technologies

MDPI

Article

An Artificial Bee Colony Algorithm for Coordinated Scheduling of Production Jobs and Flexible Maintenance in Permutation Flowshops

Asma Ladj [1], Fatima Benbouzid-Si Tayeb [2], Alaeddine Dahamni [2] and Mohamed Benbouzid [3,4,*]

[1] Railenium Research and Technology Institute, 59540 Valenciennes, France; asma.ladj@railenium.eu
[2] Laboratoire des Méthodes de Conception de Systèmes (LMCS), Ecole Nationale Supérieure d'Informatique (ESI), BP 68M, Oued Smar, Algiers 16270, Algeria; f_sitayeb@esi.dz (F.B.-S.T.); fa_dahamni@esi.dz (A.D.)
[3] Institut de Recherche Dupuy de Lôme (UMR CNRS 6027), University of Brest, 29238 Brest, France
[4] Logistics Engineering College, Shanghai Maritime University, Shanghai 201306, China
[*] Correspondence: mohamed.benbouzid@univ-brest.fr

Abstract: This research work addresses the integrated scheduling of jobs and flexible (non-systematic) maintenance interventions in permutation flowshop production systems. We propose a coordinated model in which the time intervals between successive maintenance tasks as well as their number are assumed to be non-fixed for each machine on the shopfloor. With such a flexible nature of maintenance activities, the resulting joint schedule is more practical and representative of real-world scenarios. Our goal is to determine the best job permutation in which flexible maintenance activities are properly incorporated. To tackle the NP-hard nature of this problem, an artificial bee colony (ABC) algorithm is developed to minimize the total production time (Makespan). Experiments are conducted utilizing well-known Taillard's benchmarks, enriched with maintenance data, to compare the proposed algorithm performance against the variable neighbourhood search (VNS) method from the literature. Computational results demonstrate the effectiveness of the proposed algorithm in terms of both solution quality and computational times.

Keywords: permutation flowshop scheduling problem; flexible maintenance; integrated scheduling of production and maintenance; artificial bee colony

check for updates

Citation: Ladj, A.; Benbouzid-Si Tayeb, F.; Dahamni, A.; Benbouzid, M. An Artificial Bee Colony Algorithm for Coordinated Scheduling of Production Jobs and Flexible Maintenance in Permutation Flowshops. *Technologies* **2024**, *12*, 45. https://doi.org/10.3390/technologies12040045

Academic Editor: Francesco Aggogeri

Received: 17 January 2024
Revised: 12 March 2024
Accepted: 18 March 2024
Published: 25 March 2024

1. Introduction

Production scheduling is a crucial problem that has been extensively explored in the literature. This problem refers to the assignment of production jobs (tasks) to be processed on machines (resources). The aim of machine scheduling problems in production (MSPP) is to find a sequence of jobs to be processed on machines in a way that optimizes a set of objectives [1].

For manufacturers, it is crucial to optimize equipment utilization by ensuring efficient schedules at the operational level. Nonetheless, in MSPP theory, the majority of studies assume that resources are continuously available for processing tasks throughout the scheduling horizon. However, in realistic situations, machines may be unavailable during certain periods due to deterministic or stochastic reasons. In the deterministic case, the machine may not be available because of some planned machine stopping, such as inspections, repair, tool change, preventive maintenance (PM), etc., where the starting times and durations may be known or estimated in advance. Whereas in the stochastic case, the unavailability of machines may be due to unpredictable machine stopping or breakdowns, implying the inability to process the tasks until the restart of the machine. This work deals with the first category of scheduling, where the deterministic aspect implies that the factors influencing the schedule are not subject to randomness or uncertainty, providing a clear and predictable environment for scheduling decisions. In this area, preventive maintenance

(PM) is the major reason for machine unavailability. PM tasks aim to avoid failures and production stopping by trading off between planned unproductive downtimes dedicated to performing maintenance interventions, and the risk of unscheduled downtimes due to machine breakdown.

Deterministic scheduling problems with unavailability intervals have received considerable attention since the beginning of the 1990s. Several comprehensive reviews of scheduling under availability constraints in different kinds of production systems are provided [2–5]. When unavailability constraints are due to PM, two different ways of modelling the unavailability intervals have been considered in different machine environments: (i) the intervals are fixed, and (ii) the intervals are flexible. The first type is the interrelated model, which considers maintenance as a constraint, not a decision. The second type is the integrated model, which corresponds to the situation where unavailability intervals due to maintenance interventions also have decision variables. Moreover, regarding the possibility for an operation to be interrupted by an unavailability period, jobs may be resumable, where the interrupted operation can be continued without any penalty after the end date of the unavailability period, or non-resumable, where the operation needs to be fully restarted [6].

While much research has been done on the interrelated model, there are fundamental limitations in this approach. For instance, the machine availability constraints must be set ahead, which is challenging to know in practice and may affect the search space where optimal solutions exist. Hence, it is more advantageous to adopt the integrated model, which considers maintenance and production scheduling simultaneously [7–11]. All these studies assume that the optimal PM intervals are regular, which means that after a constant period, the PM actions must take place to renew the machine. This strategy may lead to poor performance because it constrains the flexibility of the PM schedule, which does not coordinate the production scheduling and maintenance well. To address this issue, the non-periodic PM intervals are proposed [12–23].

Recently, predictive maintenance has emerged as an intelligent strategy based on the actual condition of the equipment. Unlike time-based PM, which attempts to predefined regular or flexible intervals based on reliability analysis or manufacturer recommendations, predictive maintenance is scheduled based on the prognostics and health management (PHM) process [24]. PHM seeks to analyze the health status of equipment which allows us to define the optimal moment to schedule a maintenance intervention. This proactive approach enables early detection of potential failures, minimizing downtime, optimizing maintenance schedules, and extending the lifespan of equipment for improved operational efficiency. Concisely, the production system's health state is continuously (or periodically) monitored by analyzing signals collected from embedded sensors (or inspection information). After that, the current state is inferred and the future progression of failure is predicted to estimate the time before failure, known as the remaining useful life (RUL). The development of effective PHM approaches is a significant research field. Indeed, PHM benefits are heavily reliant on decision-making based on prognosis information, a process known as post-prognostic decision (PPD) [25]. PPD may occur as the integration of predictive maintenance planning and production schedule, to enhance safety, and minimize downtimes and inopportune maintenance spending.

Building upon the above advancement in the field of predictive maintenance and its superior advantages over traditional PM, this paper focuses on the optimization of integrated production scheduling and predictive maintenance planning in permutation flowshop, one of the most common types of manufacturing layouts. In such an organization, the goal of production scheduling is to determine the optimal order of jobs to be processed by the machines to minimize the overall completion time, referred to as Makespan or C_{max}. When coupled with predictive maintenance decisions, this job sequence must take into account the integration of maintenance interventions for each machine. Concretely, each machine on the floor is run to process jobs in the form of several production batches (or processing periods). Upon completion of a production batch and reaching a high level of degradation, the machine is stopped and a maintenance period takes place, during which, a predictive maintenance activity is performed to renew the machine before it can resume processing the next production batch. The pivotal decision to conclude a processing period and initiate a maintenance period is based on the health state of the machine, provided by prognostics information. Machines are permitted to operate as long as possible to optimize their utilization. However, considering the objective of minimizing the Makespan, the starting dates of maintenance operations are not fixed, which leads to more flexibility in the schedule.

The PFSP with predictive maintenance is recognized as NP-hard as it extends the classical flowshop scheduling problem under availability constraints [26]. Consequently, metaheuristic algorithms have emerged as the most practical approach for addressing this challenge. Hence, we propose tackling this issue by employing a population-based metaheuristic, specifically the artificial bees colony (ABC) algorithm [27]. ABC is a promising metaheuristic algorithm that simulates the specific intelligent and collective behaviour of honey bee swarms. Since ABC was proposed, it has mainly been used to solve unconstraint and constraint continuous function optimization problems [28,29]. Compared with other metaheuristic algorithms, there are some remarkable characteristics of ABC such as its effective global exploration capability, excellent local exploitation ability, and fast convergence speed. Due to its simple structure and convenient implementation, ABC has demonstrated efficiency in handling combinatorial optimization problems [30], and particularly scheduling problems [31]. Its successful applications on scheduling prove that ABC possesses an excellent searchability and convergence rate. These factors motivated our idea to use ABC to solve the integrated PFSP and predictive maintenance planning. However, ABC cannot be directly used to solve this problem because it is completely different from other scheduling problems. The algorithm needs to be modified according to our problem features. Therefore, we are committed to making improvements on ABC for solving the integrated PFSP and predictive maintenance planning, in this paper.

The remaining part of the paper is organized as follows. Related works are summarized in Section 2. Section 3 describes the research problem. Section 4 explains the proposed ABC algorithm. Section 5 is depicted to present the experimental results and discussions. Section 6 outlines the main conclusions and future research topics.

2. Related Works

In this research work, the production scheduling part deals with one of the most studied problems in the literature with extensive applications in real-life manufacturing, namely the permutation flowshop scheduling problem (PFSP). Several interesting papers reviewing the existing methods to solve the PFSP can be found in [32–37].

Whereas production scheduling is studied separately in the conventional PFSP, excluding maintenance considerations, the key contribution of this paper falls within the scope of embedding maintenance activities. As the importance of joint optimization of production and maintenance scheduling has been highlighted, this topic has attracted significant research interest for several decades. The latest research trends are directing their attention to the implementation and optimization of PHM approaches by establishing the most effective integrated scheduling of production and predictive maintenance to enhance the overall outcomes. Here, we will provide an overview of studies, focusing on flowshop scheduling problems with flexible unavailability periods due to predictive maintenance. In this area, it is noticed that relatively few works were proposed compared to conventional PM. The reader may refer to the extensive review in [38].

The first approach to tackle with PFSP with predictive maintenance was introduced in [39]. The authors proposed a new case study where machines can switch between two production modes: nominal and sub-nominal. In the second mode, the machine is slowed down to avoid early failures. As a consequence, the production tasks will be longer than expected but the RUL is increased. They developed a mixed integer linear model (MILP) that allows finding the best production and predictive maintenance scheduling optimizing the aggregated sum of makespan and maintenance delays. However, a single maintenance operation is considered per machine during the production horizon. Moreover, the defined MILP optimally solves small instances but it is not able to compute the optimal solution for instances with an important number of jobs and machines. To deal with large instances of the same problem, the study in [40] proposes a local search method that is proven to be effective and scalable compared to the exact approach (MILP). Authors in [41] designed an improved genetic algorithm (GA) to solve the PFSP with predictive maintenance, where several maintenance interventions are planned for each machine based on degradation information provided by PHM module. In a close context, we can refer to the works provided in [42,43], where authors studied the PFSP under maintenance constraints and proposed different metaheuristics to minimize the expected Makespan. However, it was assumed that the degradation value of a machine could be determined only through inspections in pre-specified intervals.

To address uncertainties of PHM information, [44,45] proposed fuzzy logic-based metaheuristics to solve the PFSP with predictive maintenance. In these studies, uncertain PHM outputs are modelled using fuzzy logic. A modified version of the PFSP with flexible maintenance is investigated in [46] where processing times of jobs are assumed to be variable due to deterioration and learning effects. Considering the dynamic aspect of real-life scheduling, a novel approach is developed in [47] to solve the PFSP with predictive maintenance. Sensor data are processed using machine learning algorithms and times to failure (RULs) are predictive. The proposed framework, combined with discrete event simulation, enables the integration of predictive maintenance constraints into the scheduling process, to minimize the Makespan.

Production scheduling is an arduous decision-making process and has been recognized as NP-hard for almost all machine configurations, constraints, and optimization criteria combinations [48,49]. Indeed, the simplest single-machine scheduling with total tardiness is NP-hard, so large-scale scheduling problems are often difficult to solve by the traditional methods and metaheuristics become the main path to solve these problems for their effectiveness in offering optimal/near-optimal results within a reasonable amount of time notably on large-scale optimization problems [50,51]. Metaheuristic algorithms mimic the processes of natural phenomena [52–55]. Evolutionary algorithms mimic the evolution behaviour of humans or animals (genetic algorithms [56], biogeography-based optimization [57], etc.), swarm intelligence algorithms imitate the activity of groups of animals (such as particle swarm optimization [58], ant colony optimization [59], fruit fly optimization [60], bat algorithm [61], grey wolf optimization [62], migrating birds optimization [63], etc.), and physics or chemical processes based algorithms emulate physics or chemical laws (such as chemical reaction optimization [64], galaxy-based search algorithm [65], black hole algorithm [66], etc.). Due to the large number of problem variants and solution methods, several survey papers provide comprehensive insights into the literature addressing various types of scheduling problems. These surveys cover solution methods such as evolutionary algorithms [67], automatically generated dispatching rules [68], and multi-population-based metaheuristics [31].

Among swarm intelligence algorithms for optimization problems, the artificial bee colony (ABC) algorithm was first proposed in 2005 to solve continuous (unconstrained) problems [27] and it was later successfully extended to constrained optimization problems [69]. Literature results show that the performance of ABC algorithm is comparable to other state-of-the-art algorithms for high dimensionality optimization [70]. ABC has become a very active research area and many modifications [28] and enhancements [71] of the original algorithm were introduced. For PFSP, ABC was applied for different versions of the problem, as a standalone method [72–77], hybridized with other approaches [46,78], to optimize single or multiple objective functions [79,80].

3. Problem Statement

We tackle the joint scheduling of production and predictive maintenance within a specific manufacturing layout: the permutation flowshop. This integrated problem involves the simultaneous optimization of the production schedule and maintenance activities planning. Hence, the goal is to identify the most effective sequencing of jobs and the strategic timing of maintenance activities for each machine. The optimization objective is to minimize the total completion time (of production jobs and maintenance activities), referred to as Makespan or C_{max}. For each machine on the shopfloor, the resulting joint schedule can be seen as a succession of production blocks, during which the machine executes the jobs included in the block, separated by unavailability periods, in which predictive maintenance interventions are planned.

In a permutation flowshop, a set $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ of n production jobs move through a series $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ of m machines, in a predetermined order without altering the order of the jobs from one machine to another, i.e. only permutation schedules are allowed. The processing time of job J_j on machine M_i is denoted p_{ij}. All jobs are assumed to be available at time zero and preemption of jobs is not allowed, i.e., once the execution of an operation is started, it cannot be interrupted neither by other operations nor by maintenance activities. Each machine processes only one action (production job or maintenance activity) at a time. Setup times are included in the processing times and are sequence-independent.

For the maintenance side, multiple unavailability periods are considered for each machine. Unavailability periods are due to predictive maintenance activities scheduled based on PHM outputs. For each machine M_i, a degradation value δ_{ij} is associated when processing each job J_j. We consider the degradation value associated to job J_j when being processed on machine M_i, $\delta_{ij} = \frac{p_{ij}}{RUL_{ij}} \times 100\%$ where $0 < \delta_{ij} < 100\%$ and RUL_{ij} is the remaining useful life provided by PHM process.

At $t = 0$, all machines are supposed to be "as good as new"; i.e., their initial degradation values are null. When processing each job J_j, the accumulated degradation of a machine M_i is updated according to the corresponding degradation value δ_{ij}. When the current accumulated degradation of a machine (i.e., the sum of degradation values of jobs that have been processed) reaches a maximal threshold $\Delta = 100\%$, a maintenance intervention must be performed. Flexible interventions are considered. The time intervals between each two consecutive maintenance activities as well as the number of maintenance activities are assumed to be decision variables and they are defined during the search process. Indeed, the maintenance planning strategy, described in Section 4.1 allows to align with the production schedule by ensuring minimal disruption of the schedule. Maintenance intervention timing is not fixed but is adapted for each machine with respect to total completion time (C_{max}) minimization. We suppose that at least one maintenance intervention is performed on each machine and no maintenance intervention is performed after the processing of the last job. After each maintenance intervention, the machine is recovered to be "as good as new".

The resulting integrated schedule is denoted $\Pi = \{\pi_1, \pi_2, \ldots, \pi_m\}$ where $\pi_i, i = 1, \ldots, m$ represents the corresponding integrated sequence of n jobs and k_i (≥ 1) flexible maintenance on machine M_i. Then π_i can be seen as a succession of $k_i + 1$ blocks of jobs B_i^l (subsets of $\mathcal{J}$) separated by predictive maintenance operations PdM_i^l:

$$\pi_i = \{B_i^1, PdM_i^1, B_i^2, PdM_i^2, \ldots, B_i^k, PdM_i^k, B_i^{k_i+1}\}, \text{where} \cup_{l=1}^{k_i+1} B_i^l = \mathcal{J}.$$

4. Proposed ABC-Based Solving Approach

The ABC algorithm has demonstrated notable success in solving scheduling problems [31]. For PFSP, ABC was applied for different versions of the problem, as a standalone method [72–77], hybridized with other approaches [46,78], to optimize single or multiple objective functions [79,80]. This success lies in its ability to leverage the principles of swarm intelligence to efficiently navigating solution spaces and finding near-optimal schedules.

Indeed, bee colonies collect information from their surroundings and adapt their behavior accordingly. When applied in the context of scheduling problems, subject to multiple constraints, this intelligent and highly dynamic behavior seems to be very efficient. ABC capacity to strike a balance between exploration and exploitation makes it well-suited for such complex optimization problems. These two key mechanisms, i.e., exploration and exploitation, are inherently included within the ABC algorithm, as will be described in the next section. Moreover, its simplicity and ease of implementation contribute to its widespread applicability. Furthermore, since the performance of metaheuristic algorithms depends on parameters tuning, a notable advantage of the ABC algorithm is that it uses only 3 control parameters (colony size, maximum cycle number, and number of iteration to drop out a solution).

Concretely, ABC algorithm employs three distinct and collective classes of artificial bees: employed bees, onlookers, and scouts. Employed bee stays on a food source (i.e., candidate solution) and examines its neighborhood. Onlookers are allocated to a food source based on the information that they gain from employed bees. If a food source does not improve for a certain number of cycles, scouts replace that food source with a new random one.

The general scheme of the proposed ABC algorithm to solve the integrated production and flexible maintenance scheduling problem is shown in Figure 1 and given in Algorithm 1. Two stopping criteria are used in the proposed ABC algorithm. The first one is the maximum cycle number *MCN*. Another option is to stop the search if the best solution found by the algorithm has not been improved for a given number of iterations *max_improv*.

The detailed description of the main features of the proposed algorithm including the initialization procedure, employed bees phase, onlookers phase, and scouts phase, are detailed in the sections below.

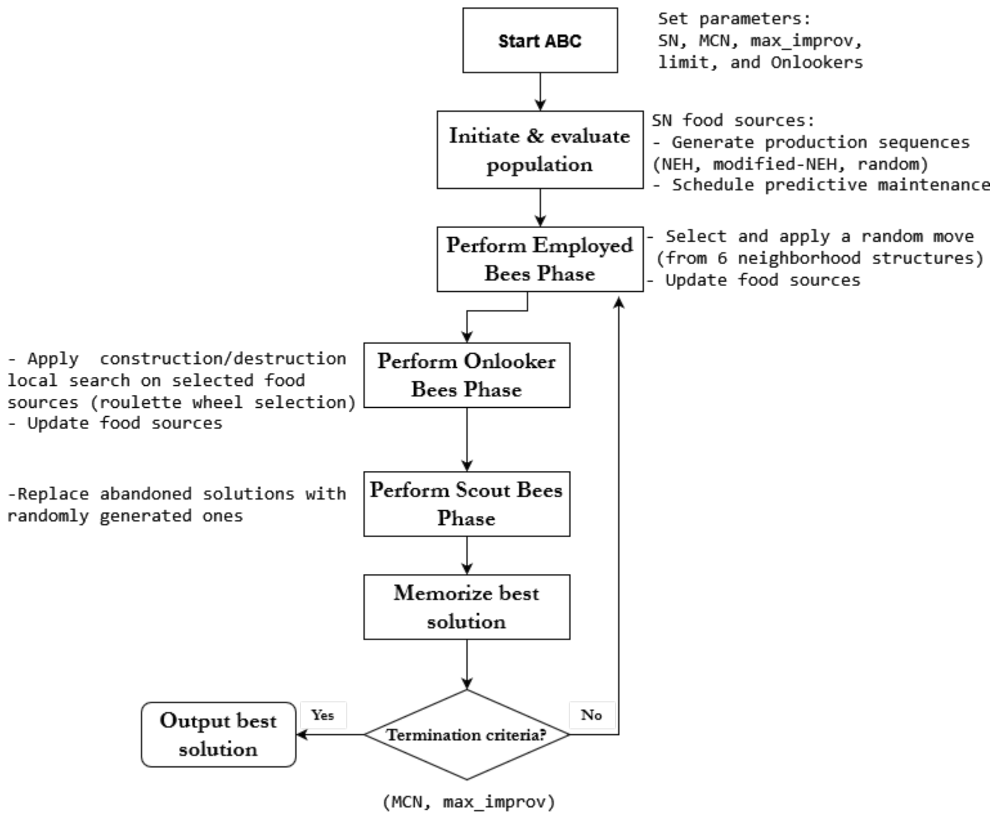
Start ABC
Set parameters: SN, MCN, max_improv, limit, and Onlookers
Initiate & evaluate population
SN food sources:
- Generate production sequences (NEH, modified-NEH, random)
- Schedule predictive maintenance
Perform Employed Bees Phase
- Select and apply a random move (from 6 neighborhood structures)
- Update food sources
- Apply construction/destruction local search on selected food sources (roulette wheel selection)
- Update food sources
Perform Onlooker Bees Phase
-Replace abandoned solutions with randomly generated ones
Perform Scout Bees Phase
Memorize best solution
Termination criteria?
(MCN, max_improv)
Yes
No
Output best solution

Figure 1. Flow chart of the proposed ABC algorithm.

Algorithm 1 The proposed ABC algorithm

```
1:  Parameters
2:  $SN$: Colony size (number of food sources)
3:  $limit$: Maximum number of iterations without improvement to drop out a food source
4:  $MCN$: Maximum cycle number of the algorithm
5:  $max\_improv$: Maximum number of iterations without improvement to stop the algorithm
6:  Initialize population with $SN$ food sources (solutions);
7:  Associate a food source to each employed bee;
8:  while $MCN$ and $max\_improv$ not reached do
9:      EMPLOYED_BEES_PHASE();
10:     ONLOOKER_BEES_PHASE();
11:     SCOUT_BEES_PHASE();
12:     Return best solution;
13: end while
```

4.1. Solution Representation, Objective Function and Initialization Procedure

In ABC, food sources represent solutions in search space, and the nectar density corresponds to the fitness value of a feasible solution. In this paper, a two-field structure is used to represent a candidate solution,i.e., food source, which jointly specifies the production job sequence and the maintenance operations positions on each machine, as proposed in [13]. The sequence of jobs to be processed by the permutation flowshop is coded using a permutation $S = \{J_{s1}, \ldots, J_{sn}\}$, where J_{sj} is the jth job to be executed. For the maintenance side, to represent the maintenance intervention positions on each machine, a binary matrix $PM(m \times n)$ is used, where each line is dedicated to a machine of the shopfloor. To indicate the presence of maintenance activity on machine M_i after the jth job of the sequence, we put $PM[i,j] = 1$, otherwise $PM[i,j] = 0$.

A single objective function is addressed in this work, which is the total completion time of the integrated schedule, after maintenance activities planning, referred to as Makespan or C_{max}.

As for any evolutionary algorithm, population (colony) initialization is a critical step for ABC algorithm. Indeed, the convergence speed and the quality of the final solution are highly affected by the diversity of the initial population. In this work, random solutions as well as good quality solutions are generated and included in the initial population. Randomly generated solutions contribute to the exploration phase of the algorithm by helping in exploring distinct regions of the solution space, which increases the chances of discovering more promising areas and avoids getting trapped in local optima. On the other side, good candidate solutions enhance the exploitation phase by focusing on promising areas where optimal or near-optimal solutions may be located. Hence, in this study, we proposed a two-step initialization procedure to generate an initial population of SN food sources (integrated solutions) as follows:

1. Generation of production sequences In this work, SN production sequences are generated as follows:
   - *Good quality solutions.* One production sequence is generated using the well-known NEH heuristic [81] considered one of the best constructive methods for minimizing Makespan in PFSP. Furthermore, $5\% \times SN$ sequences are generated using the modified NEH heuristic [82], inspired by the original NEH heuristic, which has also proved good performance;
   - *Randomly generated solutions.* The remaining part of the initial production sequences is generated randomly.
2. Generation of maintenance activities planning
   To build a feasible solution, i.e., an integrated production and maintenance schedule, a maintenance matrix PM is associated with each production sequence S, obtained from the previous step. The maintenance planning is achieved based on PHM data, in the form of degradation values. We use the heuristic proposed in [41]. The main

idea of this heuristic is to schedule maintenance operations in ascending order, i.e., starting from the first machine of the flowshop, by evaluating the total accumulated degradation of the machine when processing jobs. Jobs in the sequence S are scanned one by one and maintenance tasks are inserted, on each machine, according to the current cumulative degradation δ_i with respect to the threshold ($\Delta = 100\%$). Hence, for each machine M_i in the shopfloor, two cases could be observed as shown in the illustrative example in Figure 2:

- If $\delta_i < \Delta$, no predictive maintenance intervention has to be scheduled because the full degradation has not been achieved yet. So, we scan the next production job in the sequence. This is observed in example of Figure 2, when performing jobs J_1, J_3, J_8, the cumulative degradation of machine M_1 doesn't reach the full degradation Δ, hence, no maintenance operation is scheduled;
- If $\delta_i \geq \Delta$. The machine reaches the total degradation threshold ($\Delta = 100\%$) and a high risk of machine failure is faced. Hence, we schedule a maintenance intervention either before or after the current job, according to the position that best optimizes the Makespan. This is shown in the example of Figure 2 when processing job J_5: $\delta_{11} + \delta_{13} + \delta_{18} + \delta_{15} \geq \Delta$ so we evaluate the two options of scheduling a maintenance before or after current job J_5 and we choose the one minimizing C_{max}.

This strategy seeks to use machines at their full potential by planning maintenance only when necessary. Moreover, this enhances schedule flexibility and minimizes production disturbance by considering various options for maintenance insertion for the Makespan optimization.

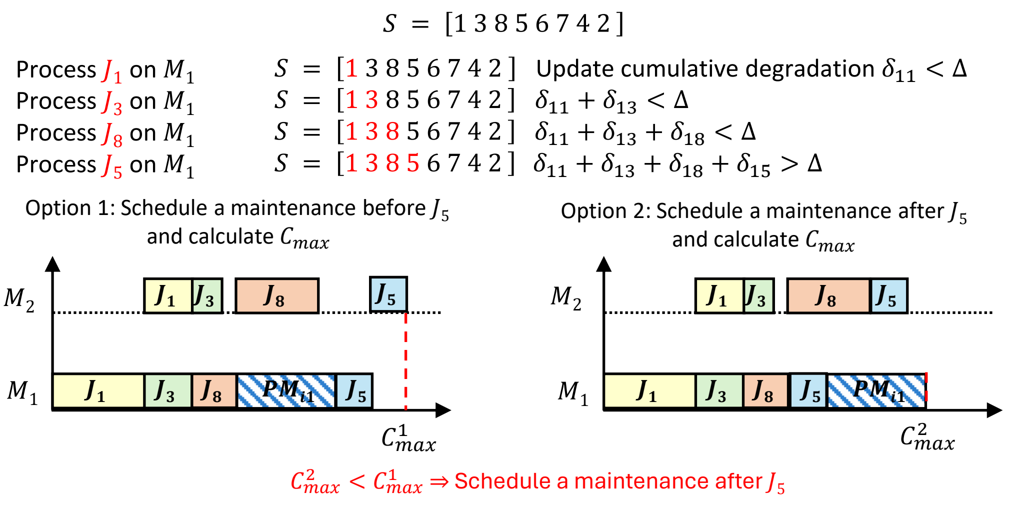
$S = [1\ 3\ 8\ 5\ 6\ 7\ 4\ 2]$
Process J_1 on M_1 — $S = [1\ 3\ 8\ 5\ 6\ 7\ 4\ 2]$ — Update cumulative degradation $\delta_{11} < \Delta$
Process J_3 on M_1 — $S = [1\ 3\ 8\ 5\ 6\ 7\ 4\ 2]$ — $\delta_{11} + \delta_{13} < \Delta$
Process J_8 on M_1 — $S = [1\ 3\ 8\ 5\ 6\ 7\ 4\ 2]$ — $\delta_{11} + \delta_{13} + \delta_{18} < \Delta$
Process J_5 on M_1 — $S = [1\ 3\ 8\ 5\ 6\ 7\ 4\ 2]$ — $\delta_{11} + \delta_{13} + \delta_{18} + \delta_{15} > \Delta$
Option 1: Schedule a maintenance before J_5 and calculate C_{max}
Option 2: Schedule a maintenance after J_5 and calculate C_{max}
M_2
M_1
J_1
J_3
J_8
J_5
PM_{i1}
C^1_{max}
C^2_{max}
$C^2_{max} < C^1_{max} \Rightarrow$ Schedule a maintenance after J_5

Figure 2. Maintenance insertion strategy.

Once the initial population is created, the search process is launched. Each iteration consists of three stages: the employed bees phase, the onlooker bees phase, and the scout bees phase, which will be presented in the next sections.

4.2. Employed Bees Phase

The employed bees phase is the first step in the search process of ABC algorithm. In this phase, a food source, i.e., an integrated schedule within the current population, is associated with each employed bee. Hence, the number of employed bees is equal to the number of food sources SN. The main role of employed bees is to move around the neighborhood of their assigned food sources to discover new and potentially better solutions. Hence, move strategies need to be defined to navigate in the search space. In this work, we use two complementary and efficient neighborhood structures from the existing literature, which have proven to be efficient in exploring solutions in PFSP. These structures allow exploration of production sequences neighborhood as follows:

- One-swapping based neighborhood. To obtain a neighbor solution, a swap move is carried out by exchanging the positions of two production jobs, randomly selected from the production sequence of the current solution, as shown in Figure 3.
  Let S be the production sequence of the current solution. p and q are two distinct positions, randomly selected from 1 to n (size of S). When applying one-swapping move, all jobs are maintained but jobs corresponding to positions p and q are interchanged to obtain the new sequence S':

$$S'[p] = S[q]$$
$$S'[q] = S[p]$$
$$S'[k] = S[k] \text{ for } k \neq p, q$$

- Double-swapping based neighborhood. In this case, two swap moves are carried out successively on the production jobs sequence, as shown in Figure 3.

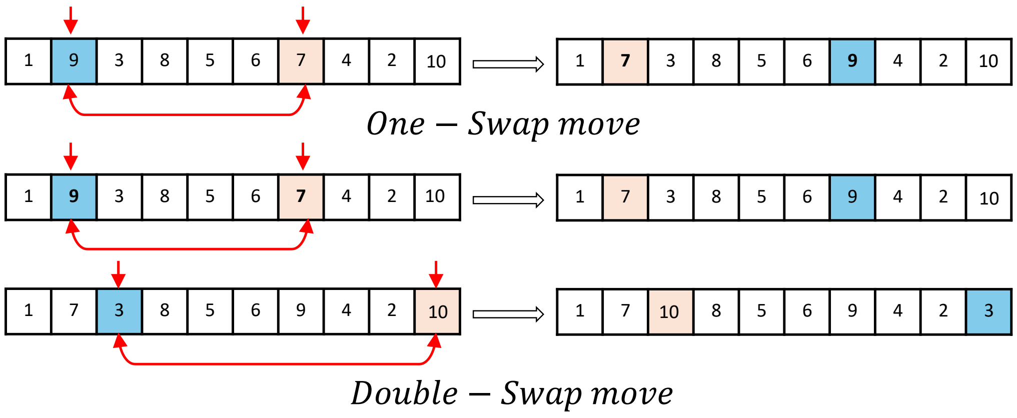

Figure 3. Illustrative examples of One-swap and Double-swap moves.

- One-insertion-based neighborhood. A production job is randomly selected from the production sequence, deleted from its current position, and then inserted into a new randomly selected one, as shown in Figure 4.
  Let S be the production sequence of the current solution. p and q are two distinct positions, randomly selected from 1 to n (size of S). When applying one-insertion move, job corresponding to position p is deleted from the sequence S and then inserted in position q.
  If $p < q$:

$$S'[i] = \begin{cases} S[i] & for\ \ i < p\ \ or\ \ i > q \\ S[i+1] & for\ \ p \leq i < q \\ S[p] & if\ \ i = q \end{cases} \tag{1}$$

  If $p > q$:

$$S'[i] = \begin{cases} S[i] & for\ \ i < q\ \ or\ \ i > p \\ S[i-1] & for\ \ q < i \leq p \\ S[q] & if\ \ i = p \end{cases} \tag{2}$$

- Double-insertion based neighborhood. It consists of two insert moves, carried out successively on the production jobs sequence, as shown in Figure 4.

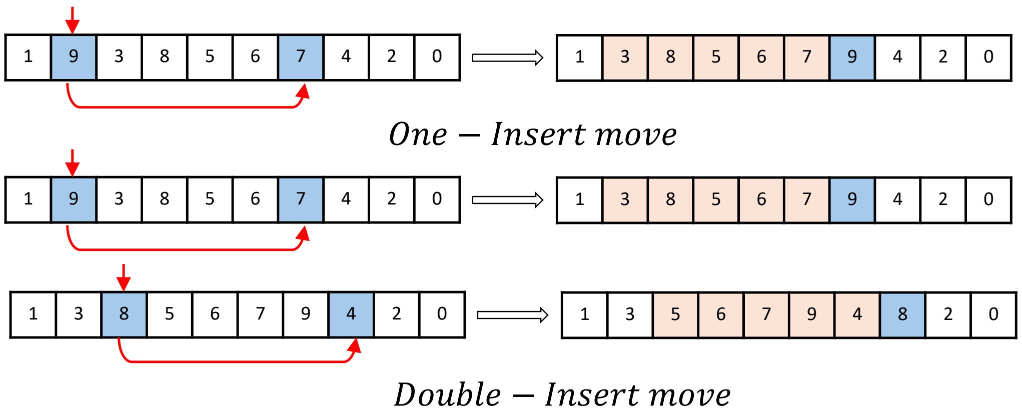

Figure 4. Illustrative examples of One-insert and Double-insert moves.

To explore solutions neighborhood based on maintenance planning, two other structures are used as shown in Figure 5:

- Right shift-based neighborhood. In this move, a maintenance operation planned on a selected machine is randomly chosen from the predictive maintenance matrix, and then shifted to the right, i.e., moved forward to be rescheduled after the next job.
- Left shift-based neighborhood. A maintenance operation planned on a selected machine is randomly chosen and then shifted to the left, i.e., moved backward to be rescheduled before the current job.

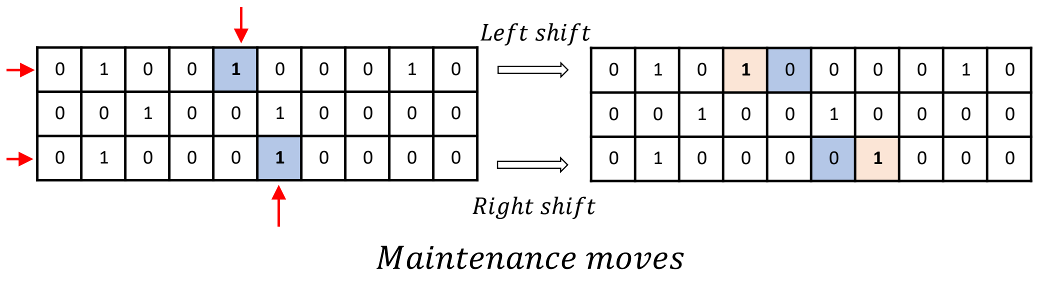

Figure 5. Illustrative examples of right and left shift moves.

Once the initial population is generated as explained before, each employed bee flies to a unique food source (candidate solution). Throughout the search process iteration, each employed bee seeks to find a new food source within its neighborhood. For this reason, the employed bee randomly selects one of the previous neighborhood structures and uses it to generate a new solution. Note that if maintenance operations are altered, they will be readjusted. A greedy selection mechanism is used to choose between the current and the new food sources. In other words, if the Makespan of the current solution is improved, then it is replaced by the new solution, which will be then assigned to the employed bee in subsequent iterations. Otherwise, the current solution is retained. To control the evolution of solutions manipulated by the employed bees and their eventual stagnation in local optima, a counter *trials_count* is associated with each solution and incremented whenever its objective function (Makespan) doesn't improve. Employed bees share their findings (solution information) with onlooker bees. The process of employed bees phase in given in Algorithm 2.

Algorithm 2 Employed bees phase algorithm

```
1:  procedure EMPLOYED_BEES_PHASE()
2:      for all Employed bees $EB_i, i = 1, .., SN$ do
3:          Let $SN_i$ be the food source associated to $B_i$
4:          Randomly select a neighborhood structure move from (swap, double swap, insert,
    double insert, leftshift, rightshift)
5:          Use move to generate new solution $SN_i^*$ from $SN_i$
6:          Reschedule maintenance operations of $SN_i^*$
7:          if $fitness(SN_i^*) > fitness(SN_i)$ then
8:              Update food solution $SN_i$ with $SN_i^*$
9:              Reset trials_count of $SN_i$ to 0
10:         else
11:             Increment trials_count of $SN_i$;
12:         end if
13:     end for
14: end procedure
```

4.3. Onlooker Bees Phase

The onlooker bees phase in the ABC algorithm deals with the exploitation of the search process. This phase involves the selection of promising food sources by onlooker bees based on the information shared by employed bees. Onlooker bees then generate new solutions through a local search method, and if the new solution is better in term of the Makespan, the corresponding food source is updated, as detailed in Algorithm 3. Similarly, if maintenance operations are modified, they are readjusted after the local search process.

Algorithm 3 Onlooker bees phase algorithm

```
1:  procedure ONLOOKER_BEES_PHASE()
2:      Onlookers: Number of onlooker bees
3:      for all Onlooker bees $OB_i, i = 1, .., Onlookers$ do
4:          Select a food source $SN$ using roulette wheel scheme
5:          $SN^* \leftarrow LOCAL\_SEARCH(SN, d)$
6:          Reschedule maintenance activities of $SN^*$
7:          if $fitness(SN^*) > fitness(SN)$ then
8:              Update food solution $SN$ with $SN^*$
9:              Reset trials_count of $SN$ to 0
10:         else
11:             Increment trials_count of $SN$
12:         end if
13:     end for
14:     Apply the same process on the best food source in the population
15: end procedure
```

The selection process is performed by onlooker bees using a roulette wheel approach, where each food source is associated with a selection probability determined by its fitness evaluation. This allows food sources with better fitness to have a higher chance of being selected. Note that for Makespan (C_{max}) minimization, the considered fitness function is $fitness = 1/C_{max}$. The following formula is used to evaluate the selection probability p_i of a food source fs_i:

$$p_i = \frac{fitness(fs_i)}{\sum_{k=1}^{SN} fitness(fs_k)} \tag{3}$$

Once each onlooker bee selects a food source, it starts generating a new solution through a local search method. Various local search algorithms are available in the literature. In this work, the proposed local search is inspired by the construction/destruction mechanism introduced in [73]. This procedure consists of removing a predetermined number of jobs from the production sequence and then rescheduling them based on the well-known NEH heuristic [81]. Since NEH heuristics allows finding the best emplacement of a job in a sequence to minimize C_{max}, this local search method enhances the quality of the solution found by the employed bees, promoting then the exploitation of the search process. Two distinct stopping criteria are considered for the local search: *first improve* criterion, where the local search concludes upon achieving the first improved solution. This criterion is applied to all selected food sources at each iteration. The second criterion involves conducting an exhaustive search, by inserting the removed jobs into all possible positions. Given that this criterion is greedy in terms of computational time, it is applied only to the best solution in each iteration. Similarly to the employed bees phase, if a solution is not improved after the local search procedure, its respective counter (*trials_count*) is incremented. The local search procedure integrated in onlooker bees phase is given Algorithm 4.

Algorithm 4 Local search algorithm

```
1:  procedure LOCAL_SEARCH($SN, d$)
2:      $SN$: food source (solution)
3:      $d$: number of jobs to be rescheduled
4:      Let $S = \{J_{s1}, .., J_{sn}\}$ be the job sequence associated to solution $SN$
5:      for $i \leftarrow 1$ to $d$ do
6:          Select a random job position $j$ and delete corresponding job $J_{sj}$
7:          for $k \leftarrow 1$ to $n - 1$ do
8:              Insert $J_{sj}$ in position $k$ and evaluate new Makespan $(C^*_{max})$
9:              if $(C^*_{max} < C_{max})$ then
10:                 Update solution $SN$
11:             end if
12:             if $SN$ is not the best solution in the population then
13:                 stop procedure (first improvement scheme)
14:             end if
15:         end for
16:     end for
17: end procedure
```

4.4. Scout Bees Phase

The scout bees phase is the last stage of ABC algorithm iteration. During this phase, scout bees fly to candidate solutions with *trials_count* exceeding the allowed maximum threshold, denoted *limit*. These solutions are abandoned and replaced by new randomly generated ones in the subsequent iteration, as given in Algorithm 5.

Algorithm 5 Scout bees phase algorithm

```
1: procedure SCOUT_BEES_PHASE()
2:     for all Food sources $SN_i, i = 1, .., SN$ do
3:         if $trials\_count$ of $SN_i > limit$ then
4:             Generate new random solution $SN^*$;
5:             Update food solution $SN_i$ with $SN^*$;
6:             Reset $trials\_count$ of $SN$ to 0;
7:         end if
8:     end for
9: end procedure
```

5. Computational Results

This section is devoted to presenting the results of computational experiments conducted to calibrate and evaluate the newly proposed ABC algorithm for the integrated scheduling of production and predictive maintenance. All algorithms and tests were developed using Python and executed on Microsoft Azure cloud using the Standard *NC6_Promo* virtual machine under the Ubuntu 18 operating system, equipped with 6 cores and 6 gigabytes of RAM.

We first describe the data benchmarks used in this study. Next, we outline the statistical analysis conducted for the configuration of ABC parameters. Finally, we analyze the performance of the proposed ABC algorithm when compared to a VNS (variable neighborhood search) method from the literature [44].

5.1. Data Benchmarks for the Integrated Scheduling Problem

We apply a similar data generation approach as proposed in [44]. For production data, we use the well-known Taillard's instances [83]. This benchmark consists of 120 instances of the PFSP, where 12 configurations of flowhsop $(n \times m)$ are considered: $n \in \{20, 50, 100, 200, 500\}$ and $m \in \{5, 10, 20\}$. For each configuration, 10 instances are included. For predictive maintenance data, PHM information in form of degradation values δ_{ij} are generated for each job when being executed by each machine, according to its processing time p_{ij}. This involves three cases: Low, Medium, and High degradation values as follows:

- Jobs with processing times $p_{ij} < 20$ induce machine degradation δ_{ij} uniformly distributed in $[0.02, 0.03]$;
- Jobs with processing times $20 \leq p_{ij} < 50$ induce machine degradation δ_{ij} uniformly distributed in distributed in $[0.03, 0.06]$;
- Jobs with processing times $p_{ij} \geq 50$ induce machine degradation δ_{ij} uniformly distributed in $[0.06, 0.1]$.

For maintenance operations processing times, two variants are considered. In the first mode, medium maintenance durations are generated from the uniform distribution $U[50, 100]$. In the second mode, long maintenance durations are generated from the uniform distribution $U[100, 150]$. Hence, we obtain a set of 120 instances of the integrated scheduling problem for each maintenance mode, and then a total of 240 instances.

Three metrics are calculated to assess the performance of the proposed ABC algorithm:

(1) The relative percentage deviation (RPD) of the Makespan (C_{max}) obtained by the ABC algorithm, with respect to a theoretical lower bound LB. The RPD is calculated as follows:

$$RPD = \frac{C_{max} - LB}{LB} \times 100 \tag{4}$$

The proposed lower bound LB is based on the lower bound suggested by Taillard in [83], taking into account the maintenance operations. Let a_i be the minimum duration that machine M_i must wait before it can start processing (waiting for the previous machines in the flowshop), b_i the minimum duration the machine M_i remains inactive after completing processing all jobs (waiting for the next machines in the flowshop), k_i the total number of maintenance activities of duration D and T_i be the total processing time of machine M_i including both production and maintenance operations. Then:

$$a_i = \min_{\text{j}}(\sum_{k=1}^{i-1} P_{kj}) \tag{5}$$

$$b_i = \min_{\text{j}}(\sum_{k=i+1}^{m} P_{kj}) \tag{6}$$

$$T_i = \sum_{j=1}^{n} P_{ij} + k_i \times D \tag{7}$$

where m is the number of machines in the flowshop, n is the number of jobs, and p_{ij} is the processing time of task J_j on machine M_i.

The sum $a_i + T_i + b_i$ represents the time a machine should wait before it starts processing (a_i), added to the total time it needs to process all production and maintenance tasks (T_i), added to the minimum period it waits till the next machines of the flowshop finish their execution (b_i). Given that no waiting times are considered between tasks, C_{max} must exceeds the maximum of $a_i + T_i + b_i$. On the other hand, C_{max} also exceeds the maximum period required for jobs to pass through all machines of the flowshop. Hence, the lower bound LB is obtained as follows:

$$LB = max\{\max_{\mathrm{i}}(a_i + T_i + b_i), \max_{\mathrm{j}} \sum_{i=1}^{m} P_{ij}\} \tag{8}$$

(2) The average percentage of maintenance operations earliness/tardiness (ET). It is calculated according to the machine's accumulated degradation at the moment of predictive maintenance intervention scheduling. An ideal maintenance operation is scheduled when the machine is fully exploited, i.e., its accumulated degradation is equal to 100%. Otherwise, a maintenance operation is either early if the accumulated machine degradation is less than 100% or tardy if the accumulated machine degradation is greater than 100%. Hence, the ET of maintenance operations on machine M_i is:

$$ET_i = \frac{\sum_{k=1}^{n_i} |\delta_k - 1|}{n_i} \tag{9}$$

where n_i is the number of predictive maintenance operations scheduled on machine M_i and δ_k is the value of the accumulated degradation of the machine when the kth predictive maintenance is scheduled.

The average percentage of ET of all maintenance operations of the schedule is calculated as follows:

$$ET = \frac{\sum_{i=1}^{m} ET_i}{m} \times 100 \tag{10}$$

(3) Computation time CPU.

5.2. Statistical Analysis for ABC Parameters Tuning

The ABC algorithm has three parameters, namely, the number of food sources (SN) which is equal to the number of employed bees, the number of iterations after which a non-improved solution is abandoned $limit$, and the maximum cycle number (MCN). In most cases, the number of onlooker bees is equal to the number of employed bees. In this work, to limit the computation time and enhance the balance between exploration and exploitation for the specific needs of our optimization problem, the number of onlooker bees (denoted $Onlookers$) is also considered as a parameter.

Four parameters (SN, MCN, $limit$, and $Onlookers$) are assessed in the calibration procedure. Various levels are considered for each parameter: $SN \in \{50, 70, 100\}$, $MCN \in \{100, 150, 200\}$, $limit \in \{5, 10, 50\}$, and $Onlookers \in \{0.2 \times SN, 0.3 \times SN, 0.4 \times SN\}$ of the total number of food sources SN. A full factorial design was conducted, in which all possible combinations of various parameters were tested. The results were then analyzed using analysis of variance method statistical method, with the least significant difference intervals at 95%, to set each factor to its best level.

To allow fair parameter tuning, the algorithm calibration tests are executed on a new set of instances. Production data are generated following procedure outlined in [83], including 20 combinations of $n \times m$, where $n \in \{20, 70, 120, 170, 220\}$ and $m \in \{5, 10, 15, 20\}$. Two instances are generated for each combination, resulting in a total of 40 instances. The same approach described in Section 5.1 is used for maintenance data generation (the first maintenance mode is adopted).

Applying a full factorial design, we have $3 \times 3 \times 3 \times 3 = 81$ parameter combinations to test on 40 instances, with 5 runs per instance, resulting in 16.200 executions. The response variable for analysis of variance is the Makespan deviation RPD. Results show that the best combination of ABC parameters is: $SN = 70$, $MCN = 200$, $limit = 5$, and $Onlookers = 0.4 \times SN$.

5.3. ABC Performance Analysis

In this section, we will provide insights into results obtained by the proposed ABC algorithm when applied to solve the 120 instances of the integrated scheduling problem in the two maintenance modes. To assess the performance of the ABC approach, it is compared to a variable neighborhood approach (VNS) from the literature [44]. This work was selected as it tackles the same integrated scheduling problem of production and predictive maintenance. To make a fair comparison, the VNS algorithm was re-executed on the same instances and in the same environment (machine configuration). Results obtained by ABC and VNS in terms of RPD and ET are depicted in Tables 1 and 2, for maintenance modes 1 and 2, respectively.

Table 1. RPD and ET results for ABC and VNS algorithms in maintenance mode 1 (in %).

Instance	RPD ABC	RPD VNS	ET ABC	ET VNS
20 × 5	**4.62**	11.52	**2.05**	2.08
20 × 10	**12.82**	23.01	**2.67**	3.07
20 × 20	**25.03**	28.19	3.84	**1.58**
50 × 5	1.94	**1.72**	3.83	**0.82**
50 × 10	9.8	**8.24**	3.7	**1.85**
50 × 20	19.69	**18.5**	4.28	**2.01**
100 × 5	**1.17**	1.35	4.7	**0.69**
100 × 10	**4.37**	5.09	4.73	**1.47**
100 × 20	**13.47**	15	4.87	**1.81**
200 × 10	**1.75**	3.13	5	**1.77**
200 × 20	**7.25**	11.21	4.93	**2.33**
500 × 20	**3**	-	5.04	-
Average	**9.26**	11.55	4.14	**1.85**

Table 2. RPD and ET results for ABC and VNS algorithms in maintenance mode 2 (in %).

Instance	RPD ABC	RPD VNS	ET ABC	ET VNS
20 × 5	**3.68**	10.617	**2.06**	2.8
20 × 10	**14.05**	24.199	4.13	**3.12**
20 × 20	**27.52**	32.982	3.14	**1.72**
50 × 5	3.03	**1.933**	4.66	**1.16**
50 × 10	12.39	**9.955**	4.23	**1.93**
50 × 20	24.9	**19.276**	4.22	**2.15**
100 × 5	1.47	**1.463**	4.56	**0.85**
100 × 10	6.23	**5.483**	4.67	**1.61**
100 × 20	16.43	**15.614**	4.8	**1.97**
200 × 10	**2.52**	3.429	4.88	**1.93**
200 × 20	**9.64**	12.549	4.89	**2.33**
500 × 20	**3.04**	-	**5.01**	-
Average	**11.57**	12.5	4.27	**1.96**

For Makespan (C_{max}) minimization, the proposed ABC algorithm outperforms the VNS method for the majority of instances, in both maintenance modes. On average, ABC is the best, with an average RPD of 9.26% against 11.55% for VNS in maintenance mode 1, and 11.57% against 12.5% in maintenance mode 2. Hence we could conclude that ABC is more efficient in C_{max} minimization. It is important to note that we could not execute the VNS algorithm for instances 500 × 20 due to excessively long execution times.

To statistically validate the significance of the difference in RPD values provided by the proposed ABC algorithm and VNS, Wilcoxon's statistical analysis of variance was conducted. Results reject the null hypothesis, i.e., significant variance is validated between the two algorithms at a confidence interval of 90% ($p\text{-}value$) was set to 0.1). Figure 6 shows the mean plots of both algorithms for RPD optimization.

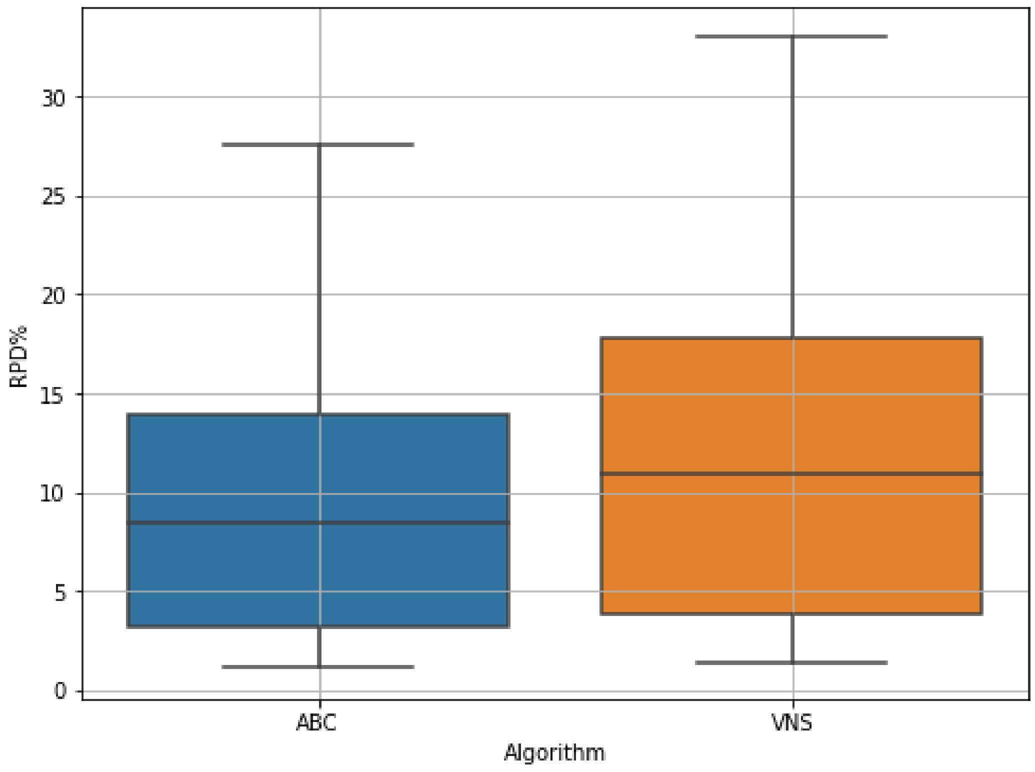

Figure 6. Box-plots for ABC and VNS algorithms for RPD optimization.

On the other hand, when considering maintenance average ET minimization, the VNS method provides better results in most cases, with an average ET of 1.85% against 4.14% for the proposed ABC in maintenance mode 1, and 1.96% against 4.27% for maintenance mode 2. It is worth noting that the maintenance criterion (average ET) was only calculated in this study, but single objective optimization, of C_{max}, was considered. However, when jointly observing both criteria, we can confirm that ABC allows achieving a good trade-off between production completion time and maintenance ET while maintaining good results for C_{max}.

When comparing maintenance modes, we notice that C_{max} deviations increase in mode 2, while maintenance average ET remains steady. Indeed, C_{max} deviations are influenced by the duration of maintenance interventions: greater disruptions occur when longer maintenance activities are scheduled. However, maintenance ET depends solely on the timing of maintenance interventions and remains unaffected by their duration.

A critical feature of an optimization problem is the convergence phenomenon. To analyze the behaviour of the proposed ABC algorithm through the search iterations, we provide in Figure 7 an illustrative example of several independent executions of the ABC algorithm on one given instance of the integrated problem of size 100×10 in maintenance mode 1. We can observe that the ABC algorithm is effectively progressing towards finding the near-optimal possible solution. We notice that the convergence rate is high at the first ≈ 50 iterations, after that, the search progress slows down. In most cases, the search process globally converges towards the same best solution ($RPD = 2.56\%$). Some local convergences (in local optima) are also observed, but the ABC algorithm is most often able to boost the search process. Otherwise, based on the second termination criterion $max\_improv = 40$ iterations, the search process is stopped if the algorithm is stacked in local optimum for more than 40 iterations.

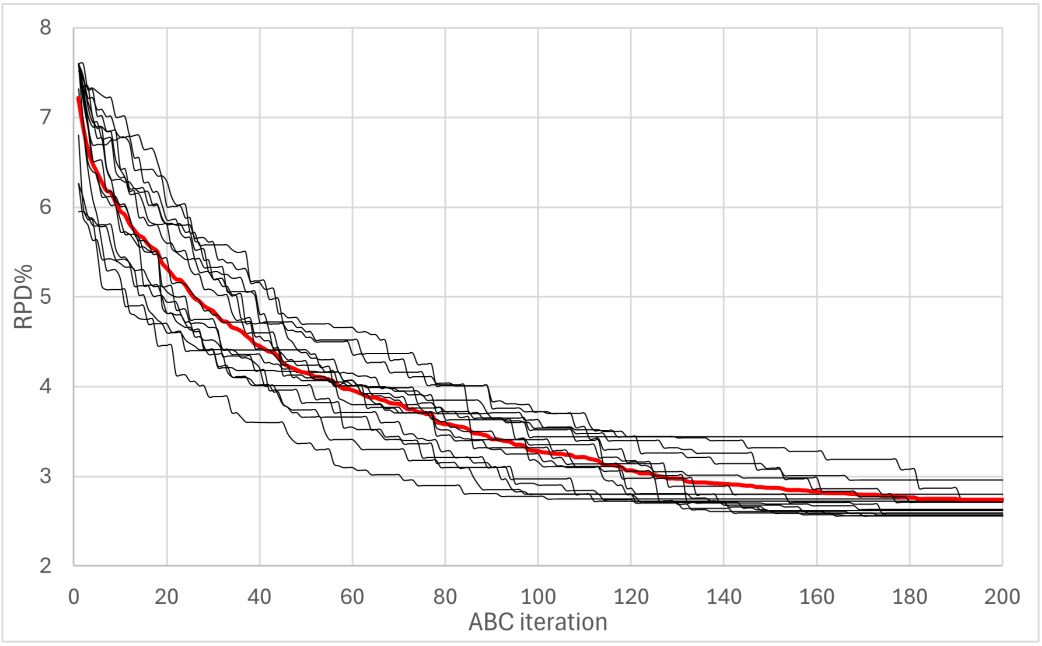

Figure 7. Convergence curves of the ABC algorithms and average curve (in red) for one instance of size 100 × 10.

To assess the computational performance of both methods, *CPU* time is evaluated in Table 3. This metric aids in identifying potential trade-offs between solution quality and computational speed. For small instances ((20 × 5), (20 × 10), (20 × 20), (50 × 5)), VNS is faster than ABC. When the instance size increases, ABC becomes faster with an average difference of 2915.51 s, which can go up to 5.07 h (for benchmark (200 × 20). Moreover, for the set of instances 500 × 20, we were enabled to run the VNS algorithm. For small instances, the CPU time difference can be explained by the fact that VNS applies its research process on a single solution at each iteration, while ABC manipulates a whole population, which makes it slower. For larger instances, exhaustive neighbourhood exploration during the VNS research process makes it slower, because the neighbourhood size explodes. As a consequence, when considering the scalability of both methods, we can confirm that the proposed ABC algorithm is more practical.

Table 3. CPU for ABC and VNS algorithms (in seconds).

Instance	CPU ABC	CPU VNS
20 × 5	15.94	**0.85**
20 × 10	22.47	**1.97**
20 × 20	34.38	**5.54**
50 × 5	29.56	**19.32**
50 × 10	**50.32**	55.93
50 × 20	**72.73**	198.85
100 × 5	**53.2**	178.05
100 × 10	**103.36**	569.9
100 × 20	**166.37**	2375.4
200 × 10	**234.4**	10,116
200 × 20	**262.61**	20,808
500 × 20	**1761.34**	-
Average	**233.89**	3149.4

6. Conclusions

This work deals with the integrated scheduling problem of production and predictive maintenance in permutation flowshops. This problem is characterized by flexible maintenance activities scheduled based on PHM outputs. Metaheuristic approaches seem to be the best and most practical choice to solve such an NP-hard problem. Among a large number of swarm intelligence-inspired methods, artificial bee colony (ABC), simulating the foraging behaviour of honey bees, has been successfully applied to different variants of scheduling problems. Hence, in this study, we develop an ABC algorithm with an efficient search process, to minimize the total completion time (Makespan) of the schedule. The main idea is to design adapted search procedures with appropriate structures to balance the exploration and exploitation of the search. A statistical analysis was used to accurately calibrate the ABC parameters. Comprehensive experiments were carried out to assess the performance of the proposed ABC compared to a VNS algorithm from the literature, dealing with the same problem. Results analysis shows that the proposed ABC is more efficient, providing better solutions in terms of Makespan minimization, in a shorter computational time.

While the proposed ABC algorithm has demonstrated good results, some improvements can be considered. One potential enhancement axis involves the integration of a more robust selection scheme of the neighborhood structures by the employed bees. Moreover, the hybridization of ABC with other local search methods allows them to leverage their complementary strengths. Considering the average earliness/tardiness of maintenance interventions, the proposed ABC seems to be less performant. Indeed, this criterion was not tackled in this paper. Hence, a promising perspective would be to develop a multi-objective ABC algorithm where both production and maintenance criteria are simultaneously optimized.

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

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All data and code supporting the findings of this study are available from the corresponding author M. Benbouzid on request.

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

Abbreviations

The following abbreviations are used in this manuscript:

ABC	Artificial bee colony
CPU	Central processing unit
ET	Earliness/tardiness
GA	Genetic algorithm
MILP	Mixed-integer linear programming
MSPP	Machine scheduling problem in production
PFSP	Permutation flowshop scheduling problem
PHM	Prognostics and health management
PM	Preventive maintenance
PPD	Post prognostic decision
RPD	Relative percentage deviation
RUL	Remaining useful life
VNS	Variable neighborhood search

References

1. Abedinnia, H.; Glock, C.H.; Grosse, E.H.; Schneider, M. Machine scheduling problems in production: A tertiary study. *Comput. Ind. Eng.* **2017**, *111*, 403–416. [CrossRef]
2. Sanlaville, E.; Schmidt, G. Machine scheduling with availability constraints. *Acta Inform.* **1998**, *35*, 795–811. [CrossRef]

3. Schmidt, G. Scheduling with limited machine availability. *Eur. J. Oper. Res.* **2000**, *121*, 1–15. [CrossRef]
4. Ma, Y.; Chu, C.; Zuo, C. A survey of scheduling with deterministic machine availability constraints. *Comput. Ind. Eng.* **2010**, *58*, 199–211. [CrossRef]
5. Huo, Y.; Zhao, H. Two machine scheduling subject to arbitrary machine availability constraint. *Omega* **2018**, *76*, 128–136. [CrossRef]
6. Lee, C.Y. Two-machine flowshop scheduling with availability constraints. *Eur. J. Oper. Res.* **1999**, *114*, 420–429. [CrossRef]
7. Cassady, C.R.; Kutanoglu, E. Integrating preventive maintenance planning and production scheduling for a single machine. *IEEE Trans. Reliab.* **2005**, *54*, 304–309. [CrossRef]
8. Ruiz, R.; García-Díaz, J.C.; Maroto, C. Considering scheduling and preventive maintenance in the flowshop sequencing problem. *Comput. Oper. Res.* **2007**, *34*, 3314–3330. [CrossRef]
9. Wang, S.; Liu, M. Two-machine flow shop scheduling integrated with preventive maintenance planning. *Int. J. Syst. Sci.* **2016**, *47*, 672–690. [CrossRef]
10. Xiao, L.; Song, S.; Chen, X.; Coit, D.W. Joint optimization of production scheduling and machine group preventive maintenance. *Reliab. Eng. Syst. Saf.* **2016**, *146*, 68–78. [CrossRef]
11. Seif, J.; Yu, A.J.; Rahmanniyay, F. Modelling and optimization of a bi-objective flow shop scheduling with diverse maintenance requirements. *Int. J. Prod. Res.* **2018**, *56*, 3204–3225. [CrossRef]
12. Chen, J.S. Scheduling of nonresumable jobs and flexible maintenance activities on a single machine to minimize makespan. *Eur. J. Oper. Res.* **2008**, *190*, 90–102. [CrossRef]
13. Benbouzid-Si Tayeb, 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.
14. Bock, S.; Briskorn, D.; Horbach, A. Scheduling flexible maintenance activities subject to job-dependent machine deterioration. *J. Sched.* **2012**, *15*, 565–578. [CrossRef]
15. Vahedi-Nouri, B.; Fattahi, P.; Tavakkoli-Moghaddam, R.; Ramezanian, R. A general flow shop scheduling problem with consideration of position-based learning effect and multiple availability constraints. *Int. J. Adv. Manuf. Technol.* **2014**, *73*, 601–611. [CrossRef]
16. Benbouzid-Si Tayeb, F.; Belkaaloul, W. Towards an artificial immune system for scheduling jobs and preventive maintenance operations in flowshop problems. In Proceedings of the 2014 IEEE 23rd International Symposium on Industrial Electronics (ISIE), Istanbul, Turkey, 1–4 June 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 1065–1070.
17. Khatami, M.; Zegordi, S.H. Coordinative production and maintenance scheduling problem with flexible maintenance time intervals. *J. Intell. Manuf.* **2017**, *28*, 857–867. [CrossRef]
18. Detti, P.; Nicosia, G.; Pacifici, A.; de Lara, G.Z.M. Robust single machine scheduling with a flexible maintenance activity. *Comput. Oper. Res.* **2019**, *107*, 19–31. [CrossRef]
19. Wang, H.; Yan, Q.; Zhang, S. Integrated scheduling and flexible maintenance in deteriorating multi-state single machine system using a reinforcement learning approach. *Adv. Eng. Inform.* **2021**, *49*, 101339. [CrossRef]
20. Costa, A.; Fernandez-Viagas, V. A modified harmony search for the T-single machine scheduling problem with variable and flexible maintenance. *Expert Syst. Appl.* **2022**, *198*, 116897. [CrossRef]
21. Yan, Q.; Wu, W.; Wang, H. Deep reinforcement learning for distributed flow shop scheduling with flexible maintenance. *Machines* **2022**, *10*, 210. [CrossRef]
22. Penz, L.; Dauzère-Pérès, S.; Nattaf, M. Minimizing the sum of completion times on a single machine with health index and flexible maintenance operations. *Comput. Oper. Res.* **2023**, *151*, 106092. [CrossRef]
23. Jia, Y.; Yan, Q.; Wang, H. Q-learning driven multi-population memetic algorithm for distributed three-stage assembly hybrid flow shop scheduling with flexible preventive maintenance. *Expert Syst. Appl.* **2023**, *232*, 120837. [CrossRef]
24. Atamuradov, V.; Medjaher, K.; Dersin, P.; Lamoureux, B.; Zerhouni, N. Prognostics and health management for maintenance practitioners-Review, implementation and tools evaluation. *Int. J. Progn. Health Manag.* **2017**, *8*, 1–31. [CrossRef]
25. Iyer, N.; Goebel, K.; Bonissone, P. Framework for post-prognostic decision support. In Proceedings of the 2006 IEEE Aerospace Conference, Big Sky, MT, USA, 4–11 March 2006; IEEE: Piscataway, NJ, USA, 2006; p. 10.
26. Kubiak, W.; Błażewicz, J.; Formanowicz, P.; Breit, J.; Schmidt, G. Two-machine flow shops with limited machine availability. *Eur. J. Oper. Res.* **2002**, *136*, 528–540. [CrossRef]
27. Karaboga, D.; Basturk, B. A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm. *J. Glob. Optim.* **2007**, *39*, 459–471. [CrossRef]
28. Karaboga, D.; Akay, B. A modified artificial bee colony (ABC) algorithm for constrained optimization problems. *Appl. Soft Comput.* **2011**, *11*, 3021–3031. [CrossRef]
29. Aslan, S.; Badem, H.; Karaboga, D. Improved quick artificial bee colony (iqABC) algorithm for global optimization. *Soft Comput.* **2019**, *23*, 13161–13182. [CrossRef]
30. Kaya, E.; Gorkemli, B.; Akay, B.; Karaboga, D. A review on the studies employing artificial bee colony algorithm to solve combinatorial optimization problems. *Eng. Appl. Artif. Intell.* **2022**, *115*, 105311. [CrossRef]
31. Lei, D.; Cai, J. Multi-population meta-heuristics for production scheduling: A survey. *Swarm Evol. Comput.* **2020**, *58*, 100739. [CrossRef]

32. Framinan, J.M.; Gupta, J.N.; Leisten, R. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *J. Oper. Res. Soc.* **2004**, *55*, 1243–1255. [CrossRef]
33. Ruiz, R.; Maroto, C. A comprehensive review and evaluation of permutation flowshop heuristics. *Eur. J. Oper. Res.* **2005**, *165*, 479–494. [CrossRef]
34. Pan, Q.K.; Ruiz, R. A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. *Comput. Oper. Res.* **2013**, *40*, 117–128. [CrossRef]
35. Yenisey, M.M.; Yagmahan, B. Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends. *Omega* **2014**, *45*, 119–135. [CrossRef]
36. Fernandez-Viagas, V.; Ruiz, R.; Framinan, J.M. A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation. *Eur. J. Oper. Res.* **2017**, *257*, 707–721. [CrossRef]
37. Zaied, A.N.H.; Ismail, M.M.; Mohamed, S.S. Permutation flow shop scheduling problem with makespan criterion: Literature review. *J. Theor. Appl. Inf. Technol* **2021**, *99*, 830–848.
38. 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]
39. Varnier, C.; Zerhouni, N. Scheduling predictive maintenance in flow-shop. In Proceedings of the IEEE 2012 Prognostics and System Health Management Conference (PHM-2012 Beijing), Beijing, China, 23–25 May 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 1–6.
40. Ecoretti, A.; Ceschia, S.; Schaerf, A. Local Search for Integrated Predictive Maintenance and Scheduling in Flow-Shop. In Proceedings of the Metaheuristics International Conference, Syracuse, Italy, 11–14 July 2022; Springer: Cham, Switzerland, 2022; pp. 260–273.
41. Ladj, A.; Benbouzid-Si Tayeb, F.; Varnier, C. Tailored genetic algorithm for scheduling jobs and predictive maintenance in a permutation flowshop. In Proceedings of the 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), Turin, Italy, 4–7 September 2018; IEEE: Piscataway, NJ, USA, 2018; Volume 1, pp. 524–531.
42. 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]
43. 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]
44. Ladj, A.; Benbouzid-Si Tayeb, F.; 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]
45. Ladj, A.; Benbouzid-Si Tayeb, F.; 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]
46. 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. [CrossRef]
47. Azab, E.; Nafea, M.; Shihata, L.A.; Mashaly, M. A machine-learning-assisted simulation approach for incorporating predictive maintenance in dynamic flow-shop scheduling. *Appl. Sci.* **2021**, *11*, 11725. [CrossRef]
48. Kan, A.R. *Machine Scheduling Problems: Classification, Complexity and Computations*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2012.
49. Pinedo, M.L. *Scheduling*; Springer: Berlin/Heidelberg, Germany, 2012; Volume 29.
50. Gogna, A.; Tayal, A. Metaheuristics: Review and application. *J. Exp. Theor. Artif. Intell.* **2013**, *25*, 503–526. [CrossRef]
51. Rabadi, G. *Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling*; Springer: Berlin/Heidelberg, Germany, 2016; Volume 236.
52. Cuevas, E.; Fausto, F.; González, A.; Cuevas, E.; Fausto, F.; González, A. An introduction to nature-inspired metaheuristics and swarm methods. In *New Advancements in Swarm Algorithms: Operators and Applications*; Springer: Cham, Switzerland, 2020; pp. 1–41.
53. Dragoi, E.N.; Dafinescu, V. Review of metaheuristics inspired from the animal kingdom. *Mathematics* **2021**, *9*, 2335. [CrossRef]
54. Peres, F.; Castelli, M. Combinatorial optimization problems and metaheuristics: Review, challenges, design, and development. *Appl. Sci.* **2021**, *11*, 6449. [CrossRef]
55. Darvishpoor, S.; Darvishpour, A.; Escarcega, M.; Hassanalian, M. Nature-Inspired Algorithms from Oceans to Space: A Comprehensive Review of Heuristic and Meta-Heuristic Optimization Algorithms and Their Potential Applications in Drones. *Drones* **2023**, *7*, 427. [CrossRef]
56. Godinho Filho, M.; Barco, C.F.; Tavares Neto, R.F. Using Genetic Algorithms to solve scheduling problems on flexible manufacturing systems (FMS): A literature survey, classification and analysis. *Flex. Serv. Manuf. J.* **2014**, *26*, 408–431. [CrossRef]
57. Liu, S.; Wang, P.; Zhang, J. An improved biogeography-based optimization algorithm for blocking flow shop scheduling problem. *Chin. J. Electron.* **2018**, *27*, 351–358. [CrossRef]
58. 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]
59. Neto, R.T.; Godinho Filho, M. Literature review regarding Ant Colony Optimization applied to scheduling problems: Guidelines for implementation and directions for future research. *Eng. Appl. Artif. Intell.* **2013**, *26*, 150–161. [CrossRef]

60. Shao, Z.; Pi, D.; Shao, W. Hybrid enhanced discrete fruit fly optimization algorithm for scheduling blocking flow-shop in distributed environment. *Expert Syst. Appl.* **2020**, *145*, 113147. [CrossRef]
61. Marichelvam, M.; Prabaharan, T.; Yang, X.S.; Geetha, M. Solving hybrid flow shop scheduling problems using bat algorithm. *Int. J. Logist. Econ. Glob.* **2013**, *5*, 15–29. [CrossRef]
62. Jiang, T.; Zhang, C. Application of grey wolf optimization for solving combinatorial problems: Job shop and flexible job shop scheduling cases. *IEEE Access* **2018**, *6*, 26231–26240. [CrossRef]
63. Zhang, B.; Pan, Q.K.; Gao, L.; Zhang, X.L.; Sang, H.Y.; Li, J.Q. An effective modified migrating birds optimization for hybrid flowshop scheduling problem with lot streaming. *Appl. Soft Comput.* **2017**, *52*, 14–27. [CrossRef]
64. Li, J.Q.; Pan, Q.K. Chemical-reaction optimization for flexible job-shop scheduling problems with maintenance activity. *Appl. Soft Comput.* **2012**, *12*, 2896–2912. [CrossRef]
65. Zahmani, M.H. An adaptation of the galaxy-based search algorithm for solving the single machine total weighted tardiness problem. *Int. J. Manuf. Res.* **2021**, *16*, 399–413. [CrossRef]
66. Jeet, K.; Dhir, R.; Singh, P. Hybrid black hole algorithm for bi-criteria job scheduling on parallel machines. *Int. J. Intell. Syst. Appl.* **2016**, *8*, 1–17. [CrossRef]
67. Hart, E.; Ross, P.; Corne, D. Evolutionary scheduling: A review. *Genet. Program. Evolvable Mach.* **2005**, *6*, 191–220. [CrossRef]
68. Branke, J.; Nguyen, S.; Pickardt, C.W.; Zhang, M. Automated design of production scheduling heuristics: A review. *IEEE Trans. Evol. Comput.* **2015**, *20*, 110–124. [CrossRef]
69. Karaboga, D.; Basturk, B. Artificial bee colony (ABC) optimization algorithm for solving constrained optimization problems. In Proceedings of the International Fuzzy Systems Association World Congress, Cancun, Mexico, 18–21 June 2007; Springer: Berlin/Heidelberg, Germany, 2007; pp. 789–798.
70. Karaboga, D.; Basturk, B. On the performance of artificial bee colony (ABC) algorithm. *Appl. Soft Comput.* **2008**, *8*, 687–697. [CrossRef]
71. Brajevic, I.; Tuba, M. An upgraded artificial bee colony (ABC) algorithm for constrained optimization problems. *J. Intell. Manuf.* **2013**, *24*, 729–740. [CrossRef]
72. Pan, Q.K.; Tasgetiren, M.F.; Suganthan, P.N.; Chua, T.J. A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. *Inf. Sci.* **2011**, *181*, 2455–2468. [CrossRef]
73. 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]
74. Pan, Q.K.; Wang, L.; Li, J.Q.; Duan, J.H. A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimisation. *Omega* **2014**, *45*, 42–56. [CrossRef]
75. Li, H.; Li, X.; Gao, L. A discrete artificial bee colony algorithm for the distributed heterogeneous no-wait flowshop scheduling problem. *Appl. Soft Comput.* **2021**, *100*, 106946. [CrossRef]
76. Zuo, Y.; Fan, Z.; Zou, T.; Wang, P. A novel multi-population artificial bee colony algorithm for energy-efficient hybrid flow shop scheduling problem. *Symmetry* **2021**, *13*, 2421. [CrossRef]
77. Zuo, Y.; Wang, P.; Li, M. A Population Diversity-Based Artificial Bee Colony Algorithm for Assembly Hybrid Flow Shop Scheduling with Energy Consumption. *Appl. Sci.* **2023**, *13*, 10903. [CrossRef]
78. Li, H.; Gao, K.; Duan, P.Y.; Li, J.Q.; Zhang, L. An improved artificial bee colony algorithm with q-learning for solving permutation flow-shop scheduling problems. *IEEE Trans. Syst. Man Cybern. Syst.* **2022**, *53*, 2684–2693. [CrossRef]
79. Gong, D.; Han, Y.; Sun, J. A novel hybrid multi-objective artificial bee colony algorithm for blocking lot-streaming flow shop scheduling problems. *Knowl.-Based Syst.* **2018**, *148*, 115–130. [CrossRef]
80. Baysal, M.E.; Sarucan, A.; Büyüközkan, K.; Engin, O. Artificial bee colony algorithm for solving multi-objective distributed fuzzy permutation flow shop problem. *J. Intell. Fuzzy Syst.* **2022**, *42*, 439–449. [CrossRef]
81. Nawaz, M.; Enscore Jr, E.E.; Ham, I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega* **1983**, *11*, 91–95. [CrossRef]
82. Ruiz, R.; Maroto, C.; Alcaraz, J. Two new robust genetic algorithms for the flowshop scheduling problem. *Omega* **2006**, *34*, 461–476. [CrossRef]
83. Taillard, E. Benchmarks for basic scheduling problems. *Eur. J. Oper. Res.* **1993**, *64*, 278–285. [CrossRef]