MATHEMATICAL MODELING AND COMPUTING, Vol. 12, No. 4, pp. 1393–1405 (2025)

An Iterative Greedy Algorithm Involving a Q-Learning Mechanism for Solving the Permutation Flow Shop Scheduling Problem with Sequence-Dependent Setup Times

Mesmar K.[1,*], Lebbar M.[1], Allali K.[2]

[1] *ROSDM Research Team, LISTD – Laboratory of Systems Engineering and Digital Transformation, ENSMR Rabat, Morocco*
[2] *Laboratory of Mathematics, Computer Science and Applications, FST Mohammedia, Hassan II University, Morocco*
[*] *Corresponding author: khadija.mesmar@enim.ac.ma*

(Received 8 May 2025; Revised 8 December 2025; Accepted 10 December 2025)

This paper proposes a novel hybrid framework, Q-IG, to solve the permutation flow shop scheduling problem with sequence-dependent setup times (PFSP-SDST). Recent advancements in learning-based methods demonstrate significant potential in addressing flow shop scheduling, yet they often struggle with the enormous solution space and the design of effective reward functions. To overcome these challenges, Q-IG integrates the iterated greedy metaheuristic (IG) with Q-learning. It begins by applying the Nawaz–Enscore–Ham (NEH) heuristic to generate high-quality initial solutions. During each IG iteration, the schedules are partially destroyed and reconstructed, and Q-learning is used to guide the selection of neighborhood moves that are expected to minimize makespan. We evaluated both standalone IG and the Q-IG hybrid in the Taillard benchmark suite, measuring performance through a relative percentage deviation from the best-known makespan. The results demonstrate that Q-IG outperforms its competitors, confirming its effectiveness for PFSP-SDST and highlighting the promise of incorporating Q-learning into traditional metaheuristic approaches.

Keywords: *iterated greedy; Q-learning; permutation flow shop scheduling (PFSP); sequence-dependent setup times (SDST); makespan.*

2010 MSC: 68M20, 90B35, 37M05, 65K05, 14N10 **DOI:** 10.23939/mmc2025.04.1393

1. Introduction

The flow shop scheduling problem (FSP) represents a fundamental challenge in production scheduling, where a collection of jobs is required to follow a specific sequence through a set of machines. The primary objective is often to minimize the total completion time, also known as the makespan. Throughout the years, this model has undergone generalization, initially to flexible or hybrid flow shops (HFSP), where each processing stage can include multiple parallel machines, and subsequently to variations that specifically consider sequence-dependent setup times (SDST). In numerous practical manufacturing settings, the duration needed for tooling changes, machine cleaning, work-in-process repositioning, or process parameter adjustments is influenced by the specific job that has just been completed. Instances range from modifying molds for various tire dimensions to fine-tuning baking temperatures for breads with diverse dough formulations. When setup times are substantial, overlooking them can utilize more than 20% of available capacity and severely impact the effectiveness of any schedule [1]. The permutation flow shop with sequence-dependent setup times (PFSP-SDST), represented in the standard three-field notation as $F_m|permu, s_{ijk}|C_{\max}$, is classified as NP-hard [2]. This problem involves a scenario where every job adheres to a consistent sequence of machines, with the stipulation that jobs cannot surpass one another due to a permutation constraint. Furthermore, prior to the processing of job j on machine k, a setup time s_{ijk} determined by the job i that was the most recently processed on that machine must be observed. This model addresses the expenses associated with transitioning between different families of components within group technology en-

vironments, playing a crucial role in lot-sizing choices when changeovers are significant. Industrial studies spanning the glass, metallurgical, paper, textile, chemical, and aerospace sectors indicate that nearly 70% of schedulers are required to manage SDST in practice, and neglecting these tasks significantly hinders shop performance [3]. Although it has significant practical relevance, studies on PFSP-SDST are still limited compared to traditional FSP. Many current flow shop investigations typically either presume fixed setups or concentrate on total tardiness and other metrics related to due dates. Several studies have established complexity results for both single and two-machine scenarios involving sequence-dependent setup times, and have validated by simulation the impact of sequence-dependent changeovers on throughput and lateness [4]. However, there remains a need for effective metaheuristic or learning-based approaches specifically designed to address the combinatorial structure of PFSP-SDST.

This paper presents a solution to the PFSP-SDST through the introduction of Q-IG, an innovative hybrid algorithm that merges the intensification and diversification capabilities of an iterated greedy (IG) metaheuristic with the adaptive decision-making features of Q-learning. We present the following contributions:

1. The PFSP-SDST problem is formulated as $F_m|permu, s_{ijk}|C_{\max}$, integrating sequence-dependent setup times within the flow shop framework.
2. A Q-learning module is developed to dynamically direct the destruction-reconstruction phases of IG, acquiring knowledge of which jobs to eliminate and the best locations for their reinsertion.
3. We carry out comprehensive computational experiments on benchmark instances, showing that Q-IG surpasses leading metaheuristics in makespan minimization within the PFSP-SDST framework.

The subsequent sections of this paper are structured as outlined below. Section 2 offers an in-depth examination of the pertinent literature. Section 3 provides a formal definition of PFSP-SDST. In Section 4, the Q-IG algorithm is discussed in full detail. In Section 5, we present the details of our experimental setup along with the results obtained. In conclusion, Section 6 summarizes the findings and suggests potential directions for further investigation.

2. Review of the literature

Flow shop scheduling represents a well-established NP-hard problem where jobs are required to be processed sequentially on a set of machines. Since Johnson's [5] groundbreaking two-machine study, a substantial body of literature has emerged concerning both exact and heuristic approaches to the permutation flow shop problem (PFSP). Initial precise methodologies - dynamic programming for two machines with sequence-dependent setup times (SDST) [6], lexicographic search [7], and mixed-integer linear programming (MILP) models [8,9] are confined to very small instances. Branch-and-cut and branch-and-bound algorithms [10, 11] demonstrate effectiveness in solving up to ten jobs on six machines; however, they become impractical when addressing larger problems.

Due to the combinatorial explosion associated with the PFSP [12], there has been a shift in focus toward heuristics and metaheuristics in the field of study. The Nawaz, Enscore, and Ham (NEH) heuristic [13] and the Campbell–Dudek–Smith (CDS) [14] method are prominent examples of constructive rules that are frequently utilized. Szwarc and Gupta modified the multi-sort techniques derived from the traveling salesman problem (TSP) [15]; Simons introduced the TOTAL and SETUP rules based on TSP [16]; and Das et al. introduced a savings index heuristic [17]. Ruiz and Maroto present an in-depth examination of these and additional PFSP heuristics [18].

Techniques such as simulated annealing (SA) [19], tabu search [20], and genetic algorithms (GA) [21] have undergone adaptations and hybridizations to enhance the quality of solutions. For instance, Lei [22] presented an effective decoding method for genetic algorithms applied to flexible job shops with fuzzy time parameters; Engin et al. [23] created innovative mutation operators; Mirsanei et al. [24] and Defersha et al. [25] improved simulated annealing by introducing new search moves for flexible job shop scheduling problems; and Qin et al. [26] and Han et al. [27] formulated iterated greedy algorithms aimed at enhancing energy efficiency in distributed flow shop environments. Hybrid schemes

that integrate genetic algorithms, simulated annealing, variable neighborhood search, estimation of distribution, and domain descent heuristics [28–31] have gained significant traction. Knowledge-driven approaches, such as iterated greedy rules that account for learning and forgetting effects [32] and migratory bird optimization algorithms [33], enhance the metaheuristic toolkit significantly.

Although advancements have been made with respect to the PFSP, flow shops utilizing SDST have not garnered the same level of focus. In addition to the two-machine exact methods and small-scale MILP models, the main notable metaheuristic identified for SDST-PFSP is a greedy randomized adaptive search procedure (GRASP) developed by Ríos–Mercado and Bard [34]. To the best of our knowledge, no genetic algorithms have been published specifically designed for SDST flow shops, and the availability of fair comparative evaluations is limited due to the use of different instance sets and computing platforms.

Recently, machine learning techniques, especially reinforcement learning (RL) and deep reinforcement learning (DRL), have been applied to scheduling. Q-learning and deep Q-networks (DQN) have been applied in real-time scheduling [35], PFSP [36–38], as well as in hybrid approaches that integrate DQN within estimation-of-distribution frameworks [39]. Graph neural networks (GNNs) have been utilized [40–42] to acquire representations of job-machine graphs. Imitation learning and multi-expert architectures have tackled the challenges of distributed heterogeneous flow shops with family setups.

However, purely end-to-end learning methods often encounter difficulties in generalizing across different scales and still do not surpass the performance of the leading metaheuristics on established PFSP benchmarks. This suggests an opportunity to integrate efficient metaheuristic components into learning-based frameworks: expert heuristics can guide RL or DRL policies toward high-quality regions of the search space, improving both solution quality and convergence speed.

Currently, existing networks based on end-to-end learning are difficult to generalize to PFSPs of different scales, and the solving accuracy still needs to be further improved. Consequently, it is important to explore the application of effective metaheuristics to enhance the quality of the solution in learning-based approaches.

3. Problem description

Table 1. Notation used in the PFSP-SDST.

Symbol	Definition
n	Number of jobs
m	Number of machines
$i \in M$	Index of machines, where $M = \{1, 2, \ldots, m\}$
$j, k \in J,\ j \neq k$	Indices of jobs, where $J = \{1, 2, \ldots, n\}$
J	Set of jobs $\{1, 2, \ldots, n\}$
M	Set of machines $\{1, 2, \ldots, m\}$
$\pi = (\pi_1, \pi_2, \ldots, \pi_n)$	A permutation representing the processing order of jobs
$p_{i,j}$	Processing time of job j on machine i
$s_{i,j,k}$	Sequence-dependent setup time on machine i from job j to job k
$C_{i,j}$	Completion time of job j on machine i
$S_{i,j}$	Starting time of job j on machine i
$C_{\max}$	Makespan: total time to complete all jobs

Table 1 provides a summary of the notation utilized in this section. The permutation sequence-dependent setup times flow shop problem (PFSP-SDST) involves a collection $J = \{J_1, J_2, \ldots, J_n\}$ of n jobs that need to be processed without interruption on a group $M = \{M_1, M_2, \ldots, M_m\}$ of m machines. Each machine processes the jobs in the same fixed order. Every job J_j is assigned a specific processing duration $p_{j,i}$ on machine M_i, and when a machine transitions from job J_k to job J_ℓ, it incurs sequence-dependent setup times $s_{i,k,\ell}$ on machine M_i. A schedule is defined by a permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ of jobs. The objective is to find the optimal sequence π^* that minimizes the makespan $C_{\max}(\pi)$, which is the time at which the final job is completed on the last machine.

Let $\mathcal{Q}$ denote the set of all $n!$ permutations of J. The optimization goal is to find $\pi \in \mathcal{Q}$ such that:

$$C_{\max}(\pi) \leqslant C_{\max}(\pi'), \quad \forall \pi' \in \mathcal{Q}. \tag{1}$$

For a given permutation π, let $C(\pi_j; i)$ represent the completion time of job π_j on machine M_i. The makespan is then:

$$C_{\max}(\pi) = C(\pi_n; m). \tag{2}$$

The completion times $C(\pi_j; i)$ can be computed recursively once π is fixed:

— **Initial job on the first machine:**

$$C(\pi_1; 1) = p(\pi_1; 1). \tag{3}$$

— **Initial job on machines** $i = 2, \ldots, m$ (no setup required):

$$C(\pi_1; i) = C(\pi_1; i-1) + p(\pi_1; i). \tag{4}$$

— **Subsequent jobs on the first machine** $(j = 2, \ldots, n)$:

$$C(\pi_j; 1) = \sum_{q=1}^{j-1} \left[p(\pi_q; 1) + s_{1,\pi_q,\pi_{q+1}}\right] + p(\pi_j; 1). \tag{5}$$

— **Remaining jobs on machines** $i = 2, \ldots, m$ and $j = 2, \ldots, n$:

$$C(\pi_j; i) = \max\left\{C(\pi_j; i-1),\ C(\pi_{j-1}; i) + s_{i,\pi_{j-1},\pi_j}\right\} + p(\pi_j; i). \tag{6}$$

Equation (6) enforces that job π_j can start on machine M_i only after its completion on machine M_{i-1}, the previous job π_{j-1} has finished on M_i, and the setup time s_{i,π_{j-1},π_j} has elapsed. Once all completion times $C(\pi_j; i)$ are calculated, the makespan is determined as $C(\pi_n; m)$.

4. Related approaches

4.1. Iterated greedy algorithm

The iterated greedy algorithm (IG) is a single-solution metaheuristic that was initially introduced by Ruiz and Stützle [43], specifically designed to address the permutation flow shop scheduling problem (PFSP). The pseudo-code can be found in Algorithm 1. The process begins with the creation of an initial permutation of n jobs, which can be achieved randomly or, more frequently, through a constructive heuristic that produces a high-quality starting solution. Following this initial sequence, it subsequently engages in a cycle of perturbation, local optimization, acceptance testing, and termination verification.

During the perturbation phase of each iteration, the algorithm randomly selects and removes d jobs from the current permutation β, resulting in two subsets: β_D, which includes the remaining $n-d$ jobs, and β_R, the set of extracted jobs d. A local search may be applied to β_D to enhance the partial sequence prior to reconstruction. Subsequently, in the constructive (reinsertion) phase, each job in β_R is reinserted individually into the optimal position in β_D, specifically, the position that minimizes the makespan $C_{\max}$ of the resulting partial permutation. In instances where multiple positions result in identical $C_{\max}$, a secondary rule or random selection can be used to resolve ties, thus improving diversification.

After all the jobs have been reinserted, the algorithm applies a local search to the reconstructed sequence, guiding it towards a new local optimum. Subsequently, an acceptance mechanism evaluates this candidate solution in relation to the current one employing criteria such as enhancement in $C_{\max}$ or a probabilistic rule to determine if the existing solution should be replaced. The procedure continues until a predetermined stopping criterion is satisfied, which may include a maximum iteration count, a time constraint, or a defined number of iterations lacking improvement. Using a cycle of selective destruction and strategic reconstruction, along with local search and acceptance tests, the iterated greedy algorithm adeptly navigates the solution space and transcends local optima, ultimately identifying high-quality schedules for PFSP.

Algorithm 1 Iterated greedy (IG) algorithm.

Require: $d \in \mathbb{N}$
Ensure: $\Pi^* \in S(N)$
1: $\Pi \leftarrow$ `initialSolution()`
2: $\Pi \leftarrow$ `localSearch`(Π)
3: $\Pi^* \leftarrow \Pi$
4: **while** `terminationCriterion()`
// — Perturbation Phase —
5: $D \leftarrow$ `destruction`(Π, d)
6: $\Pi \setminus D \leftarrow \Pi \setminus D$
7: $\Pi \setminus D \leftarrow$ `localSearch`$(\Pi \setminus D)$
// — Construction Phase —
8: $\Pi' \leftarrow$ `construction`$(\Pi \setminus D, D)$
9: $\Pi' \leftarrow$ `localSearch`(Π')
// — Acceptance —
10: **if** `accept`(Π', Π) **then**
11: $\Pi \leftarrow \Pi'$
12: **if** `better`(Π', Π^*) **then**
13: $\Pi^* \leftarrow \Pi'$
14: **return** Π^*

4.2. Q-learning algorithm

Reinforcement learning (RL) is a structured approach to decision making in which an agent engages in trial-and-error interactions with its environment to determine the optimal actions to take in specific states, with the aim of maximizing cumulative rewards [44]. At each time step t, the agent perceives the current state s_t, chooses an action a_t from the available action set, and subsequently the environment transitions to a new state s_{t+1} while providing a scalar reward R_{t+1}. Through this iterative process of observation, action, reward and transition, the agent progressively uncovers a policy π that maps states to actions to maximize expected long-term return (see Figure 1).

Traditional reinforcement learning methods rely on a complete model of the environment, including the state space, action space, transition probabilities, and reward function, allowing for the derivation of optimal policies using dynamic programming. However, in many practical situations-particularly combinatorial optimization problems-such a model is not available [45]. Model-free approaches, such as Monte Carlo and Temporal Difference (TD) learning, overcome this limitation by estimating value functions from sampled experience [46].

Fig. 1. Model of Q-learning.

Q-learning, introduced by Watkins [47], is an RL algorithm based on temporal-difference learning and model-free value estimation. The objective is to learn the optimal action value function $Q^*: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, where $Q^*(s,a)$ represents the maximum expected cumulative reward obtainable by taking action a in state s and subsequently following the optimal policy. The agent maintains a Q-table with all initial entries set to zero. When executing the action a_t in state s_t, receiving the reward R_{t+1}, and transitioning to state s_{t+1}, the Q value is updated using the following rule:

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot Q(s_t, a_t) + \alpha \cdot \left[R_{t+1} + \gamma \cdot \max_{a'} Q(s_{t+1}, a') \right]. \tag{7}$$

Alternatively, the incremental form is often used:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot \left[r + \gamma \cdot \max_{a'} Q(s', a') - Q(s,a) \right]. \tag{8}$$

In this context, $\alpha \in (0,1]$ is the learning rate, which determines how much new information is overriding old estimates. The discount factor $\gamma \in (0,1]$ governs the importance of future rewards. The term $\max_{a'} Q(s_{t+1}, a')$ approximates the optimal expected return of the next state.

A key challenge in Q-learning is balancing exploration and exploitation. Exploration allows the agent to try less visited actions to improve value estimates, while exploitation selects actions currently believed to be optimal. The ε-greedy policy addresses this by choosing a random action with probability ε, and with probability $1-\varepsilon$, selecting $\arg\max_a Q(s,a)$. Under standard assumptions, such as decaying α, bounded rewards, and sufficient exploration, Q-learning is guaranteed to converge to the optimal action value function Q^* with probability one [47]. Once Q^* is obtained, the optimal policy is given by:

$$\pi^*(s) = \arg\max_a Q^*(s,a). \tag{9}$$

The ε-greedy strategy [46] effectively balances exploration and exploitation by assigning a small probability ε to the selection of random actions. With probability $1-\varepsilon$, the agent selects the action that currently appears to be the best, while with probability ε, it explores other actions uniformly at random. This policy can be formally defined as:

$$a = \begin{cases} \arg\max_{a \in \mathcal{A}} Q(s,a), & \text{with probability } 1-\varepsilon, \\ \text{randomly select } a \in \mathcal{A}, & \text{with probability } \varepsilon. \end{cases} \tag{10}$$

This policy yields the highest expected cumulative reward over time.

4.3. IG algorithm with Q-learning

This work introduces Q-IG, an innovative variant of the traditional Iterated Greedy (IG) algorithm tailored for the Permutation Flow Shop Scheduling Problem (PFSP), which incorporates a RL-inspired operator selection mechanism during its perturbation phase. In contrast to the original IG (refer to Algorithm 1), which consistently removes and reinserts a predetermined number of jobs d at each iteration, thus maintaining a steady exploration rate regardless of search progress, Q-IG dynamically adjusts its level of disruption by employing Q-learning to select from a set of perturbation operators.

Reinforcement learning is a structured paradigm in which an agent interacts with its environment by taking actions, receiving feedback, and updating its policy to maximize long-term cumulative rewards. In Q-IG, each application of a perturbation operator is considered an action, the current solution represents the state, and the resulting change in the objective function (typically the makespan) serves as the reward. After each perturbation, the Q-table is updated accordingly, ensuring that future operator selections balance between exploiting high-reward actions and exploring less frequently used alternatives.

Figure 2 presents the overall Q-IG framework, which, similar to the canonical IG, is composed of three primary phases: perturbation, local search, and acceptance. The key innovation (see Algorithm 2) lies in the perturbation phase, where Q-learning is used to select from k different removal/reinsertion operators. In more detail, a collection of varying values of d within the constructive heuristic is interpreted as the set of actions $\mathcal{A}$, defined as:

$$\mathcal{A} = \{1, 2, \ldots, d_{\max}\}, \tag{10}$$

where $d_{\max}$ represents the maximum number of jobs that can be removed during the destruction phase. The motivation behind exploring different values of d is to establish a diverse set of perturbation operators, each with varying degrees of exploration intensity. As the value of d increases, a larger portion of the solution is disrupted by removing and reinserting more jobs, thereby enhancing the algorithm's ability to escape local optima through more aggressive search diversification. Following the destruction step, a partial solution is re-optimized using the same insertion-based local search method adopted in the standard IG.

Algorithm 2 Q-IG: Q-learning-based Iterated Greedy Algorithm.

Require: $\varepsilon \in \mathbb{R}$ // Exploration rate
Require: $\beta \in \mathbb{R}$ // Epsilon decay factor
Require: $\alpha \in \mathbb{R}$ // Learning rate
Require: $\gamma \in \mathbb{R}$ // Discount factor
Require: $E \in \mathbb{N}$ // Episode length
Require: $\eta \in \mathbb{R}$ // Improvement weight
Require: A // Set of possible actions (perturbation levels)
Ensure: $\Pi^* \in S(N)$ // Best solution found

$\Pi \leftarrow$ `initialSolution()`
$\Pi \leftarrow$ `localSearch`(Π)
$\Pi^* \leftarrow \Pi$
$Q \leftarrow$ zero-initialized Q-table
$s \leftarrow 0$
$d \leftarrow$ `randomChoice`(A)
while not `terminationCriterion()`
  $C_{\text{prev}} \leftarrow C_{\max}(\Pi)$
  $C^*_{\text{prev}} \leftarrow C_{\max}(\Pi^*)$
  $C \leftarrow C_{\text{prev}}$
  $C^* \leftarrow C^*_{\text{prev}}$
  for $e = 1$ to E
    $D \leftarrow$ `destruction`(Π, d)
    $\Pi \setminus D \leftarrow \Pi \setminus D$
    $\Pi \setminus D \leftarrow$ `localSearch`$(\Pi \setminus D)$
    $\Pi' \leftarrow$ `construction`$(\Pi \setminus D, D)$
    $\Pi' \leftarrow$ `localSearch`(Π')
    if `accept`(Π', Π) **then**
      $\Pi \leftarrow \Pi'$
      $C \leftarrow \min(C, C_{\max}(\Pi))$
    if `better`(Π', Π^*) **then**
      $\Pi^* \leftarrow \Pi'$
      $C^* \leftarrow C_{\max}(\Pi^*)$
  $(Q, s, d) \leftarrow$ `Q-learning`$(C_{\text{prev}}, C^*_{\text{prev}}, C, C^*, \varepsilon, \alpha, \gamma, \beta, \eta, A, s, d)$
return Π^*

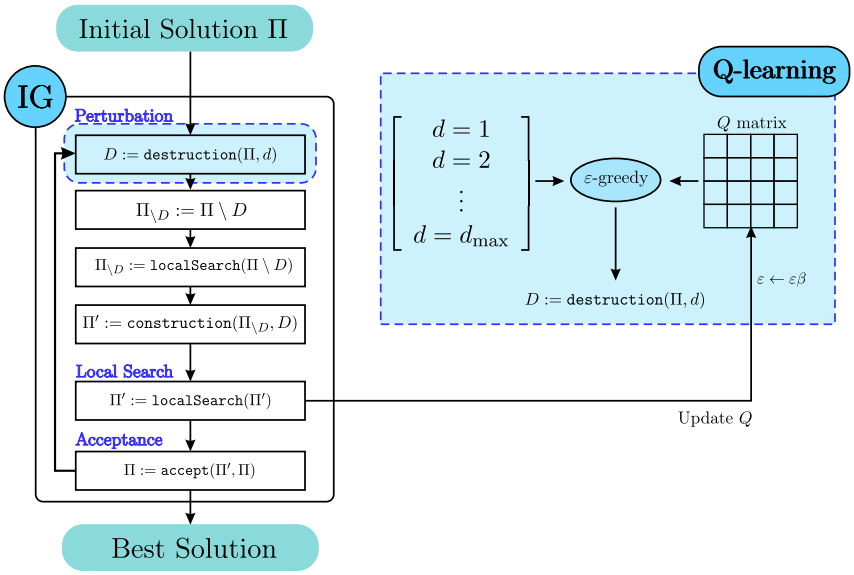

Fig. 2. The methodology of the suggested Q-IG algorithm.

The Q-IG algorithm is initialized with a solution generated by the Nawaz–Enscore–Ham (NEH) heuristic, a well-known and effective constructive method for PFSP. During both the reconstruction and local search phases, tie-breaking is handled using the Fernandez–Viagas and Framinan rule, which selects the job that minimizes cumulative idle time. By integrating Q-learning-based adaptive perturbation and established construction strategies, Q-IG dynamically modulates its exploration intensity and demonstrates increased capability in escaping local optima and improving scheduling performance.

5. Experimental evaluation

The implementation of all algorithms was carried out using MATLAB R2024b, and executed on a personal computer equipped with an Intel Core i7-10700 @ 2.90 GHz CPU and 16 GB of RAM. To guarantee an equitable and consistent assessment, each instance was assigned a maximum computational duration of 3 600 s. The quality of the solutions was evaluated using the percentage deviation (PD) metric, which is defined as:

$$\mathrm{PD} = \frac{C_{\max} - \mathrm{LB}}{\mathrm{LB}} \times 100, \tag{11}$$

where $C_{\max}$ represents the makespan obtained by the algorithm, and LB denotes the best-known lower bound. This standardized scale-independent metric allows for performance comparison between problem instances of different sizes and complexities.

Reinforcement learning-guided metaheuristic was tested on four well-established Taillard-based flow shop datasets that incorporate sequence-dependent setup times (SDST). These datasets are publicly available at: `http://soa.iti.es/problem-instances`. Each dataset is defined by the maximum ratio between the setup time and the processing time, as follows:

— **SDST10:** Setup times limited to 10% of the original maximum processing time;
— **SDST50:** Setup times limited to 50%;
— **SDST100:** Setup configurations limited to 100%;
— **SDST125:** Setup configurations limited to 125%.

The algorithm applies an ε-greedy policy to select among a set of perturbation operators: with a probability of ε, the agent performs exploratory actions, whereas with a probability of $1-\varepsilon$, it exploits the currently most effective operator. A preliminary parameter tuning procedure was performed using grid search, yielding the following configuration:

— Exploration rate: $\varepsilon = 0.80$;
— Decay factor of exploration: $\beta = 0.996$;
— Learning rate for Q-value updates: $\alpha = 0.60$;
— Discount factor for future rewards: $\gamma = 0.80$;
— Episode length: $E = 6$ perturbation steps;
— Balance weight for local/global improvement: $\eta = 0.30$.

This configuration was consistently applied in all experimental runs. The computational results for instances with fewer than 100 jobs are presented in Table 2, which reports both the absolute makespan values ($C_{\max}$) and the corresponding PD values. In addition, the average standard deviations are included to assess the robustness and stability of the algorithm in multiple trials. These results provide a comprehensive perspective on the effectiveness of the proposed approach in varying instance complexities and SDST intensities.

The comparative performance of the iterated greedy (IG) algorithm and the proposed Q-learning-enhanced iterated greedy (Q-IG) algorithm is presented in Table 2, which encompasses 40 benchmark instances of the permutation flow shop scheduling problem with sequence-dependent setup times (PFSP-SDST). The instances are grouped into four categories **SDST10**, **SDST50**, **SDST100**, and **SDST125** based on the relative dominance of setup times with respect to processing times.

Across all instance categories, Q-IG consistently outperforms the classical IG in terms of percentage deviation (PD) from the known lower bounds. This improvement highlights the effectiveness of reinforcement learning in adaptively controlling the strength of the disturbance, thus enhancing both the exploratory and exploitative capabilities of the search process.

Table 2. Results of IG and Q-IG on 40 benchmark instances of the PFSP-SDST.

Set	Ins	$n \times m$	LB	IG		Q-IG		Set	Ins	$n \times m$	LB	IG		Q-IG	
				$C_{\max}$	PD	$C_{\max}$	PD					$C_{\max}$	PD	$C_{\max}$	PD
SDST10	ta001	20×5	1330	1369	2.93	1360	2.26	SDST50	ta011	20×10	2009	2065	2.79	2045	1.79
	ta002	20×5	1401	1424	1.64	1413	0.86		ta012	20×10	2065	2112	2.28	2089	1.16
	ta003	20×5	1161	1206	3.88	1205	3.79		ta013	20×10	1897	1948	2.69	1936	2.06
	ta004	20×5	1370	1421	3.72	1381	0.80		ta014	20×10	1794	1844	2.79	1832	2.12
	ta005	20×5	1303	1329	2.00	1316	1.00		ta015	20×10	1842	1889	2.55	1877	1.90
	ta006	20×5	1269	1311	3.31	1278	0.71		ta016	20×10	1816	1868	2.86	1851	1.93
	ta007	20×5	1294	1328	2.63	1301	0.54		ta017	20×10	1858	1918	3.23	1896	2.05
	ta008	20×5	1282	1301	1.48	1292	0.78		ta018	20×10	1962	2022	3.06	2005	2.19
	ta009	20×5	1313	1360	3.58	1315	0.15		ta019	20×10	1985	2045	3.02	2016	1.56
	ta010	20×5	1178	1198	1.70	1182	0.34		ta020	20×10	2013	2067	2.68	2047	1.69
Average				3.21			**2.69**	Average					3.67		**2.74**
SDST100	ta031	50×5	3893	4028	3.47	3973	2.05	SDST125	ta041	50×10	5275	5532	4.87	5530	4.83
	ta032	50×5	4056	4239	4.51	4205	3.67		ta042	50×10	5177	5397	4.25	5328	2.92
	ta033	50×5	3900	4217	8.13	4192	7.49		ta043	50×10	5193	5431	4.58	5401	4.01
	ta034	50×5	4020	4302	7.01	4278	6.42		ta044	50×10	5286	5463	3.35	5410	2.35
	ta035	50×5	4014	4296	7.03	4184	4.24		ta045	50×10	5236	5456	4.20	5406	3.25
	ta036	50×5	4073	4319	6.04	4204	3.22		ta046	50×10	5262	5479	4.12	5402	2.66
	ta037	50×5	3999	4214	5.38	4195	4.90		ta047	50×10	5340	5537	3.69	5509	3.16
	ta038	50×5	3966	4187	5.57	4063	2.45		ta048	50×10	5317	5508	3.59	5498	3.40
	ta039	50×5	3808	4102	7.72	4005	5.17		ta049	50×10	5194	5359	3.18	5318	2.39
	ta040	50×5	4022	4322	7.46	4222	4.97		ta050	50×10	5334	5524	3.56	5456	2.29
Average				6.23			**4.46**	Average					3.94		**3.13**

SDST10 and SDST50 (20-job instances). For the moderately complex SDST10 and SDST50 sets, Q-IG achieves notable gains over IG. Specifically:
— The average PD for SDST10 is reduced from 3.21% (IG) to 2.69% (Q-IG).
— In SDST50, the PD decreases from 3.67% to 2.74%, representing an average improvement of approximately one percentage point.

These gains can be primarily attributed to the reinforcement learning mechanism, which dynamically selects perturbation operators based on their historical impact on the quality of the solution. Since these smaller instances allow for more iteration cycles within the 3 600-second time limit, the algorithm has more opportunities to refine its policies and converge toward high-quality solutions.

SDST100 and SDST125 (50-job instances). The performance gap between IG and Q-IG becomes more pronounced as the size of the instance and the complexity of the setup times increase:
— In SDST100, the average PD decreases from 6.23% (IG) to 4.46% (Q-IG), resulting in an improvement of nearly 1.8 percentage points.
— In SDST125, the average PD is reduced from 3.94% to 3.13%, suggesting that Q-IG offers more stable and efficient convergence even with greater dominance of setup time.

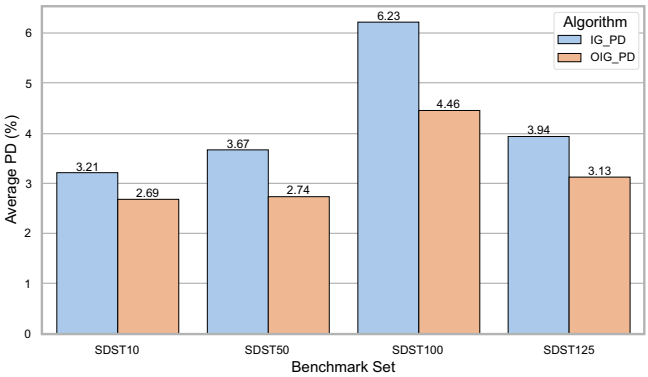

Fig. 3. Average PD comparison of IG vs Q-IG on benchmark sets.

These results confirm that Q-IG is capable of maintaining high-quality solutions across a broad range of problem complexities. The learned operator-selection policy enables the algorithm to adapt its behavior based on the current state of the search and the effectiveness of previous perturbations, thus enhancing its robustness and scalability.

Figure 3 illustrates that the Q-IG algorithm consistently attains lower average percentage deviation (PD) values compared to the classical IG in all PFSP-SDST benchmark sets. The improvements are modest for SDST10 and SDST50, whereas they become significant in more complex cases like SDST100 (from 6.23% to 4.46%) and SDST125. The findings indicate that Q-IG delivers superior solution quality, especially in complex scheduling scenarios characterized by significant setup-time variability.

6. Conclusion and future work

This study presents a Q-learning-enhanced variant of the iterated greedy algorithm (IG), referred to as Q-IG, for solving the permutation flow shop scheduling problem with sequence-dependent setup times (PFSP-SDST). By integrating a reinforcement learning mechanism into the perturbation phase, the proposed Q-IG algorithm dynamically adapts the perturbation size based on previous search performance, thereby enabling a more effective exploration–exploitation trade-off. Extensive experiments conducted on 40 benchmark instances demonstrate that Q-IG consistently outperforms the classical IG in terms of percentage deviation (PD) from the best-known lower bounds. The performance gains are particularly pronounced for larger and more complex instances, such as those in the SDST100 and SDST125 categories, highlighting the robustness and scalability of the proposed approach. Overall, the results confirm that the use of Q-learning to guide perturbation operator selection constitutes an effective enhancement for adaptive metaheuristic design in complex scheduling environments.

Future research may explore several promising directions:
— Extending the Q-IG framework to multi-objective or stochastic variants of the PFSP.
— Investigating alternative reinforcement learning paradigms, such as deep Q-networks or policy gradient methods.
— Applying the proposed Q-IG approach to other combinatorial optimization problems, including job-shop, open-shop, or vehicle routing problems.

— Enhancing the learning mechanism by incorporating state representations based on solution features or search history to improve generalization and learning efficiency.

In summary, this work provides strong evidence that hybridizing metaheuristics with learning-based decision-making mechanisms is a promising strategy for tackling large-scale and highly complex scheduling problems.

[1] Pinedo M., Hadavi K. Scheduling: theory, algorithms and systems development. Operations Research Proceedings 1991. 35–42 (1992).

[2] Lenstra J. K., Rinnooy Kan A. H. G. Complexity of vehicle routing and scheduling problems. Networks. **11** (2), 221–227 (1981).

[3] Luh P. B., Gou L., Zhang Y., Nagahora T., Tsuji M., Yoneda K., Hasegawa T., Kyoya Y., Kano T. Job shop scheduling with group-dependent setups, finite buffers, and long time horizon. Annals of Operations Research. **76**, 233–259 (1998).

[4] Kim S. C., Bobrowski P. M. Impact of sequence-dependent setup time on job shop scheduling performance. The International Journal of Production Research. **32** (7), 1503–1520 (1994).

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

[6] Corwin B. D., Esogbue A. O. Two machine flow shop scheduling problems with sequence dependent setup times: A dynamic programming approach. Naval Research Logistics Quarterly. **21** (3), 515–524 (1974).

[7] Gupta J. N. D. Search algorithm for the generalized flowshop scheduling problem. Computers & Operations Research. **2** (2), 83–90 (1975).

[8] Srikar B. N., Ghosh S. A MILP model for the n-job, M-stage flowshop with sequence dependent set-up times. International Journal of Production Research. **24**, 1459–1474 (1986).

[9] Stafford E. F. Jr, Tseng F. T. On the Srikar-Ghosh MILP model for the iVx M SDST flowshop problem. International Journal of Production Research. **28** (10), 1817–1830 (1990).

[10] Ríos-Mercado R. Z., Bard J. F. Computational experience with a branch-and-cut algorithm for flowshop scheduling with setups. Computers & Operations Research. **25** (5), 351–366 (1998).

[11] Rios-Mercado R. Z., Bard J. F. A branch-and-bound algorithm for permutation flow shops with sequence-dependent setup times. IIE Transactions. **31** (8), 721–731 (1999).

[12] Garey M. R., Johnson D. S., Sethi R. The Complexity of Flowshop and Jobshop Scheduling. Mathematics of Operations Research. **1** (2), 117–129 (1976).

[13] Nawaz M., Enscore E. E. Jr., Ham I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega. **11** (1), 91–95 (1983).

[14] Campbell H. G., Dudek R. A., Smith M. L. A Heuristic Algorithm for the n Job, m Machine Sequencing Problem. Management Science. **16** (10), B-579–B-697 (1970).

[15] Szwarc W., Gupta J. N. D. A flow-shop problem with sequence-dependent additive setup times. Naval Research Logistics (NRL). **34** (5), 619–627 (1987).

[16] Simons J. V. Jr. Heuristics in flow shop scheduling with sequence dependent setup times. Omega. **20** (2), 215–225 (1992).

[17] Das S. R., Gupta J. N. D., Khumawala B. M. A savings index heuristic algorithm for flowshop scheduling with sequence dependent set-up times. Journal of the Operational Research Society. **46** (11), 1365–1373 (1995).

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

[19] Osman I. H., Potts C. N. Simulated annealing for permutation flow-shop scheduling. Omega. **17** (6), 551–557 (1989).

[20] Widmer M., Hertz A. A new heuristic method for the flow shop sequencing problem. European Journal of Operational Research. **41** (2), 186–193 (1989).

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

[22] Lei D. A genetic algorithm for flexible job shop scheduling with fuzzy processing time. International Journal of Production Research. **48** (10), 2995–3013 (2010).

[23] Engin O., Ceran G., Yilmaz M. K. An efficient genetic algorithm for hybrid flow shop scheduling with multiprocessor task problems. Applied Soft Computing. **11** (3), 3056–3065 (2011).

[24] Mirsanei H. S., Zandieh M., Moayed M. J., Khabbazi M. R. A simulated annealing algorithm approach to hybrid flow shop scheduling with sequence-dependent setup times. Journal of Intelligent Manufacturing. **22**, 965–978 (2011).

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

[26] Qin H.-X., Han Y.-Y., Zhang B., Meng L.-L., Liu Y.-P., Pan Q.-K., Gong D.-W. An improved iterated greedy algorithm for the energy-efficient blocking hybrid flow shop scheduling problem. Swarm and Evolutionary Computation. **69**, 100992 (2022).

[27] Han X., Han Y., Chen Q., Li J., Sang H., Liu Y., Pan Q., Nojima Y. Distributed Flow Shop Scheduling with Sequence-Dependent Setup Times Using an Improved Iterated Greedy Algorithm. Complex System Modeling and Simulation. **1** (3), 198–217 (2021).

[28] Chen X., Li J., Wang Z., Li J., Gao K. A genetic programming based cooperative evolutionary algorithm for flexible job shop with crane transportation and setup times. Applied Soft Computing. **169**, 112614 (2024).

[29] Mou J., Duan P., Gao L., Liu X., Li J. An effective hybrid collaborative algorithm for energy-efficient distributed permutation flow-shop inverse scheduling. Future Generation Computer Systems. **128**, 521–537 (2022).

[30] Sun K., Zheng D., Song H., Cheng Z., Lang X., Yuan W., Wang J. Hybrid genetic algorithm with variable neighborhood search for flexible job shop scheduling problem in a machining system. Expert Systems with Applications. **215**, 119359 (2023).

[31] Wang L., Wang S., Zheng X. A hybrid estimation of distribution algorithm for unrelated parallel machine scheduling with sequence-dependent setup times. IEEE/CAA Journal of Automatica Sinica. **3** (3), 235–246 (2016).

[32] Li J.-Q., Du Y., Gao K.-Z., Duan P.-Y., Gong D.-W., Pan Q.-K., Suganthan P. N. A hybrid iterated greedy algorithm for a crane transportation flexible job shop problem. IEEE Transactions on Automation Science and Engineering. **19** (3), 2153–2170 (2021).

[33] Benkalai I., Rebaine D., Gagné C., Baptiste P. Improving the migrating birds optimization metaheuristic for the permutation flow shop with sequence-dependent set-up times. International Journal of Production Research. **55** (20), 6145–6157 (2017).

[34] Ríos-Mercado R. Z., Bard J. F. Heuristics for the flow line problem with setup costs. European Journal of Operational Research. **110** (1), 76–98 (1998).

[35] Shiue Y.-R., Lee K.-C., Su C.-T. Real-time scheduling for a smart factory using a reinforcement learning approach. Computers & Industrial Engineering. **125**, 604–614 (2018).

[36] Marchesano M. G., Guizzi G., Santillo L. S., Vespoli S. A Deep Reinforcement Learning approach for the throughput control of a Flow-Shop production system. IFAC-PapersOnLine. **54** (1), 61–66 (2021).

[37] Yang Y., Qian B., Hu R., Zhang D. Deep Reinforcement Learning Algorithm for Permutation Flow Shop Scheduling Problem. Intelligent Computing Methodologies. 473–483 (2022).

[38] Li H., Gao K., Duan P., Li J.-Q., Zhang L. 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 (2022).

[39] Du Y., Li J., Li C., Duan P. A Reinforcement Learning Approach for Flexible Job Shop Scheduling Problem With Crane Transportation and Setup Times. IEEE Transactions on Neural Networks and Learning Systems. **35** (4), 5695–5709 (2022).

[40] Kool W., Van Hoof H., Welling M. Attention, learn to solve routing problems! Preprint ArXiv:1803.08475 (2018).

[41] Liu Z., Wang Y., Liang X., Ma Y., Feng Y., Cheng G., Liu Z. A graph neural networks-based deep Q-learning approach for job shop scheduling problems in traffic management. Information Sciences. **607**, 1211–1223 (2022).

[42] Li J., Li J., Gao K., Duan P. A hybrid graph-based imitation learning method for a realistic distributed hybrid flow shop with family setup time. IEEE Transactions on Systems, Man, and Cybernetics: Systems. **54** (12), 7291–7304 (2024).

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

[44] Kaelbling L. P., Littman M. L., Moore A. W. Reinforcement learning: A survey. Journal of Artificial Intelligence Research. **4**, 237–285 (1996).

[45] Wauters T., Verbeeck K., De Causmaecker P., Vanden Berghe G. Boosting Metaheuristic Search Using Reinforcement Learning. Hybrid Metaheuristics. 433–452 (2013).

[46] Sutton R. S., Barto A. G. Reinforcement learning: An introduction. MIT Press Cambridge (1998).

[47] Watkins C. Learning from delayed rewards. King's College, Cambridge United Kingdom (1989).

Ітераційний жадібний алгоритм, що використовує механізм Q-навчання, для розв'язання задачі планування потокового виробництва з фіксованим порядком завдань та часом налаштування, залежним від послідовності

Месмар К.[1], Леббар М.[1], Аллалі К.[2]

[1] *Дослідницька група ROSDM, LMAID – Лабораторія прикладної математики та бізнес-аналітики, ENSMR Рабат, Марокко*
[2] *Лабораторія математики, інформатики та застосувань, FST Мохаммедія, Університет Хасана II, Марокко*

Ця стаття пропонує новий гібридний фреймворк, Q-IG, для розв'язання задачі планування потокового виробництва з фіксованим порядком завдань та часом налаштування, залежним від послідовності (PFSP-SDST). Нещодавні досягнення у методах, заснованих на навчанні, демонструють значний потенціал у вирішенні задач планування потокового виробництва, однак вони часто стикаються з проблемою величезного простору розв'язків та складністю розробки ефективних функцій винагороди. Щоб подолати ці виклики, Q-IG інтегрує ітераційну жадібну метаевристику (IG) із Q-навчанням. Алгоритм розпочинає свою роботу із застосування евристики Навази–Енскора–Гама (NEH) для генерації високоякісних початкових рішень. Протягом кожної ітерації IG розклади частково руйнуються та реконструюються, а Q-навчання використовується для спрямування вибору сусідніх кроків пошуку, які, як очікується, мінімізують загальну тривалість виконання. Оцінено як автономний IG, так і гібрид Q-IG на еталонному наборі задач Таїлларда, вимірюючи продуктивність через відносне відхилення у відсотках від найкращої відомої тривалості виконання. Результати демонструють, що Q-IG перевершує своїх конкурентів, підтверджуючи його ефективність для PFSP-SDST та підкреслюючи перспективність включення Q-навчання до традиційних метаевристичних підходів.

Ключові слова: *ітераційний жадібний метод; Q-навчання; задача планування потокового виробництва з фіксованим порядком завдань (PFSP); час налаштування, залежний від послідовності (SDST); тривалість виконання.*