


Article

# A Reinforcing-Learning-Driven Artificial Bee Colony Algorithm for Scheduling Jobs and Flexible Maintenance under Learning and Deteriorating Effects

Nesrine Touafek <sup>1,\*</sup> , Fatima Benbouzid-Si Tayeb <sup>1</sup> and Asma Ladj <sup>2</sup>

<sup>1</sup> Laboratoire des Méthodes de Conception de Systèmes (LMCS), Ecole Nationale Supérieure d'Informatique (ESI), Algiers BP 68M-16270, Algeria; f\_sitayeb@esi.dz (F.B.-S.T.); asma.ladj@railenium.eu (A.L.)

<sup>2</sup> Railenium Research and Technology Institute, 59540 Valenciennes, France

\* Correspondence: en\_touafek@esi.dz

**Abstract:** In the last decades, the availability constraint as well as learning and deteriorating effects were introduced into the production scheduling theory to simulate real-world case studies and to overcome the limitation of the classical models. To the best of our knowledge, this paper is the first in the literature to address the permutation flowshop scheduling problem (PFSP) with flexible maintenance under learning and deterioration effects to minimize the makespan. Firstly, we address the PFSP with flexible maintenance and learning effects. Then, the deteriorating effect is also considered. Adaptive artificial bee colony algorithms (ABC) enhanced with Q-learning are proposed, in which the Nawaz–Enscore–Ham (NEH) heuristic and modified NEH heuristics are hybridized with a maintenance insertion heuristic to construct potential integrated initial solutions. Furthermore, a Q-learning (QL)-based neighborhood selection is applied in the employed bees phase to improve the quality of the search space solutions. Computational experiments performed on Taillard's well-known benchmarks, augmented with both prognostic and health management (PHM) and maintenance data, demonstrate the effectiveness of the proposed QL-driven ABC algorithms.

**Keywords:** permutation flowshop problem; flexible maintenance; learning effect; deteriorating effect; artificial bee colony algorithm; Q-learning



**Citation:** Touafek, N.; Benbouzid-Si Tayeb, F.; Ladj, A. A Reinforcing-Learning-Driven Artificial Bee Colony Algorithm for Scheduling Jobs and Flexible Maintenance under Learning and Deteriorating Effects. *Algorithms* **2023**, *16*, 397. <https://doi.org/10.3390/a16090397>

Academic Editor: Marc Sevaux

Received: 11 July 2023

Revised: 18 August 2023

Accepted: 19 August 2023

Published: 22 August 2023



**Copyright:** © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Scheduling is an important tool for manufacturers and production engineers since it deals with the allocation of human and machine resources to satisfy consumer demand. The objective is to monitor and optimize workloads, as well as to reduce production time and costs while adhering to all production constraints.

In traditional scheduling problems and models, job processing times are assumed to be a constant value. However, this hypothesis is not suitable for many realistic situations in which the job processing time may change owing to job deterioration and/or learning effect phenomena. Indeed, when the employees execute the same task repeatedly, they learn how to perform more efficiently. On the other hand, machines wear out after long periods of processing. Therefore, recent trends in scheduling theory have focused on modeling real-world production scheduling problems where human learning, as well as machine deteriorating effects, are taken into account when studying scheduling problems [1,2]. The learning effect [3] is a natural human-oriented phenomenon appearing among operators after repeating similar tasks frequently. They gain experience and the execution time decreases accordingly. Ref. [4] proposed the first modeling of this phenomenon with a learning curve and since then, multiple models have been proposed in the literature. Refs. [3,5] were among the pioneers that investigated the learning effect in scheduling problems, and this was continued in the literature in hundreds of studies with different

machine configurations [6,7]. On the other hand, the deteriorating effect [8,9] is a machine-oriented phenomenon appearing after processing tasks are conducted on machines for a long time. The machine performances wear out and therefore additional delays are required to execute the production schedule. Refs. [8,10] seem to be the first to have investigated this effect in production scheduling. The interested reader can find extensive surveys of different scheduling models and problems involving deteriorating jobs in [11,12]. More recently, Ref. [13] presented an updated survey of the results on scheduling problems with deteriorating jobs and time-dependent processing times. As for learning and/or deterioration effects, Ref. [7] reviews the mathematical models for scheduling problems. It is worth noting that most proposed learning or deteriorating effects functions are either position-dependent, time-dependent, or sum-of-processing-times-dependent [6,13,14].

To counter the deteriorating effect, increase machines' availability, and reduce the overall production costs, maintenance activities should be planned along the production process [15]. However, unavailability periods for machines appear since the maintenance activities may take up a potential production time that could otherwise be allocated to completing production tasks. Therefore, to compensate for this deficiency, maintenance tasks should be scheduled in an effective way to improve machine availability and minimize disruption to the production process [16]. Ref. [17] established that appropriate maintenance planning can preserve machine performance and save 20–30% of total operating costs.

Recently, maintenance strategies have moved towards a smart aspect. In fact, modern sensor, diagnosis, and prognostic and health management (PHM) technologies allow the ideal planning of interventions and reduce inopportune spending. Simply put, signals gathered from embedded sensors (or inspection information) are analyzed in order to continuously (or periodically) monitor the health state of the production system. Following that, the current state is inferred and the future progression till failure is predicted in order to estimate the time left before the occurrence of failure, known as the remaining useful life (RUL) [18]. Then, the PHM outputs (i.e., RUL) are exploited in the decision making that concerns maintenance planning. The main advantage of this strategy, namely the decision making based on prognosis information, which is referred to as a post-prognostic decision (PPD) [19], is that it maintains the system only when necessary, which reduces the cost of inopportune maintenance interventions.

The current PPD research on scheduling problems is focused on two main streams. First, the intent is to make the best PPD regarding maintenance planning. In order to improve maintenance planning, PHM findings are thus used [20–23]. On the other hand, the second stream looks into the issue of integrating production jobs and maintenance operations when making decisions. This can help to avoid production downtime and improve system availability by establishing the most appropriate integrated production and maintenance schedule [16,18,24–27].

Metaheuristics have emerged as a common solution approach for scheduling in industry and manufacturing due to the fact that most problems are computationally hard [28]. Recently, bio-inspired metaheuristics, including evolutionary algorithms and swarm intelligence algorithms, have become increasingly popular [29]. Evolutionary algorithms start with a set of random candidate solutions and iteratively generate descendant solutions until an acceptable one is found. Common metaheuristics in this category that have many applications in scheduling problems are genetic algorithms [30,31], the differential evolution algorithm [32], and more recently, biogeography-based optimization (BBO) [33]. Swarm Intelligence is an advanced artificial intelligence discipline that uses the behavior of living swarms such as ants, bees, butterflies, and birds to solve complex problems [34]. Several metaheuristics in this category have been applied to solve scheduling problems such as Ant Colony Optimization (ACO) [35], Particle Swarm Optimization (PSO) [36], and more recently, Grey Wolf Optimization (GWO [37]). The artificial bee colony (ABC) algorithm [38], which has recently been introduced to formulate NP-hard problems, simulates the social behavior of bees when searching for promising food sources associated with higher nectar amounts. The ABC algorithm has been extensively applied to solve

various flowshop scheduling problems [39–41]. On the other hand, machine learning (ML) is an expansive field within artificial intelligence that has gained significant popularity due to its straightforward algorithms, enabling programs to learn patterns, behaviors, models, and functions. This acquired knowledge is then utilized to enhance future decision making and actions. ML can be categorized into three main types: supervised learning, unsupervised learning, and reinforcement learning. In recent years, reinforcement learning (RL) methods have been applied to machine scheduling problems [42]. It is noteworthy that among scheduling problems, jobshop scheduling, flowshop scheduling, unrelated parallel machine scheduling, and single-machine scheduling have emerged as the most extensively studied ones [42].

While integrating maintenance decisions in production scheduling problems makes existing models and solution methods more realistic, it also increases their complexity. Furthermore, incorporating effects further increases the problem's complexity. Additionally, while most research on deteriorating jobs and learning effects has focused on single-machine problems [1], it is the multimachine problems that are more interesting and closer to real-world problems. Therefore, the intent of this paper is to explore the permutation flowshop scheduling problem (PFSP) with flexible maintenance under learning and deteriorating effects, through the perspective of reinforcement learning algorithms. In this regard, we propose two Q-learning (QL)-driven artificial bee colony (ABC) algorithms to solve the defined PFSP with only the learning effect and the defined PFSP with simultaneous consideration of learning and deteriorating effects. The main advantages of the proposed hybrid metaheuristic approaches are the following:

1. To be close to realistic production workshops, we propose a novel interpretation of PHM outputs, where RULs and degradation values are estimated based on the operating conditions of machines during the processing of production jobs;
2. The use of the QL algorithm, one of the major RL algorithms, to guide the search process in the employed bees phase of the ABC algorithms;
3. The design of a new heuristic, named INEH, based on the hybridization of the NEH heuristic [43] and maintenance insertion heuristic [44] concepts, to generate initial solutions of good quality;
4. A maintenance cost function to be optimized when readjusting maintenance emplacements after being perturbed in employed and onlooker bees phases;

The remainder of this paper is organized as follows. A literature review and the research gaps are given in Section 2. The problem formulation and objective function are provided in Section 3. Section 4 details the proposed ABC-based solving approaches. Section 5 discusses the performance analyses and presents our computational results. Finally, some conclusions and future works are discussed in Section 6.

## 2. Brief Literature Review and Research Gaps

Scheduling problems are considered to be of very complex resolution. Not only are they NP-complete [45], but they are also actually among the most difficult problems to solve because of the difficulty of computing solutions [46]. The joint scheduling problem addressed in this paper is even more complex as it involves not only assigning production jobs and maintenance activities to machines but also accounting for the impact of learning and deteriorating effects on processing times.

### 2.1. Metaheuristic Solution Approach

As a result of their efficiency and ability to be used in real-world production environments, metaheuristics as approximation methods have made breakthroughs in solving various complex optimization problems in the industry and become the main methods used to solve large-scale scheduling problems [47]. Nonetheless, it is worth noting that few studies in the literature have proposed metaheuristics to solve scheduling problems with learning and/or deteriorating effects [1,2].

The existing literature can be placed into three main streams according to the considered effect, namely the learning effect, the deteriorating effect, and finally, the combined effects. Reference [48] proposed a tabu search algorithm to solve the single-machine scheduling problem with a time-dependent learning effect. Ref. [49] developed four metaheuristics, namely the genetic algorithm (GA), simulated annealing, ABC, and iterated greedy algorithms to solve a two-stage, three-machine assembly scheduling problem with a position-dependent learning effect. On the other hand, Ref. [41] proposed an ABC algorithm to solve the flexible flowshop scheduling problem with a step deteriorating effect, while [50] proposed three metaheuristics including dynamic differential evolution, the simulated annealing algorithm, and the cloud-theory-based simulated annealing algorithm, to solve the two-stage assembly scheduling problem under a time-dependent deteriorating effect. In practical settings, combined effects (learning and deteriorating) are also considered. In this regard, we can quote the work of [51], which developed a GA to solve parallel machine scheduling problems under the effects of position-dependent learning and time-dependent deteriorating effects.

Joint planning of production and maintenance activities, taking into account learning and/or deteriorating effects has also been studied in the literature. However, works in this area are even more scarce. From the learning perspective, Ref. [52] solved the joint flowshop scheduling problem with a hybrid metaheuristic algorithm based on the simulated annealing algorithm and firefly algorithm. For the deteriorating effect, Ref. [53] developed a GA to deal with the joint single-machine scheduling problem taking into account the deteriorating effect. Finally, considering the case of combined effects, Ref. [54] proposed an ABC algorithm to solve the joint flowshop scheduling problem under the time-dependent deteriorating effect and position-dependent learning effect. Ref. [55] studied the joint parallel batching scheduling problem under a combined position-dependent learning effect and time-dependent deteriorating effect. A hybrid algorithm combining GA and differential evolution (DE) was proposed for resolution.

Based on the findings of the literature, it has been observed that scheduling problems frequently feature position-dependent learning effects and/or time-dependent deteriorating effects [2]. Furthermore, variable maintenance durations due to learning or deteriorating effects is also an important research stream in the scheduling literature. While variable maintenance durations due to the deterioration effect have most often been considered [56–58], variable maintenance durations due to the learning effect have scarcely been studied [54,59].

## 2.2. Metaheuristic and RL Solution Approach

Recently, there has been a growing focus on using reinforcement learning (RL) [60], a class of ML algorithms, as an emerging intelligent optimization method for addressing complex combinatorial optimization problems. The RL algorithms have been successfully applied to solve a wide range of production scheduling problems [42,61]. Ref. [62] proposed a Q-learning (QL) algorithm to solve the classical PFSP. The authors reported competitive results compared to the optimum values proposed in the literature. Ref. [63] applied the QL algorithm to the makespan hybrid flowshop scheduling problem. To validate the method, two real-life applications were made. Ref. [64] proposed a multi-agent reinforcement learning framework for the jobshop scheduling problem. The best results are returned by the approach using well-known benchmark instances.

In the literature, integrating RL algorithms into metaheuristics has been proven to offer high performance in terms of search robustness, solution quality, and convergence speed [42]. Metaheuristics can rely on RL algorithms to calibrate their key parameters [31,65] or support their search process [66,67]. Despite extensive efforts to incorporate RL algorithms into metaheuristics for scheduling problem resolution, little research has addressed constraints such as maintenance, learning, and deteriorating effects [42]. Furthermore, despite the extensive literature on flowshop scheduling problems, to the best of our knowledge, references to the problem addressed in this paper, that is, the PFSP

with flexible maintenance under learning and deteriorating effects, are very scarce. Hence, this paper introduces several enhancements to the ABC algorithm based on the specific characteristics of the problem under investigation. These improvements, including the hybrid initialization method and QL-based neighborhood search, aim to improve the search capability and overall performance.

### 3. Integrated Problem Description and Effects Mathematical Models

In this section, we first describe the integrated permutation flowshop scheduling problem (PFSP) with flexible maintenance that we are dealing with. We focus on the variable duration of maintenance and job processing times due to learning and deteriorating effects. Furthermore, the assumptions made in this paper are provided. Finally, we discuss the most salient aspects of learning and deteriorating effects. The following notations are used for the remainder of this paper:

$J$	indicates a set of production jobs to be processed.
$j$	indicates the index of a job in the set $J$ ( $1 \leq j \leq n$ ).
$M_i$	indicates the $i$ th machine, $i$ is used as the machine index ( $1 \leq i \leq m$ ).
$k_i$	indicates the number of planned maintenance operations on machine $i$ .
$M_{ic}$	indicates the planned maintenance operation on machine $i$ at the $c$ th position, where $c$ is used as the maintenance position indicator ( $1 \leq c \leq k_i$ ).
$p_{ij}$	indicates the normal processing time of job $j$ on machine $i$ .
$p_{ij}(t)$	is the time-dependent deteriorating processing time of job $j$ on machine $i$ , where $t$ is its start time.
$\beta$	is the deteriorating index.
$pm_{ic}$	is the normal time of the $c$ th maintenance on machine $i$ .
$pm_{ic}(c)$	is the position-dependent learning time of the $c$ th maintenance on machine $i$ .
$\alpha$	is the learning index.
$\sigma_{ij}$	indicates the degradation of machine $i$ after processing the job $j$ .
$B_{il}$	indicates a subset of $J$ with $n'$ jobs where $n' < n$ .
$C_{max}$	indicates the total completion time.

#### 3.1. The Integrated Scheduling of Production and Maintenance Problem

The PFSP is one of the most thoroughly studied scheduling problems in the operation research literature. Many production scheduling problems resemble a flowshop, and therefore, it has been extensively studied due to its application in industry [68].

Formally, the PFSP can be described as follows. A set  $J = \{j, 1 \leq j \leq n\}$  of  $n$  independent jobs needs to be processed on a set  $M = \{M_i, 1 \leq i \leq m\}$  of  $m$  independent machines in the same order on the  $m$  machines. We assume that all jobs are available at time zero and no preemption is allowed. In order to ensure the system's reliability and to prevent the occurrence of fatal breakdowns during the processing horizon, machines are monitored continuously by a PHM module. This module is able to predict, for each machine  $M_i$ , the relative time before failure after processing a job  $j$ , noted as  $RUL_{ij}$ , and the resulting degradation value,  $\sigma_{ij}$ , is then deduced. When the cumulative jobs degradation reaches a certain threshold,  $\Delta$ , a maintenance operation should be planned.

The resulting integrated sequence is denoted  $\Pi = \{\pi_1, \pi_2, \dots, \pi_m\}$  where  $\pi_i$  represents the corresponding integrated sequence of  $n$  jobs and  $k_i$  ( $\geq 1$ ) flexible maintenance on machine  $M_i$ . Then  $\pi_i$  can be seen as a succession of  $k_i + 1$  blocks of jobs ( $B_{il} = \{\text{subset of } J\}$ ) separated by maintenance operations ( $M_{ic}$ ):

$$\pi_i = \{B_{i1}, M_{i1}, B_{i2}, M_{i2}, \dots, B_{ik_i}, M_{ik_i}, B_{i(k_i+1)}\}, \text{ where } \bigcup_{l=1}^{k_i+1} B_{il} = J.$$

The general assumptions of the integrated problem are given as follows:

- At time zero, all jobs are available and ready for processing and no preemption is allowed;
- Each machine can handle at most one action (production or maintenance) and each job can be processed by at most one machine;
- Each production job  $j$  (maintenance operation  $M_{ic}$ ) requires a basic amount of processing time on each machine  $i$ , represented by  $p_{ij}$  ( $pm_{ic}$ );

- We fix the degradation value associated to job  $j$  after being processed on machine  $i$ ,  $\sigma_{ij} = p_{ij} / RUL_{ij}$ ; where  $0 < \sigma_{ij} < 1$ .
- The maximal authorized threshold degradation  $\Delta$  is fixed to 1 for all machines;
- At least one maintenance intervention is performed on each machine and no maintenance intervention is performed after the processing of the last job;
- After a maintenance intervention, the machine is recovered to an "As good as new" state.

### 3.2. Learning and Deteriorating Effects' Mathematical Models

Due to the learning effect, we suppose that the maintenance durations will decrease gradually as they are scheduled later in the sequence. This is the position-dependent learning model (Equation (1)). On the other hand, for the deteriorating effect, we suppose that the production job processing times will increase gradually over time. This is the time-dependent deteriorating model (Equation (2)).

$$pm_{ic}(c) = pm_{ic} \times c^{\alpha}, -1 < \alpha < 0 \quad (1)$$

$$p_{ij}(t) = p_{ij} + \beta \times t, \beta > 0 \quad (2)$$

## 4. Proposed QL-Driven ABC-Based Solving Approaches

The ABC algorithm [38] is a competitive bio-inspired algorithm that simulates the behavior of bee colonies when searching for promising food sources associated with higher nectar amounts. In the ABC algorithm, the food source represents a candidate solution to the problem and the nectar amount of a source corresponds to the quality, known as fitness, of the related solution. The ABC search process is conducted by three bee swarms, namely: employed, onlooker, and scout bees. A review of its fundamentals and some applications can be found in [69]. In [70], the ABC algorithm showed effective performances over the differential evolution (DE) algorithm, Particle Swarm Optimization (PSO), and evolutionary algorithm (EA). The main advantages of the ABC algorithm are its ability to balance exploitation and exploration, it having few control parameters, its fast convergence, its strong robustness, and its ease of being combined with other methods [71]. It is also easy to implement compared to GA and Ant Colony Optimization (ACO) [72].

In an attempt to solve the integrated PFSP with a flexible maintenance problem under learning and deteriorating effects in a reasonable time, in this paper, we propose two efficient ABC algorithms enhanced with the RL mechanism. The first ABC algorithm, named IQABC-LE for Integrated QL-based ABC under Learning Effect, takes into account the learning effect. After generating the initial solutions, the search process of IQABC-LE starts with the employed bees phase, which applies an RL mechanism to drive the neighborhood exploitation of the associated solutions in order to update the search space with optimal solutions. After that, the onlooker bees select the best individuals from the search space to exploit their neighborhood for further optimization. To allow exploration of the search space, the scout bees randomly replace the solutions that have not been updated after a defined number of trials. The search process is iterated until a termination condition is met. The general scheme of the IQABC-LE algorithm is given in Algorithm 1 and its flow chart is indicated in Figure 1. The second ABC algorithm, named IQABC-LDE for Integrated QL-based ABC under Learning and Deteriorating Effects, follows the same steps as IQABC-LE but also takes into account the deteriorating effect. The main features of the proposed algorithms are detailed in the sections below.

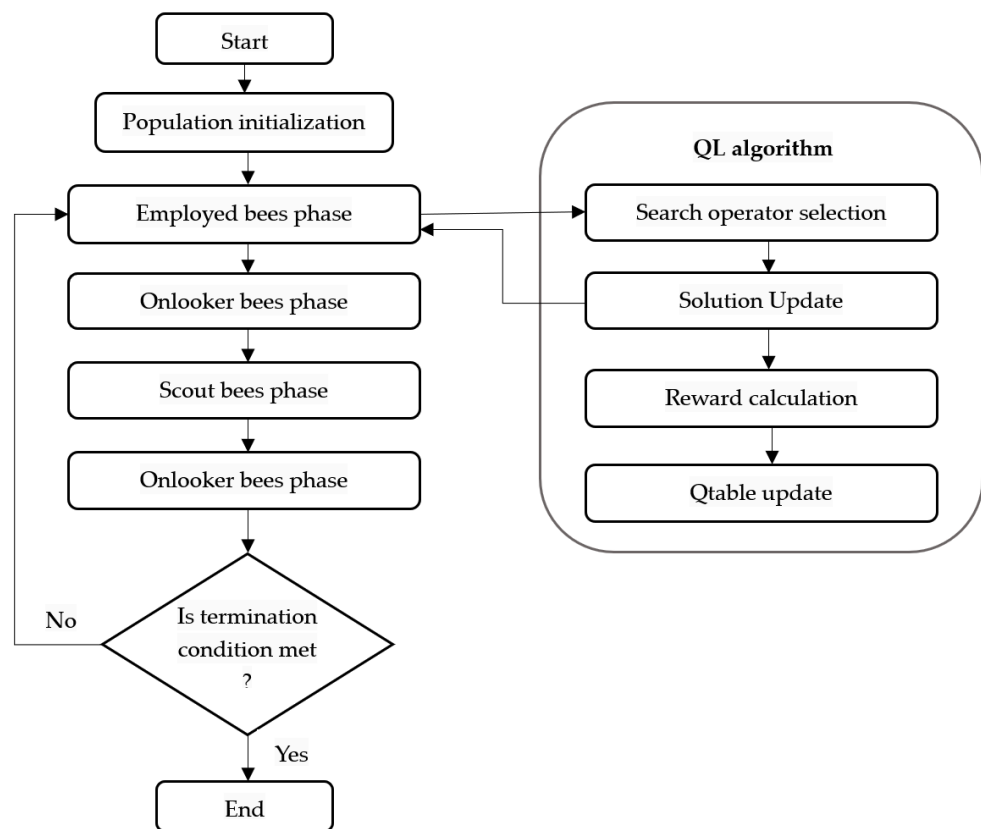


Figure 1. Flow chart of the IQABC-LE algorithm.

4.1. Food Source Representation and Evaluation

A suitable solution representation is based on the work of [73]. A candidate solution *Sol*, i.e., food source, jointly specifies the production job sequence and the maintenance operations’ positions on each machine and is coded by a two-field structure. The first field is a vector  $S = \{j, j \in [1, n]\}$  representing the production job order. The second field represents the scheduling of maintenance operations on each machine by a binary matrix  $P$  of  $m \times n$  size. An element  $P[i, j]$  of  $P$  indicates if a maintenance intervention is scheduled on machine  $M_i$  after job  $j$ , and then  $P[i, j] = 1$ , or not, and then  $P[i, j] = 0$ . Let

$$S = (1 \ 9 \ 3 \ 8 \ 5 \ 6 \ 7 \ 4 \ 2 \ 0), \text{ and}$$

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

represent a solution of an integrated flowshop scheduling problem with  $m = 3$  machines and  $n = 10$  jobs. Then, the execution order of the ten jobs and maintenance operations on the three machines is the following:

- Machine 1: 1, 9,  $M_{11}$ , 3, 8, 5,  $M_{12}$ , 6, 7, 4, 2,  $M_{13}$ , 0.
- Machine 2: 1, 9, 3,  $M_{21}$ , 8, 5, 6,  $M_{22}$ , 7, 4, 2, 0.
- Machine 3: 1, 9,  $M_{31}$ , 3, 8, 5, 6,  $M_{32}$ , 7, 4, 2, 0.

**Algorithm 1** IQABC-LE()

**Require:** Population size (PopSize), Onlooker bees percentage (onlook%), Number of generations (max\_iterations), Abandonment number of trials (limit)

```

Begin
Ebees[1] ← INEH()                                ▷ Population initialization step
for i ← 1 to alpha%PopSize do
    Ebees[i] ← ModifiedNeh()
end for
for j ← 1 to (PopSize-alpha%PopSize-1) do
    Ebees[j] ← RandomGeneration()
end for
iter ← 1
while iter ≤ max_iterations do
    for i ← 1 to PopSize do                                ▷ employed bees phase
        NeighborEbee ← QL(Ebees[i])                        ▷ Qlearning selection (Algorithm 3)
        if NeighborEbee ≥ Ebees[i] then
            Ebees[i] ← NeighborEbee
        end if
    end for
    for i ← 1 to onlook%PopSize do                        ▷ onlooker bees phase
        Olook[i] ← probaSelection(Ebees)
        Olook ← ILS(Olook[i])
        if Olook ≥ Olook[i] then
            Ebees[i] ← Olook
        end if
    end for
    for i ← 1 to PopSize do                                ▷ scout bees phase
        if CheckLimit(Ebees[i],limit)=True then
            Ebees[i] ← RandomGeneration()
        end if
    end for
end while
Return Best Solution()
End

```

The objective is to find a solution *Sol* that minimizes the amount of nectar (the measure of the quality of a solution), i.e., the best sequencing of jobs and maintenance operations to be processed for each machine in order to minimize the total completion time of the schedule,  $C_{max}$ , taking into account maintenance interventions planning under learning and deteriorating effects. Considering the learning and the deteriorating effects, and the fact that the total completion time  $C_{max}$  depends mainly on the sum of the job processing times and maintenance durations (see Formula (3)), the  $C_{max}$  while considering the learning effect and both effects simultaneously are given by Formulas (4) and (5), respectively.

$$C_{max} = IT + \sum_{j=1}^{j=n} p_{mj} + \sum_{c=1}^{c=k_m} pm_{mc} \quad (3)$$

$$C_{max} = IT + \sum_{j=1}^{j=n} p_{mj} + \sum_{c=1}^{c=k_m} pm_{mc}(c) = IT + \sum_{j=1}^{j=n} p_{mj} + \sum_{c=1}^{c=k_m} (pm_{mc} \times c^\alpha) \quad (4)$$

$$C_{max} = IT + \sum_{j=1}^{j=n} p_{mj}(t) + \sum_{c=1}^{c=k_m} pm_{mc}(c) = IT + \sum_{j=1}^{j=n} (p_{mj} + \beta \times t) + \sum_{c=1}^{c=k_m} (pm_{mc} \times c^\alpha) \quad (5)$$

where: *IT* refers to the total idle time of the last machine *m* waiting for jobs arrival and  $k_m$  is the number of inserted maintenance activities on the last machine *m*.



#### 4.2. Initial Population Generation

To allow quality and diversity in the search space, we generated an initial population of candidate solutions at random and “seeded” it with some good ones. Therefore, in this work, we suggest the following three-step initialization procedure to generate an initial population of  $PopSize$  complete integrated solutions ( $S, P$ ):

- **Step 1.** Firstly, we generated the first candidate food source of the initial population with a newly designed heuristic, named INEH, a greedy approximate heuristic based on the well-known NEH heuristic [43], by modifying its second phase.
- **Step 2.** Generate  $\alpha\%PopSize$  production sequences using a modified NEH heuristic [74]. To generate complete food sources, maintenance activities are scheduled using the PHM-based greedy heuristic of [44]. In this heuristic, maintenance operations are inserted according to the current accumulated degradation of the machine estimated by the PHM module, starting from the first machine  $M_1$  to the last one  $M_m$ .
- **Step 3.** The rest of the population is generated randomly with production sequences generated at random and maintenance operations inserted using the heuristic of [44].

The main idea of the INEH heuristic is to build from scratch the sequence of production jobs and maintenance operations giving priority to jobs with higher processing times to be scheduled first. In the first phase, the jobs are ordered according to the LPT rule in a list  $L$ . Then, a partial sequence is generated with the first job of  $L$ . The second phase is an insertion procedure to iteratively construct a complete integrated sequence. This phase starts first, by scanning the current partial sequence, and maintenance operations are inserted when necessary with respect to the threshold  $\Delta=1$  on each machine. Then, by testing the second job of  $L$  in all available positions of the integrated partial sequence. The position that yields the minimum makespan is chosen to be inserted, and the new partial integrated sequence overwrites the last one. This insertion step is repeated iteratively with the rest of the jobs. Detailed steps of the INEH heuristic are given in Algorithm 2.

---

#### Algorithm 2 INEH Heuristic

---

**Require:** A Set of production jobs

**Begin**

Let  $L$  the list of ranked production jobs in decreasing order of total processing time (LPT rule).

Insert the first job of  $L$  in the first position of the current partial empty sequence.

Update  $L$

**while**  $L$  not Empty **do**

    Insert maintenance operations in the appropriate positions of the current partial sequence with respect to the threshold  $\Delta$

    Insert the first job of  $L$  in all the  $k+1$  possible positions of the current partial sequence

    Evaluate all of  $k+1$  resulting partial sequences

    Keep the best sequence as a new current partial sequence

    Update  $L$

**end while**

Return Integrated sequence

**End**

---

#### 4.3. QL-Based Employed Bees Phase

RL studies the sequential decision process of interaction between the agent and the environment [75]. Q-learning (QL) is one of the major RL algorithms that allows agents to learn an optimal policy through trial and error in an unknown environment. It does not rely on prior knowledge of the environment’s dynamics [76]. The algorithm uses a value function called the  $Q$ -function to estimate the expected cumulative reward for taking specific actions in certain states and following the optimal policy thereafter.

During the learning process, the agent explores the environment, taking actions and observing the resulting rewards and next states. These experiences are used to iteratively update the Q-values, which represent the estimated quality of actions in different states. The update process is guided by the Bellman equation, which relates the current Q-value to the expected future Q-values.

Motivated by the above consideration, this paper introduces the QL principle to the selection of neighborhood structures in order to guide the search process in the employed bees phase of the proposed IQABC algorithms. Employed bees are in charge of searching for new and hopefully better food sources. To this end, in the original ABC algorithm, each employed bee generates one new candidate solution in the neighborhood of the current solution in one food source. The new candidate replaces the old solution if it is better. In this paper, each employed bee is placed on a solution from the population and applies one of the neighborhood structures on it, to generate a new solution. The choice of the neighborhood structure is performed by the application of a QL algorithm. The new solution replaces the current one if its quality is better, otherwise, a non-optimization trials counter, *nb\_trials*, is incremented.

#### 4.3.1. Neighborhood Structures

Neighborhood structures are designed for further optimization of the initial solutions. Based on previous studies on scheduling, it has been found that the insert neighborhood is generally more effective for the PFSP [77]. Furthermore, it is advised to incorporate complementary neighborhood structures to ensure a thorough exploration of the search space and enhance the effectiveness of the search process. Therefore, in this paper, insert and swap neighborhoods are used. These neighborhood structures allow the generation of new solutions by changing the execution order of production jobs. Additionally, a third type of neighborhood structure involving the shifting of maintenance tasks is utilized. Based on these neighborhoods, six local search operators are proposed as follows:

1. **Swap move on production jobs.** It consists of swapping the positions of two production jobs selected randomly from the production sequence.
2. **Double swap move on production jobs.** It consists of making two consecutive swap moves on the production sequence.
3. **Insert move on production.** It consists of selecting a production job randomly and then inserting it into another randomly selected position in the production sequence.
4. **Double insert move on production jobs.** It consists of making two consecutive insert moves.
5. **Right shift move on maintenance activities.** It consists of selecting a maintenance operation in a selected machine and then shifting it to the right, i.e., after the next job in the sequence.
6. **Left shift move on maintenance activities.** It consists of selecting a maintenance operation in a selected machine and then randomly shifting it to the left, i.e., before the previous job in the sequence.

#### 4.3.2. Q-Learning Algorithm

In the proposed IQABC algorithms, a QL algorithm is used to guide the choice of the neighborhood structures in the employed bees phase. The states of the QL model are defined by the set of population individuals, while the six neighborhood structures indicate the set of actions. At each iteration, the QL agent observes the current solution (*s*), selects an action (*a*) (a neighborhood structure), which generates a new solution (*s'*), and consequently, it receives a reward/penalty (*r*) signal according to the quality of the generated solution (see Formula (6)). The agent records its learning experience by a Q-value ( $Q(s,a)$ ), which is stored in a Q-table. The update of the Q-value is given in Formula (7), where  $\alpha_{ql}$  is the Q-learning rate and  $\gamma$  is the discount factor.

$$r = 1 + C_{max}(S) - C_{max}(S') \quad (6)$$

$$Q(s, a) = Q(s, a) + \alpha_{ql} \times (r + \gamma \times \max_{a'} Q(s', a') - Q(s, a)) \quad (7)$$

The action selection mechanism is implemented using an  $\epsilon$ -greedy strategy, where the agent can choose the best action considering the associated  $Q$ -value (exploitation), or can select one action randomly (exploration) with an  $\epsilon$  probability. Thus, the  $\epsilon$ -greedy strategy enables both exploitation and exploration. That is, the agent has to exploit what it already knows to obtain a reward, but it also needs to explore in order to make better action selections in the future. The employed bees stage with the QL algorithm is recalled in Algorithm 3.

---

### Algorithm 3 QL-based Employed bees phase

---

**Require:** Search space,  $Q$ -table of the population

**Begin**

**for** Each solution  $S$  from the search space **do** ▷  $S$  is the current state

Generate a random number  $a$

**if**  $a \leq \epsilon$  **then**

Choose the best action (Neighbor structure  $NS$ ) associated with the best  $Q$ -value of the  $Q$ -table.

**else**

Choose the best action  $NS$  randomly

**end if**

Generate New Solution  $Sol'$  using  $NS$  ▷  $Sol'$  is the next state

**end for**

$r \leftarrow 1 + \text{makespan}(Sol) - \text{makespan}(Sol')$  ▷ reward formula

Update the  $Q$ -value of the tuple  $(Sol, NS)$  as specified in Formula (7)

**if**  $\text{makespan}(Sol') \leq \text{makespan}(Sol)$  **then**

Replace  $Sol$  with  $Sol'$

**end if**

update the search space

**End**

---

To implement the QL algorithm, we used the “pandas” python library. The  $Q$ -table is stored in a DataFrame structure, which is a two dimensional table where indexes (lines) point to food sources as represented in Section 4.1, and columns refer to a defined list of neighborhood structures as presented in Section 4.3.1.

#### 4.4. Onlooker Bees Phase

Similar to the employed bees phase, the onlooker bees phase aims to enhance the intensification of the local search process. Firstly, the onlooker bees select candidate solutions from the employed bees stage according to the roulette wheel method. Then a local search heuristic borrowed from the iterated local search (ILS) algorithm [78] with destruction and reconstruction procedures is applied to selected solutions for exploitation. The probability that a solution  $(Sol_i)$  is allowed to be selected is defined in Formula (8).

$$P(Sol_i) = \frac{fitness(Sol_i)}{\sum_{k=1}^n fitness(Sol_k)} \quad (8)$$

#### 4.5. Scout Bees Phase

In the scout bees phase, candidate solutions that have not been updated after a defined number of trials, let this be *limit*, are abandoned and replaced by new random ones.

#### 4.6. Readjustment Procedure for Maintenance Cost Minimization

The perturbations on the candidate solutions, through the neighborhood structures within the employed bees stage, and the local search heuristic within the onlooker bees

phase may disturb the positions of the required maintenance interventions, which can either be advanced or delayed. In both cases, a maintenance cost needs to be considered. An early maintenance intervention is detected when the cumulative degradation caused by the production jobs scheduled before it is less than the threshold  $\Delta$ . In this case, we define the maintenance cost as follows:

$$\text{maint\_cost} = 200 \times |\Sigma\sigma - \Delta| + 100$$

In the case of tardy maintenance intervention, i.e, when the cumulative degradation caused by the production jobs scheduled before it is greater than the threshold  $\Delta$ , we define the maintenance cost as follows:

$$\text{maint\_cost} = 400 \times |\Sigma\sigma - \Delta| + 100$$

An optimal maintenance cost is reached when the cumulative degradation is equal to  $\Delta$ .

In this regard, we propose to apply a readjustment procedure (Algorithm 4) after the generation of new solutions in the employed and onlooker bees phase that browses complete solutions to detect early/tardy maintenance interventions and tries to readjust them in the appropriate positions that optimize the maintenance cost.

---

#### Algorithm 4 Readjustment procedure

---

**Require:** Sol (a returned solution with job permutation vector S and the associated maintenance matrix P),  $\Sigma\sigma$  the cumulative machine degradation.

**Begin**

$\Sigma\sigma \leftarrow 0$

**for** Each machine  $M_i (i \leq m)$  **do**

$\Sigma\sigma \leftarrow 0$

**for** Each job  $j$  from S **do**

$\Sigma\sigma \leftarrow \Sigma\sigma + \sigma_{ij}$

**if**  $P[i,j]=1$  **then**

$\Sigma\sigma \leftarrow 0.$

**end if**

**if**  $\Sigma\sigma \geq \Delta$  **then**

Insert the maintenance before or after  $j$  in order to minimize the maintenance cost.

Update the matrix P with the maintenance emplacement.

Remove the next maintenance:  $P[i,(\text{next maintenance position})] \leftarrow 0.$

$\Sigma\sigma \leftarrow 0.$

**end if**

**end for**

**end for**

**End**

---

#### 4.7. Termination Condition

The termination condition of the proposed IQABC is a maximum number of iterations (*max\_iterations*) of the algorithm or when the limit of the number of iterations without improving the best solution (*stagnation*) is reached.

### 5. The Computational Results and Analysis

In this section, the performance of the designed QL-driven ABC algorithms IQABC-LE and IQABC-LDE is demonstrated through numerical experiments on a well-designed test bed. The experiments are conducted in Python 3.9.5 on a personal computer with Windows 10 operating system, Intel i5 CPU (2.10 GHz), and 8-GB RAM.

In the following, we first describe how test data are generated. Secondly, to analyze the performance of our newly proposed algorithms, we conducted two sets of experiments. In the first set, the calibration process, and a comparison between IQABC without learning and deteriorating effects and a standard ABC were performed. In the second, we validated our IQABC algorithms with effects, namely IQABC-LE and IQABC-LDE, by comparing them with several representative algorithms. We ran, independently, all the algorithms five times on each instance. The complete details are reported in the next sections.

### 5.1. Data Generation

We used two types of data to assess our proposed methods. The first was about production, while the second was about PHM and maintenance data. For the first type, we ran our experiments on a set of 110 known Taillard instances [79] of different sizes ( $n \times m$ ), with  $n \in \{20, 50, 100, 200\}$  and  $m \in \{5, 10, 20\}$ . To complete the production data with the PHM data and the maintenance duration, we proceeded as follows:

- For each instance, we generated three levels of machines' degradation : (1) from the uniform distribution  $\mathcal{U}[0.02; 0.03]$  for job processing times below 20; (2) from the uniform distribution  $\mathcal{U}[0.03; 0.06]$  for job processing times between 20 and 50; and from the uniform distribution  $\mathcal{U}[0.06; 0.1]$  for job processing times above 50.
- Each instance was tested twice, the first time with medium maintenance durations generated from the uniform distribution  $\mathcal{U}[50, 100]$ , designated as the first maintenance mode *M1*, and the second time with long maintenance durations generated from the uniform distribution  $\mathcal{U}[100, 150]$ , designated as the second maintenance mode *M2*.

It is challenging to make a fair comparison between our results and those of other authors. This difficulty arises from the fact that only a small number of authors have addressed the same problem with the same constraints and objectives, and even fewer with the same instances for comparison purposes. Therefore, to demonstrate the effectiveness of our newly proposed QL-driven ABC algorithms, we report the Average Relative Percentage Deviations (ARPD) provided by each algorithm (after maintenance operations) with respect to the best-known solution for the Taillard instance without maintenance operations (see Equation (9)). Taillard's acceleration technique [80] was applied to compute the makespan. It allows the CPU time to be reduced by calculating all the partial schedules in a given iteration in a single step.

$$ARPD = \frac{1}{R} \times \sum_{i=1}^R \frac{C_{max}^{Sol_i} - C_{max}^{best}}{C_{max}^{best}} \times 100. \quad (9)$$

where  $C_{max}^{best}$  is Taillard's best-known solution and  $C_{max}^{Sol_i}$  is the returned value, and *R* is the number of similar scaled instances running.

### 5.2. IQABC Features Analysis

For the first set of experiments, we undertook a sensitive performance analysis of our newly proposed algorithms IQABC by varying different parameters. These parameters were experimentally found to be good and robust for the problems tested. We chose a full factorial design in which all possible combinations of the following ABC factors were tested:

- Population size *PopSize*: three levels (50, 70, and 100);
- Onlooker bees percentage *onlook%*: three levels (20%, 30%, and 40%);
- Abandonment number of trials *limit*: three levels (5, 10, and 50);
- Number of iterations *max\_iterations*: three levels (100, 150, and 200).

All the cited factors resulted in a total of  $3 \times 3 \times 3 \times 3 = 81$  different combinations run on a set of  $n \times m$  Taillard instances with  $n \in \{20; 70; 120; 170; 220\}$  and  $m \in \{5; 10; 15; 20\}$ . The job processing times of the instances were generated uniformly in the interval  $[1, 99]$ , and maintenance durations were generated according to the first maintenance mode (*M1*) described in Section 5.1. Each instance was executed five times, which meant a total of

16,200 executions. The ARPD was calculated as a response variable. The resulting experiment was analyzed by means of a multifactor analysis of variance (ANOVA) technique [81] with the least significant difference (LSD) intervals at the 95% confidence level.

Each parameter was set to its most adequate level as follows:  $PopSize = 70$ ;  $onlook\% = 40\%$ ;  $Limit = 5$ ; and  $max\_iterations = 200$ .

For the QL factors, namely the Q-learning index  $\alpha_{ql}$ , the discount factor  $\gamma$ , and the probability for the QL mechanism  $\epsilon$ , based on the study of [82], four typical combinations for the QL algorithm were tested:

- C1:  $\alpha = 0.1, \gamma = 0.8, \epsilon = 0.2$ ;
- C2:  $\alpha = 0.1, \gamma = 0.8, \epsilon = 0.1$ ;
- C3:  $\alpha = 0.1, \gamma = 0.9, \epsilon = 0.1$ ;
- C4:  $\alpha = 0.1, \gamma = 0.9, \epsilon = 0.2$ .

After executing different experiments for each combination, we decided to keep the second combination, as it was able to report better results. Finally, the stagnation factor was set at 80% since a large number of convergence curves for multiple executions showed a stabilization of the results up to this value.

### 5.3. Performance Analysis of IQABC Algorithm without Learning and Deteriorating Effects

To prove the performance of the integrated QL-driven ABC algorithm without learning and deterioration effects (IQABC), we compared it with three different competing algorithms for the same problem and constraints without considering the learning and deteriorating effects. The comparative algorithms are : (1) a population-based metaheuristic, which is the ABC algorithm of [54] with random neighborhood selection; (2) a unique solution-based metaheuristic, which is the Variable Neighbor Search (VNS) algorithm of [83]; (3) and finally, INEH, our newly designed constructive heuristic. The comparison was carried out by means of 1100 runs of the test benchmark. The resulting ARPD values reported in Table 1 were calculated based on the  $C_{max}$  value given in Formula (3), with no consideration of the learning and deteriorating effects, and with respect to the first and second maintenance modes (M1 and M2).

**Table 1.** Performance offset from Taillard’s upper bounds of IQABC , ABC, VNS, and INEH algorithms for maintenance mode 1 and 2.

Instance	M1				M2			
	ABC	VNS	INEH	IQABC	ABC	VNS	INEH	IQABC
20x5	4.69	11.52	6.77	3.11	3.43	10.61	8.87	3.34
20x10	13.34	23.01	16.22	11.45	13.97	24.20	19.09	12.33
20x20	25.35	28.19	27.31	23.16	27.03	32.98	29.46	24.84
50x5	2.02	1.72	2.66	0.54	2.83	1.93	2.83	−0.001
50x10	9.93	8.24	12.11	5.91	12.62	9.95	15.79	6.91
50x20	19.79	18.5	21.94	15.12	24.65	19.28	25.01	16.93
100x5	1.16	1.35	1.47	0.93	1.40	1.46	1.93	0.96
100x10	4.34	5.09	5.56	2.90	6.37	5.48	10.27	3.82
100x20	13.41	15	17.36	11.40	16.31	15.61	21.53	13.22
200x10	1.77	3.13	3.10	1.94	2.58	3.43	6.45	2.68
200x20	7.31	11.21	12.6	6.58	9.53	12.55	16.92	8.94
500x20	2.99	-	-	3.08	2.98	-	-	3.64
average	8.84	11.54	11.55	7.18	10.30	12.50	14.38	8.13

Table 1 shows that the INEH heuristic produces competitive ARPD results compared to the VNS metaheuristic, even though the INEH is a constructive method. In fact, for small instances (20x5, 20x10, 20x20), INEH achieves the best ARPD values: for Mode 1, an average of 16.76 against 20.90 for VNS; for Mode 2, an average of 19.14 for INEH against 22.59 for VNS. Nonetheless, it fails to reach the results obtained by the VNS algorithm for large instances (50x5–200x20): for Mode 1, an average of 9.60 against 8.03 for the VNS; and

for Mode 2, an average of 12.59 for INEH against 8.71 for VNS. The great advantage of the INEH heuristic is its ability to obtain near-optimal solutions in a reasonable computation time (on average 6.99 time units (TU) compared to 3120.89 TU for VNS).

Although the ABC metaheuristic of [54] gave better solutions than the VNS algorithm, which is considered an efficient metaheuristic for the problem [83,84], the IQABC algorithm outperforms it, as can clearly be seen from Table 1 for the two maintenance modes. This is proved by the employed bees’ local search procedures and the QL-driven neighborhood selection. To further verify the statistical significance of the difference in ARPD values between ABC and IQABC for the two maintenance modes, we performed a Friedman’s two-way analysis of variance test [85]. The result summary in Table 2 as well as the mean plots displayed in Figure 2 show the significant differences between IQABC and ABC. Therefore, the IQABC algorithm is significantly superior to the ABC, VNS, and INEH algorithms. However, the ABC and IQABC algorithms are population-based methods, which means an important amount of time is used to effectively explore the search space. Furthermore, the QL mechanism in the IQABC algorithm needs more time to learn optimal policies. Therefore, it is not meaningful to compare the running times of the IQABC and ABC algorithms with those of the VNS and INEH heuristics.

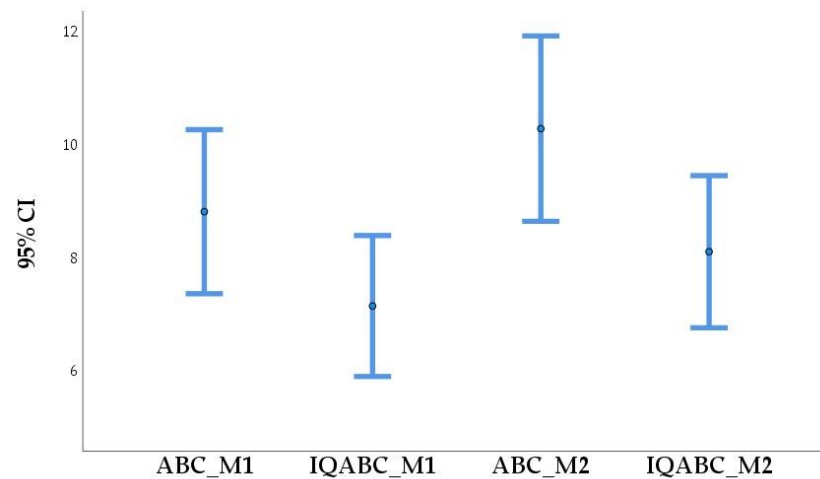


Figure 2. Mean plots of ABC and IQABC for the two maintenance modes.

In Figures 3–6, we also compare the convergence curves of the IQABC and ABC algorithms. We note that the ABC algorithm is more likely to become trapped in premature convergence, while the IQABC algorithm still finds better optimal solutions at a relatively more efficient rate, and the better results can be obtained at the final stage. We can conclude that the QL algorithm in the employed bees phase helps the IQABC algorithm to converge in a robust and efficient way. This conclusion is supported by the convergence curves of 10 trials on the instance 50x20\_M1 as well as the average convergence curve in bold in Figure 7.

Table 2. IQABC-ABC Friedman’s two-way analysis of variance summary for M1 (left) and M2 (right).

	M1	M2	
<b>Total N</b>	120	<b>Total N</b>	120
<b>Test statistic</b>	36,300	<b>Test statistic</b>	28,033
<b>Degree of freedom</b>	1	<b>Degree of freedom</b>	1
<b>Asymptotic sig</b>	<0.001	<b>Asymptotic sig</b>	<0.001
<b>Decision</b>	Reject the null hypothesis	<b>Decision</b>	Reject the null hypothesis

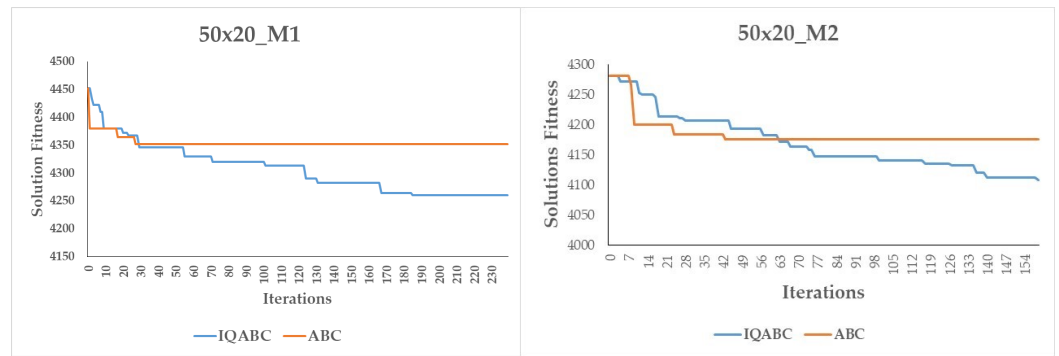


Figure 3. Convergence curves of ABC and IQABC for 50x20\_M1 and 50x20\_M2 instances.

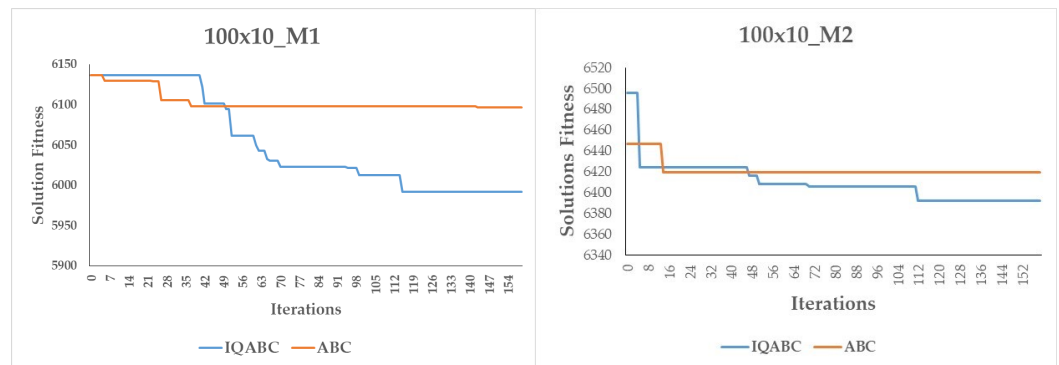


Figure 4. Convergence curves of ABC and IQABC for 100x10\_M1 and 100x10\_M2 instances.

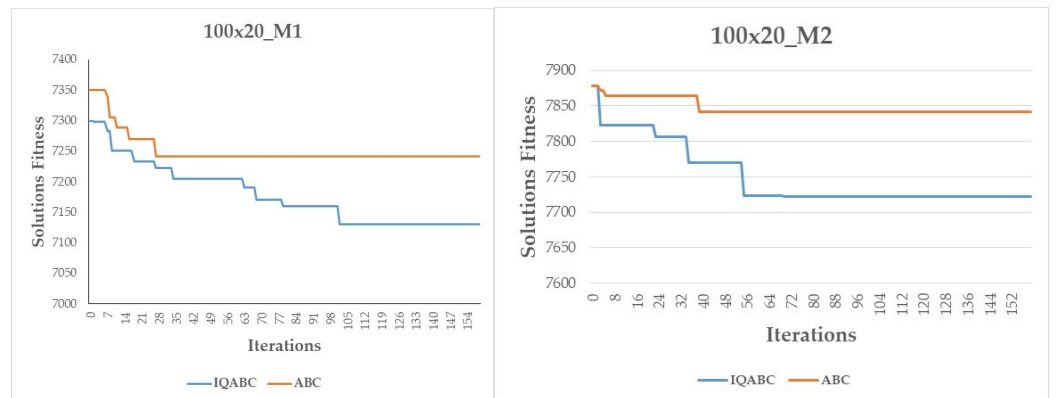


Figure 5. Convergence curves of ABC and IQABC for 100x20\_M1 and 100x20\_M2 instances.

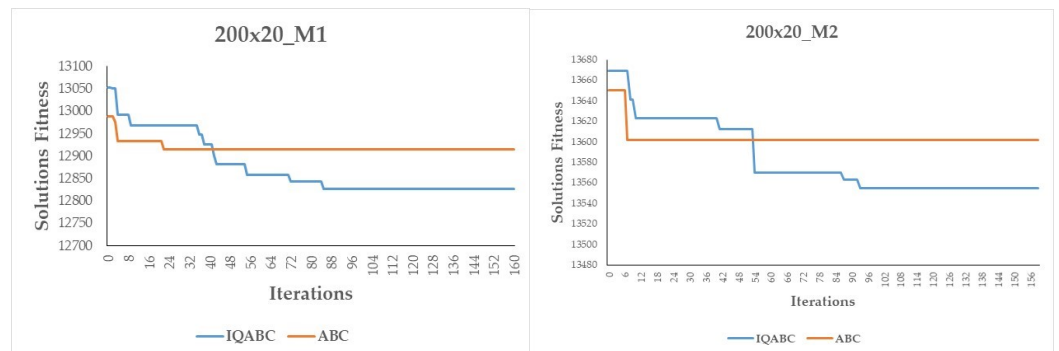
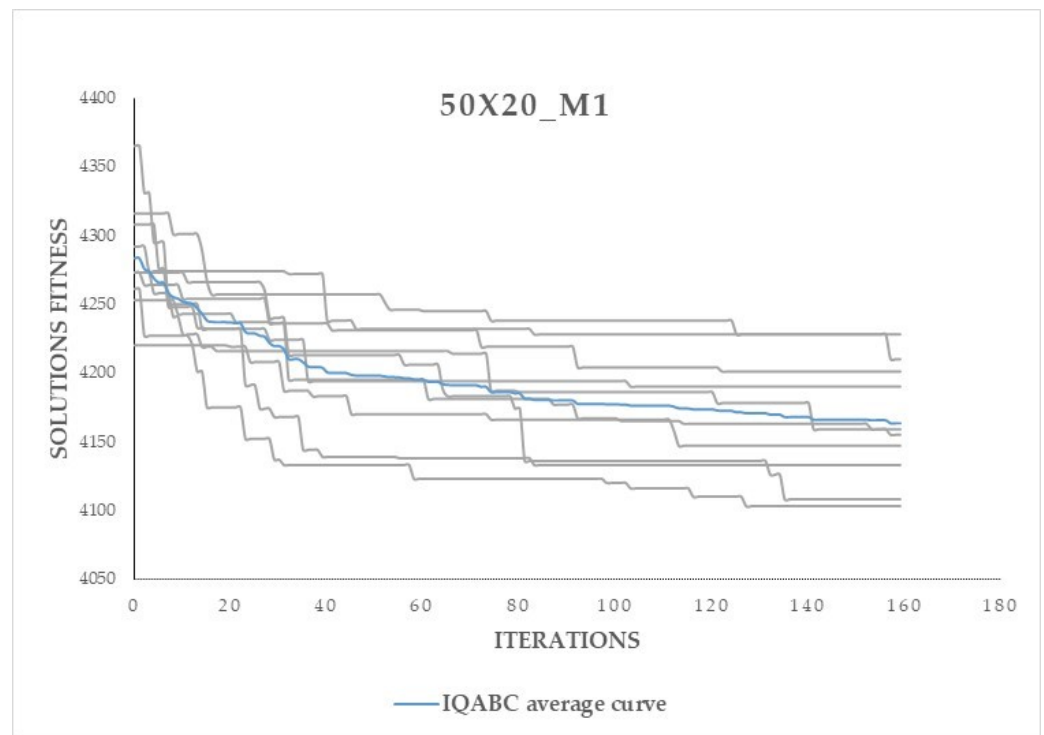


Figure 6. Convergence curves of ABC and IQABC for 200x20\_M1 and 200x20\_M2 instances.





**Figure 7.** Average convergence of the IQABC for the 50x20\_M1 instance.

It is important to note that high ARPD values do not reflect poor solution quality since they are calculated relative to the best-known Taillard solutions, without taking into account maintenance tasks and learning/deterioration effects. Therefore, the presented ARPD deviations (for all experiments) are much higher than they would be if they were calculated with respect to a more accurate lower bound that takes into account maintenance tasks and learning/deterioration effects.

#### 5.4. Performance Analysis of the QL-Driven ABC Algorithm with Learning Effect

The performance of the first algorithm, namely the IQABC-LE, is proven through three scenarios and 1100 executions per scenario on the designed benchmarks completed with learning indexes ( $\alpha$ ). First, we assume a small common learning rate among maintenance operators, so we generate small learning indexes (SF) from the uniform distribution  $\mathcal{U}[0; 0.20]$ . Then, assuming that the operators learn very quickly at the same rate, large learning indices (LF) are generated from the uniform distribution  $\mathcal{U}[0.80; 1]$ . Finally, we assume that the learning phenomenon is random and depends on the machine characteristics, so the learning indexes are generated from the uniform distribution  $\mathcal{U}[0; 1]$  per machine (FPM).

The execution results are compared with the ABC learning algorithm (ABC-LE) reported in [54]. Indeed, the IQABC-LE algorithm implements a QL mechanism to guide the neighbor structure selection in the employed bees phase, instead of random selection as in ABC-LE. In addition, the IQABC-LE uses the INEH heuristic to generate one complete solution, while the ABC-LE algorithm applies the NEH heuristic [43] to generate first, the production sequence, then, the maintenance heuristic [44] to complete the solution with maintenance emplacements.

The comparison between the two algorithms is made according to the ARPD values, which are calculated on the basis of the  $C_{max}$  values given in Formula (4).

The results presented in Tables 3–5 clearly show the superiority of the IQABC-LE algorithm over the ABC-LE algorithm. An important note comparing the results of the IQABC algorithm when considering a learning effect and no effect, is the significant

difference in ARPD values, which increasingly reinforce the need to take into account real constraints when studying scheduling problems.

From Figure 8, it can be seen that the IQABC-LE algorithm with large learning indexes (LF) optimizes the  $C_{max}$  very well compared to the small and random learning indexes. Consequently, it is of great importance for a workshop to improve the ability of its employees to learn.

**Table 3.** Performance offset from Taillard’s upper bounds of IQABC-LE and ABC-LE considering small learning indexes.

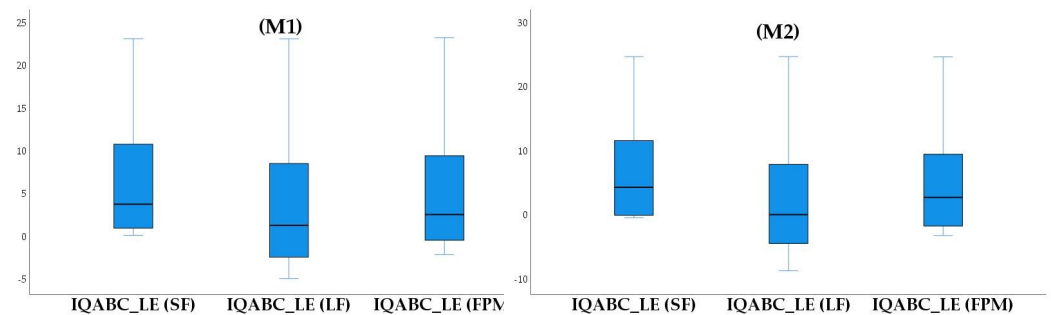
Instance	M1		M2	
	ABC-LE(SF)	IQABC-LE(SF)	ABC-LE(SF)	IQABC-LE(SF)
20x5	4.47	2.79	3.35	3.33
20x10	12.81	11.36	14.44	12.27
20x20	24.76	23.24	27.20	24.88
50x5	1.87	0.27	2.24	−0.25
50x10	8.98	5.43	12.26	6.27
50x20	19.43	14.65	23.37	16.26
100x5	0.39	0.25	0.38	−0.14
100x10	3.7	2.37	4.93	2.50
100x20	12.39	10.46	14.46	11.28
200x10	0.43	0.76	0.74	−0.13
200x20	6.31	5.01	7.16	5.66
500x20	1.26	1.43	0.23	0.42
average	8.07	6.50	9.23	6.86

**Table 4.** Performance offset from Taillard’s upper bounds of IQABC-LE and ABC-LE considering large learning indexes.

Instance	M1		M2	
	ABC-LE(LF)	IQABC-LE(LF)	ABC-LE(LF)	IQABC-LE(LF)
20x5	4.73	3.06	3.29	3.51
20x10	13.34	11.47	14.22	12.52
20x20	25.1	23.23	26.95	24.88
50x5	0.21	−1.06	−0.43	−2.53
50x10	6.85	3.70	8.15	2.97
50x20	16	12.35	18.69	12.39
100x5	−2.82	−3.15	−4.3	−4.94
100x10	−0.4	−1.45	−1.62	−3.60
100x20	7.71	5.83	6.52	3.75
200x10	−3.87	−4.13	−6.36	−7.05
200x20	0.7	−0.22	−1.36	−2.52
500x20	−4.39	−4.79	−8.17	−8.52
average	5.26	3.73	4.63	2.57

**Table 5.** Performance offset from Taillard’s upper bounds of IQABC-LE and ABC-LE considering random learning indexes per machine.

Instance	M1		M2	
	ABC-LE(FPM)	IQABC-LE(FPM)	ABC-LE(FPM)	IQABC-LE(FPM)
20x5	4.4	2.95	3.38	3.44
20x10	13.69	11.42	14.20	12.23
20x20	25.11	23.38	27.28	24.84
50x5	0.76	−0.47	1.24	−1.48
50x10	7.63	4.51	9.86	4.39
50x20	17.66	13.44	20.96	14.34
100x5	−1.47	−1.24	−1.55	−3.02
100x10	1.84	0.47	1.52	−0.38
100x20	9.9	7.70	9.30	7.05
200x10	−1	−2.01	−2.10	−2.74
200x20	3.45	2.40	3.19	2.41
500x20	−0.52	−0.16	−2.49	−1.61
average	6.79	5.19	7.06	4.95



**Figure 8.** Box plot of the IQABC\_LE algorithm with SF, LF, and FPM for both maintenance modes.

5.5. Performance Analysis of the QL-Driven ABC Algorithm with Learning and Deteriorating Effects

In this section, the IQABC-LDE algorithm with a learning and deteriorating effect is evaluated on the designed benchmarks completed with random learning and deteriorating indexes generated from a uniform distribution  $\mathcal{U}[0, 1]$ . The results of the 1100 executions are summarized in Table 6 and compared to those of the ABC-LDE reported in [54]. The ABC-LDE follows the same steps as the ABC-LE, with the addition of the deteriorating effect. The ARPD values reported in 6 are calculated on the basis of the  $C_{max}$  values given in Formula (5).

Once again, the IQABC-LDE is superior to the ABC-LDE, proving the effectiveness and robustness of the QL-based neighbor selection mechanism when combined with the ABC metaheuristic. As with the learning effect, we observe a significant difference in the results obtained with the IQABC algorithm if the learning and degradation effects are not taken into account, thus emphasizing the need to integrate these effects into the study. In fact, the deteriorating effect, as opposed to the learning effect, extends processing times. It is therefore important for manufacturers to implement an efficient maintenance strategy to prevent machines from deteriorating and additional delays.

**Table 6.** Performance offset from Taillard’s upper bounds of IQABC-LDE and ABC-LDE considering random learning and deteriorating indexes per machine.

instance	M1		M2	
	ABC-LDE	IQABC-LDE	ABC-LDE	IQABC-LDE
20x5	13.04	9.64	12.45	9.24
20x10	23.66	19.31	23.02	19.05
20x20	37.3	30.82	36.17	32.10
50x5	11.30	7.96	12.09	8.59
50x10	18.49	14.09	19.32	13.85
50x20	28.54	23.09	30.06	23.17
100x5	10.84	7.85	11.45	5.62
100x10	13.29	10.41	13.32	8.87
100x20	21.41	17.43	20.32	15.55
200x10	12.86	9.56	8.91	7.18
200x20	17.69	12.92	15.01	12.08
500x20	13.04	9.84	12.45	8.71
average	18.45	14.40	17.88	13.66

## 6. Conclusions

We addressed the makespan PFSP with flexible maintenance under learning and deteriorating effects. The maintenance operations are inserted to prevent machine failures with an intelligent PHM-based policy. In addition, the learning effect on the maintenance duration is considered and modeled by a decreasing position-dependent function. On the other hand, the deteriorating effect is assumed to extend the job processing times over time. For the resolution, the well-known ABC algorithm, which has been successfully applied to solve the PFSP without learning and deterioration effects, is adapted to the problem with flexible maintenance, learning, and deteriorating effects in order to find a faster and near-optimal or optimal solution to the problem. To improve the local search in the ABC-employed bees phase, a QL algorithm, considered a common potential tool of the RL mechanism, is carefully integrated into the ABC metaheuristic. Furthermore, a newly designed heuristic, the INEH, is applied to enrich the search space with a high-quality complete solution. The adapted ABC algorithm with the new strategies is used to first solve the integrated PFSP with a learning effect (IQABC-LE) and then also with a deteriorating effect (IQABC-LDE). We conducted a comprehensive experimental study to evaluate the IQABC algorithm without considering the learning and deteriorating effects, the IQABC-LE and IQABC-LDE algorithms. The objective of evaluating these three algorithms is, on the one hand, to justify the significance of our study when we have considered the learning and deteriorating effects, and on the other hand, to prove the effectiveness and robustness of our newly proposed algorithms. For the latter purpose, the experimental results are compared with those of two existing and competitive metaheuristics in the literature: ABC and VNS. The computational results and comparisons demonstrate that the new strategies of the IQABC approach really improve its search performance and IQABC is a competitive approach for the considered problem. Moreover, the need to consider learning and deterioration effects for a more realistic and accurate study is also demonstrated and emphasized. For future work, we propose to integrate more realistic constraints such as the forgetting effect and human resource constraints in the study of the PFSP. We also expect to apply other recently introduced bio-inspired metaheuristics, such as BBO and GWO, to solve the studied scheduling problem, and compare the results with the ABC algorithm.

**Author Contributions:** Conceptualization, N.T.; data curation, N.T.; formal analysis, N.T., F.B.-S.T., and A.L.; methodology, F.B.-S.T. and A.L.; project administration, F.B.-S.T.; software, N.T.; supervision, F.B.-S.T. and A.L.; validation, F.B.-S.T. and A.L.; writing—original draft, N.T.; writing—review and editing, F.B.-S.T. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** No new data were created or analyzed in this study. Data sharing is not applicable to this article.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Paredes-Astudillo, Y.A.; Montoya-Torres, J.R.; Botta-Genoulaz, V. Taxonomy of Scheduling Problems with Learning and Deterioration Effects. *Algorithms* **2022**, *15*, 439. [[CrossRef](#)]
2. Pei, J.; Zhou, Y.; Yan, P.; Pardalos, P.M. A concise guide to scheduling with learning and deteriorating effects. *Int. J. Prod. Res.* **2022**, *61*, 1–22. [[CrossRef](#)]
3. Biskup, D. Single-machine scheduling with learning considerations. *Eur. J. Oper. Res.* **1999**, *115*, 173–178. [[CrossRef](#)]
4. Wright, T.P. Factors affecting the cost of airplanes. *J. Aeronaut. Sci.* **1936**, *3*, 122–128. [[CrossRef](#)]
5. Cheng, T.E.; Wang, G. Single machine scheduling with learning effect considerations. *Ann. Oper. Res.* **2000**, *98*, 273–290. [[CrossRef](#)]
6. Azzouz, A.; Ennigrou, M.; Said, L.B. Scheduling problems under learning effects: Classification and cartography. *Int. J. Prod. Res.* **2017**, *56*, 1642–1661. [[CrossRef](#)]
7. Biskup, D. A state-of-the-art review on scheduling with learning effects. *Eur. J. Oper. Res.* **2008**, *188*, 315–329. [[CrossRef](#)]
8. Gupta, J.N.; Gupta, S.K. Single facility scheduling with nonlinear processing times. *Comput. Ind. Eng.* **1988**, *14*, 387–393. [[CrossRef](#)]
9. Xu, H.; Li, X.; Ruiz, R.; Zhu, H. Group Scheduling with Nonperiodical Maintenance and Deteriorating Effects. *IEEE Trans. Syst. Man, Cybern. Syst.* **2021**, *51*, 2860–2872. [[CrossRef](#)]
10. Browne, S.; Yechiali, U. Scheduling deteriorating jobs on a single processor. *Oper. Res.* **1990**, *38*, 495–498. [[CrossRef](#)]
11. Alidaee, B.; Womer, N.K. Scheduling with time dependent processing times: Review and extensions. *J. Oper. Res. Soc.* **1999**, *50*, 711–720. [[CrossRef](#)]
12. Cheng, T.E.; Ding, Q.; Lin, B.M. A concise survey of scheduling with time-dependent processing times. *Eur. J. Oper. Res.* **2004**, *152*, 1–13. [[CrossRef](#)]
13. Gawiejnowicz, S. A review of four decades of time-dependent scheduling: Main results, new topics, and open problems. *J. Sched.* **2020**, *23*, 3–47. [[CrossRef](#)]
14. Blazewicz, J.; Ecker, K.H.; Pesch, E.; Schmidt, G.; Sterna, M.; Weglarz, J. Time-Dependent Scheduling. In *Handbook on Scheduling*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 431–474.
15. Pandey, D.; Kulkarni, M.S.; Vrat, P. Joint consideration of production scheduling, maintenance and quality policies: A review and conceptual framework. *Int. J. Adv. Oper. Manag.* **2010**, *2*, 1–24. [[CrossRef](#)]
16. Safari, E.; Sadjadi, S.J. A hybrid method for flowshops scheduling with condition-based maintenance constraint and machines breakdown. *Expert Syst. Appl.* **2011**, *38*, 2020–2029. [[CrossRef](#)]
17. Syan, C.S.; Ramsoobag, G. Maintenance applications of multi-criteria optimization: A review. *Reliab. Eng. Syst. Saf.* **2019**, *190*, 106520. [[CrossRef](#)]
18. Bougacha, O.; Varnier, C.; Zerhouni, N. A review of post-prognostics decision-making in prognostics and health management. *Int. J. Progn. Health Manag.* **2020**, *11*, 31. [[CrossRef](#)]
19. Iyer, N.; Goebel, K.; Bonissone, P. Framework for post-prognostic decision support. In Proceedings of the 2006 IEEE Aerospace Conference, Piscataway, NJ, USA, 4–11 March 2006; p. 10.
20. Lei, X.; Sandborn, P.A. Maintenance scheduling based on remaining useful life predictions for wind farms managed using power purchase agreements. *Renew. Energy* **2018**, *116*, 188–198. [[CrossRef](#)]
21. Skima, H.; Varnier, C.; Dedu, E.; Medjaher, K.; Bourgeois, J. Post-prognostics decision making in distributed MEMS-based systems. *J. Intell. Manuf.* **2019**, *30*, 1125–1136. [[CrossRef](#)]
22. Gerum, P.C.L.; Altay, A.; Baykal-Gürsoy, M. Data-driven predictive maintenance scheduling policies for railways. *Transp. Res. Part C Emerg. Technol.* **2019**, *107*, 137–154. [[CrossRef](#)]
23. Sprong, J.P.; Jiang, X.; Polinder, H. Deployment of Prognostics to Optimize Aircraft Maintenance—A Literature Review. *J. Int. Bus. Res. Mark.* **2020**, *5*, 26–37. [[CrossRef](#)]
24. Ladj, A.; Tayeb, F.B.S.; Varnier, C. Hybrid of metaheuristic approaches and fuzzy logic for the integrated flowshop scheduling with predictive maintenance problem under uncertainties. *Eur. J. Ind. Eng.* **2021**, *15*, 675–710. [[CrossRef](#)]
25. Zhai, S.; Kandemir, M.G.; Reinhart, G. Predictive maintenance integrated production scheduling by applying deep generative prognostics models: Approach, formulation and solution. *Prod. Eng.* **2022**, *16*, 65–88. [[CrossRef](#)]
26. Ladj, A.; Varnier, C.; Tayeb, F.B.S.; Zerhouni, N. Exact and heuristic algorithms for post prognostic decision in a single multifunctional machine. *Int. J. Progn. Health Manag.* **2017**, *8*, 2620. [[CrossRef](#)]
27. Safari, E.; Jafar Sadjadi, S.; Shahanaghi, K. Scheduling flowshops with condition-based maintenance constraint to minimize expected makespan. *Int. J. Adv. Manuf. Technol.* **2010**, *46*, 757–767. [[CrossRef](#)]
28. Xhafa, F.; Abraham, A. *Metaheuristics for Scheduling in Industrial and Manufacturing Applications*; Springer: Berlin/Heidelberg, Germany, 2008; Volume 128.

29. Ma, H.; Shen, S.; Yu, M.; Yang, Z.; Fei, M.; Zhou, H. Multi-population techniques in nature inspired optimization algorithms: A comprehensive survey. *Swarm Evol. Comput.* **2019**, *44*, 365–387. [[CrossRef](#)]
30. Fu, Y.; Wang, H.; Huang, M.; Ding, J.; Tian, G. Multiobjective flow shop deteriorating scheduling problem via an adaptive multipopulation genetic algorithm. *Proc. Inst. Mech. Eng. Part B J. Eng. Manuf.* **2018**, *232*, 2641–2650. [[CrossRef](#)]
31. Chen, R.; Yang, B.; Li, S.; Wang, S. A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem. *Comput. Ind. Eng.* **2020**, *149*, 106778. [[CrossRef](#)]
32. Onwubolu, G.; Davendra, D. Scheduling flow shops using differential evolution algorithm. *Eur. J. Oper. Res.* **2006**, *171*, 674–692. [[CrossRef](#)]
33. Zhang, Y.; Gu, X. A biogeography-based optimization algorithm with modified migration operator for large-scale distributed scheduling with transportation time. *Expert Syst. Appl.* **2023**, *231*, 120732. [[CrossRef](#)]
34. Monga, P.; Sharma, M.; Sharma, S.K. A comprehensive meta-analysis of emerging swarm intelligent computing techniques and their research trend. *J. King Saud Univ.-Comput. Inf. Sci.* **2022**, *34*, 9622–9643. [[CrossRef](#)]
35. Lin, B.; Lu, C.; Shyu, S.; Tsai, C. Development of new features of ant colony optimization for flowshop scheduling. *Int. J. Prod. Econ.* **2008**, *112*, 742–755. [[CrossRef](#)]
36. Marichelvam, M.; Geetha, M.; Tosun, Ö. An improved particle swarm optimization algorithm to solve hybrid flowshop scheduling problems with the effect of human factors—A case study. *Comput. Oper. Res.* **2020**, *114*, 104812. [[CrossRef](#)]
37. Luo, S.; Zhang, L.; Fan, Y. Energy-efficient scheduling for multi-objective flexible job shops with variable processing speeds by grey wolf optimization. *J. Clean. Prod.* **2019**, *234*, 1365–1384. [[CrossRef](#)]
38. Karaboga, D. *An Idea Based on Honey Bee Swarm for Numerical Optimization*; Technical Report-tr06; Computer Engineering Department, Engineering Faculty, Erciyes University: Kayseri, Turkey, 2005.
39. Arık, O.A. Artificial bee colony algorithm including some components of iterated greedy algorithm for permutation flow shop scheduling problems. *Neural Comput. Appl.* **2021**, *33*, 3469–3486. [[CrossRef](#)]
40. Li, Y.; Li, X.; Gao, L.; Zhang, B.; Pan, Q.K.; Tasgetiren, M.F.; Meng, L. A discrete artificial bee colony algorithm for distributed hybrid flowshop scheduling problem with sequence-dependent setup times. *Int. J. Prod. Res.* **2021**, *59*, 3880–3899. [[CrossRef](#)]
41. Xuan, H.; Zhang, H.; Li, B. An improved discrete artificial bee colony algorithm for flexible flowshop scheduling with step deteriorating jobs and sequence-dependent setup times. *Math. Probl. Eng.* **2019**, *2019*, 1–13. [[CrossRef](#)]
42. Wang, L.; Pan, Z.; Wang, J. A review of reinforcement learning based intelligent optimization for manufacturing scheduling. *Complex Syst. Model. Simul.* **2021**, *1*, 257–270. [[CrossRef](#)]
43. Nawaz, M.; Enscore, E.E., Jr.; Ham, I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega* **1983**, *11*, 91–95. [[CrossRef](#)]
44. Ladj, A.; Varnier, C.; Tayeb, F.S. IPro-GA: An integrated prognostic based GA for scheduling jobs and predictive maintenance in a single multifunctional machine. *IFAC-PapersOnLine* **2016**, *49*, 1821–1826. [[CrossRef](#)]
45. Kan, A.R. *Machine Scheduling Problems: Classification, Complexity and Computations*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2012.
46. Lawler, E.L.; Lenstra, J.K.; Rinnooy Kan, A. Recent developments in deterministic sequencing and scheduling: A survey. In *Proceedings of the Deterministic and Stochastic Scheduling: Proceedings of the NATO Advanced Study and Research Institute on Theoretical Approaches to Scheduling Problems, Durham, UK, 6–17 July 1981*; Springer: Berlin/Heidelberg, Germany, 1982; pp. 35–73.
47. Karimi-Mamaghan, M.; Mohammadi, M.; Meyer, P.; Karimi-Mamaghan, A.M.; Talbi, E.G. Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *Eur. J. Oper. Res.* **2022**, *296*, 393–422. [[CrossRef](#)]
48. Zheng, C.; Chen, H.; Xu, R. Tabu search algorithms for minimizing total completion time on a single machine with an actual time-dependent learning effect. *Nat. Comput.* **2019**, *18*, 287–299. [[CrossRef](#)]
49. Wu, C.C.; Bai, D.; Azzouz, A.; Chung, I.H.; Cheng, S.R.; Jhwueng, D.C.; Lin, W.C.; Said, L.B. A branch-and-bound algorithm and four metaheuristics for minimizing total completion time for a two-stage assembly flow-shop scheduling problem with learning consideration. *Eng. Optim.* **2020**, *52*, 1009–1036. [[CrossRef](#)]
50. Wu, C.C.; Azzouz, A.; Chung, I.H.; Lin, W.C.; Ben Said, L. A two-stage three-machine assembly scheduling problem with deterioration effect. *Int. J. Prod. Res.* **2019**, *57*, 6634–6647. [[CrossRef](#)]
51. Arık, O.A.; Toksarı, M.D. A genetic algorithm approach to parallel machine scheduling problems under effects of position-dependent learning and linear deterioration: Genetic algorithm to parallel machine scheduling problems. *Int. J. Appl. Metaheuristic Comput. (IJAMC)* **2021**, *12*, 195–211. [[CrossRef](#)]
52. Vahedi Nouri, B.; Fattahi, P.; Ramezani, R. Hybrid firefly-simulated annealing algorithm for the flow shop problem with learning effects and flexible maintenance activities. *Int. J. Prod. Res.* **2013**, *51*, 3501–3515. [[CrossRef](#)]
53. Ghaleb, M.; Taghipour, S.; Sharifi, M.; Zolfagharinia, H. Integrated production and maintenance scheduling for a single degrading machine with deterioration-based failures. *Comput. Ind. Eng.* **2020**, *143*, 106432. [[CrossRef](#)]
54. Touafek, N.; Benbouzid-Si Tayeb, F.; Ladj, A.; Dahamni, A.; Baghdadi, R. An Integrated Artificial Bee Colony Algorithm for Scheduling Jobs and Flexible Maintenance with Learning and Deteriorating Effects. In *Proceedings of the Conference on Computational Collective Intelligence Technologies and Applications, Hammamet, Tunisia, 28–30 September 2022*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 647–659.

55. Kong, M.; Liu, X.; Pei, J.; Cheng, H.; Pardalos, P.M. A BRKGA-DE algorithm for parallel-batching scheduling with deterioration and learning effects on parallel machines under preventive maintenance consideration. *Ann. Math. Artif. Intell.* **2020**, *88*, 237–267. [[CrossRef](#)]
56. Li, X.J.; Wang, J.J. Parallel machines scheduling based on the impact of deteriorating maintenance. *J. Interdiscip. Math.* **2018**, *21*, 729–741. [[CrossRef](#)]
57. Lu, S.; Liu, X.; Pei, J.; Thai, M.T.; Pardalos, P.M. A hybrid ABC-TS algorithm for the unrelated parallel-batching machines scheduling problem with deteriorating jobs and maintenance activity. *Appl. Soft Comput.* **2018**, *66*, 168–182. [[CrossRef](#)]
58. Zhang, X.; Wu, W.H.; Lin, W.C.; Wu, C.C. Machine scheduling problems under deteriorating effects and deteriorating rate-modifying activities. *J. Oper. Res. Soc.* **2017**, *69*, 439–448. [[CrossRef](#)]
59. Yang, S.J.; Yang, D.L. Minimizing the makespan on single-machine scheduling with aging effect and variable maintenance activities. *Omega* **2010**, *38*, 528–533. [[CrossRef](#)]
60. Sutton, R.S.; Barto, A.G. Reinforcement learning: An introduction. *Robotica* **1999**, *17*, 229–235. [[CrossRef](#)]
61. Usuga Cadavid, J.P.; Lamouri, S.; Grabot, B.; Pellerin, R.; Fortin, A. Machine learning applied in production planning and control: A state-of-the-art in the era of industry 4.0. *J. Intell. Manuf.* **2020**, *31*, 1531–1558. [[CrossRef](#)]
62. Fonseca-Reyna, Y.C.; Martínez-Jiménez, Y.; Nowé, A. Q-learning algorithm performance for m-machine, n-jobs flow shop scheduling problems to minimize makespan. *Investig. Oper.* **2018**, *38*, 281–290.
63. Han, W.; Guo, F.; Su, X. A reinforcement learning method for a hybrid flow-shop scheduling problem. *Algorithms* **2019**, *12*, 222. [[CrossRef](#)]
64. Martínez Jiménez, Y.; Coto Palacio, J.; Nowé, A. Multi-agent reinforcement learning tool for job shop scheduling problems. In *Proceedings of the International Conference on Optimization and Learning*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 3–12.
65. Shahrabi, J.; Adibi, M.A.; Mahootchi, M. A reinforcement learning approach to parameter estimation in dynamic job shop scheduling. *Comput. Ind. Eng.* **2017**, *110*, 75–82. [[CrossRef](#)]
66. Zhao, F.; Zhang, L.; Cao, J.; Tang, J. A cooperative water wave optimization algorithm with reinforcement learning for the distributed assembly no-idle flowshop scheduling problem. *Comput. Ind. Eng.* **2021**, *153*, 107082. [[CrossRef](#)]
67. Li, Z.; Wei, X.; Jiang, X.; Pang, Y. A kind of reinforcement learning to improve genetic algorithm for multiagent task scheduling. *Math. Probl. Eng.* **2021**, *2021*, 1–12. [[CrossRef](#)]
68. Pan, Z.; Wang, L.; Wang, J.; Lu, J. Deep Reinforcement Learning Based Optimization Algorithm for Permutation Flow-Shop Scheduling. In *IEEE Transactions on Emerging Topics in Computational Intelligence*; Springer: Berlin/Heidelberg, Germany, 2021.
69. Karaboga, D.; Gorkemli, B.; Ozturk, C.; Karaboga, N. A comprehensive survey: Artificial bee colony (ABC) algorithm and applications. *Artif. Intell. Rev.* **2014**, *42*, 21–57. [[CrossRef](#)]
70. Karaboga, D.; Basturk, B. On the performance of artificial bee colony (ABC) algorithm. *Appl. Soft Comput.* **2008**, *8*, 687–697. [[CrossRef](#)]
71. Zhao, F.; Wang, Z.; Wang, L. A Reinforcement Learning Driven Artificial Bee Colony Algorithm for Distributed Heterogeneous No-Wait Flowshop Scheduling Problem with Sequence-Dependent Setup Times. *IEEE Trans. Autom. Sci. Eng.* **2022**. [[CrossRef](#)]
72. Lei, D.; Yang, H. Scheduling unrelated parallel machines with preventive maintenance and setup time: Multi-sub-colony artificial bee colony. *Appl. Soft Comput.* **2022**, *125*, 109154. [[CrossRef](#)]
73. Benbouzid-Sitayeb, F.; Guebli, S.A.; Bessadi, Y.; Varnier, C.; Zerhouni, N. Joint scheduling of jobs and preventive maintenance operations in the flowshop sequencing problem: A resolution with sequential and integrated strategies. *Int. J. Manuf. Res.* **2011**, *6*, 30–48. [[CrossRef](#)]
74. Ruiz, R.; Maroto, C.; Alcaraz, J. Two new robust genetic algorithms for the flowshop scheduling problem. *Omega* **2006**, *34*, 461–476. [[CrossRef](#)]
75. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
76. Han, G.; Gong, A.; Wang, H.; Martínez-García, M.; Peng, Y. Multi-AUV collaborative data collection algorithm based on Q-learning in underwater acoustic sensor networks. *IEEE Trans. Veh. Technol.* **2021**, *70*, 9294–9305. [[CrossRef](#)]
77. Ruiz, R.; Stützle, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *Eur. J. Oper. Res.* **2007**, *177*, 2033–2049. [[CrossRef](#)]
78. Tasgetiren, M.F.; Pan, Q.K.; Suganthan, P.; Oner, A. A discrete artificial bee colony algorithm for the no-idle permutation flowshop scheduling problem with the total tardiness criterion. *Appl. Math. Model.* **2013**, *37*, 6758–6779. [[CrossRef](#)]
79. Taillard, E. Benchmarks for basic scheduling problems. *Eur. J. Oper. Res.* **1993**, *64*, 278–285. [[CrossRef](#)]
80. Taillard, E. Some efficient heuristic methods for the flow shop sequencing problem. *Eur. J. Oper. Res.* **1990**, *47*, 65–74. [[CrossRef](#)]
81. Borrór, C.; Montgomery, D. Mixed resolution designs as alternatives to Taguchi inner/outer array designs for robust design problems. *Qual. Reliab. Eng. Int.* **2000**, *16*, 117–127. [[CrossRef](#)]
82. Jiménez, Y.M. A Generic Multi-Agent Reinforcement Learning Approach for Scheduling Problems. Ph.D. Thesis, Vrije Universiteit Brussel, Brussels, Belgium, 2012; Volume 128.
83. Ladj, A.; Tayeb, F.B.S.; Varnier, C.; Dridi, A.A.; Selmane, N. A Hybrid of Variable Neighbor Search and Fuzzy Logic for the permutation flowshop scheduling problem with predictive maintenance. *Procedia Comput. Sci.* **2017**, *112*, 663–672. [[CrossRef](#)]

84. Jomaa, W.; Eddaly, M.; Jarboui, B. Variable neighborhood search algorithms for the permutation flowshop scheduling problem with the preventive maintenance. *Oper. Res.* **2021**, *21*, 2525–2542. [[CrossRef](#)]
85. Zimmerman, D.W.; Zumbo, B.D. Relative power of the Wilcoxon test, the Friedman test, and repeated-measures ANOVA on ranks. *J. Exp. Educ.* **1993**, *62*, 75–86. [[CrossRef](#)]

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.