Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

A hybrid bacterial foraging optimization algorithm for makespan minimization in permutation flow shop scheduling

Moisés Silva-Muñoz[a*], Mauricio Palma[a], Guillermo Fuertes[b,c], Manuel Vargas[b], Luis Quezada[b], Pedro Palominos[b] and Victor Parada[a]

[a]*Informatics Engineering Department, University of Santiago of Chile, Chile*
[b]*Industrial Engineering Department, University of Santiago de Chile, Chile*
[c]*Faculty of Engineering, Science and Technology, Universidad Bernardo O'Higgins, Chile*

C H R O N I C L E

Article history:
Received April 15 2026
Received in Revised Format May 30 2026
Accepted June 22 2026
Available online June 22 2026

Keywords:
Permutation flow shop scheduling problem
Makespan minimization
Bacterial foraging optimization
Hybrid metaheuristics

A B S T R A C T

Permutation flow shop scheduling problems (PFSP) with makespan minimization are a standard benchmark in production scheduling, where solution quality depends on effective search in a discrete permutation space. This paper proposes HBFO, a hybrid discrete bacterial foraging optimization method that implements bacterial foraging dynamics through permutation-preserving operators and integrates intensification and diversification mechanisms. HBFO is evaluated on Taillard's 120 benchmark instances using 20 independent runs per instance. Performance is assessed using the best-run makespan, the relative percentage deviation (RPD) relative to best-known reference values, and run-to-run variability. Compared with recent advanced metaheuristics reported on the same benchmark families, HBFO achieves the best or tied-best best-run makespan in 115 of the 120 instances, with five instances where another method attains a lower best-run value. The RPD results show frequent exact matches to the reference values and predominantly low-percentage deviations otherwise. Configuration-level summaries further indicate that both average deviation and variability vary across Taillard instance families.

© 2026 by the authors; licensee Growing Science, Canada

1. Introduction

Permutation flow shop scheduling problems (PFSP) model production environments in which each job must be processed on multiple machines in the same order. The scheduling decision is to select a single job permutation that determines the processing sequence on every machine (Johnson, 1954). A common objective in this setting is makespan minimization, where makespan is the completion time of the last job and is widely used as a throughput-related indicator in production scheduling (Ruiz & Maroto, 2005). Because of its practical relevance and role as a standardized benchmark, makespan-oriented PFSP has become a common testbed for assessing heuristic and metaheuristic scheduling methods under shared evaluation practices and instance libraries (Taillard, 1993; Davendra et al., 2013).

Formally, the PFSP considers a set of n jobs $J = \{1, \dots, n\}$ to be processed on a set of m machines $M = \{1, \dots, m\}$, in the same technological order for all jobs. Each job j requires a processing time $p_{j,k}$ on machine k, with the usual capacity constraints that each machine processes at most one job at a time and each job is processed by at most one machine at a time. Under the permutation assumption, a feasible schedule is a permutation π of the n jobs that is applied identically on every machine. Let $C_{j,k}(\pi)$ denote the completion time of job j on machine k under π; these values are obtained using the standard flow shop completion-time recursions (Marichelvam et al., 2017; Framinan et al., 2004).

The makespan is $C_{\max}(\pi) = C_{\pi(n),m}(\pi)$. For a fixed permutation π, feasibility is enforced by two conditions: machine capacity (no overlap on the same machine) and technological precedence (each job must follow the machine order). Accordingly, job $\pi(i)$ can start on machine k only when the machine becomes available after $\pi(i-1)$ completes on k and

* Corresponding author
E-mail moises.silva@usach.cl (M. Silva-Muñoz)
ISSN 1923-2934 (Online) - ISSN 1923-2926 (Print)
2026 Growing Science Ltd.
doi: 10.5267/j.ijiec.2026.6.005

when the job becomes available after completing on machine $k-1$; the start time is therefore determined by the later of these two completion times, after which processing $p_{\pi(i),k}$ yields completion on machine k (Zaied et al., 2021).

As problem size grows, the number of feasible permutations increases rapidly, making exhaustive optimization impractical beyond small instances (Garey et al., 1976). This motivates the use of heuristic and metaheuristic search strategies for competitive makespan minimization under limited computational budgets (Palmer, 1965; Gupta, 1971; Nawaz et al., 1983; Khurshid et al., 2021). Consequently, modern PFSP research commonly evaluates algorithms on standardized benchmark suites and emphasizes comparative evidence obtained under consistent reporting conventions, rather than isolated best-case schedules (Ruiz & Maroto, 2005).

Competitive PFSP metaheuristics, therefore, focus strongly on how permutation representations are manipulated during search. Performance often depends more on the effectiveness of permutation-level operators and neighborhood structures—those that enable controlled diversification and strong local refinement—than on the metaheuristic's high-level metaphor. This principle is reflected in successful hybrid and memetic PFSP approaches that combine global exploration with explicit, neighborhood-based intensification to achieve robust makespan performance across benchmark families (Zobolas et al., 2009).

Bacterial Foraging Optimization (BFO) is a biomimetic, population-based framework that models search through chemotaxis-like movements, reproduction, and elimination–dispersion events (Passino, 2002). It has inspired a broad family of variants and applications, but transferring bacterial foraging dynamics to PFSP is nontrivial because PFSP search occurs in a discrete permutation space. As a result, chemotactic "moves" must be redefined as permutation-preserving transformations and paired with systematic refinement to be effective on scheduling landscapes (Zhao et al., 2016). More broadly, surveys of BFO variants underscore that practical performance often depends on carefully designed movement and population-update mechanisms to improve convergence and robustness (Sayadi et al., 2010; Guo et al., 2021).

Building on these considerations, this paper proposes HBFO, a hybrid discrete BFO method tailored to PFSP with the makespan objective. HBFO implements bacterial foraging dynamics with PFSP-compatible permutation operators and incorporates intensification and diversification mechanisms consistent with hybrid PFSP design practice. The method is evaluated on Taillard's benchmark suite using standard reporting conventions (best-run makespan, relative percentage deviation (RPD) from best-known reference values, and run-to-run variability) and compared with competitive baselines reported in recent literature.

The primary contributions of this work are threefold. First, we formulate HBFO as a hybrid discrete BFO method for makespan-oriented PFSP and specify permutation-aware operators that implement its core foraging dynamics, including an SA-style acceptance mechanism embedded in the exploration stage. Second, we provide a benchmark-based evaluation on Taillard's PFSP instances and report solution quality using standard measures with respect to best-known reference values (e.g., RPD). Third, we complement best-run comparisons with stability-focused evidence by reporting repeated-run dispersion and convergence traces across instance configurations.

The rest of the paper is organized as follows: Section 2 reviews related work and benchmarking practices; Section 3 describes the proposed HBFO method; Section 4 presents the experimental results; Section 5 discusses the findings; and Section 6 concludes.

2. Literature review

2.1. Benchmarks and evaluation conventions in PFSP

Benchmark-driven evaluation using shared instance sets is standard in makespan-oriented PFSP and supports cross-study comparability. Taillard's benchmark suite remains the most widely used experimental basis for makespan-focused PFSP studies (Taillard, 1993), and major reviews consider standardized benchmarks and normalized reporting necessary for organizing and comparing algorithmic contributions (Ruiz & Maroto, 2005; Zaied et al., 2021). In line with this benchmark-driven practice, Table 1 provides a concise map of the methodological landscape considered in this review, linking evaluation conventions to the solution families most frequently used in makespan PFSP and to the specific gap that motivates HBFO. Table 1 is intended as a compact synthesis; the following subsections focus on the rows most directly connected to discrete BFO adaptations for PFSP and to the robustness mechanisms motivating HBFO.

A recurring issue in PFSP benchmarking is whether the reference makespan is a proven optimum or a best-known value that serves as a practical upper bound. This distinction shapes how deviations should be interpreted, particularly for widely used families such as Taillard's instances, where best-known values are often used as the comparison baseline rather than confirmed optima (Taillard, 1993). To improve comparability across instance scales, PFSP studies commonly report normalized measures such as RPD alongside the best values achieved. For stochastic methods, variability summaries from repeated runs are also reported to document dispersion under the same experimental budget (Framinan et al., 2004; Zaied et al., 2021).

2.2. Metaheuristics for PFSP

The "general PFSP metaheuristics" row in Table 1 reflects a central empirical reality: makespan-focused PFSP is dominated by approximate search because exact optimization becomes impractical as instance size grows and the number of feasible permutations increases rapidly (Garey et al., 1976). Consequently, a wide range of population- and trajectory-based methods has been applied, including evolutionary algorithms, swarm-inspired optimizers, and single-solution schemes such as simulated annealing and iterated greedy (Reeves, 1995; Lian et al., 2008; Kirkpatrick et al., 1983; Ruiz & Stützle, 2007).

Across these paradigms, effectiveness depends heavily on operating directly in permutation space via feasible neighborhood transformations, rather than on the high-level metaphor alone. Many influential PFSP solvers incorporate constructive heuristics, such as NEH, as initializers or as components of intensification, reflecting the long-standing role of problem-aware structure in guiding search under bounded computational budgets (Nawaz et al., 1983; Zaied et al., 2021). This operator-centric view provides the main bridge to hybrid frameworks: once competitive performance depends on the strength of refinement within permutation neighborhoods, method design naturally shifts toward explicit orchestration of intensification and diversification.

2.3. Hybrid frameworks for PFSP

The hybrid and memetic rows in Table 1 reflect a consistent PFSP pattern: benchmark-competitive solvers typically combine global exploration with explicit intensification over the permutation neighborhood. This design logic recurs in makespan-oriented PFSP work, where local refinement is treated as a primary driver of high-quality solutions under standardized reporting conventions (Ruiz & Maroto, 2005).

Memetic and iterated-greedy patterns exemplify this coupling by combining population-based search or structured perturbation with explicit local improvement (Zobolas et al., 2009; Tseng & Lin, 2009; Tseng & Lin, 2010; Ruiz & Stützle, 2007; Arık, 2021; Mishra & Shrivastava, 2024). Cooperative metaheuristics further demonstrate that coordinating complementary components can improve performance under shared benchmark protocols (Vallada & Ruiz, 2009). Recent studies continue to follow this principle, integrating adaptive or learning-driven components to adjust search behavior without compromising the use of permutation-feasible operators and refinement mechanisms (Li et al., 2023; Pan et al., 2021; Zhou et al., 2023; Lu et al., 2024; Lu et al., 2025). This hybrid perspective is directly relevant to discretizing BFO: if competitive PFSP performance is typically achieved through explicit intensification within a controlled diversification schedule, a discrete BFO design must specify how chemotactic "movement" implements constructive neighborhood refinement while preserving budget-aware exploration.

2.4. BFO and discrete adaptations for permutation problems

The BFO foundations row in Table 1 highlights a structured search cycle—chemotaxis, reproduction, and elimination–dispersion—that naturally alternates refinement with rediscovery (Passino, 2002; Liu & Passino, 2002). Subsequent work examined these foundations and documented variants to improve convergence speed and robustness across problem types, often by modifying movement rules and population-update mechanisms (Das et al., 2009; Guo et al., 2021).

However, transferring BFO to permutation problems requires redefining "movement" as feasible discrete transformations and ensuring that these transformations support constructive neighborhood refinement rather than disruptive changes. This makes operator design and intensification choices central to implementing bacterial foraging dynamics in scheduling landscapes. Numerous "improved BFO" variants and hybridizations have been proposed—including chaos-enhanced schemes and learning-augmented movement or population-update strategies—for a range of optimization and modeling tasks (Zhang et al., 2018; Gan et al., 2020; Chen et al., 2017; Chen et al., 2020; Meng et al., 2024; Xiong et al., 2023; Awad & Hafez, 2022). For PFSP, the methodological question is not whether BFO can be improved in general, but which discrete operators, acceptance policies, and population-management mechanisms are required to achieve benchmark-competitive refinement within a fixed iteration budget.

2.5. BFO-based methods for PFSP and remaining limitations

The "closest prior" row in Table 1 identifies Zhao et al. (2016) as the primary reference for adapting bacterial foraging to PFSP. Their CLS-BFO formulation preserves the canonical BFO cycle but implements chemotaxis via a continuous-space update mechanism that is then mapped to a job permutation using a continuous-to-discrete encoding (SPV). Within this chemotaxis design, Zhao et al. augment the standard tumble/swim behavior with (i) a Differential Evolution (DE) operator invoked when tumble updates fail to improve, and (ii) a chaotic local search stage based on logistic mapping applied after chemotaxis, in which candidate positions generated by the chaotic sequence are accepted if they yield improvement. As a result, local refinement in Zhao et al. is not expressed as an explicit insertion/swap neighborhood descent directly on permutations, but rather as continuous perturbation plus chaos-driven search, followed by SPV-based decoding to a feasible PFSP permutation (Zhao et al., 2016).

While Zhao et al. (2016) represent the closest PFSP-oriented BFO reference, hybridizations of bacterial foraging with simulated annealing have been reported more broadly in the scheduling literature, particularly for shop scheduling variants

such as the flexible job shop problem, where SA is typically introduced as an auxiliary intensification/diversification component to reduce premature stagnation (Vital-Soto et al., 2020). However, these hybrids are generally presented as an external local-search module or as a separate hybrid stage within the overall metaheuristic pipeline, rather than as an SA-style acceptance rule embedded in the PFSP-specific foraging movement itself. Consistent with this distinction, the closest PFSP-oriented BFO reference (Zhao et al., 2016) enhances chemotaxis through DE and chaos-based search mechanisms, without reporting an SA-type probabilistic acceptance mechanism as part of the PFSP BFO cycle. This gap motivates the explicit integration of SA-style acceptance within HBFO's exploration stage under fixed-budget PFSP benchmarking.

Motivated by this gap, HBFO defines bacterial foraging directly in permutation space and instantiates chemotaxis as systematic, permutation-level refinement. Chemotactic movement is realized through PFSP-feasible neighborhood operators and hybrid intensification (alternating insertion and swap), so refinement is performed on the schedule representation itself rather than on continuous coordinates followed by decoding. Moreover, HBFO couples this permutation-level chemotaxis with explicit population-management mechanisms under a fixed iteration budget—phase-dependent move generation with bounded uphill acceptance during exploration, GA-based reproduction, and duplicate control—so that diversification and robustness are managed within the discrete foraging cycle rather than delegated to continuous-space exploration plus mapping. From the perspective of PFSP benchmarking practice (Sections 2.2–2.3), this distinction is relevant because robust performance often depends on how intensification, diversification, and population control are coordinated under a fixed computational budget. Accordingly, the remaining gap concerns robustness mechanisms under fixed budgets—particularly acceptance control, reproduction design, and duplicate control—beyond merely defining discrete chemotactic moves and local improvement in a BFO-to-PFSP adaptation (Table 1).

Table 1
Summary of related PFSP solution families and BFO-based adaptations, highlighting the gap motivating HBFO.

Theme/family	Representative references	Core idea (what it does)	Limitation relevant to this paper
Benchmarking & reporting conventions (PFSP)	Taillard (1993); Ruiz & Maroto (2005)	Shared benchmark suites and normalized metrics enable cross-paper comparison	Best-known values are often used as references; cross-paper comparisons depend on heterogeneous protocols
General PFSP metaheuristics (makespan)	Framinan et al. (2004); Zaied et al. (2021)	Metaheuristics dominate due to combinatorial growth; permutation-operator design drives performance	Reviews describe broad families but do not prescribe a unified hybrid design tailored to BFO dynamics
Evolutionary / swarm-inspired PFSP solvers	Reeves (1995); Lian et al. (2008)	Population-based search adapted to permutation space via specialized operators	Performance is highly sensitive to permutation operators and intensification design
Hybrid & memetic PFSP frameworks	Zobolas et al. (2009); Tseng & Lin (2009); Ruiz & Stützle (2007)	Global exploration combined with explicit neighborhood refinement (memetic/IG-style)	Performance depends on how intensification and diversification are orchestrated under fixed budgets
Learning-assisted PFSP pipelines	Pan et al. (2021); Lu et al. (2025); Zhou et al. (2023)	Learning is used as a policy or guidance module while refinement remains metaheuristic	Often relies on training data; reproducibility depends on model and training details
BFO foundations & improved variants	Passino (2002); Guo et al. (2021); Zhang et al. (2018)	Chemotaxis–reproduction–elimination–dispersal cycle; variants target convergence and stability	Standard BFO is continuous-space; direct transfer to permutation problems is nontrivial
BFO adapted to PFSP (closest prior)	Zhao et al. (2016)	Implements BFO for PFSP via continuous-space chemotaxis with DE/chaotic local search and SPV decoding to permutations	Relies on continuous-to-permutation mapping; room for robustness mechanisms under fixed budgets (acceptance, reproduction, duplicate control)
HBFO (PFSP makespan)	Proposed method	Phase-dependent dispersal with insertion/swap refinement, embedded SA-style acceptance, GA-based reproduction, and duplicate control	Limits: makespan/Taillard scope; competitor stability metrics unavailable (comparisons rely on published best-run values)

3. Methods

3.1. Problem setting and HBFO overview

This work addresses the PFSP under the permutation assumption and the makespan criterion. A candidate solution is a job permutation $\pi = [\pi(1), \ldots, \pi(n)]$, and its objective value is the induced makespan $C_{max}(\pi)$, computed from the standard flow shop completion-time recursions. In HBFO, each bacterium represents one permutation π, and fitness is given directly by $C_{max}(\pi)$.

HBFO instantiates the canonical bacterial foraging cycle—chemotaxis, reproduction, and elimination–dispersion—within a discrete permutation space. Chemotaxis performs structured refinement of permutations using PFSP-feasible neighborhood moves and an acceptance policy. Reproduction updates the population via genetic recombination and mutation of permutations. Elimination–dispersion reintroduces diversity through rank-biased resetting while preserving protected elites. The algorithm operates under a fixed iteration budget shared across all instances, ensuring a consistent termination rule across the benchmark suite.

Algorithm 1 summarizes the complete HBFO workflow, from mixed seeding and initial refinement to the main epoch loop, which features phase-dependent movement, GA-based reproduction, and rank-biased elimination–dispersion under duplicate control. The following subsections detail the key components of Algorithm 1.

Algorithm 1
HBFO workflow for PFSP, including phase-dependent movement, GA reproduction, and rank-biased elimination–dispersion under duplicate control.

Require: PFSP instance; population size S; epochs N_{ep}; reproduction cycles N_{re}; SA acceptance parameters $(T_{0,\text{base}}, \gamma_{\max})$; mutation probability P_m; duplicate threshold ε; rank-based elimination controls

Ensure: Best permutation π^{best} and makespan C^{best}

Initialization (**Seed = MIX**)
Generate heuristic seeds $\{\pi^{NEH}, \pi^{\text{PALMER}}, \pi^{\text{GUPTA}}\}$ and place them in protected slots.
Fill the remaining S positions with randomized permutations (root-anchored generator)
Evaluate all bacteria using $C_{max}(\cdot)$; cull near-duplicates using Kendall $-\tau$ distance threshold ε
Apply hybrid local refinement (alternating insertion and swap) to selected individuals
Set (π^{best}, C^{best}) to the best bacterium
for $e = 1$ **to** N_{ep} **do** ▷ Epoch loop (elimination–dispersion blocks)
  Determine phase (exploration early, exploitation late)
  for $r = 1$ **to** N_{re} **do** ▷ Reproduction-cycle loop
    Chemotaxis/movement (non-protected bacteria)
    for each non-protected bacterium π_i **do**
      if exploration phase **then**
        Propose candidate via *exploration-phase move generator* (swap-chains + barriers + SA-style acceptance)
      Else
        Propose candidate via *exploitation-phase move generator* (greedy improvements only)
      end if
      Refine candidate with hybrid local search (alternating insertion and swap)
      Update π_i; update (π^{best}, C^{best}) if improved
    end for
    Reproduction (GA-based)
    Rank bacteria by fitness; generate offspring via permutation crossover
    Mutate offspring with probability P_m (randomly chosen permutation moves)
    Replace non-protected bacteria according to fitness and duplicate-control policy
  end for
  Elimination–dispersion (rank-biased)
  for each non-protected bacterium π_i **do**
    Eliminate with probability increasing in rank; if eliminated, reinitialize via root-anchored generator.
  end for
  Optionally cull near-duplicates and reinsert current best if needed.
end for
return (π^{best}, C^{best})

3.2. Chemotaxis as hybrid permutation refinement

Chemotaxis is implemented as a reproducible hybrid local improvement process applied to the current permutation π. Starting from π, HBFO explores a combined neighborhood by alternating insertion and swap moves under a best-improving policy, so that each accepted step strictly decreases $C_{max}(\pi)$. The refinement stops when no improving move is found in the explored neighborhood (or when the configured effort limit is reached), and the resulting locally improved permutation becomes the bacterium's state for the next stages of the foraging cycle.

3.3. Phase-dependent move generation and SA-style acceptance

To balance exploration and exploitation within a fixed iteration budget, HBFO uses phase-dependent move generation. In the exploration phase, a candidate permutation is generated by applying a short swap-chain (a sequence of k random swap moves) to the current permutation π. Candidate proposals are screened by barrier rules to avoid wasted evaluations and overly disruptive perturbations: (i) near-duplicate candidates whose normalized Kendall–τ distance to previously evaluated permutations is below ε are rejected, and (ii) candidates whose makespan deterioration exceeds a fixed cap are discarded. Improving candidates are accepted deterministically; otherwise, candidates that pass the barrier rules may be accepted under a simulated annealing (SA) rule, allowing occasional uphill transitions during exploration. In the exploitation phase, the same candidate-generation mechanism is used, but acceptance is greedy (only improving moves), shifting the search toward systematic refinement and reducing diversification. Any accepted candidate can then be refined using the hybrid insertion/swap local improvement routine.

In the implementation, duplicate control uses $\varepsilon = 0.02$, and the exploration stage is active for the first half of the elimination–dispersion epochs (EXPLOR_FRAC $= 0.5$). During exploration, SA-style acceptance is initialized with $T_{0,\text{base}} = 75.0$ and is combined with a direct-worsening barrier $\gamma_{\max} = 0.02$ and up to TRY_EXPLOR $= 5$ candidate proposals per movement step. The effective temperature is reduced across epochs by scaling $T_0 = T_{0,\text{base}} \max\{0.2,\ 1 - e/N_{ep}\}$, which gradually tightens acceptance as the search progresses. In contrast, the rank-biased elimination–dispersion operator is intentionally more conservative because it is already a disruptive diversification mechanism: it uses a lower local temperature $T_0 = 60.0$, together

with a stricter worsening cap of 3% (applied as a hard cutoff on the makespan deterioration) to prevent excessive degradation when reinitializing or perturbing non-elite individuals. This separation reflects two roles: exploration in the main movement loop tolerates limited uphill transitions to escape local basins under controlled barriers, whereas elimination–dispersion restricts deterioration more tightly to preserve baseline solution quality while still injecting diversity.

3.4. Initialization and elite protection

HBFO initializes the population using a mixed seeding strategy (Seed = MIX) that combines heuristic constructions with randomized permutations. Under the default mixed-seeding configuration, three heuristic permutations are generated using the NEH, Palmer, and Gupta procedures and placed in the final three population positions reserved for protected elite seeds. The remaining individuals are generated by a randomized constructor to promote diversity. Protected individuals are excluded from disruptive operators and are preserved across reproduction and elimination–dispersion, providing elite protection throughout the run.

3.5. Reproduction, mutation, and duplicate control

After movement/refinement, HBFO updates the population through a GA-based reproduction step. Parents are selected from the ranked population via tournament selection, and offspring are generated using a permutation crossover consistent with the Order Crossover (OX) pattern: a contiguous segment [a..b] is copied from the first parent into the child, and the remaining positions are filled by scanning the second parent in circular order while skipping already-used jobs; an analogous construction produces the second child (Eiben & Smith, 2015). Mutation is then applied with probability P_m by sampling one of three permutation-preserving operators (swap, insertion, or segment inversion). To mitigate stagnation, offspring are filtered against near-duplicates using the same Kendall $-\ \tau$ criterion ($\varepsilon = 0.02$); if an offspring is too similar to its replacement target or to the current best global individual, additional mutation attempts are forced before insertion.

3.6. Rank-biased elimination–dispersion and termination

Elimination–dispersion is applied once per epoch as a rank-biased diversification mechanism: lower-ranked non-elite bacteria have a higher probability of being dispersed, while protected elites are excluded. Concretely, bacteria are ranked by fitness and a normalized rank fraction rankFrac $\in [0,1]$ is computed (best $\rightarrow 0$, worst $\rightarrow 1$); the elimination probability follows $p_{\text{elim}} = p_{ed} \cdot (\text{rankFrac})^{\beta}$, with $p_{ed} = 0.75$ and $\beta = 1.5$, so the probability increases smoothly with worse rank. To avoid repeatedly disrupting high-quality regions, candidates that are near-duplicates of the current global best are not dispersed, and a small elite subset is explicitly protected (including heuristic seeds and a small fraction of top-ranked individuals). After each elimination–dispersion pass, the base probability is cooled multiplicatively as $p_{ed} \leftarrow 0.90 * p_{ed}$ with a floor at $p_{ed} = 0.01$, reducing the aggressiveness of dispersion as epochs advance.

Dispersed individuals are regenerated using the same randomized constructor as initialization, and the resulting candidates are subject to the same duplicate-control policy. The algorithm terminates after the fixed number of epochs and reproduction cycles specified by the parameter configuration, with bounded inner-loop lengths for local refinement and SA-style exploration, ensuring a consistent computational budget across benchmark instances.

4. Results

HBFO was evaluated on Taillard's 120 PFSP benchmark instances from OR-Library using $R = 20$ independent runs per instance under a fixed termination budget. For instance i, algorithm a, and run r, the makespan is denoted $C_{i,a}^{(r)}$, and we report the best-run value as $C_{i,a}^{min} = min_{r=1,\dots,R} C_{i,a}^{(r)}$. Solution quality is normalized via the RPD with respect to the reference value C_i^*; when C_i^* corresponds to a best-known upper bound, RPD is computed against that upper bound. Results are summarized at complementary levels: Table 2 reports configuration-level best-run outcomes (Best/Tied-best/Worse), Table 3 details the five instances where HBFO is not best, Table 4 reports configuration-level averages of RPD and run-to-run dispersion with 95% confidence intervals, and Figure 1 presents boxplots of instance-level RPD by configuration. Statistical significance of paired best-run differences against each competitor is reported in Table 5. Convergence behavior is presented through the representative-instance traces in Figure 2 and summarized quantitatively in Table 6.

4.1. Configuration-level best-run outcomes

Table 2 aggregates the instance-level best-run comparisons into configuration-level counts for each Taillard family (10 instances per configuration). The classification is based on the per-instance best-run makespan $C_{i,a}^{\min}$ reported for each method: Best indicates that HBFO achieves a strictly smaller best-run makespan than all competing methods for that instance, Tied-best indicates that HBFO matches the minimum best-run makespan achieved by at least one competitor, and Worse indicates that another method attains a smaller $C_{i,a}^{\min}$. Across all 120 instances, HBFO is strictly best in 91 cases, tied-best in 24 cases, and not best in 5 cases. At the configuration level, the number of strictly best outcomes ranges from 3 (C1) to 10 (C8, C11, and C12), while non-best outcomes occur only in a subset of families—specifically C5 (2 instances), C6 (1 instance), and C9 (2 instances). The following subsection isolates these non-best instances and reports the corresponding best competitor and absolute gap (Table 3).

Table 2
Configuration-level counts of HBFO Best/Tied-best/Worse outcomes on Taillard's PFSP benchmark.

Configuration	Best	Tied-best	Worse	Total
C1	3	7	0	10
C2	7	3	0	10
C3	8	2	0	10
C4	5	5	0	10
C5	7	1	2	10
C6	9	0	1	10
C7	5	5	0	10
C8	10	0	0	10
C9	8	0	2	10
C10	9	1	0	10
C11	10	0	0	10
C12	10	0	0	10
Overall	**91**	**24**	**5**	**120**

4.2. Instances where HBFO is not best

Table 3 isolates the five benchmark instances for which HBFO does not achieve the lowest best-run makespan. For each instance, the table reports HBFO's best-run value $C_{i,HBFO}^{\min}$, the best competing method (the method achieving the smallest best-run value $C_{i,a}^{\min}$ among the compared algorithms), the competitor's best-run value $C_{i,comp}^{\min}$, and the absolute gap $\Delta = C_{i,HBFO}^{\min} - C_{i,comp}^{\min}$. Three of the non-best cases correspond to instances where PFSP_NET_IGA achieves a lower best-run value (Ta042, Ta046, Ta055), and two correspond to instances where HMSA achieves a lower best-run value (Ta081, Ta090). Across these five instances, the absolute gaps range from 4 to 29 makespan units. To complement this instance-level identification of non-best cases, the next subsection reports configuration-level averages and measures of dispersion.

Table 3
Taillard instances where HBFO does not achieve the best-run makespan, including the best competitor and the absolute gap.

Instance	Best competitor	$C_{i,HBFO}^{min}$	$C_{i,comp}^{min}$	Gap Δ
Ta042	PFSP_NET_IGA	2909	2904	5
Ta046	PFSP_NET_IGA	3019	3006	13
Ta055	PFSP_NET_IGA	3646	3642	4
Ta081	HMSA	6310	6306	4
Ta090	HMSA	6525	6496	29

4.3. Configuration-level performance with confidence intervals

Table 4 summarizes HBFO performance by Taillard configuration (10 instances per configuration). For each $(n \times m)$ family, it reports the mean RPD (%) across the 10 instances and the mean run-to-run standard deviation of makespan (in makespan units), both averaged across the same instances, together with 95% confidence intervals computed using a t-interval across instances (df = 9). Because the intervals are t-based and computed from a sample of ten instances per configuration, they may extend slightly below zero due to sampling variability.

Across configurations, mean RPD ranges from 0.000 in C2, C4, and C7 to 1.663 in C9, with intermediate values of 0.013 (C3), 0.058 (C10), 0.952 (C6), and 1.410 (C11). The mean run-to-run standard deviation ranges from 0.462 (C4) and 1.051 (C1) to 12.544 (C6), 15.661 (C9), 16.728 (C11), and 21.419 (C12), with corresponding confidence intervals reported in Table 4. Here, "Mean Std. Dev." denotes the average (across the 10 instances in a configuration) of the per-instance run-to-run standard deviation computed from the 20 independent runs.

The configuration-level averages in Table 4 are not strictly monotonic with respect to n or m. Figure 1 complements Table 4 by showing the distribution of instance-level RPD values within each configuration.

Table 4
Mean RPD and mean run-to-run makespan standard deviation by Taillard configuration, with 95% confidence intervals.

Instance Configuration	Mean RPD (%) [95% CI]	Mean Std. Dev. [95% CI]
C1(20 × 5)	0.041 [–0.051, 0.132]	1.051 [–0.264, 2.366]
C2(20 × 10)	0.000 [0.000, 0.000]	1.447 [0.572, 2.323]
C3(20 × 20)	0.013 [–0.017, 0.043]	2.396 [1.780, 3.013]
C4(50 × 5)	0.000 [0.000, 0.000]	0.462 [–0.149, 1.072]
C5(50 × 10)	0.624 [0.296, 0.953]	5.915 [4.028, 7.801]
C6(50 × 20)	0.952 [0.801, 1.103]	12.544 [8.699, 16.389]
C7(100 × 5)	0.000 [0.000, 0.000]	1.410 [–0.141, 2.961]
C8(100 × 10)	0.144 [0.046, 0.243]	6.847 [3.580, 10.114]
C9(100 × 20)	1.663 [1.356, 1.969]	15.661 [13.346, 17.976]
C10(200 × 10)	0.058 [0.023, 0.094]	11.253 [6.807, 15.698]
C11(200 × 20)	1.410 [1.159, 1.661]	16.728 [14.443, 19.014]
C12(500 × 20)	0.577 [0.452, 0.702]	21.419 [18.134, 24.705]

4.4. Distribution of instance-level RPD across configurations

Figure 1 presents boxplots of HBFO's instance-level RPD values grouped by Taillard configuration (C1–C12). For each configuration, the box summarizes the distribution of the 10 instance-level RPD values within that configuration (median and interquartile range), with whiskers indicating the spread beyond the quartiles. Several configurations show distributions concentrated near zero RPD (e.g., C2, C4, and C7), whereas others exhibit higher medians and wider within-configuration dispersion, including configurations whose instance-level distributions are centered above 1% (notably C9 and C11). Intermediate distributions are observed in configurations such as C5 and C12, whereas low but nonzero RPD values with limited spread are observed in C8 and C10. We next report paired instance-wise significance tests comparing HBFO to each competitor.

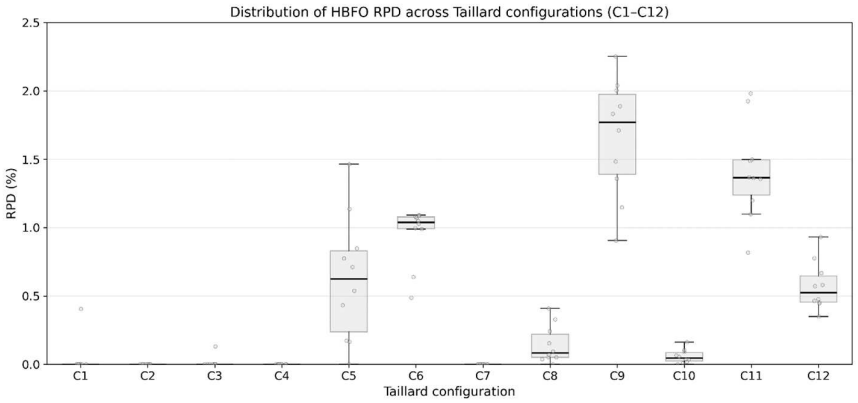

Fig. 1. Boxplots of HBFO instance-level RPD across Taillard configurations (C1–C12).

4.5. Statistical significance of best-run differences

Table 5 reports two-sided Wilcoxon signed-rank tests for paired, per-instance best-run makespan differences between HBFO and each competitor across the 120 Taillard instances. For each instance i, the paired difference is defined as $\Delta_i = C_{i,HBFO}^{\min} - C_{i,comp}^{\min}$, so negative values favor HBFO. Instances with $\Delta_i = 0$ (ties) are excluded from the signed-rank calculation and are reflected in the effective nonzero sample size N. The table reports the median paired difference (in makespan units), the signed-rank statistic W, two-sided p-values with Holm correction for the seven comparisons, and the rank-biserial effect size.

Across competitors, the Wilcoxon tests indicate that HBFO's paired best-run differences are consistently favorable at the instance level. In particular, against SGA, EGA, HGA, DSOMA, and DWWO, the statistic collapses to $W = 0$ with strongly negative median differences and rank-biserial effects essentially equal to -1, reflecting uniformly smaller best-run makespans for HBFO across the full set of instances. Against HMSA and PFSP_NET_IGA, the differences remain statistically significant after Holm correction, with negative median differences and large-magnitude rank-biserial effects; however, the effect sizes are slightly smaller, and PFSP_NET_IGA shows a reduced N due to a larger number of ties. Because competitor data are available only as per-instance best-run values from the literature, these tests quantify instance-wise consistency of best-run differences rather than run-to-run distributional differences across methods.

Table 5
Wilcoxon signed-rank tests for paired best-run makespan differences between HBFO and each competitor on the 120 Taillard instances.

Competitor	N (nonzero)	Median Δ (HBFO−Comp)	W	p-value	p (Holm)	Effect (rank-biserial)
SGA	120	−163.5	0.0	1.97E−21	1.38E−20	−1.000
EGA	120	−117.5	0.0	1.97E−21	1.38E−20	−1.000
HGA	120	−1710.5	0.0	1.97E−21	1.38E−20	−1.000
DSOMA	120	−642.0	0.0	1.97E−21	1.38E−20	−1.000
DWWO	120	−1676.0	0.0	1.97E−21	1.38E−20	−1.000
HMSA	120	−24.0	74.5	1.26E−20	2.51E−20	−0.979
PFSP_NET_IGA	96	−23.0	78.0	1.98E−16	1.98E−16	−0.966

4.6. Convergence traces on representative instances

Fig. 2 shows the convergence of HBFO on one representative instance from each Taillard configuration (C1–C12), plotting the best-so-far RPD (%) over 25 recorded chemotaxis steps across 20 independent runs. For each configuration, the thick line denotes the per-step median trajectory, the shaded region indicates the interquartile range (Q1–Q3), and the dashed lines show the per-step minimum and maximum across runs, making step-to-step variability explicit. Insets provide a zoomed view of low-RPD configurations to improve readability on a common y-axis scale. The 25 recorded steps correspond to the initial population state (step 0) plus one recorded best-so-far update after each of the 24 main-loop movement cycles in the algorithm (4 epochs × 6 reproduction cycles per epoch); all subplots use the same RPD y-axis scale.

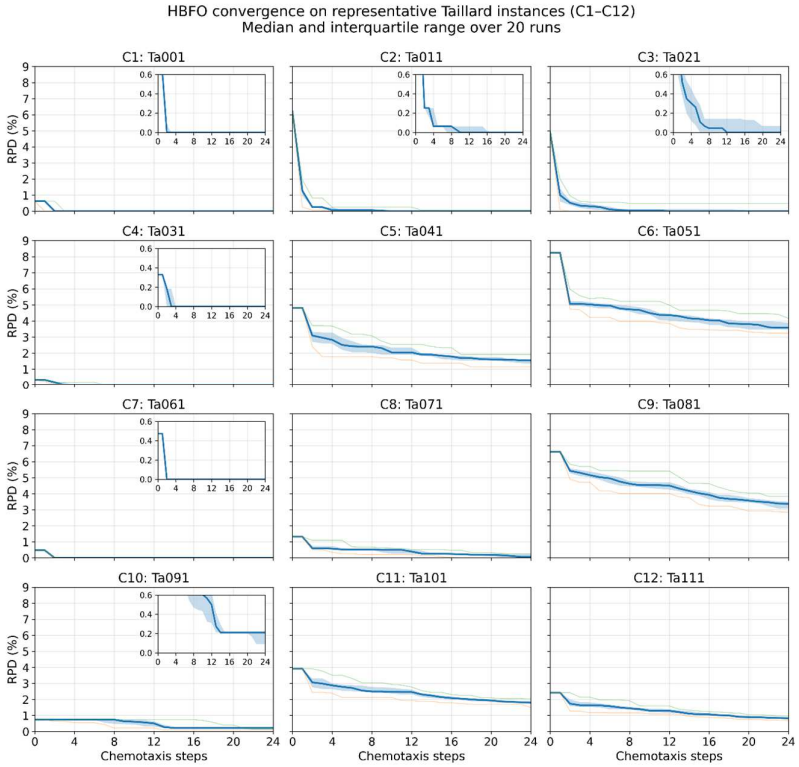

Fig. 2. Convergence traces of HBFO on representative Taillard instances (one per configuration). Median best-so-far RPD with interquartile range over 20 runs; insets zoom low-RPD cases.

4.7. Convergence summary on representative instances

Table 6 complements the convergence traces in Fig. 2 by providing a run-level summary for one representative instance per Taillard configuration (20 independent runs).

Table 6
Convergence summary on representative Taillard instances (one per configuration; 20 runs): final best-so-far RPD median/IQR and median step to 95% improvement.

Config	Instance	Final RPD median (%)	Final RPD IQR (%)	Step to 95% improvement (median)
C1	Ta001	0.000	0.000	2
C2	Ta011	0.000	0.000	2
C3	Ta021	0.000	0.065	4
C4	Ta031	0.000	0.000	3
C5	Ta041	1.538	0.259	17
C6	Ta051	3.580	0.438	18
C7	Ta061	0.000	0.000	2
C8	Ta071	0.061	0.212	18
C9	Ta081	3.374	0.436	21
C10	Ta091	0.212	0.120	17
C11	Ta101	1.797	0.146	20
C12	Ta111	0.826	0.115	21

For each instance, the table reports the median and interquartile range (IQR) of the final best-so-far RPD (%) across runs, together with the median "step to 95% improvement." Here, "step to 95% improvement" is computed per run as the first recorded step at which the best-so-far RPD reaches 95% of the total improvement achieved by the end of that run, and the table reports the median of this step across the 20 runs. The final RPD summaries range from configurations with median 0.000 (e.g., C1, C2, C3, C4, and C7) to configurations with higher final medians (notably C6 and C9), with dispersion captured by the reported IQR. The median step to 95% improvement occurs early in the lowest-RPD cases (typically within the first few steps) and shifts to later steps for configurations where improvements accumulate more gradually.

5. Discussion

Experimental evidence on Taillard's benchmark shows that HBFO achieves competitive best-run makespan performance while exhibiting configuration-dependent differences in deviation and stability. This section interprets these patterns in terms of hybrid refinement in permutation space, the role of population management under a fixed iteration budget, and the limits

imposed by benchmark-driven comparison protocols. The paired Wilcoxon results further indicate that these best-run differences are consistently favorable at the instance level, although they should be interpreted under the protocol heterogeneity of literature-sourced competitor values.

A central design feature of HBFO is the explicit coupling of population-level exploration with permutation-aware intensification. In PFSP, strong benchmark performance typically depends less on the "metaheuristic metaphor" and more on the quality of neighborhood exploitation and the scheduling of diversification under a bounded budget. From this perspective, HBFO's discrete realization of bacterial foraging—where chemotaxis is implemented through feasibility-preserving permutation moves and systematic refinement—aligns with the established need for structured intensification in stochastic search. Prior BFO-to-PFSP work similarly emphasizes that discrete operator design and local improvement are essential when transferring bacterial foraging dynamics to scheduling landscapes (Zhao et al., 2016).

The small subset of instances where HBFO is not best is informative rather than anomalous. These cases are consistent with the broader observation that hybrid PFSP designs can specialize across instance families under the same stopping rule. Methods with stronger or differently structured intensification components may yield additional gains once refinement becomes incremental, whereas learning-assisted approaches may benefit when instance regularities can be exploited by guidance mechanisms. Accordingly, the non-best cases indicate that alternative intensification or guidance strategies can be advantageous for particular instance structures, even when a method remains broadly competitive across the benchmark set.

The configuration-level heterogeneity and run-to-run dispersion patterns further underscore PFSP performance's sensitivity to instance-family characteristics. Under a fixed iteration budget, the same balance between diversification and permutation-neighborhood intensification can yield near-zero deviations for some $(n \times m)$ families yet leave others noticeably less stable or less accurate. In discrete BFO designs, this sensitivity is shaped by how movement operators, acceptance rules, and reproduction/elimination pressures interact with the landscape, affecting both the depth of refinement and the consistency of convergence toward high-quality basins. The convergence traces provide complementary evidence of this behavior, with many runs showing rapid early progress followed by diminishing gains and configuration-dependent stabilization rates.

The non-monotonic configuration-level averages reported in Table 4 should therefore be interpreted as an instance-family effect rather than as evidence that PFSP difficulty scales uniformly with n or m. Under the same fixed iteration budget, different Taillard $(n \times m)$ families can produce search landscapes that interact differently with the balance between diversification controls and permutation-neighborhood intensification, thereby affecting both mean deviation and run-to-run variability. Accordingly, some larger configurations may still allow consistent refinement under the implemented operator and acceptance schedule, whereas other families show greater dispersion and larger residual deviations despite comparable dimensions.

Compared with recent PFSP solvers, HBFO occupies a practical middle ground between purely handcrafted permutation heuristics and learning-assisted pipelines. Learning-based components can be integrated either as direct decision policies or as guidance modules within metaheuristic frameworks, whereas HBFO remains model-free and relies on explicit operator design and hybrid population management rather than training. Under standard benchmarking conventions, the reported results indicate that carefully designed hybrid stochastic search remains competitive for makespan-oriented PFSP, while leaving open the question of whether additional guidance or adaptive mechanisms could improve robustness on the more challenging instance families.

Several limitations should be noted when interpreting these findings. First, the evaluation is limited to Taillard's benchmark under the makespan objective, so the conclusions do not directly extend to other criteria or PFSP variants. Second, competitor values are drawn from published best-run results rather than recomputed under a unified codebase and computing environment, implying reliance on the original protocols and limiting direct, runtime-normalized comparisons. Third, the statistical comparisons are necessarily based on paired best-run values because run-to-run outcome distributions for competitors are not available in the published tables; therefore, stability comparisons across methods remain constrained by data availability. Finally, while the convergence traces provide time-to-quality insight under the adopted termination budget, broader conclusions about efficiency would require matched runtime reporting and normalized computational budgets across all methods.

6. Conclusions

This work proposed HBFO, a hybrid discrete BFO approach for permutation flow shop scheduling with the makespan criterion. The approach operates directly in permutation space using PFSP-compatible operators and structured intensification/diversification mechanisms. On Taillard's benchmark suite, HBFO achieves broadly competitive performance under standard PFSP evaluation conventions, attaining the best or tied-best makespan in the best run on most instances and exhibiting predominantly low deviations from best-known reference values. Configuration-level summaries further show that both deviation and run-to-run variability depend on the instance family. At the same time, the empirical evidence is constrained by the evaluation scope and comparison basis: results are reported for Taillard instances using the makespan objective; competitor values are drawn from prior publications rather than recomputed in a unified runtime environment; and systematic analyses of parameter sensitivity and budget-normalized time-to-quality behavior remain to be established.

Practical implications:

- HBFO is suitable when a fixed-parameter, fixed-budget metaheuristic is required, and extensive per-instance retuning is undesirable, particularly in benchmark-style PFSP makespan settings.
- When solution stability matters (e.g., repeated runs under the same budget), reporting dispersion and convergence behavior alongside best-run values provides a more informative characterization of performance than best-run results alone.
- For instance families exhibiting higher dispersion, robustness-oriented mechanisms (e.g., tighter acceptance caps or adjusted diversification pressure) are promising directions to improve consistency without departing from permutation-feasible operators.

Future work will focus on improving robustness on the more challenging instance families, conducting controlled sensitivity and runtime-normalization studies, and extending the method to broader PFSP variants and additional objectives under comparable benchmarking protocols.

Acknowledgement

This research was conducted with the support from DICYT (Scientific and Technological Research Bureau) (Cód. 062517PB_REG.) of the University of Santiago of Chile (USACH) and the Department of Industrial Engineering.

Code availability

The HBFO implementation code is available from the corresponding author upon reasonable request.

References

Arık, O. A. (2021). Artificial bee colony algorithm including some components of iterated greedy algorithm for permutation flow shop scheduling problems. *Neural Computing and Applications*, *33*(8), 3469–3486.

Chen, H., Zhang, Q., Luo, J., Xu, Y. & Zhang, X. (2020). An enhanced bacterial foraging optimization and its application for training kernel extreme learning machine. *Applied Soft Computing*, *86*, 105884.

Chen, Y.-P., Li, Y., Wang, G., Zheng, Y.-F., Xu, Q., Fan, J.-H. & Cui, X.-T. (2017). A novel bacterial foraging optimization algorithm for feature selection. *Expert Systems with Applications*, *83*, 1–17.

Das, S., Biswas, A., Dasgupta, S., Abraham, A. (2009). *Bacterial Foraging Optimization Algorithm: Theoretical Foundations, Analysis, and Applications*, in: Abraham, A., Hassanien, AE., Siarry, P., Engelbrecht, A. (eds), Foundations of Computational Intelligence Volume 3. Studies in Computational Intelligence, vol 203. Springer, Berlin, Heidelberg, pp. 23–55.

Davendra, D. & Bialic-Davendra, M. (2013). Scheduling flow shops with blocking using a discrete self-organising migrating algorithm. *International Journal of Production Research*, *51*(8), 2200–2218.

Eiben, A. E. & Smith, J. E. (2015). *Introduction to evolutionary computing*. Springer.

Framinan, J. M., Gupta, J. N. & Leisten, R. (2004). A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *Journal of the Operational Research Society*, *55*(12), 1243–1255.

Gan, X., Xiao, B. (2020). *Improved Bacterial Foraging Optimization Algorithm with Comprehensive Swarm Learning Strategies*, in: Tan, Y., Shi, Y., Tuba, M. (eds) Advances in Swarm Intelligence. ICSI 2020. Lecture Notes in Computer Science(), vol 12145. Springer, Cham, pp. 325–334.

Garey, M. R., Johnson, D. S. & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, *1*(2), 117–129.

Guo, C., Tang, H., Niu, B. & Lee, C. B. P. (2021). A survey of bacterial foraging optimization. *Neurocomputing*, *452*, 728–746.

Gupta, J. N. (1971). A functional heuristic algorithm for the flowshop scheduling problem. *Journal of the Operational Research Society*, *22*(1), 39–47.

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, *1*, 61–68.

Khurshid, B., Maqsood, S., Omair, M., Sarkar, B., Ahmad, I. & Muhammad, K. (2021). An improved evolution strategy hybridization with simulated annealing for permutation flow shop scheduling problems. *IEEE Access*, *9*, 94505–94522.

Kirkpatrick, S., Gelatt Jr, C. D. & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*(4598), 671–680.

Li, H., Gao, K., Duan, P.-Y., Li, J.-Q. & Zhang, L. (2023). An improved artificial bee colony algorithm with Q-learning for solving permutation flow-shop scheduling problems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *53*(5), 2684–2693.

Lian, Z., Gu, X. & Jiao, B. (2008). A novel particle swarm optimization algorithm for permutation flow-shop scheduling to minimize makespan. *Chaos, Solitons & Fractals*, *35*(5), 851–861.

Liu, Y. & Passino, K. (2002). Biomimicry of social foraging bacteria for distributed optimization: models, principles, and emergent behaviors. Journal of Optimization Theory and Applications, *115*(3), 603–628.

Lu, Y., Yuan, Y., Sitahong, A., Chao, Y. & Wang, Y. (2024). An Optimization Method for Green Permutation Flow Shop Scheduling Based on Deep Reinforcement Learning and MOEA/D. *Machines*, *12*(10), 721.

Lu, Y., Yuan, Y., Yasenjiang, J., Sitahong, A., Chao, Y. & Wang, Y. (2025). An optimized method for solving the green permutation flow shop scheduling problem using a combination of deep reinforcement learning and improved genetic algorithm. *Mathematics*, *13*(4), 545.

Marichelvam, M., Tosun, Ö. & Geetha, M. (2017). Hybrid monkey search algorithm for flow shop scheduling problem under makespan and total flow time. *Applied Soft Computing*, *55*, 82–92.

Meng, G., Cong, Z., Li, T., Wang, C., Zhou, M. & Wang, B. (2024). Dynamic bayesian network structure learning based on an improved bacterial foraging optimization algorithm. *Scientific Reports*, *14*(1), 8266.

Mishra, A. & Shrivastava, D. (2024). Parameter less hybrid ig-jaya approach for permutation flow shop scheduling problem. *Journal of Industrial and Management Optimization*, *20*(2), 606–629.

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.

Palmer, D. S. (1965). Sequencing jobs through a multi-stage process in the minimum total time—a quick method of obtaining a near optimum. *Journal of the Operational Research Society*, *16*(1), 101–107.

Pan, Z., Wang, L., Wang, J. & Lu, J. (2021). Deep reinforcement learning based optimization algorithm for permutation flow-shop scheduling. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *7*(4), 983–994.

Passino, K. M. (2002). Biomimicry of bacterial foraging for distributed optimization and control. *IEEE Control Systems Magazine*, *22*(3), 52–67.

Reeves, C. R. (1995). A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, *22*(1), 5–13.

Ruiz, R. & Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, *165*(2), 479–494.

Ruiz, R. & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, *177*(3), 2033–2049.

Sayadi, M. K., Ramezanian, R. & Ghaffari-Nasab, N. (2010). A discrete firefly meta-heuristic with local search for makespan minimization in permutation flow shop scheduling problems. *International Journal of Industrial Engineering Computations*, *1*(1), 1–10.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, *64*(2), 278–285.

Tseng, L.-Y. & Lin, Y.-T. (2009). A hybrid genetic local search algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, *198*(1), 84–92.

Tseng, L.-Y. & Lin, Y.-T. (2010). A genetic local search algorithm for minimizing total flowtime in the permutation flowshop scheduling problem. *International Journal of Production Economics*, *127*(1), 121–128.

Vallada, E. & Ruiz, R. (2009). Cooperative metaheuristics for the permutation flowshop scheduling problem. *European Journal of Operational Research*, *193*(2), 365–376.

Vital-Soto, A., Azab, A., & Baki, M. F. (2020). Mathematical modeling and a hybridized bacterial foraging optimization algorithm for the flexible job-shop scheduling problem with sequencing flexibility. *Journal of Manufacturing Systems*, *54*, 74–93.

Xiong, R., Wang, S., Yu, C., Fernandez, C., Xiao, W. & Jia, J. (2023). A novel nonlinear decreasing step-bacterial foraging optimization algorithm and simulated annealing-back propagation model for long-term battery state of health estimation. *Journal of Energy Storage*, *59*, 106484.

Zaied, A. N. H., Ismail, M. M. & Mohamed, S. S. (2021). Permutation flow shop scheduling problem with makespan criterion: literature review. *Journal of Theoretical and Applied Information Technology*, *99*(4), 830–848.

Zhang, Q., Chen, H., Luo, J., Xu, Y., Wu, C. & Li, C. (2018). Chaos enhanced bacterial foraging optimization for global optimization. *IEEE Access*, 6, 64905–64919.

Zhang, T., Guo, X. & Ji, G. (2023). Permutation flow shop scheduling optimization method based on cooperative games. *IEEE Access*, *11*, 47377–47389.

Zhao, F., Liu, Y., Shao, Z., Jiang, X., Zhang, C. & Wang, J. (2016). A chaotic local search based bacterial foraging algorithm and its application to a permutation flow-shop scheduling problem. *International Journal of Computer Integrated Manufacturing*, *29*(9), 962–981.

Zhou, T., Luo, L., Ji, S. & He, Y. (2023). A reinforcement learning approach to robust scheduling of permutation flow shop. *Biomimetics*, *8*(6), 478.

Zobolas, G., Tarantilis, C. D. & Ioannou, G. (2009). Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. *Computers & Operations Research*, *36*(4), 1249–1267.

© 2026 by the authors; licensee Growing Science, Canada. This is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC-BY) license (http://creativecommons.org/licenses/by/4.0/).