_j$
    $TL_{\ell^*} \leftarrow TL_{\ell^*} + P_j$
// Phase 4: Feasibility Repair
for all line $\ell$ in L do
    sort $S[\ell]$ by release time $r_j$
    prev_comp[1..|M|] $\leftarrow 0$
    for all job j in $S[\ell]$ do
        for k = 1 to |M| do
            if k = 1 then
                $s_{j,1} \leftarrow$ max(r_j, prev_comp[1])
                if $s_{j,1} - r_j > W_{max}$ then
                    $s_{j,1} \leftarrow r_j + W_{max}$
            else
                $s_{j,k} \leftarrow \max(prev\_comp[k], s_{j,k-1} + p_{j,k-1})$
            prev_comp[k] $\leftarrow s_{j,k} + p_{j,k}$
    $CT_\ell \leftarrow$ prev_comp[|M|]
// Phase 5: Objective Evaluation
$C_{max} \leftarrow \max_{\ell \in L} CT_\ell$
E/T $\leftarrow 0$
for all line $\ell \in L$ do
    for all job $j \in S[\ell]$ do
        $C_j \leftarrow s_{j,|M|} + p_{j,|M|}$
        E/T $\leftarrow$ E/T + $\max(0, d_j - C_j) + \max(0, C_j - d_j)$
return S, C_max, E/T
```

4.2 Integration with MOO algorithms

In the proposed approach, the Load Balancing Procedure (LBP) is embedded within both NSGA-II and MOPSO as a constraint-aware decoding mechanism. In NSGA-II, LBP is applied during the fitness evaluation steps, specifically in Step 2 to decode each randomly generated individual in the initial population, and Step 6 to evaluate the offspring population after crossover and mutation. Similarly, in MOPSO, LBP is applied in Step 2 to evaluate the initial swarm and in Step 7 to assess each particle after position updates. In both algorithms, LBP constructs feasible schedules by assigning jobs to parallel production lines while enforcing the maximum wait time constraint (Wmax) and then computes the makespan and total earliness/tardiness to form the objective vector. As shown in Figure 1, this consistent application of LBP ensures that all candidate solutions, regardless of their origin, are decoded into valid, constraint-compliant schedules. This enables both NSGA-II and MOPSO to maintain feasibility throughout the search process and effectively converge toward well-distributed, Pareto-optimal fronts in the solution space.

4.3 Taguchi method for parameter tuning

The Taguchi Method, developed by Dr. Genichi Taguchi, employs orthogonal arrays and signal-to-noise (S/N) ratios [33], to systematically optimize algorithm parameters, S/N ratios were computed using the smaller-the-better formula $(-10\, log_{10}(\frac{1}{n}\sum y_i{}^2))$ to align with our minimization objectives. Sensitivity analysis using ($\Delta S/\mathrm{N} = \max - \min$) revealed that mutation type had the strongest influence on Enhanced NSGA-II-LBP performance, with swap operations outperforming other operators by 9.9%. Population size and crossover type also showed notable effects, while crossover probability had a comparatively limited impact.

For the Enhance MOPSO-LBP algorithm, the specific configuration of the nGrid parameter demonstrated the highest level of influence among all the factors that were evaluated. In particular, the finer settings—specifically those set at values of 0.1, 5, and 3.0—were observed to significantly improve the overall algorithm performance, resulting in a measurable enhancement of approximately 25 to 26 percent. Additionally, the adjustment of velocity limits, as well as the cognitive coefficient, also exhibited considerable influence on the behavior and outcomes of the algorithm. In contrast, however, the social coefficient consistently showed much weaker sensitivity throughout the testing process, which was notably contrary to our initial expectations and earlier assumptions.

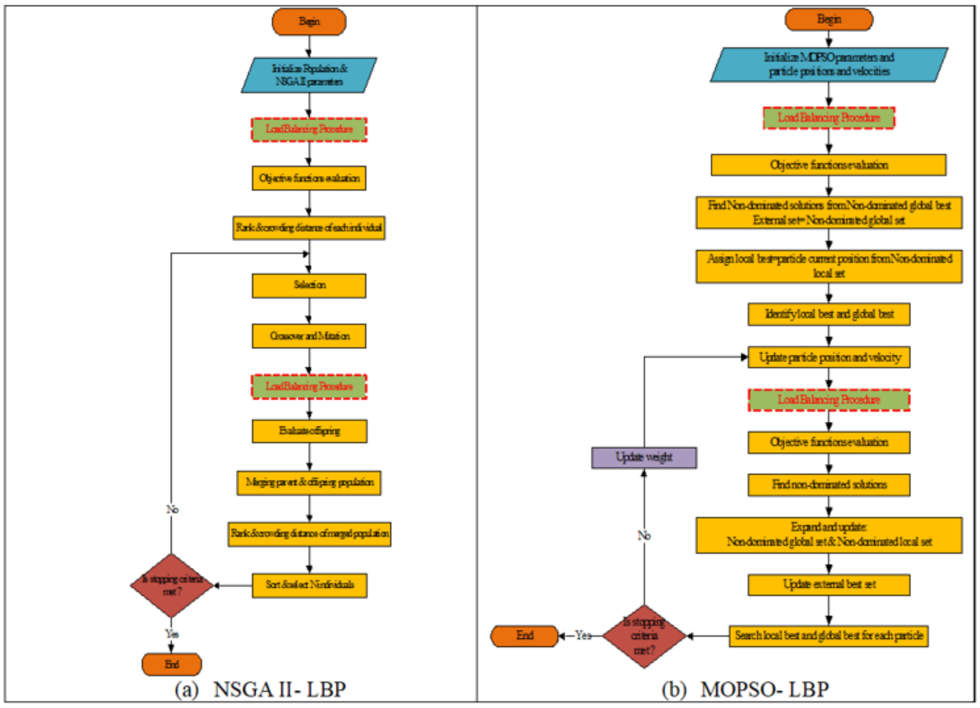

Begin
Initialize Population & NSGA II parameters
Load Balancing Procedure
Objective functions evaluation
Rank & crowding distance of each individual
Selection
Crossover and Mutation
Load Balancing Procedure
Evaluate offspring
Merging parent & offspring population
Rank & crowding distance of merged population
Sort & select N individuals
Is stopping criteria met?
No
Yes
End
(a) NSGA II- LBP
Begin
Initialize MOPSO parameters and particle positions and velocities
Load Balancing Procedure
Objective functions evaluation
Find Non-dominated solutions from Non-dominated global best External set= Non-dominated global set
Assign local best=particle current position from Non-dominated local set
Identify local best and global best
Update particle position and velocity
Load Balancing Procedure
Objective functions evaluation
Find non-dominated solutions
Expand and update: Non-dominated global set & Non-dominated local set
Update external best set
Search local best and global best for each particle
Is stopping criteria met?
Update weight
No
Yes
End
(b) MOPSO- LBP

Figure 1. Flowcharts of enhanced algorithms

Table 3 summarizes optimal parameters for NSGA-II variants, with sensitivity rankings confirming the critical role of mutation type. Enhanced NSGA-II requires a larger population size (100 vs. baseline 80) to improve Pareto front coverage, higher crossover probability (0.9 vs. 0.85) to maintain diversity under workload imbalance, and swap mutation to preserve schedule feasibility, structural choices proving more impactful than probability tuning. PMX crossover and tournament selection maintained consistent superiority across configurations, though with moderate sensitivity. Table 4 outlines Enhanced MOPSO's settings, where sensitivity analysis justified larger swarm sizes (100 vs. 80) for solution diversity, elevated inertia weight (0.75 vs. 0.45) for exploration balance, expanded velocity limits (12 vs. 10) to reduce machine idle time, and stronger cognitive coefficient (c_1=1.1) for particle coordination.

Figure 2 represents main effects plots visually corroborate these findings through slope steepness.

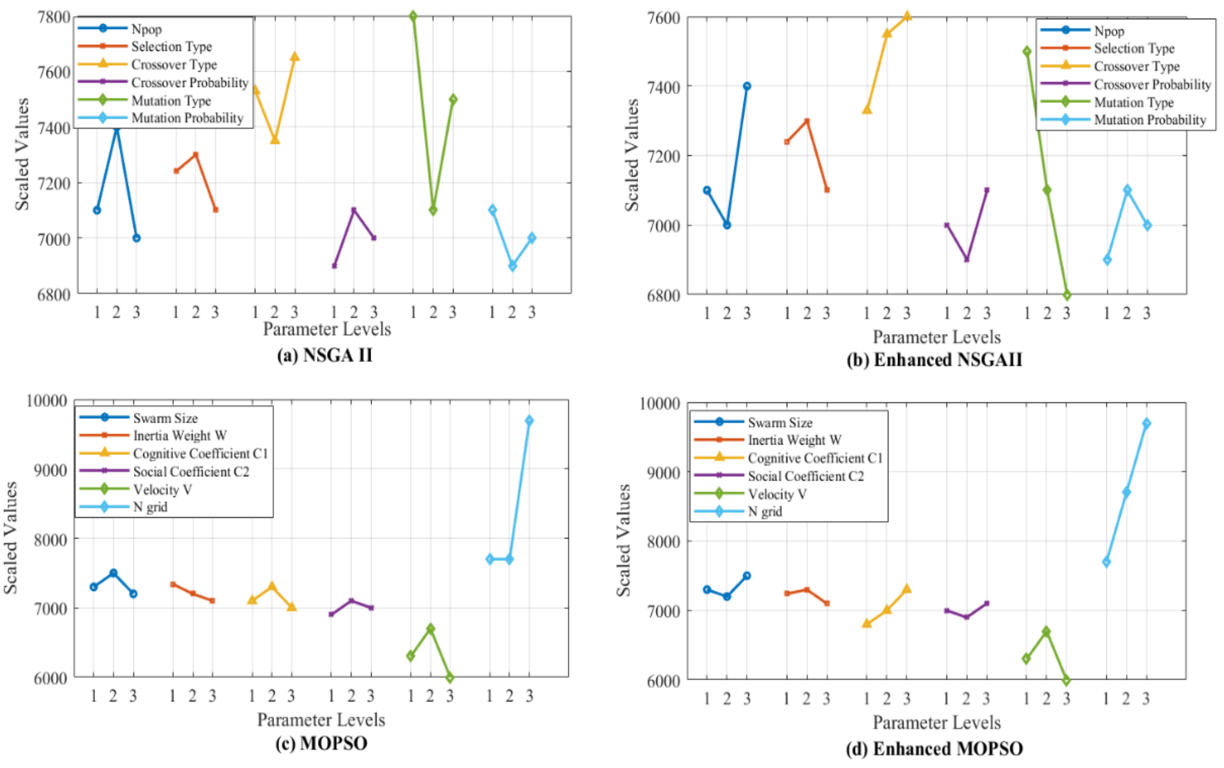


Figure 2. Main effects plot for S/N ratios of each algorithm

Table 3. Optimal parameters for NSGA-II and enhanced NSGA-II

Parameter	Level 1	Level 2	Level 3	Optimal Level for NSGA II	Optimal Level for Enhanced NSGA II	ΔS/N (db)	Rank
Population Size (N_{Pop})	50	80	100	80	100	400	2
Selection Type	Rank-Based	Tournament	Roulette Wheel	Tournament	Tournament	200	4
Crossover Type	Single-Point	Two-Point	PMX	PMX	PMX	270	3
Crossover Probability	0.75	0.85	0.9	0.85	0.9	200	4
Mutation Type	Swap	Shift	Inversion	Swap	Swap	700	1
Mutation Probability	0.12	0.18	0.22	0.12	0.18	200	4

Table 4. Optimal parameters for MOPSO and enhanced MOPSO

Parameter	Level 1	Level 2	Level 3	Optimal Level for MOPSO	Optimal Level for Enhanced MOPSO	ΔS/N (db)	Rank
Swarm Size (N_{Swarm})	50	80	100	80	100	300	4
Inertia Weight (W_w)	0.45	0.75	0.95	0.45	0.75	200	5
Cognitive Coefficient (c_1)	0.5	0.85	1.1	1.1	1.1	500	3
Social Coefficient (c_2)	0.45	0.65	1.05	0.65	1.05	200	5
Velocity Limits (V_v)	8	10	12	10	12	700	2
nGrid	(0.07, 3, 1.0)	(0.09, 4, 2.0)	(0.1, 5, 3.0)	(0.09, 4, 2.0)	(0.1, 5, 3.0)	**2000**	**1**

5. EXPERIMENTAL SETUP

This part presents the experimental setup used to evaluate the proposed approach.

5.1 Benchmark instances

The computational study used six benchmark instances obtained from https://github.com/chneau/go-taillard/tree/master/instances (accessed on 16 December 2024) to evaluate algorithm performance under varying job–machine configurations. Instance sizes included: 20_05, 20_10, 20_20, 50_05, 50_10, and 50_20. All scenarios used three identical parallel production lines.

Experiments were conducted using MATLAB R2018b on a computer with an Intel® Core™ i7-1185G7 CPU (3.00/1.80 GHz) and 32 GB RAM.

Each instance comprised 10 cases with variations in processing times, due dates, release dates, and wait-time constraints. For each case, 100 iterations were executed. The 10 best performing runs were selected and averaged to obtain a representative performance value for each case. Instance characteristics are summarized in Table 5.

Table 5. Instance characteristics

Instances Groups	Jobs (n)	Machines (m)	Parallel Lines (k)	Processing Time Range	Due Date Range	Release Date Range	Wait Time (W_{max})	Processing Time Variance (σ^2)
20_05	20	5	3	[5, 40]	[100,600]	[0, 20]	8	102.1
20_10	20	10	3	[8, 50]	[150,800]	[0, 30]	12	144.0
20_20	20	20	3	[10, 60]	[200,1000]	[0, 40]	15	208.3
50_05	50	5	3	[5, 40]	[200,1200]	[0, 20]	8	102.1
50_10	50	10	3	[8, 50]	[300,1600]	[0, 30]	12	144.0
50_20	50	20	3	[10, 60]	[400,2000]	[0, 40]	15	208.3

5.2 Performance metrics

Performance metrics are critical to impartially evaluate trade-offs between conflicting objectives in multi-objective optimization [34]. They quantify convergence (proximity to optimal solutions), diversity (spread across objectives), and uniformity (balanced distribution) [34]. For parallel flow shops, these performance metrics assess how effectively algorithms resolve workload imbalance while simultaneously minimizing objective functions.

5.2.1 Number of Pareto Solutions (NPS)

Counts non-dominated solution $x \in Q$, where no $y \in Q$ dominates $x(y \prec x)$, Higher NPS indicates broader diversity, as defined in Eq. (10):

$$\text{NPS} = \left|\{x \in Q \mid \nexists y \in Q\}\right| \text{ such that } y \prec x\}| \tag{10}$$

5.2.2 Hypervolume (HV)

Measures the volume in the objective space covered by the obtained Pareto front relative to a reference point a (nadir point). Higher HV reflects a better balance of convergence and diversity, as shown by Eq. (11):

$$\mathrm{HV}(Q)=\mathrm{Volume}\left(\bigcup_{x\in PF}\left[f_1(x),a_1\right]\times\left[f_2(x),a_2\right]\right) \tag{11}$$

5.2.3 Inverted Generational Distance (IGD)

Measures the proximity of the obtained Pareto front to the true Pareto front O^*, Lower IGD corresponds to superior convergence, as presented in Eq. (12):

$$\mathrm{IGD}\left(Q,O^*\right)=\frac{1}{\left|O^*\right|}\sum_{o\in O^*}\min_{q\in Q}\|o-q\| \tag{12}$$

where, Q represents the obtained Pareto front, O^* is the reference (true) Pareto front, and $\|o-q\|$ defines the Euclidean distance between solutions o and q.

5.2.4 Spread (Δ)

Assesses the distribution uniformity of solutions across the Pareto front. It's formulated in Eq. (13) to quantify the spread.

$$\Delta=\frac{h_f+h_l+\sum_{i=1}^{N-1}\left|h_i-\bar{h}\right|}{h_f+h_l+(N-1)\bar{h}} \tag{13}$$

where, h_f and h_l are distances to extreme solutions of the Pareto front. h_i is distance between consecutive solutions, $\bar{h}$ is the average of all h_i and N is the number of solutions.

6. RESULTS AND DISCUSSION

Figure 3 presents the performance distributions across four key metrics NPS, HV, IGD and Δ, comparing baseline algorithms with their enhanced counterparts.

(1) Number of Pareto Solutions (NPS): Enhanced NSGA-II generates 40–70% more non-dominated solutions than the baseline (5.8–10.5 vs. 3.5–7.6), while enhanced MOPSO achieves a 40–50% increase (5.2–10 vs. 3.5–7.3). This indicates that both improved algorithms sustain significantly larger and more diverse Pareto sets. A strong negative correlation with IGD (r = –0.92 for NSGA-II-LBP; r = –0.94 for MOPSO-LBP) further confirms that increased diversity contributes to better convergence.

(2) Hypervolume (HV): Both enhanced algorithms show a 30–35% increase in HV compared to their respective baselines (0.41–0.84 vs. 0.30–0.65 for NSGA-II; 0.37–0.78 vs. 0.28–0.59 for MOPSO), reflecting stronger convergence–diversity performance. This is supported by a strong negative correlation with IGD (r = –0.93), indicating that broader coverage of the Pareto front coincides with closer proximity to the true front.

(3) Inverted Generational Distance (IGD): Enhanced NSGA-II and MOPSO achieve IGD reductions of 45–65% and 40–65%, respectively, indicating significant gains in convergence (e.g., 3.88–6.35 vs. 7.31–18.41 for NSGA-II; 4.10–7.30 vs. 6.68–20.58 for MOPSO). A strong positive correlation with Spread (r = 0.89 for NSGA-II-LBP; r = 0.91 for MOPSO-LBP) reveals that convergence improvements are accompanied by more uniform solution distributions.

(4) Spread (Δ): LBP reduces Spread by 10–25%, with NSGA-II-LBP ranging from 0.26–0.49 (vs. 0.35–0.55) and MOPSO-LBP from 0.28–0.50 (vs. 0.38–0.56). This reflects improved distribution uniformity, essential for balanced scheduling. A strong negative correlation with HV (r = –0.85 for NSGA-II-LBP; r = –0.88 for MOPSO-LBP) confirms that uniformity enhances coverage quality.

The correlation coefficients were calculated based on Spearman's rank correlation approach, as presented in Eq. (14):

$$\rho=1-\frac{6\sum d_i^2}{n\left(n^2-1\right)} \tag{14}$$

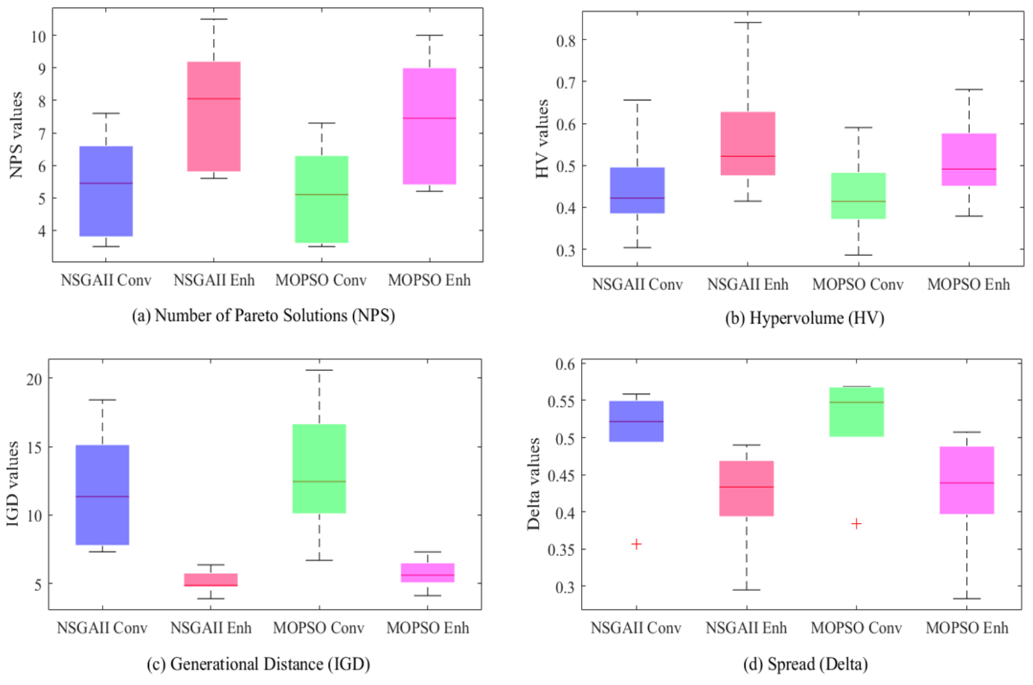


Figure 3. Boxplots of comparative performance metrics

These relationships highlight LBP's ability to reconcile historically conflicting objectives. The strong HV–IGD correlation (r ≈ –0.93) confirms that convergence and coverage improve jointly, driven by LBP's criticality scoring, which selects jobs that enhance both. The positive IGD–Δ correlation (r > 0.89) reflects LBP's capacity to produce evenly distributed, high-quality solutions through constraint-aware sequencing. Finally, the weak NPS–Δ correlation (r ≈ –0.40) demonstrates that diversity and distribution can improve concurrently a direct result of LBP's two-phase structure, which decouples job selection from load balancing.

Together, these findings confirm that the proposed approach resolves the convergence, diversity, and distribution trade-off, offering a coherent multi-objective optimization framework for complex scheduling environments.

The Pareto front visualization in Figure 4 provides valuable insight into solution quality for the 50_20 instance, illustrating how the proposed heuristic enhancements reshape the distribution of trade-offs between makespan and earliness–tardiness. Enhanced NSGA-II achieves the most balanced compromise, delivering solutions particularly well-suited for practical scheduling environments where both efficiency and due-date compliance are critical. These solutions reflect marginal trade-offs, making them highly operationally relevant. Enhanced MOPSO follows closely, also achieving a balance between objectives, though with a slight preference toward minimizing makespan. Collectively, the enhanced algorithms expand the Pareto front by 31.5% in hypervolume compared to baseline methods, indicating broader solution diversity and improved exploration of extreme trade-offs. The Pareto front of the enhanced methods forms a clear convex shape, reinforcing the soundness of the proposed Load Balancing Procedure (LBP).

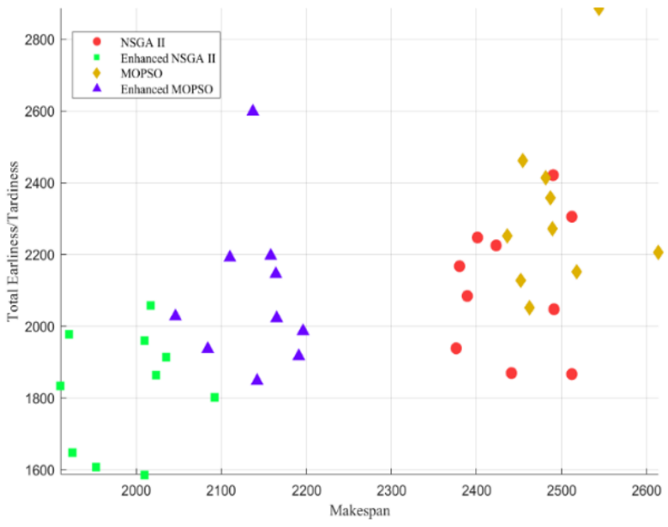
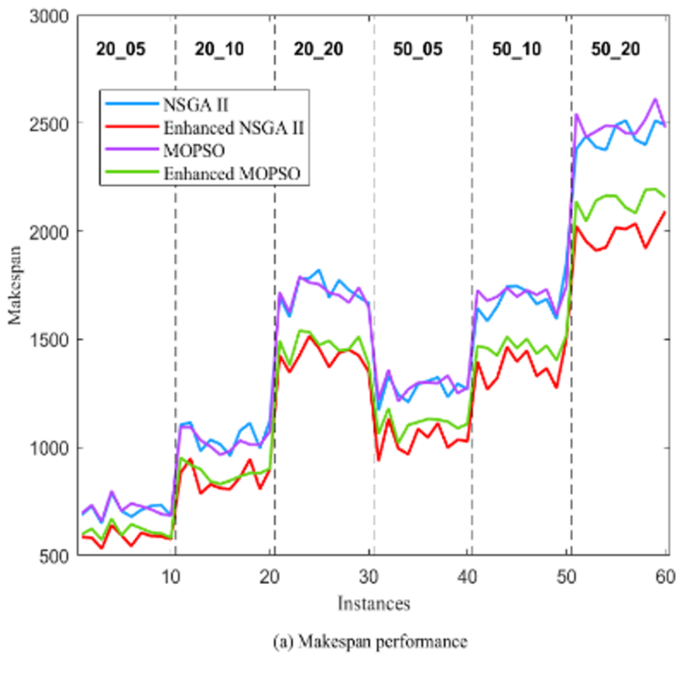


Figure 4. Pareto front for instance 50_20



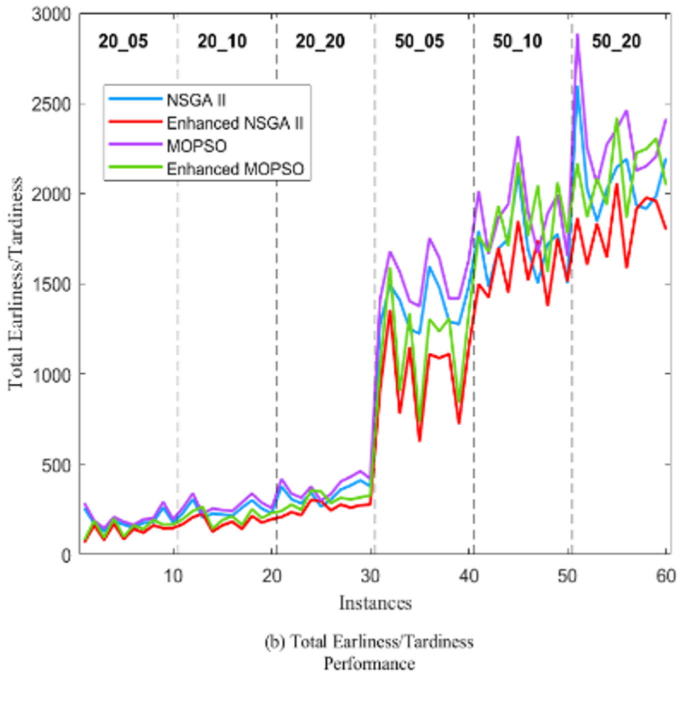


Figure 5. Comparative analysis of makespan and total earliness/tardiness

Figure 5 presents a comparative analysis of makespan and total earliness/tardiness performance. Figure 5(a) illustrates makespan values, while Figure 5(b) displays total earliness/tardiness. The results in Figure 5(a) confirm that Enhanced NSGA-II-LBP achieves an average makespan reduction of 19.2% relative to conventional NSGA-II, with improvements reaching 28.9% in large-scale instances (e.g., Instance 20_20_10: 1667 → 1350). These gains are consistent across complexity levels, ranging from 8.3% to 28.9%. In parallel, Enhanced MOPSO-LBP yields an average makespan reduction of 13.7%, with peak improvements of 22.4% (e.g., Instance 50_20_8: 2518 → 2191), and a tighter performance spread (3.1% to 22.4%), reflecting uniform scalability.

Figure 5(b) highlights similar trends for total earliness/tardiness. Enhanced NSGA-II-LBP achieves a 22.8% average reduction, with maximum gains of 39.2% in high-tardiness scenarios (e.g., Instance 50_20_5: 2421 → 2058), underscoring its strength in mitigating due-date violations through effective load balancing. Likewise, Enhanced MOPSO-LBP provides a robust 18.4% reduction, with peak improvements of 38.2% (e.g., Instance 50_20_6: 2462 → 2191), and demonstrates consistent performance across instance groups. These results emphasize the benefits of LBP integration in both algorithms, offering notable improvements in convergence quality and scheduling stability.

The runtime analysis CPU (Central Processing Unit) in Figure 6 reveals Enhanced NSGA-II and NSGA-II exhibit comparable computational times (1.1–2.8 seconds), indicating minimal overhead from the load-balancing heuristic. MOPSO consistently outperforms both NSGA-II variants, with 30–50% faster execution (e.g., 0.484–1.588 vs. 1.1–2.8 seconds for NSGA-II), attributed to swarm-based parallelism. Enhanced MOPSO incurs a marginal runtime increase over MOPSO (e.g., 0.614–1.792 vs. 0.484–1.786 seconds). Notably, all algorithms scale linearly with instance complexity (e.g., 20_05: ~1.1 seconds; 50_20: ~2.8 seconds), but MOPSO's lower absolute times highlight its suitability for real-time scheduling.

6.1 Algorithm comparison

This demonstrative example, shown in Figure 7, is based on a small instance of three parallel flow shop lines, each with five machines and a total of 21 jobs. The Gantt charts illustrate the impact of the Load Balancing Procedure (LBP) on job scheduling. Figure 7(a) represents the schedule generated by Enhanced NSGA-II, while Figure 7(b) corresponds to the baseline NSGA-II.

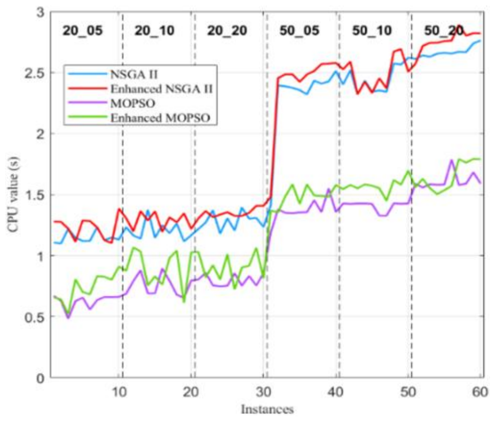


Figure 6. Comparative CPU time

In Figure 7(a), all three lines exhibit compact, well-coordinated job sequences, with operations tightly packed across machines. Final operations complete near a unified makespan threshold (~800), as indicated by the red dashed line, reflecting effective synchronization and load balancing. In contrast, Figure 7(b) displays fragmented scheduling with visible idle gaps and poor load distribution, particularly in Flow Shop Lines 2 and 3, where machine inactivity extends the makespan beyond 890. The rightward-pointing dashed arrows highlight delays in completing final operations. Furthermore, job sequencing lacks continuity, resulting in inefficient resource utilization and disrupted flow. Final job completion times vary substantially across lines, ranging from 600 to nearly 900, indicating a significant imbalance. This discrepancy results in early completions on some lines, leading to premature inventory and delays on others, ultimately compromising overall schedule stability.

Figure 8 illustrates the load distribution performance of the enhanced algorithms across three identical parallel production lines. Both enhanced approaches produce tightly clustered workload values (e.g., Enhanced NSGA-II: [771.5, 757.5, 757.5]; Enhanced MOPSO: [854.0, 830.5, 819.0]), with inter line variance remaining below 5%. This outcome reflects the effectiveness of the integrated heuristic in achieving balanced utilization across resources. In contrast, the baseline algorithms exhibit pronounced disparities in workload allocation (e.g., NSGA-II: [484.5, 417.5, 258.5]; MOPSO: [1250, 1625, 1300]), with deviations exceeding 40–60%, thereby highlighting the limitations of conventional methods in achieving load balance.

6.2 Statistical validation

To evaluate the effectiveness of the proposed heuristic enhancements, statistical analyses were performed using Analysis of Variance (ANOVA) and paired t-tests across multiple benchmark instances (Tables 6 and 7). The ANOVA results include partial eta-squared (η^2) values and 95% confidence intervals (CIs) to quantify the effect size of algorithmic differences. Paired t-tests are further supported by Cohen's d to indicate the magnitude of observed differences.

Table 6 presents the ANOVA outcomes across all evaluated metrics. The p-values were consistently below 0.001, leading to the rejection of the null hypothesis of equal performance among algorithms. Large effect sizes (η^2 = 0.38–0.93) with tight confidence intervals suggest that algorithm choice substantially influences performance across benchmark instances. Notably, hypervolume (HV) and inverted generational distance (IGD) show strong discriminative power, with η^2 reaching 0.72 [0.67, 0.77] for IGD, indicating that the enhancements significantly improved convergence and diversity. The most pronounced values were observed for CPU time ($\eta^2 > 0.83$), reflecting measurable runtime differences due to the Load Balancing.

Following the ANOVA results, which decisively rejected the null hypothesis H_0, post-hoc paired t-tests were conducted to evaluate whether the mean performance of two related algorithmic approaches differs significantly. To control the family-wise error rate (FWER) arising from multiple comparisons, the Bonferroni correction was applied, adjusting the significance threshold, as in Eq. (15):

$$\alpha_{\text{adjusted}} = \frac{\alpha_{desired}}{number \text{ of comparisions}} = \frac{0.05}{6} \approx 0.0083 \qquad (15)$$

As shown in Table 7, statistically significant results (α = 0.0083) are accompanied by large or very large effect sizes in most cases ($|d| > 0.8$). For example, Enhanced NSGA-II exhibited strong gains in solution quantity (NPS) over the baseline (Cohen's d = 1.19–2.59), while Enhanced MOPSO achieved substantial reductions in CPU time compared to NSGA-II (d = –3.55 to –12.61). Differences between Enhanced MOPSO and its baseline were negligible ($|d| < 0.2$), confirming comparable efficiency. Enhanced NSGA-II also maintained similar computational times to its baseline, with small effect sizes (d = –0.27 to 0.99), suggesting the added procedures introduce only minor runtime variation.

Overall, these findings demonstrate that the proposed enhancements significantly improve solution quality across multiple objectives. While some increase in computational time is observed, the overall benefits in convergence, diversity, and robustness clearly outweigh the cost, particularly in applications where scheduling quality is a critical requirement.

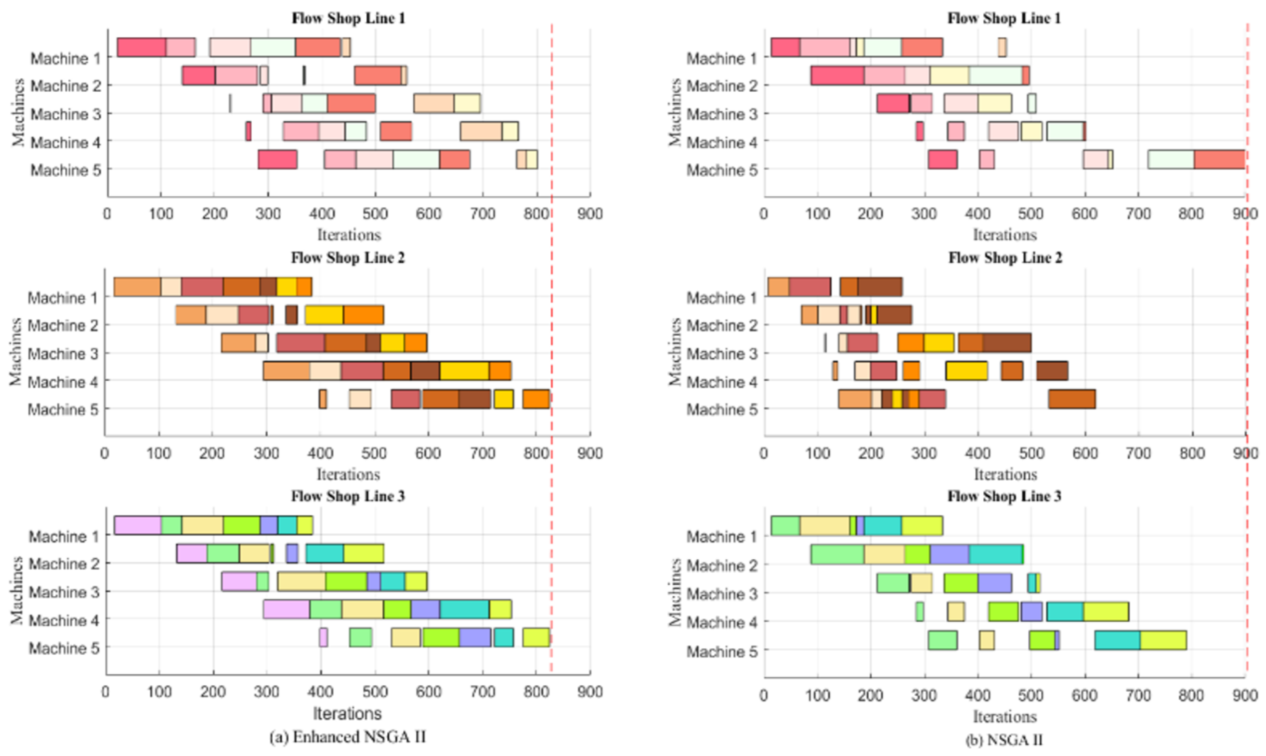


Figure 7. Gantt chart comparison of NSGA II–LBP and NSGA-II

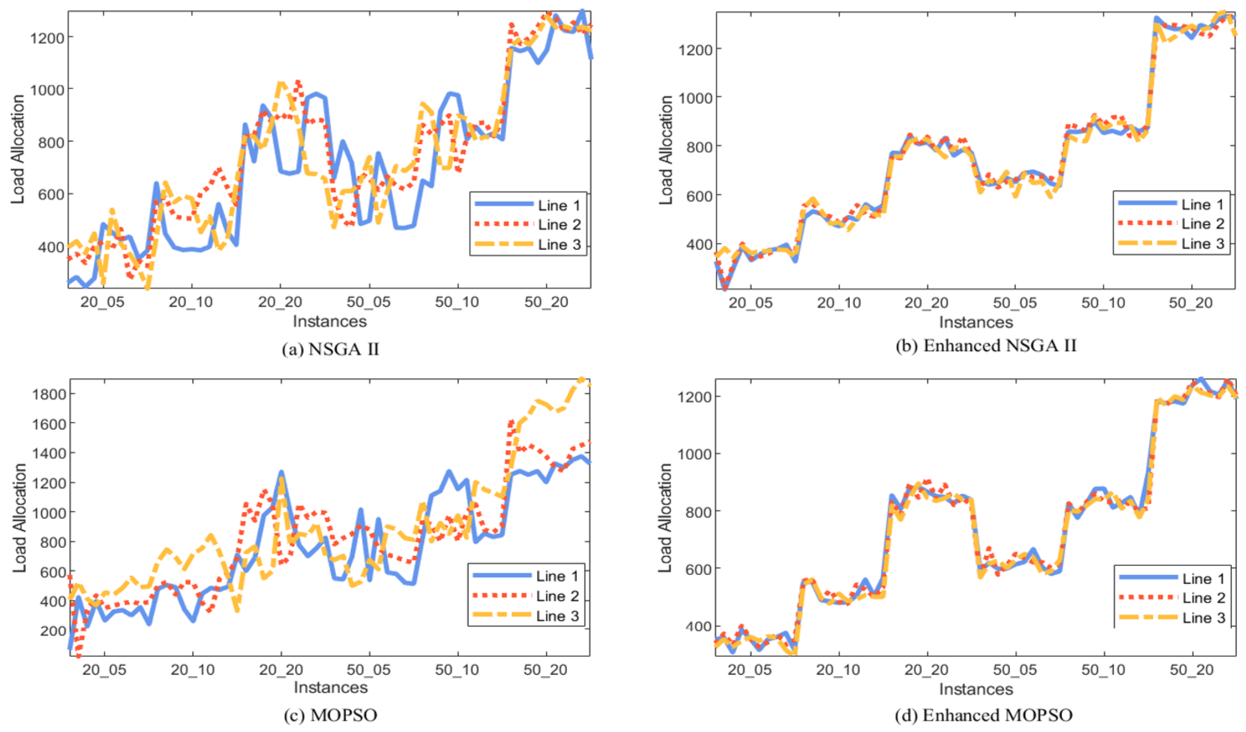


Figure 8. Comparison of load distribution between enhanced and conventional algorithms

Table 6. ANOVA summury table

Instance Groups	**Metrics**														
	NPS			**HV**			**Δ**			**IGD**			**CPU**		
	F-stat	**P-value**	**η² [95% CI]**	**F-stat**	**P-value**	**η² [95% CI]**	**F-stat**	**P-value**	**η² [95% CI]**	**F-stat**	**P-value**	**η² [95% CI]**	**F-stat**	**P-value**	**η² [95% CI]**
20_05	10.85	3.2E-05	0.48 [0.43, 0.53]	16.20	7.9E-07	0.57 [0.52, 0.62]	9.21	0.0001	0.43 [0.38, 0.48]	30.45	5.55E-10	0.72 [0.67, 0.77]	129.47	2.4E-19	0.92 [0.89, 0.95]

20_10	11.31	2.3E-05	0.49 [0.44, 0.54]	15.07	1.6E-06	0.56 [0.51, 0.61]	9.49	9.3E-05	0.44 [0.39, 0.49]	19.50	1.11E-07	0.62 [0.57, 0.67]	60.34	4E-14	0.83 [0.79, 0.87]
20_20	10.44	4.4E-05	0.47 [0.42, 0.52]	10.11	5.7E-05	0.46 [0.41, 0.51]	7.56	0.0005	0.39 [0.34, 0.44]	15.31	1.39E-06	0.56 [0.51, 0.61]	135.08	1.2E-19	0.92 [0.89, 0.95]
50_05	8.76	0.0002	0.42 [0.37, 0.47]	7.49	0.0005	0.38 [0.33, 0.43]	9.63	8.3E-05	0.45 [0.40, 0.50]	22.27	2.50E-08	0.65 [0.60, 0.70]	75.22	1.4E-15	0.86 [0.82, 0.90]
50_10	9.43	9.7E-05	0.44 [0.39, 0.49]	8.89	0.0002	0.43 [0.38, 0.48]	11.83	1.5E-05	0.50 [0.45, 0.55]	27.65	1.87E-09	0.70 [0.65, 0.75]	167.55	3.3E-21	0.93 [0.90, 0.96]
50_20	15.75	1E-06	0.57 [0.52, 0.62]	10.42	4.4E-05	0.46 [0.41, 0.51]	15.80	1E-06	0.57 [0.52, 0.62]	14.58	2.24E-06	0.55 [0.50, 0.60]	134.77	1.3E-19	0.92 [0.89, 0.95]

Table 7. Paired t-test results

Algorithm Comparison	**Metrics**	**Instances Groups**											
		20_05		**20_10**		**20_20**		**50_05**		**50_10**		**50_20**	
		P-value	**Cohen's d**	**P-value**	**Cohen's d**	**P-value**	**Cohen's d**	**P-value**	**Cohen's d**	**P-value**	**Cohen's d**	**P-value**	**Cohen's d**
Improved NSGA-II vs. NSGA-II	NPS	0.0002	2.59	0.0005	1.19	0.0001	–1.33	0.0002	1.59	0.0002	1.73	0.0001	1.56
	HV	0.0001	1.88	0.0001	1.57	0.0009	1.41	0.0003	1.37	0.0019	1.02	0.0036	1.23
	Δ	0.0028	-1.85	0.0002	-1.90	0.0001	-1.45	0.0001	-1.88	0.0001	-1.26	0.0001	-2.05
	IGD	0.0006	-1.86	0.0001	-2.28	0.0002	-2.40	0.0002	-1.83	0.0001	-2.10	0.0002	-1.27
	CPU	0.0197	0.77	0.0178	0.72	0.0453	0.72	0.6136	0.43	0.3996	-0.27	0.0194	0.99
MOPSO vs. NSGA-II	NPS	1.0000	0.00	0.6410	–0.12	0.8536	–0.08	0.2778	-0.43	0.6600	-0.15	0.6236	-0.19
	HV	0.2417	-0.68	0.6158	-0.16	0.8200	-0.10	0.4576	-0.18	0.5566	-0.23	0.5684	-0.19
	Δ	0.3836	0.36	0.7419	0.14	0.1934	0.42	0.3826	0.33	0.3344	0.46	0.7206	-0.15
	IGD	0.1309	0.40	0.0347	1.02	0.2336	-0.43	0.1355	-0.41	0.3237	1.02	0.1396	0.704
	CPU	1.6E-14	-5.77	4.1E-10	-3.55	2.7E-12	-5.39	2.4E-11	-6.88	3.5E-09	-5.82	7.2E-19	-12.61
Improved MOPSO vs. NSGA-II	NPS	0.0055	1.17	0.0021	0.97	0.0036	–1.05	0.0315	0.78	0.0080	1.02	0.0021	1.16
	HV	0.0037	1.00	0.0004	1.68	0.0013	0.92	0.0081	0.53	0.0106	0.80	0.0347	0.88
	Δ	0.0055	-1.06	0.0036	-1.26	0.0026	-0.98	0.0074	-1.10	0.0058	-0.91	0.0004	-1.84
	IGD	0.0010	-1.63	0.0002	-2.13	0.0030	-1.12	0.0004	-1.81	0.0002	-1.91	0.0006	-1.31
	CPU	5.6E-09	-3.21	2.6E-05	-1.66	4.8E-08	-2.78	1.5E-10	-5.08	5.9E-09	-5.41	1.7E-07	-5.29
Improved NSGA-II vs. MOPSO	NPS	0.0001	1.26	0.0002	1.41	0.0003	1.25	0.0001	1.36	0.0001	1.74	0.0000	1.61
	HV	0.0001	2.31	0.0001	1.54	0.0000	1.84	0.0001	2.07	0.0006	1.47	0.0000	1.94
	Δ	0.0008	-1.86	0.0004	-1.69	0.0018	-2.01	0.0009	-1.44	0.0000	-1.64	0.0002	-2.05
	IGD	0.0000	-3.33	0.0007	-1.53	0.0000	-2.63	0.0001	-2.08	0.0000	-2.27	0.0002	-1.51
	CPU	8.6E-13	5.52	7.4E-12	4.19	1.6E-16	4.76	1.7E-06	3.85	2.7E-08	5.03	2.3E-08	-5.85
Improved NSGA-II vs. Improved MOPSO	NPS	0.2895	0.42	0.6669	0.11	0.4093	0.28	0.4143	0.24	0.7907	0.11	0.3860	0.27
	HV	0.0023	1.47	0.0601	0.61	0.4962	0.20	0.1962	0.55	0.3404	1.18	0.1356	0.52
	Δ	0.4164	-0.34	0.9318	0.06	0.8658	-0.09	0.8579	0.08	0.3500	-0.30	0.1399	-0.72
	IGD	0.1511	-1.17	0.0994	-0.57	0.0111	-1.39	0.0569	-0.74	0.0392	-0.61	0.0130	-0.85
	CPU	4.0E-09	3.17	8.2E-07	2.13	1.1E-07	2.43	5.6E-06	3.14	7.2E-08	3.81	2.5E-09	-5.39
Improved MOPSO vs. MOPSO	NPS	0.0041	1.04	0.001	1.59	0.0058	1.01	0.0080	1.46	0.0044	1.72	0.0001	1.65
	HV	0.0011	1.50	0.0005	1.27	0.0015	0.95	0.0474	0.64	0.0027	1.18	0.0000	2.78
	Δ	0.0014	1.21	0.0042	-1.44	0.0092	-1.62	0.0056	-1.15	0.0021	-1.18	0.0001	-2.12
	IGD	0.0000	-3.30	0.0008	-1.56	0.0052	-1.25	0.0003	-1.72	0.0001	-2.00	0.0003	-1.51
	CPU	0.0131	1.29	0.0203	0.97	0.0146	0.97	0.0087	1.06	1.2E-06	3.43	0.2245	-0.49

7. CONCLUSION AND FUTURE WORK

This study addressed the Identical Parallel Flow Shop Scheduling Problem (IPFSSP) using a bi-objective model that minimizes makespan and total earliness–tardiness under waiting time constraints. A dynamic load-balancing heuristic was integrated into NSGA-II and MOPSO to improve solution quality. The enhanced algorithms, NSGA-II–LBP and MOPSO–LBP, showed clear gains in convergence, diversity, and workload distribution, as confirmed by numerical results and statistical tests. MOPSO–LBP, in particular, achieved better computational efficiency, making it suitable for real-time scheduling in manufacturing systems.

Future work should adapt these approaches to dynamic settings with uncertain job arrivals, machine failures, or varying processing times. In such cases, predictive and adaptive strategies may support real-time decision-making. The efficiency of MOPSO–LBP also suggests potential for use in time-constrained industrial environments. Overall, the proposed method offers a strong and scalable solution for complex scheduling problems.

REFERENCES

[1] Komaki, G.M., Sheikh, S., Malakooti, B. (2018). Flow shop scheduling problems with assembly operations: A review and new trends. International Journal of Production Research, 57(10): 2926-2955. https://doi.org/10.1080/00207543.2018.1550269

[2] Lachtar, N., Driss, I. (2023). Application of ant colony optimization for job shop scheduling in the pharmaceutical industry. Journal Européen des Systèmes Automatisés, 56(5): 713-723. https://doi.org/10.18280/jesa.560501

[3] Bedhief, A.O. (2021). Comparing mixed-integer programming and constraint programming models for the hybrid flow shop scheduling problem with dedicated machines. Journal Européen des Systèmes Automatisés, 54(4): 591-597. https://doi.org/10.18280/jesa.540408

[4] Ren, J.F., Ye, C.M., Li, Y. (2020). A two-stage optimization algorithm for multi-objective job-shop scheduling problem considering job transport. Journal Européen des Systèmes Automatisés, 53(6): 915-924. https://doi.org/10.18280/jesa.530617

[5] Graham, R.L. (1969). Bounds on multiprocessing timing anomalies. SIAM Journal on Applied Mathematics, 17(2): 416-429. https://doi.org/10.1137/0117039

[6] Ribas, I., Companys, R. (2021). A computational evaluation of constructive heuristics for the parallel blocking flow shop problem with sequence-dependent setup times. International Journal of Industrial Engineering Computations, 12(3): 321-328. https://doi.org/10.5267/j.ijiec.2021.1.004

[7] Ribas, I., Companys, R., Tort-Martorell, X. (2017). Efficient heuristics for the parallel blocking flow shop scheduling problem. Expert Systems with Applications, 74: 41-54. https://doi.org/10.1016/j.eswa.2017.01.006

[8] Ribas, I., Companys, R., Tort-Martorell, X. (2021). An iterated greedy algorithm for the parallel blocking flow shop scheduling problem and sequence-dependent setup times. Expert Systems with Applications, 184. https://doi.org/10.1016/j.eswa.2021.115535

[9] Kamarposhti, M.A., Lorenzini, G., Solyman, A.A.A. (2021). Locating and sizing of distributed generation sources and parallel capacitors using multiple objective particle swarm optimization algorithm. Mathematical Modelling of Engineering Problems, 8(1): 10-24. https://doi.org/10.18280/mmep.080102

[10] Xhafa, F., Abraham, A. (2010). Computational models and heuristic methods for grid scheduling problems. Future Generation Computer Systems, 26(4): 608-621. https://doi.org/10.1016/j.future.2009.11.005

[11] Lama, P., Zhou, X.B. (2012). Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud. In Proceedings of the 9th International Conference on Autonomic Computing, pp. 63-72. https://doi.org/10.1145/2371536.2371547

[12] Krauter, K. Buyya, R. Maheswaran, M. (2002). A taxonomy and survey of grid resource management systems for distributed computing. Software: Practice and Experience, 32(2): 135-164. https://doi.org/10.1002/spe.432

[13] Varela, M.L.R., Trojanowska, J., Carmo-Silva, S., Costa, N.M.L., Machado, J. (2017). Comparative simulation study of production scheduling in the hybrid and the parallel flow. Management and Production Engineering Review, 8(2): 69-80. https://doi.org/10.1515/mper-2017-0019

[14] Al-Salem, A. (2004). A heuristic to minimize makespan in proportional parallel flow shops. International Journal of Computing & Information Sciences, 2(2): 98.

[15] Zhang, X.D., Velde, S.V.D. (2012). Approximation algorithms for the parallel flow shop problem. European Journal of Operational Research, 216(3): 544-552. https://doi.org/10.1016/j.ejor.2011.08.007

[16] Dong, J.M., Tong, W.T., Luo, T.B., Wang, X.S., Hu, J.L., Xu, Y.F., Lin, G.H. (2017). An FPTAS for the parallel two-stage flowshop problem. Theoretical Computer Science, 657: 64-72. https://doi.org/10.1016/j.tcs.2016.04.046

[17] Tong, W.T., Miyano, E. Goebel, R. Lin, G.H. (2018). An approximation scheme for minimizing the makespan of the parallel identical multi-stage flow-shops. Theoretical Computer Science, 734: 24-31. https://doi.org/10.1016/j.tcs.2017.09.018

[18] Dong, J.M., Jin, R.Y., Luo, T.B., Tong, W.T. (2020). A polynomial-time approximation scheme for an arbitrary number of parallel two-stage flow-shops. European Journal of Operational Research, 281(1): 16-24. https://doi.org/10.1016/j.ejor.2019.08.019

[19] Ribas, I., Companys, R., Tort-Martorell, X. (2019). An iterated greedy algorithm for solving the total tardiness parallel blocking flow shop scheduling problem. Expert Systems with Applications, 121: 347-361. https://doi.org/10.1016/j.eswa.2018.12.039

[20] Kim, M.G., Yu, J.M., Lee, D.H. (2014). Scheduling algorithms for remanufacturing systems with parallel flow-shop-type reprocessing lines. International Journal of Production Research, 53(6): 1819-1831. https://doi.org/10.1080/00207543.2014.962112

[21] Hatami, S., Calvet, L., Fernández-Viagas, V., Framiñán, J.M., Juan, A.A. (2018). A simheuristic algorithm to set up starting times in the stochastic parallel flowshop problem. Simulation Modelling Practice and Theory, 86: 55-71. https://doi.org/10.1016/j.simpat.2018.04.005

[22] Rajeswari, N., Shahabudeen, P. (2008). Bicriteria parallel flow line scheduling using hybrid population-based heuristics. The International Journal of Advanced Manufacturing Technology, 43: 799-804. https://doi.org/10.1007/s00170-008-1754-4

[23] Wang, H.F., Fu, Y.P., Huang, M., Huang, G.Q., Wang, J.W. (2017). A NSGA-II based memetic algorithm for multiobjective parallel flowshop scheduling problem. Computers & Industrial Engineering, 113: 185-194. https://doi.org/10.1016/j.cie.2017.09.009

[24] Johnson, S.M. (1954). Optimal two - and three - stage production schedules with setup times included. Naval Research Logistics Quarterly, 1(1): 61-68. https://doi.org/10.1002/nav.3800010110

[25] Nawaz, M., Enscore Jr, E.E., Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega, 11(1): 91-95. https://doi.org/10.1016/0305-0483(83)90088-9

[26] Campbell, H.G., Dudek, R.A., Smith, M.L. (1970). A heuristic algorithm for the n job, m machine sequencing problem. Management Science, 16(10): B630-B637. http://www.jstor.org/stable/2628231.

[27] Li, F.N., Lang, S., Tian, Y., Hong, B.Y., Rolf, B., Noortwyck, R., Schulz, R., Reggelin, T. (2024). A transformer-based deep reinforcement learning approach for dynamic parallel machine scheduling problem with family setups. Journal of Intelligent Manufacturing. https://doi.org/10.1007/s10845-024-02470-8

[28] Li, G.Z., Zhang, L. (2024). Discrete-event simulation integrates an improved NEH algorithm for practical flowshop scheduling problems in the satellite industry. Applied Sciences, 14(21): 9755. https://doi.org/10.3390/app14219755

[29] Razali, L.F., Nawawi, A. (2024). Optimization of Permutation Flowshop Schedulling Problem (PFSP) using First Sequence Artificial Bee Colony (FSABC) algorithm. Progress in Engineering Application and Technology, 5(1): 369-377. https://doi.org/10.30880/peat.2024.05.01.039

[30] Khan, A.R. (2024). Dynamic load balancing in cloud computing: Optimized RL-based clustering with multi-objective optimized task scheduling. Processes, 12(3): 519. https://doi.org/10.3390/pr12030519

[31] Jena, U.K., Das, P.K., Kabat, M.R. (2022). Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment. Journal of King Saud University - Computer and Information Sciences, 34(6): 2332-2342. https://doi.org/10.1016/j.jksuci.2020.01.012

[32] Pradhan, A., Bisoy, S.K. (2022). A novel load balancing technique for cloud computing platform based on PSO. Journal of King Saud University - Computer and Information Sciences, 34(7): 3988-3995. https://doi.org/10.1016/j.jksuci.2020.10.016

[33] Aswal, A., Jha, A., Tiwari, A., Modi, Y.K. (2019). CNC turning parameter optimization for surface roughness of aluminium-2014 alloy using Taguchi methodology. Journal Européen des Systèmes Automatisés, 52(4): 387-390. https://doi.org/10.18280/jesa.520408

[34] Najlaoui, B., Basha, M.S., Raouf, E.A., El-Hafez, H.A. (2024). An improved multi-objective approach based on competitive optimization algorithm and its engineering design application. Journal Européen des Systèmes Automatisés, 57(5): 1337-1347. https://doi.org/10.18280/jesa.570509