energies

MDPI

Article

Solving Scheduling Problem in a Distributed Manufacturing System Using a Discrete Fruit Fly Optimization Algorithm

Xiaohui Zhang [1], Xinhua Liu [1], Shufeng Tang [2,*], Grzegorz Królczyk [3] and Zhixiong Li [4,5,*]

1 School of Mechanical and Electrical Engineering, China University of Mining & Technology, Xuzhou 221116, China
2 School of Mechanical Engineering, Inner Mongolia University of Technology, Hohhot 010051, China
3 Department of Manufacturing Engineering and Automation Products, Opole University of Technology, 45758 Opole, Poland
4 Suzhou Automotive Research Institute, Tsinghua University, Suzhou 215134, China
5 School of Mechanical, Materials, Mechatronic and Biomedical Engineering, University of Wollongong, NSW 2522, Australia
* Correspondence: tangshufeng@imut.edu.cn (S.T.); zhixiong_li@uow.edu.au (Z.L.); Tel.: +61-405-840751 (Z.L.); Fax: +61-703025 (Z.L.)

Received: 20 July 2019; Accepted: 22 August 2019; Published: 23 August 2019

Abstract: This study attempts to optimize the scheduling decision to save production cost (e.g., energy consumption) in a distributed manufacturing environment that comprises multiple distributed factories and where each factory has one flow shop with blocking constraints. A new scheduling optimization model is developed based on a discrete fruit fly optimization algorithm (DFOA). In this new evolutionary optimization method, three heuristic methods were proposed to initialize the DFOA model with good quality and diversity. In the smell-based search phase of DFOA, four neighborhood structures according to factory reassignment and job sequencing adjustment were designed to help explore a larger solution space. Furthermore, two local search methods were incorporated into the framework of variable neighborhood descent (VND) to enhance exploitation. In the vision-based search phase, an effective update criterion was developed. Hence, the proposed DFOA has a large probability to find an optimal solution to the scheduling optimization problem. Experimental validation was performed to evaluate the effectiveness of the proposed initialization schemes, neighborhood strategy, and local search methods. Additionally, the proposed DFOA was compared with well-known heuristics and metaheuristics on small-scale and large-scale test instances. The analysis results demonstrate that the search and optimization ability of the proposed DFOA is superior to well-known algorithms on precision and convergence.

Keywords: energy saving and efficiency; distributed manufacturing system; blocking constraint; distributed flow shop scheduling; fruit fly optimization algorithm

1. Introduction

The well-known blocking flowshop scheduling problem (BFSP) [1] has gained sustained attention since it better reflects the real-life characteristics in most manufacturing systems [2]. Under a BFSP environment, a job, having completed its operation, is expected to enter into the next machine for processing immediately. When the condition is not met, i.e., the next machine is occupied, the job must stay on the machine and block itself until the next machine is available. According to the three-field notation introduced by Graham [3], BFSP with makespan criterion under study can be denoted as $F_m|\text{blocking}|C_{\max}$.

In the above discussion, there always exists a universal hypothesis, as follows: The BFSP model is established on one production workshop, center, or factory, and all jobs are assumed to be processed in the same factory. In real life, however, the emergence of concurrent or large-scale production makes the pattern of distributed manufacturing necessary. This environment enables manufacturers to dispatch the general task among independent production units, with the view of raising productivity, lowering management risks, and reducing the manufacturing cost. We denote BFSP in a distributed environment as DBFSP (distributed blocking flowshop scheduling problem) throughout this paper. DBFSP contains multiple identical flow shops with a blocking constraint. That is, the machine configurations are the same in each flow shop. The processing time of one specific job on one fixed machine is the same as that on identical machines of other shops. The jobs can be assigned to any flow shops before the sequence of jobs in each shop are determined. Every single shop in such a distributed environment follows the character of a typical blocking flow shop. Figure 1 demonstrates an example of a Gantt chart for DBFSP with two factories.

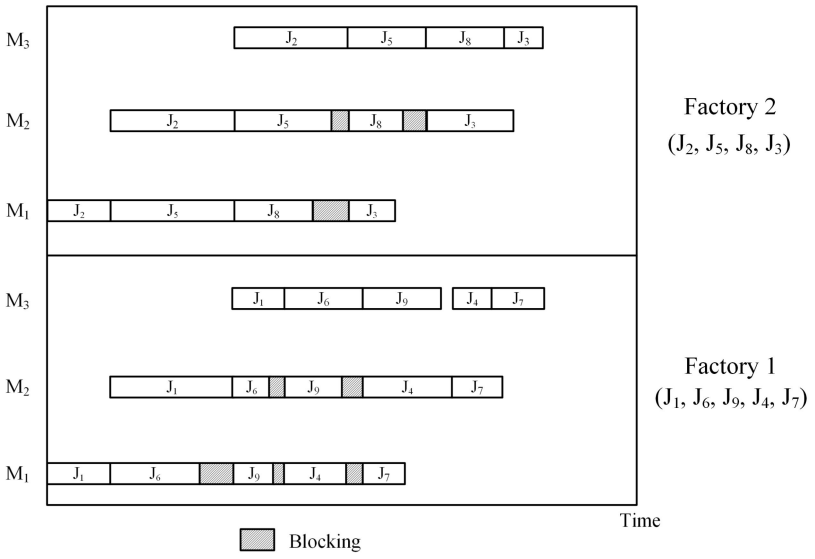

Figure 1. Example of a Gantt chart for DBFSP with two factories.

Compared with conventional BFSP, scheduling in a distributed system is more complicated. In a single factory, jobs only need to be sequenced on a set of machines, whereas for DBFSP, an additional decision is especially required to determine the assignment of jobs to each factory. Apparently, both problems are highly related and cannot be solved without considering the sequential characteristics. Since $F_m|\text{blocking}|C_{\max}$ has been verified to be NP-hard, even only for two machines [4], DBFSP is also NP-hard. Consequently, the traditional exact algorithms are not effective and applicable. For solving such complex and comprehensive scheduling problems, there is an urgent need for an effective yet simple algorithm with efficiency.

Bearing the above observations, this paper aims to tackle DBFSP with makespan criterion using a novel and effective discrete fruit fly optimization algorithm (DFOA). The proposed algorithm makes the following contributions:

(1) An initialization strategy based on problem-specific characteristics is proposed to generate an initial population with good quality and diversity.
(2) In the smell-based search phase of DFOA, a novel neighborhood strategy is designed with the view of extending the exploration.
(3) To enhance the exploitation of DFOA, an adaptive VND-based local search strategy is highlighted.
(4) In the vision-based search phase, an effective elite-based update criterion is proposed, which helps DFOA converge faster.

The rest of the components of this paper are outlined as follows. Section 2 systematically reviews the relevant literature. Section 3 states the DBFSP. The details of DFOA are presented in Section 4. Section 5 provides the numerical experiments and statistical analysis. The conclusions and future work are summarized in Section 6.

2. Related Work

To the best of our knowledge, few literature investigations have studied DBFSP. Thus, recent publications relevant to this paper are mainly concerned with the following three research streams: BFSP, distributed flowshop scheduling problem (DFSP), and fruit fly algorithm (FOA). In this section, the correlative literature is concluded.

2.1. Blocking Flowshop Scheduling Problem (BFSP)

BFSP is encountered in a rather wide range of industrial sectors, such as iron [5], chemical and gear industries [6], just-in-time production lines and in-line robotic cells [7], and computing resource management [8]. The optimal solution for BFSP can be harvested through exact algorithms, such as integer programming or brunch and bound. However, they are limited to small instances due to computational complexity. Therefore, recent research has focused on heuristics (including constructive and improvements heuristics) or metaheuristics. Heuristics completes the solution on a partial sequence or makes improvement according to the problem-specific characteristics, while metaheuristics is a combination of stochastic mechanism and local search algorithms [9]. Heuristics terminates spontaneously after a given number of steps regardless of the time limit [10], yet metaheuristics takes certain termination criterion as inputs. Moreover, metaheuristics usually needs fast initial solutions, of which the quality is known to affect the performance of the metaheuristics, to begin the search process. Hence, the initial solutions for the metaheuristics are often provided by heuristics.

In 2001, Caraffa et al. [11] proposed a genetic algorithm to solve BFSP with a makespan criterion. In 2007, Grabowski and Pempera [12] used a Tabu search (TS) algorithm to tackle the same problem. The experimental results demonstrated that TS was superior to GA. After that, Wang et al. [13] presented a hybrid discrete differential evolution algorithm (HDDE) for BFSP. The HDDE was proven to be more effective and efficient than algorithms in previous literature. Later, Ribas et al. [14] presented a simple yet effective iterated greedy (IG) algorithm combining with the NEH method [15]. In 2012, a discrete particle swarm algorithm (DPSO) [16] and an improved discrete artificial bee colony (IABC) algorithm [17] were proposed. Han et al. [18] further proposed a discrete artificial bee colony (DABC) incorporating a differential evolution (DE) strategy in 2015. In 2016, Han et al. [19] applied a novel FOA to solve BFSP and the results reported that 67 out of 90 new upper bounds of the Taillard benchmark [20] were improved. Recently, Shao et al. [21] proposed a novel discrete invasive weed optimization for BFSP.

2.2. Distributed Flowshop Scheduling Problem (DFSP)

DFSP is more complicated than the traditional FSP since there is an additional decision for job-to-factory assignment in the solution space. This problem-specific characteristic also requires a counter-change on the neighborhood search strategy and the optimization procedure. Naderi et al. [22] addressed DFSP for the first time in 2010. The authors developed six alternative mixed integer linear programming (MILP) models and 12 dispatching-based heuristics with two effective job-to-factory assignment rules. These two rules are described as follows:

(1) Assign job j to the factory with the lowest current makespan C_{max} (not including job j).
(2) Assign job j to the factory that completes it at the earliest time, i.e., the factory resulting in the lowest makespan C_{max} (after assigning job j).

Moreover, two iterative methods based on variable neighborhood descent (VND) [23] are proposed to search for better neighborhoods of the solution. Following this pioneering work, several approaches were surveyed and applied for DFSP, such as electromagnetism-like mechanism algorithm [24], hybrid genetic algorithm [25], Tabu search algorithm [26], estimation of distribution algorithm [27], scatter search algorithm [28], immune algorithm [29], chemical reaction optimization algorithm [30], and iterated greedy algorithm [31,32]. It is worth noting that Fernandez et al. [33] investigated DFSP with the total flow time (TFT) criterion for the first time in 2015. The authors also presented six job-to-factory assignment rules that were highly related to TFT and tested the performances of the rules in the experiments.

DBFSP is developed from DFSP by extending the permutation constraint to blocking. Heretofore, few literatures have focused on this field. Zhang et al. [34] designed a discrete differential evolution algorithm in 2018. Except for the DBFSP, Shao et al. [35] optimized the distributed no-wait flowshop scheduling problem (DNWFSP) with an improved IG algorithm for the first time.

2.3. Fruit Fly Algorithm (FOA)

FOA is a novel nature-inspired evolutionary algorithm proposed by Pan [36]. It is inspired by the foraging behavior of fruit fly swarms using sensitive osphresis and version. A fruit fly has two important organs, the olfactory and visual organs. The olfactory organ is used for smelling all types of odor in the air so that the fruit fly can fly towards the target locations. Afterward, it flies towards the food source with the help of the visual organ. The foraging behavior process of a fruit fly swarm is shown in Figure 2.

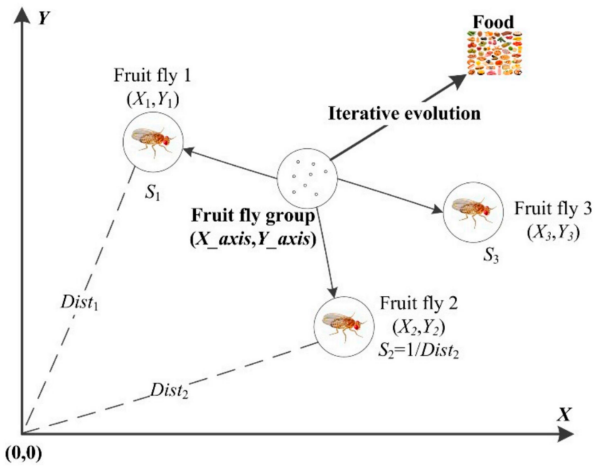

Figure 2. Graphical presentation of foraging behavior of a fruit fly swarm.

According to the algorithm structure, FOA has two search phases, as follows: The smell-based search phase and the vision-based search phase. This parallel search framework enables the researchers to embed a number of heuristics, local search strategies, and solution generation operators in it, so that the exploration and exploitation of the framework can be enhanced. Compared with other metaheuristics, FOA is simple and has few parameters to implement. The experiment results from many literature examples have reported that FOA is competitive and appropriate for optimization problems. The searching procedure of FOA is outlined in the following steps:

Step 1. Initialization of parameters: Set the population size and the number of generations.
Step 2. Initialization of the fruit fly population with location.
Step 3. Smell-based search phase: The fruit fly exploits N locations (i.e., food sources) randomly. Evaluate the N locations with the smell concentration values as fitness values.
Step 4. Vision-based search phase: Replace the current best population location when a better location is found. The population flies towards the new best location.

Step 5. Termination criterion: End the procedure if the maximum generation number is reached; otherwise, back to Step 3. The detailed search procedure of FOA refers to Reference [36].

Heretofore, FOA has been successfully applied to a variety of fields, including antenna arrays synthesis [37], traffic flow forecasting [38], web auction logistics [39], and multidimensional knapsack problems [40]. For scheduling problems, FOA was modified by Zheng et al. [41] to solve the semiconductor final testing scheduling problem, which can be abstracted as a flexible job shop scheduling problem (FJSP) with setup time and resource allocation constraints. Later, Zheng et al. [42] adopted a modified FOA to address the FJSP with dual resource constraints. The authors embedded the knowledge system into FOA so that the fruit fly can be guided towards the food source quickly in the vision-based search phase. Li et al. [43] proposed a hybrid FOA with an adaptive neighborhood strategy for scheduling problems in a steelmaking factory, where machine breakdown and disruptions were considered in real life.

2.4. Discussion

Although various approaches were developed to tackle the scheduling problems in distributed environments, there are the following observations in the considered fields. First, a majority of current literature focuses on DFSP and the variants of DFSP have not been fully investigated. Second, the increasing number of jobs will bring a huge obstacle when scheduling in a complex distributed environment. When solving DBFSP, it is better to apply a simple and effective algorithm with few parameters and low mathematical requirements.

With the above motivations, this paper presents a novel DFOA to solve DBFSP in a discrete manner. To the best of our knowledge, FOA has not been applied to DBFSP. In the proposed DFOA framework, a central location for the fruit fly population is generated in the initialization phase firstly. In the smell-based search phase, it is expected that the individual fruit fly can search in a random direction (neighborhood structure) around the center location in the decision space. When a fruit fly is getting close to a new location, a local search strategy is emphasized to guide it towards the best location. In the vision-based search phase, the aim of the algorithm is to guide the population to a better searching space quickly.

3. Problem Statement

In this section, we state DBFSP on the basis of BFSP. The derivation procedure of makespan is discussed.

3.1. Problem Description of BFSP and DBFSP

BFSP is described as follows. There is a job set $J = \left\{J_j|1,2,\ldots,n\right\}$ consisting of n jobs and a machine set $M = \{M_i|1,2,\ldots,m\}$ with m different machines. Each job J_n will be sequentially processed on the machines $1, 2, \ldots, m$. The aim of solving BFSP is to minimize the makespan with a processing sequence. There are the following assumptions when solving BFSP:

(1) The orders where all jobs to be processed is the same on each machine.
(2) If m_i is occupied, the job needs to be blocked on the current machine until m_i is available.
(3) The job cannot be interrupted once it starts operation.
(4) Each machine can only handle one job at a time.
(5) All jobs and machines are available at time zero.
(6) The setup time of jobs is included in the processing time.

Based on BFSP, DBFSP considers processing n jobs by using a factory set $F = \{F_k|k = 1,2,\ldots,f\}$, where $f \geq 2$. It is worth noting that all the factories in this study have the same machine configuration and environment. Jobs are not allowed to be removed to any other factories once they are processed in the assigned factory. A complete solution of DBFSP includes two correlative decisions, assigning jobs

to factories and sequencing jobs in each factory. Let $C_{\max}$ be the makespan in any of the factories. The aim of solving DBFSP in this paper is to minimize $C_{\max}$.

3.2. Mathematical Model of DBFSP

Let $\pi_k = \left\{\pi_{k(1)}, \pi_{k(2)}, \ldots, \pi_{k(n_k)}\right\}$ represent the partial sequence of n_k jobs that are assigned to factory F_k. Hence, a complete solution of DBFSP with F factories and n jobs can be presented as a set of partial sequences of all factories, i.e., $\pi = \left\{\pi_k \middle| (k = 1, 2, \ldots, F)\right\}$. Assume that $d_{\pi_k(l),k}$ represents the departure time of operation $O_{\pi_k(l),i}$ on the machine i with the corresponding processing time $p_{\pi_k(l),i}$. The value $S_{\pi_k(l),0}$ represents the start time of job $\pi_k(l)$ on the first machine. In factory F_k, $d_{\pi_k(l),i}$ can be deduced with the following recursive formulas:

$$S_{\pi_k(1),0} = 0, \tag{1}$$

$$d_{\pi_k(1),i} = d_{\pi_k(1),i-1} + p_{\pi_k(1),i}, \quad i = 2, 3, \ldots, m, \tag{2}$$

$$S_{\pi_k(l),0} = d_{\pi_k(l-1),1}, \quad i = 2, \ldots, n_k, \tag{3}$$

$$d_{\pi_k(l),i} = \max\left\{d_{\pi_k(l),i-1} + p_{\pi_k(l),i}, \quad d_{\pi_k(l-1),i+1}\right\} \quad l = 2, 3, \ldots, n_k \quad i = 1, 2, \ldots, m-1, \tag{4}$$

$$d_{\pi_k(l),m} = d_{\pi_k(l),m-1} + p_{\pi_k(l),m}, \quad l = 2, 3, \ldots, n_k, \tag{5}$$

where Equations (1) and (2) calculate the start and departure time of the first job $\pi_k(1)$ from machine 1 to machine m. Equations (3) and (4) represent the start and departure time of job $\pi_k(l)$ from machine 1 to machine $m-1$. Equation (5) gives the departure time of job $\pi_k(l)$ on the last machine m. By comparing the makespan of each factory F_k, the objective function for DBFSP can be expressed as follows:

$$C_{\max} = \max_{k=1}^{f} d_{\pi_k(n_k),m}. \tag{6}$$

To clearly illustrate the deduction procedure, a makespan derivation procedure of a certain factory F_k is provided as an example.

Example 1. *Assume that n (n > 4) jobs need to be processed. Firstly, they are assigned to F factories, each of which contains 3 machines. A job sequence* $\pi_k = \{1, 2, 3, 4\}$ *(*$k \in \{1, 2, \ldots F\}$*) is given to factory* F_k*. The processing time* $p_{\pi_k(l),i}$ *of each job on each machine is shown as follows:*

$$p_{\pi_k(l),i} = \begin{bmatrix} p_{\pi_k(1),1} & p_{\pi_k(1),2} & p_{\pi_k(1),3} \\ p_{\pi_k(2),1} & p_{\pi_k(2),2} & p_{\pi_k(2),3} \\ p_{\pi_k(3),1} & p_{\pi_k(3),2} & p_{\pi_k(3),3} \\ p_{\pi_k(4),1} & p_{\pi_k(4),2} & p_{\pi_k(4),3} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 3 \\ 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 1 & 3 \end{bmatrix}. \tag{7}$$

The departure time $d_{\pi_k(l),k}$ *is deduced as follows:*

$S_{\pi_k(1),0} = 0$
$d_{\pi_k(1),1} = p_{\pi_k(1),1} = 2$
$d_{\pi_k(1),2} = d_{\pi_k(1),1} + p_{\pi_k(1),2} = 2+3 = 5$
$d_{\pi_k(1),3} = p_{\pi_k(1),2} + p_{\pi_k(1),3} = 5 + 3 = 8$
$S_{\pi_k(2),0} = S_{\pi_k(1),1} = 2$
$d_{\pi_k(2),1} = \max\left\{S_{\pi_k(2),0} + p_{\pi_k(2),1}, d_{\pi_k(1),2}\right\} = \max\{3, 5\} = 5$
$d_{\pi_k(2),2} = \max\left\{d_{\pi_k(2),1} + p_{\pi_k(2),2}, d_{\pi_k(1),3}\right\} = \max\{7, 8\} = 8$
$d_{\pi_k(2),3} = d_{\pi_k(2),2} + p_{\pi_k(2),3} = 8 + 3 = 11$
$S_{\pi_k(3),0} = d_{\pi_k(2),1} = 5$
$d_{\pi_k(3),1} = \max\left\{S_{\pi_k(3),0} + p_{\pi_k(3),1}, d_{\pi_k(2),2}\right\} = \max\{8, 8\} = 8$

$$d_{\pi_k(3),2} = \max\left\{d_{\pi_k(3),1} + p_{\pi_k(3),2}, d_{\pi_k(2),3}\right\} = \max\{9, 11\} = 11$$
$$d_{\pi_k(3),3} = d_{\pi_k(3),2} + p_{\pi_k(3),3} = 11 + 2 = 13$$
$$S_{\pi_k(4),0} = d_{\pi_k(3),1} = 8$$
$$d_{\pi_k(4),1} = \max\left\{S_{\pi_k(4),0} + p_{\pi_k(4),1}, d_{\pi_k(3),2}\right\} = \max\{10, 11\} = 11$$
$$d_{\pi_k(4),2} = \max\left\{d_{\pi_k(4),1} + p_{\pi_k(4),2}, d_{\pi_k(3),3}\right\} = \max\{12, 13\} = 13$$
$$d_{\pi_k(4),3} = d_{\pi_k(4),2} + p_{\pi_k(4),3} = 13 + 3 = 16$$

Finally, the makespan of $n_k = 4$ *jobs processed in the factory* F_k *is* $C_{\max} = \max d_{\pi_k(n_k),m} = 16$.

4. Proposed Algorithm for Solving DBFSP

Although FOA has presented good performances on many engineering optimization problems, difficulties are still exposed when applying it to solving the scheduling problems. Firstly, due to the problem-specific discreteness, the continuous fitness functions in FOA cannot be directly employed and dedicated encoding and decoding schemes are necessary. Secondly, the random initialization mechanism of basic FOA reduces the quality of solutions, which further increases the difficulty in searching for the optimum value. Thirdly, the random search behavior of each fruit fly could not support the convergence performance of the population. Lastly, it misses an effective local search method to guide the fruit fly towards the best location.

To cope with the above limitations, we present a DFOA that inherits and extends the searching idea of basic FOA. The flowchart is presented in Figure 3. In the smell-based search phase, the algorithm explores the solution space. The VND-based local search aims to exploit the search space. In the vision-based search phase, DFOA updates the population with the best fruit fly found so far. DFOA balances both the exploration and exploitation. It is expected to achieve satisfactory performances for solving DBFSP.

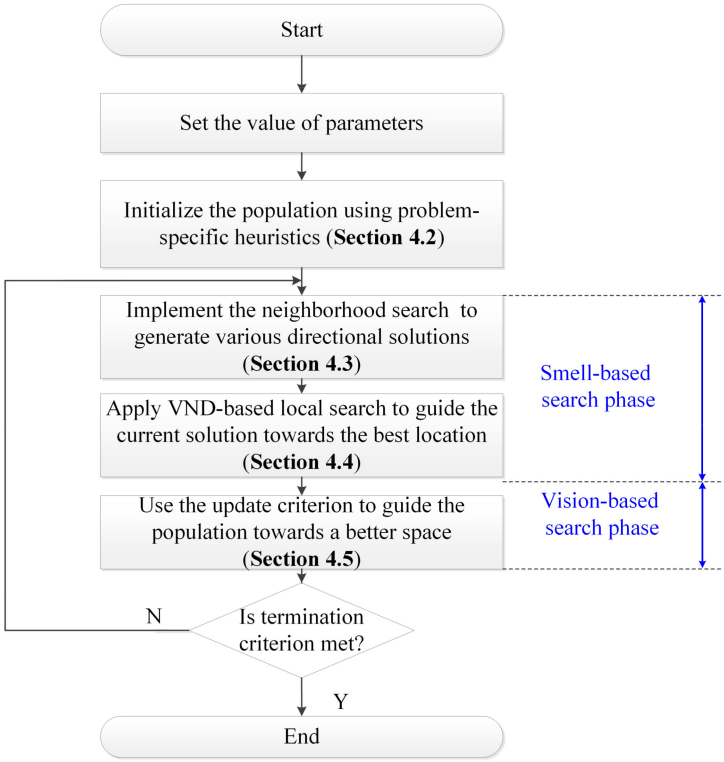

Figure 3. Flow chart of the proposed DFOA.

4.1. Solution Representation

Based on the description from Section 3, the solution for DBFSP with F factories and n jobs can be defined as a set of partial sequences of all factories, as follows:

$$\pi = \left\{\pi_k \middle| (k = 1, 2, \ldots, F)\right\}, \tag{8}$$

where $\pi_k = \left\{\pi_{k(1)}, \pi_{k(2)}, \ldots, \pi_{k(n_k)}\right\}$ denotes the partial sequence with n_k jobs assigned in the factory F_k, and $\sum_{k=1}^{f} n_k = n$. Such representation can be easily decoded to a DBFSP schedule because π_k determines not only the processing sequence, but also the assigned jobs in factory F_k.

Example 2. *A solution for DBFSP with $n = 6$ and $F = 2$ is considered. Hence, a complete solution is $\pi = \{\pi_1, \pi_2\}$, where $\pi_1 = \{J_2, J_1, J_6\}$, and $\pi_2 = \{J_3, J_5, J_4\}$. When applying the decoding scheme, jobs J_2, J_1, and J_6 are assigned to factory 1 and processed in the order of $J_2 \rightarrow J_1 \rightarrow J_6$. The decoding procedure for π_2 is the same.*

4.2. Population Initialization

When initializing the population, the decision of assigning jobs to factories should be considered. In this paper, the assignment rule implemented by Naderi and Ruiz [22], called the earliest completion factory (ECF) rule [26] is adopted. The pseudocode of the ECF rule is presented in Algorithm 1. ECF firstly arranges all the jobs according to their total processing time. Then, it assigns job j to the factory orderly that completes it at the earliest time, i.e., the lowest $C_{\max}$ after including this job as the last job. This assignment rule was proven to be more effective than other rules for makespan minimization [28,33]. In addition, the decoding rule can also balance the workload between all factories. Based on the ECF rule, three heuristic initialization methods are proposed as follows: A distributed NEH-PWT method (DNPM), a distributed NEH method (NEH2) [22] and a distributed NEH random method (DNRM).

Algorithm 1 ECF rule

Procedure ECF rule
Input Parameter (solution π, factory number F)
For k = 1 to F
    $\pi_k(1) = \pi(k)$
    $n_k = 1$
End For
For $k = F + 1$ to n
    Find the factory f that can process job $\pi(k)$ with the earliest completion time
    $n_f = n_f + 1$
    $\pi_k(n_f) = \pi(k)$
End For
Output π

DNPM: The NEH heuristic approach [15] has proven to be one of the most effective heuristics known for FSP. The NEH method requires that the job with the larger total processing time should be arranged with a higher priority than the one with the smaller total processing time. Nevertheless, it may be unsuitable for scheduling problems with blocking or buffer constraints due to the problem-specific characteristics. For BFSP, arranging the job with the larger total processing time in the forepart of a permutation may lead to a larger blocking time for its successive jobs. The increased blocking time can cause a larger makespan value.

In response to this problem, Pan et al. have proposed the NEH-PWT method [13], which gives the job with shorter total processing time higher priority when sequencing. The NEH-PWT method was

proven to be more superior to NEH for solving BFSP through the numerical experiments. Inspired by the idea, we apply the NEH-PWT method to construct DNPM, which contains the three following steps: First, all jobs to be processed are arranged in ascending order according to their total processing time, $T_j = \sum_{i=1}^{m} P_{j,i}$, to generate a job sequence, $J = [J_1, J_2, \ldots, J_n]$. Then, F partial sequences are constructed for all factories. With the ECF rule, insert each job from J_1, J_2, until J_n to all possible slots of all factories until the lowest makespan C_{max} is found.

NEH2: The procedure of the NEH2 method is similar to DNPM, except that the job sequence, $J = [J_1, J_2, \ldots, J_n]$, is arranged in descending order according to their total processing time $T_j = \sum_{i=1}^{m} P_{j,i}$.

DNRM: The procedure of DNRM is similar to DNPM, except that the job sequence $J = [J_1, J_2, \ldots, J_n]$ is generated through randomly arranging the order of jobs.

In the initialization phase, the fruit fly population with Ps individuals are produced with the above three heuristic methods. To guarantee the quality of solutions while keeping the diversity of the population, one solution is generated using DNPM, one is produced using NEH2, and the rest P_s-2 solutions are generated using DNRM.

4.3. Smell-Based Search Phase

In the smell-based search phase, each fruit fly searches for a food source in a random direction and generates a new location. The best location will be found and the whole population flies towards it. In this section, a neighborhood search strategy that contains four solution generation operators is proposed to help a fruit fly find a good location in its local region. A neighborhood structure is defined by representing the way it modifies the incumbent feasible solution to determine a new feasible solution. The design of neighborhoods in a distributed environment needs to consider the fact that the global makespan cannot be improved without involving the factory with the largest makespan (defined as critical factory fc). That is, it makes no sense, when the neighborhood search strategy is implemented within a non-critical factory or between two non-critical factories. Considering such characteristics, we exploit the neighborhood structures within the critical factory, as well as between the critical factory and other factories.

The solution generation operators are classified into two categories, as follows: One is based on the job sequence adjustment, including forward insertion and backward insertion within the critical factory; another one is based on the reassignment of jobs, including insertion and the swap between the critical factory and other factories. The four solution generation operators are depicted as follows. An example of the four solution generation operators is given in Figure 4.

Forward insertion within the critical factory fc: Stochastically select one if there is more than one critical factory. Choose two positions, $s1$ and $s2$ ($s2 > s1$), at random. The block between these two positions is referred to B_1. Move B_1 and $s2$ one position forward by turns, move $s1$ to the original position of $s2$ (please see Figure 4a).

Backward insertion within the critical factory fc: Choose two positions, $s1$ and $s2$ ($s2 > s1$), at random. The block between these two positions is referred to as B_1; move B_1 and $s1$ one position forward by turns, move $s2$ to the original position of $s1$ (please see Figure 4b).

Insertion between fc and other factories: Randomly select one job, J, at position s1 in the fc. For each of the other $F-1$ factories, randomly select one position and insert J to this position. Thus, $F-1$ solutions are generated. Choose the best one (with lowest global makespan) as the candidate solution. (please see Figure 4c).

Swap between fc and other factories: Randomly select one job J at position $s1$ in the fc. For each of the other $F-1$ factories, randomly select one job and swap J and J^*. Then, $F-1$ solutions are generated. Choose the best one as the candidate solution. (please see Figure 4d).

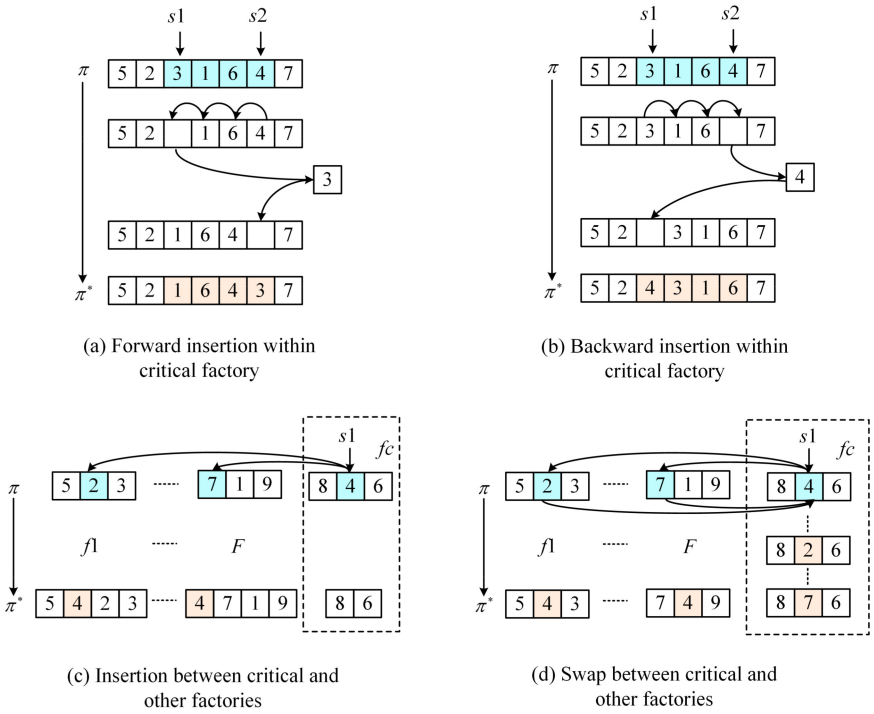

Figure 4. Solution generation operators of the neighborhood structures.

The pseudocode of the neighborhood search strategy is sketched in Algorithm 2. In general, more operators can enhance the search ability of the individual in a higher probability than a single one. This strategy also holds the diversity of the population. After generating the neighborhood structures, the candidate solutions are evaluated. The best one is considered as the new solution to undergo the local search, which aims to conduct the new solution rapidly towards the best location.

Algorithm 2 Neighborhood search strategy

Procedure Neighborhood search strategy
Input: Parameter (initialized solution π, critical factory *fc*)
Output: Solution π^*
Begin // neighborhood operation
π_1:$\pi\leftarrow$ Forward insertion operator within the critical factory
π_2:$\pi\leftarrow$ Backward insertion operator within the critical factory
π_3:$\pi\leftarrow$ Insertion operator between the critical factory and other factories
π_4:$\pi\leftarrow$ Swap operator between the critical factory and other factories
evaluate $C_{\max}(\pi_1), C_{\max}(\pi_2), C_{\max}(\pi_3), C_{\max}(\pi_4)$// makespan evaluation
$\pi^* = \text{argmin}\{C_{\max}(\pi_1), C_{\max}(\pi_2), C_{\max}(\pi_3), C_{\max}(\pi_4)\}$
Output π^*
End

4.4. VND-Based Local Search

During the procedure of FOA iteration, the whole population gathers around the best location. Each individual can only learn from the current optimal individual. This makes the algorithm very easy to trap into the local optimum. To overcome such premature problems, FOA needs to furnish with a mechanism that helps escape from the local optimum and continue searching in other solution spaces.

In most BFSP literature, insertion and swap movements are universally recognized as effective and efficient search processes to produce better solutions. Following this vein, a local search scheme based on insertion and swap variants of the VND method [23] is embedded in the proposed algorithm.

VND is a variant of variable neighborhood search (VNS), where N neighborhood structures are systematically switched in a definitive way. VND begins from the first neighborhood and undergoes the improvement procedures until a local optimum in relation to all neighborhoods is reached. When no further improvement for the current $i-th$ neighborhood is obtained and $i < N$, the search procedure carries on with the $(i+1)-th$ neighborhood. Like VNS, if a better solution is found, VND starts again from the first neighborhood. When $i+1 > N$, the search procedure terminates and rewards the final solution. VND is simple and easy to implement and it has shown high performances for many optimization problems. In this section, the concept of VND is adopted in the local search part to help the algorithm explore a larger solution space. Following the design philosophy of neighborhood structure in Section 4.3, and to avoid redundant computing procedures, two neighborhood movements for the VND are proposed. The insertion (LS_Insert) and swap movement (LS_Swap) are presented as follows.

LS_Insert: Select a job j_1 from the critical factory and insert it in the best position of a non-critical factory. The best position refers to the position that obtains the lowest makespan after insertion movement. The improvement is recognized if the makespan $C_{\max}$ is diminished. The permutation of jobs in this factory is kept and the search procedure starts with the factory that now has the maximal makespan. If the movement cannot improve the makespan, select a new job, j_2, from the original critical factory to rerun the procedure. The procedure terminates after traversing all jobs of the original critical factory. Moreover, to accelerate the insertion procedure, a modified speed-up method proposed by Reference [13] is adopted in this study. The speed-up method is applied to evaluate the n_k+1 partial sequences obtained by inserting one job, J, in all the possible positions of factory F_k that already has n_k assigned jobs. The speed-up method can reduce the time complexity from $O(mn^3)$ to $O(mn^2)$ for an insertion-based local search procedure, which is crucial for an algorithm with high performance. The detailed procedure is described as follows.

Step 1: Calculate the departure time, $d_{\pi_k(l),i}$, of the n_k jobs that are already assigned in the factory F_k with Equations (1)–(5).

Step 2: Calculate the tails, $f_{\pi_k(l),i}$, of n_k jobs that are already assigned in the factory with Equations (9)–(13), shown as follows:

$$S_{\pi_k(n_k),m+1} = 0, \tag{9}$$

$$f_{\pi_k(n_k),i} = f_{\pi_k(n_k),i+1} + p_{\pi_k(n_k),i}, i = m,\ldots,1, \tag{10}$$

$$f_{\pi_k(l),m+1} = f_{\pi_k(l+1),m}, \quad l = n_k-1,\ldots,1, \tag{11}$$

$$f_{\pi_k(l),i} = \max\left\{f_{\pi_k(l),i+1} + p_{\pi_k(l),i}, \quad f_{\pi_k(l+1),i-1}\right\} \quad l = n_k-1,\ldots,1 \quad i = m,\ldots,2, \tag{12}$$

$$f_{\pi_k(l),1} = d_{\pi_k(l),2} + p_{\pi_k(l),1}, \quad l = n_k-1,\ldots,1\,. \tag{13}$$

Step 3: Calculate the departure times, $d_{\pi_k(q),i}$, $i = 1,\ldots,m$, of jo J to be inserted in the position q of the selected factory in the current solution.

Step 4: Compare the makespan of the selected factory after inserting job J in the position q by

$$C_{\max} = \max_{i=1,\ldots,m}(d_{\pi_k(q),i} + f_{\pi_k(q),i}), \quad q = 1,\ldots,n_k+1. \tag{14}$$

Step 5: Choose the best insertion position and return the best makespan. The pseudocode of the LS_Insert process is illustrated in Algorithm 3.

LS_Swap: Select each job from the critical factory, swap j_1 with all jobs of all non-critical factories orderly, and reinsert jobs in the best position of the new factories, i.e., the position that results in the lowest makespan. The improvement is recognized if the makespan $C_{\max}$ is diminished. The procedure terminates after all jobs of the critical factory have been selected. The speed-up method used for LS_Swap is the same as one used for LS_Insert, except that a job is removed from the original factory

before a new job can be inserted. The LS_Swap process is described in Algorithm 4. Moreover, the pseudocode of the VND local search scheme is illustrated in Algorithm 5.

Algorithm 3 Local search insert

Procedure Local search insert
Input: Parameter (solution π, factory number F)
Output: Solution π^*
Begin
While stop criterion is not satisfied
  For j = 1 to n_{f_c} // traverse all jobs in the critical factory
    Select job j from the critical factory fc without repetition
      Select a factory f randomly without repetition (with sequence π_f)
        Insert j in the best position of f and obtaining $\pi_{f'}$
        If $C_{max}(\pi_{f'}) < C_{max}(\pi)$ then
          π^*: update π^* by substituting π_f with $\pi_{f'}$, remove job j from the critical factory
            Calculate the $C_{\max}(\pi^*)$ using the speed-up method and detect the new critical factory
        Else $\pi^* = \pi$, $j = j + 1$
        End If
  End For
End While
End

Algorithm 4 Local search swap

Procedure Local search swap
Input: Parameter (solution π, factory number F)
Output: Solution π^*
Begin
While the stop criterion is not satisfied
  For j = 1 to n_{fc} // job number in the critical number
    Select j from the critical factory fc without repetition
      For f = 1 to F // select a non-critical factory
        If $f \neq fc$ then
          For i = 1 to n_f // the job number in the select factory
            remove job j from fc and insert i in the best position of fc obtaining $\pi_{fc'}$, calculate $C_{max}(\pi_{fc'})$ using the speed-up method
            remove job i from f and insert j in the best position of f obtaining $\pi_{f'}$, calculate $C_{max}(\pi_{f'})$ using the speed-up method
            If $C_{\max}(\pi_{fc'}) < C_{\max}(\pi)$ and $C_{\max}(\pi_{f'}) < C_{\max}(\pi)$ then
            $C_{\max}(\pi^*) = \max\left\{C_{\max}(\pi_{fc'}), C_{\max}(\pi_{f'})\right\}$
            π^*: modify π with $\pi_{fc'}$ and $\pi_{f'}$
          Else $\pi^* = \pi$, return job j and job i to their original positions
          End If
        End For
      End If
    End For
  End For
Output π^*
End

Algorithm 5 VND-based local search

Procedure VND-based local search
Input: Parameter **(Solution** π, Nl**)**
Output: Solution π^*
Begin
Nl = {LS_Insert, LS_Swap}
  For i = 1 to Size(Nl)
    $\pi^* : \pi \leftarrow Nl(i)$
If $C_{\max}(\pi^*) < C_{\max}(\pi)$ then Output π^*, $i = 1$
    Else $i = i + 1$
End If
    If $i >$ Size(Nl)
     Break
End If
End For
End

4.5. Vision-Based Search Phase

The purpose of the vision-based search phase is to guide the fruit fly population to fly towards a superior space to further enhance the performance of the proposed algorithm. An elite-based update criterion is applied in this study. Firstly, retrieve the whole population and find the individual with the largest makespan. Secondly, replace it with the individual with lowest makespan found so far. After the vision-based search phase, DFOA finishes one iteration. The search procedure repeats until the termination criterion is reached. The pseudocode of DFOA is illustrated in Algorithm 6. The complexity of the algorithm is $O(Ps \times Nl \times m \times n^2)$.

Algorithm 6 DFOA for DBFSP

Procedure DFOA for DBFSP
Input: Parameter (population size Ps, termination time $T_{\max}$)
Output: best solution π^*
Begin
// Initialize population (**Section 4.2**)
$\pi_1 \leftarrow$ problem-specific heuristic (DNPM)
$\pi_2 \leftarrow$ problem-specific heuristic (NEH2)
$\pi_i \leftarrow$ problem-specific heuristic (DNRM)
i=3,4,...,PS
Repeat
For i = 1 to Ps // smell-based search phase (**Section 4.3**)
    $\pi' \leftarrow$ Neighborhood search strategy
      $\pi'' \leftarrow$ VND-based local search // (**Section 4.4**)
      If $C_{\max}(\pi'') < C_{\max}(\pi_i)$ then $\pi_i = \pi''$
        If $C_{\max}(\pi'') < C_{\max}(\pi_{best})$ then $\pi_{best} = \pi''$
        End If
      End If
End For
// vision-based search phase (**Section 4.5**)
  Find out the worst solution π_{worst} in the whole population
  π_{worst}= Update criterion (π_{best})
  Until the termination time $T_{\max}$ is met
End

5. Computational Experiment

5.1. Experiment Setting

Since there are no dedicated instances available, as a compromise, the computational experiments were conducted with the benchmark for DFSP modified by Naderi and Ruiz [28], which is extended from the benchmark of Taillard [20]. The benchmark comprises a set of 420 small instances (developed in 2010) and a set of 720 larger instances (developed in 2014).

Moreover, the authors appended two sets (small and large) of 50 test instances for calibration with n, m, and f values randomly sampled from the set of 720 instances. To be more specific, the benchmark contained 72 sets of 10 instances ranging from 20 jobs × 5 machines to 500 jobs × 20 machines, where n ϵ {20, 50, 100, 200, 500} and m ϵ {5, 10, 20}. The number of factories, f, is in the set {2, 3, 4, 5, 6, 7}. All instances are available on http://soa.iti.es.

In order to evaluate the effectiveness of proposed DFOA in different domains, it was compared with other heuristics and metaheuristics. In the experiment, the same experimental environment, including hardware, programming language, and termination criterion were applied. All the algorithms were coded in Python and loaded on a PC with an Intel(R) Core(TM) i7-8700 CPU and 16G RAM. The termination criterion for each compared metaheuristic was set as the maximal elapsed CPU time $T_{\max} = n \times m \times F \times 90$ milliseconds (ms). Setting this CPU time correlated with the instance size and computational complexity enables the test algorithm to have more time to address large-scale instances that may be "hard" [44]. The generated minimum makespan is recorded for calculation of the statistical indicators. Denote C_r^i as the solution provided by the r-th running of the i-th compared algorithm and C^{best} represents the best solution obtained by any of the algorithms. To estimate the computational results, the following statistical indicators are computed:

(1) Average relative percentage deviation (ARPD) is considered as a response variable that evaluates the mean quality of solutions, as follows:

$$ARPD_i = \frac{1}{R}\sum_{r=1}^{R}\frac{C_r^i - C^{best}}{C^{best}} \times 100. \quad (15)$$

(2) Standard deviation (SD) that evaluates the quality of initial solutions and the robustness of the algorithm, as follows:

$$SD_i = \sqrt{\frac{1}{R}\sum_{r=1}^{R}\left[\frac{C_r^i - C^{best}}{C^{best}} \times 100 - ARPD\right]^2}. \quad (16)$$

From Equation (15) it is clear that the lower the ARPD value is, the better is the compared algorithm performance. The experiments are conducted considering following aspects:

(1) Sensitivity analysis of parameter P_s;
(2) Comparison of the heuristics initialization methods;
(3) Comparison of the local search methods;
(4) Comparison with heuristics on small-scale instances;
(5) Comparison with metaheuristics from other literature on large-scale instances.

5.2. Sensitivity Analysis

As mentioned, one advantage of DFOA is its simplicity. Compared with other algorithms, FOA has only few parameters to adjust. Here, a sensitivity analysis of the parameter population size (P_s) is conducted on the test instances. According to usual practice, a small value of P_s brings the algorithm an insufficient search and convergence ability. On the contrary, a large value of P_s can obtain better results because more fruit flies increase the diversity and can explore more locations in the solution

space. However, an overlarge value will yield a high computational cost. As a trade-off, P_s was set as 35 according to the sensitivity analysis, which is shown in Figure 5.

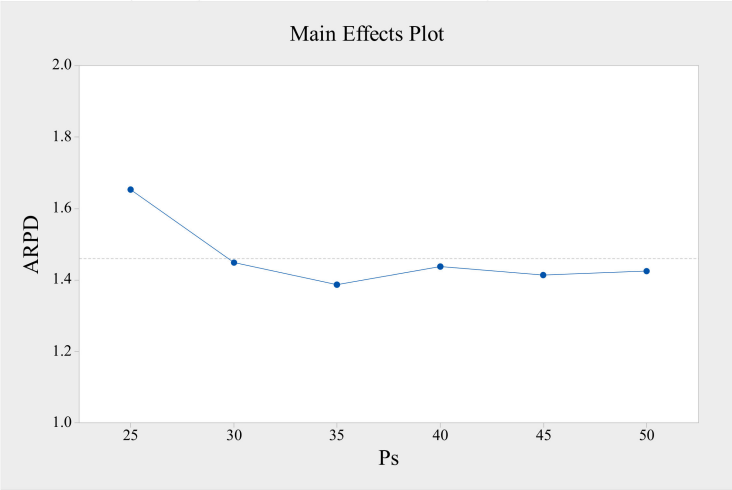

Figure 5. Sensitivity analysis of parameter P_s.

5.3. Comparison of the Heuristic Initialization Methods

To analyze the effectiveness of the heuristic initialization methods in Section 4.2, DNPM, NEH2, and DNRMwe tested separately. All of them are heuristics that need no termination criterions for implementation. However, due to randomness, DNRM, especially, was repeated 10 times and the average value is taken as the test result. The comparison results are listed in the form of ARPD values in Table 1, grouped by different factory numbers. Note that the result of each group is the average value of all test instances in this group. In Table 1, the best performing heuristic was DNPM, with 13.86 on average, and it produced smaller ARPD values for all factories. The second best heuristic was NEH2, which produced an average ARPD value of 15.87. The DNRM yielded the worst results by a value of 22.41, on average. From these results, we confirmed that the DNPM is more suitable for DBFSP. It also indicates that the blocking constrains areas different from the permutation constraints, which reminds the researchers to design dedicated methods to handle it.

Table 1. The comparison results of DNRM, NEH2, and DNPM in the algorithm.

F	ARPD			Running Time		
	DNRM	**NEH2**	**DNPM**	**DNRM**	**NEH2**	**DNPM**
2	20.85	14.38	12.05	0.003	0.003	0.003
3	21.47	15.9	13.59	0.004	0.004	0.004
4	22.93	16.19	14.25	0.006	0.006	0.006
5	23.49	17.67	15.2	0.009	0.009	0.009
6	22.5	16.32	14.47	0.010	0.010	0.010
7	23.24	15.81	13.728	0.011	0.011	0.011
Ave.	22.41	15.87	13.86	0.0072	0.0072	0.0072

In addition, Figure 6a presents the mean plot, with a 95% confidence interval, of the three heuristics. It is clear that the overall ARPDs of DNPM stayed under NEH2 and DNRM without any overlapping for all factories, which reports that the DNPM performed significantly better than the NEH2 and DNRM from the statistical view. On the other hand, in Figure 6b, we can observe the behavior of the initialization methods with the size of the instances ($n \times m$). As seen, the most influential factor is the number of machines in each factory. With the increment of m, the performances of all the initialization methods improved.

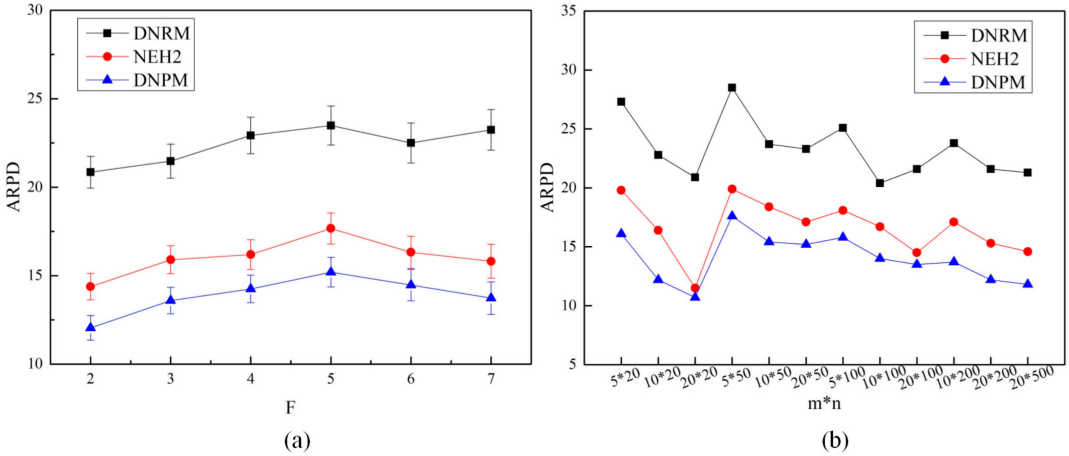

Figure 6. (**a**) Mean plot with 95% confidence interval of interaction between the initialization methods and factories; (**b**) ARPD values by different initialization methods, with $m \times n$.

5.4. Comparison of the Local Search Methods

In this section, the performances of the proposed local search methods are compared. DFOA was used as the test algorithm. For a fair comparison, the initialization methods applied are the same. That is, to overcome the randomness brought by DNRM, DFOA generates the $P_s - 2$ solutions by DNRM only once, they are applied for all the compared metaheuristics. Consequently, four metaheuristics are created, i.e., DFOA + LS_Insert (with LS_ins for short), DFOA + LS_Swap (with LS_sw for short), DFOA + no local search (with NLS for short), and DFOA + VND (with LS_V for short). Table 2 lists the ARPD values for all algorithms, grouped by different factory numbers.

Table 2. The Comparison results of LS_ins, LS_sw, NLS and LS_V in the algorithm.

F	LS_ins		LS_sw		NLS		LS_V	
	ARPD	SD	ARPD	SD	ARPD	SD	ARPD	SD
2	1.612	1.352	1.454	1.034	2.241	1.827	0.939	0.684
3	1.538	1.348	1.292	1.168	2.238	1.758	0.823	0.732
4	1.529	1.361	1.258	1.138	2.097	1.732	0.786	0.743
5	1.464	1.308	1.281	1.102	2.116	1.962	0.807	0.618
6	1.492	1.292	1.349	1.106	2.182	1.814	0.881	0.757
7	1.473	1.315	1.178	1.048	2.052	2.095	0.844	0.704
Ave.	1.518	1.329	1.302	1.136	2.154	1.865	0.847	0.706

As seen in Table 2, between LS_ins and LS_sw, LS_sw achieved better performance, with an overall ARPD value of 1.302, while the LS_ins obtained the overall ARPD value of 1.518. This demonstrates that the exploitation ability of LS_sw is relatively stronger. In most literature, the insertion movement explores larger searching spaces than the swap movement. In this experiment, however, the swap operation attached the insertion procedure after the swap movement and produced thereby better results. Additionally, the overall ARPD value of LS_V is 0.847, which shows large superiority over the results provided by the other three metaheuristics. This indicates that a better performance can be obtained by combining multiple local search methods. Since different local search methods explore different solution spaces, a hybridization strategy can help escape the local optimum. It is not surprising that, among the four metaheuristics, NLS obtained the worst performance, with an overall ARPD value of 2.154. Like some other nature-inspired algorithms, FOA has a better exploration ability but with a poor exploitation ability. An embedded local search could overcome this limitation.

Figure 7 illustrates the mean plot, with a 95% confidence level, between the compared local search methods and factory numbers. As can be seen, LS_sw gained slightly better results than LS_ins, but

the differences between them are not large enough to define the significant difference from a statistical view. The overall ARPD values of LS_V lie totally under those of LS_ins, LS_sw, and NLS without overlapping, which indicates that LS_V is significantly better than the other three metaheuristics.

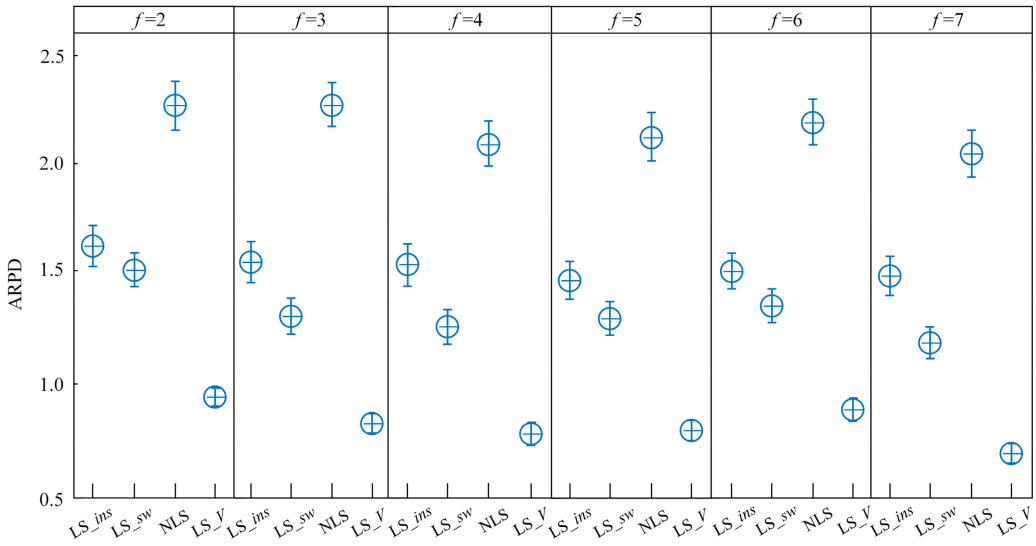

Figure 7. Mean plot with a 95% confidence interval of the interaction between local search methods and factories.

5.5. Comparison with Heuristics on Small-Scale Instances

In this section, the proposed algorithm is compared with some constructive heuristics proposed by Naderi and Ruiz in 2010 [22]. The authors have developed largest processing time heuristics (LPT2), shortest processing time heuristics (SPT2), heuristic by Johnson's rule (Johnson2), and heuristics of NEH2 and VND(a). The suffix "2" means that the second job-to-factory rule is employed, which was proven to be more effective than the first one proposed in their studies. More details about the test algorithms refer to Reference [22]. The 420 small-scale instances are used to compare DFOA with the above five heuristics. All the results are grouped by each combination of the factory number, F, and the number of jobs, n. Each group contains 20 instances. The result of each group is the average result from 20 instances. The termination criterion for DFOA is 50 iterations. From Table 3 it can be seen that DFOA is the best algorithm for solving the small-scale instances. VND(a) ranks second and the NEH2 method ranks third. LPT2 and Johnson2 obtained the largest ARPD (21.027) and the second largest ARPD (13.933), respectively. The performance comparison between LPT2 and SPT2 also reflects the problem-specific characteristics of DBFSP, which has an inverse relation when both of them are applied for DFSP.

In addition, we come to the following conclusion. (1) The instance combination $F \times n = 4 \times 4$ gives no other arrangement on the solution. Hence, the results obtained by all algorithms were the same. (2) When the complexity of the combination increased, the ARPD values increased as well. (3) It is clear that the results obtained by heuristics are far from those of metaheuristics. However, heuristics has the advantage on CPU time. Table 4 demonstrates the time consumptions of all the compared algorithms. As seen, the CPU time is especially small when applying the constructive heuristics, whereas metaheuristics consume more time due to their iterated procedure. Since our aim is to minimize the makespan, this CPU time can be accepted in practice use. It was also found that CPU time reduces with the increment of the factory number F, which indicates that the instance becomes easier to solve when F becomes larger.

Table 3. ARPD values by different algorithms on small-scale instances.

Instance ($F \times n$)	LPT2	SPT2	Johnson2	NEH2	VND(a)	DFOA
2×4	8.326	3.919	5.368	0.586	**0.000**	**0.000**
2×6	15.831	4.424	10.172	1.923	1.53	**0.000**
2×8	20.485	7.342	12.279	3.761	2.764	**0.000**
2×10	23.855	12.279	16.024	5.681	3.573	**0.000**
2×12	28.747	15.486	14.325	5.508	3.112	**0.000**
2×14	31.639	20.367	19.895	6.544	3.433	**0.000**
2×16	32.095	18.997	17.351	6.225	3.771	**0.000**
3×4	6.472	4.786	5.288	0.353	**0.000**	**0.000**
3×6	20.533	9.528	10.648	2.371	1.269	**0.000**
3×8	21.684	12.235	15.247	3.813	2.673	**0.000**
3×10	28.729	15.687	17.542	5.279	4.082	**0.000**
3×12	29.754	19.376	19.556	6.852	4.565	**0.000**
3×14	28.652	18.453	21.455	6.034	5.263	**0.000**
3×16	30.213	18.674	20.859	5.511	4.736	**0.000**
4×4	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
4×6	10.538	11.789	12.523	0.418	**0.000**	**0.000**
4×8	13.277	13.894	14.267	1.278	0.879	**0.000**
4×10	15.541	16.452	17.385	3.923	2.598	**0.000**
4×12	20.067	20.155	21.716	5.536	4.677	**0.000**
4×14	25.244	23.483	25.006	6.893	5.343	**0.000**
4×16	29.875	25.276	27.233	7.282	5.816	**0.000**
Ave.	21.027	13.933	15.435	4.084	2.8611	**0.000**

Table 4. Average time consumptions (s) on small-scale instances grouped by factory number F.

F	LPT2	SPT2	Johnson2	NEH2	VND(a)	DFOA
2	0.00151	0.00151	0.00151	0.00151	0.0174	53.14
3	0.00149	0.00149	0.00149	0.00149	0.0126	52.08
4	0.00148	0.00148	0.00148	0.00148	0.0085	49.59
Ave.	0.00149	0.00149	0.00149	0.00149	0.0128	51.6

5.6. Comparison to Other Metaheuristics on Large-Scale Instances

In this section, DFOA is compared with the well-known metaheuristics on large-scale instances. Since there is little literature published for DBFSP so far, some metaheuristics from DFSP literature are modified and applied for the DBFSP. The compared metaheuristics are the following: (1) The discrete electromagnetism-like mechanism algorithm (EM) of Liu et al. [24]; (2) the hybrid genetic algorithm (HGA) of Gao et al. [25], (3) the Tabu search (TS) of Gao et al. [26]; and (4) the discrete differential evolution algorithm (DDE) of Zhang et al. [34]. Not all metaheuristics in the reviewed literature are compared, as some of them are hard to replicate without accessing their original codes. Since all metaheuristics perform well and have a strong search ability, the comparison differences on the small-scale instances are not significant. Hence, only the comparison results on the large-scale instances are demonstrated.

Given the five algorithms tested, 720 large-scale benchmark instances and 10 replications, there are, in total, 36,000 results. It is worth mentioning that we strictly complied with the detailed description of the literature to carry out the above metaheuristics. Even though most of them were not originally developed for DBFSP, their searching essences (e.g., the architecture, the encoding and decoding scheme, the local search method) were not abandoned. Only the fitness functions are adjusted to match the problem considered. The parameters of the compared metaheuristics are listed in Table 5.

Table 5. Parameter setting of the compared algorithms.

Algorithms	Year	Parameter Setting
EM [24]	2010	$Threshold = 0.9, proMu = 0.6, UP = 0.8$
HGA [25]	2011	$Ps = 30, pm = 0.6$
TS [26]	2013	$T = 5$
DDE [34]	2018	$Ps = 100, \kappa = 0.6, C_p = 0.2, \lambda = 0.005, lstimes = 20$

In order to investigate the differences and performance trends between the above metaheuristics comprehensively, the results in Table 6 were grouped by the influential factors, i.e., the number of factories, F, jobs, n, and machines, m, respectively. As seen, DFOA obtained the lowest ARPD values in all instances by all factors. The overall ARPD of DFOA was 0.91, which is remarkably better than that of EM (4.985), HGA (3.202), TS (2.901), and DDE (1.548).

Table 6. Algorithmic comparison on large-scale instances, grouped by different factors.

Instance		EM	HGA	TS	DDE	DFOA
F	2	4.682	3.272	2.701	1.688	1.024
	3	5.047	3.443	2.889	1.852	1.084
	4	5.231	3.361	3.146	1.578	0.865
	5	5.378	3.434	3.048	1.543	1.011
	6	4.864.	3.002	2.997	1.433	0.848
	7	4.597	2.703	2.356	1.271	0.595
n	20	4.365	2.783	2.384	1.355	0.798
	50	4.702	2.908	2.722	1.524	0.806
	100	4.808	3.172	2.968	1.539	0.857
	200	5.146	3.429	3.095	1.618	1.087
	500	5.698	3.812	3.371	1.642	1.031
m	5	5.472	3.557	3.335	1.829	1.114
	10	4.966	3.038	2.899	1.447	0.896
	20	4.708	2.917	2.578	1.366	0.707
	Ave.	4.985	3.202	2.901	1.548	0.91

For the instances grouped by F, n, and m, the performance trends of metaheuristics are presented in Figure 8. From Figure 8a it can be seen that the entire performances of all metaheuristics improved with the increment of F, when the value of F was over five. It also indicates that a small factory number would not dominate their performances as expected. The algorithmic performances fluctuate when F is smaller than five. Other influential factors such as job number n, machine number m, or the algorithmic components like population size, may interfere with the performances more comprehensively. In Figure 8b, it is clear that with the increment of n, the performances of the metaheuristics became worse in general. This may because assigning more jobs to fixed machines makes the situation more complicated and results in more blocking situations. The same argument is suggested for the performance trends that were grouped by different machine numbers, which is shown in Figure 8c.

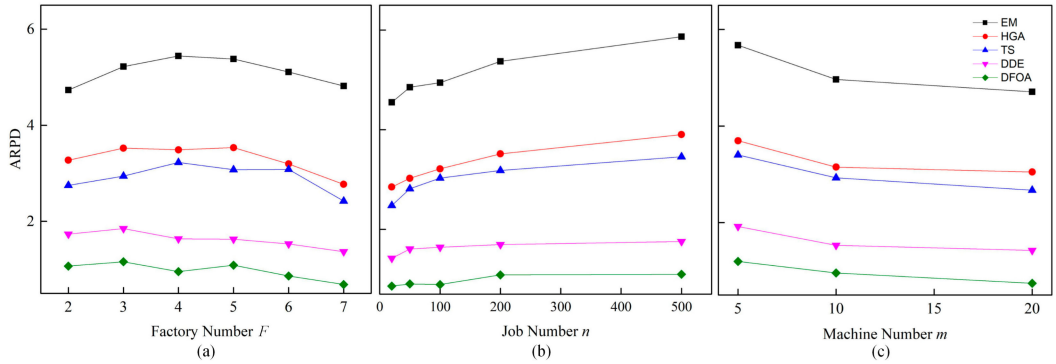

Figure 8. Performance trends of metaheuristics on different: (**a**) Factory number; (**b**) job number; and (**c**) machine number.

Figures 9–11 demonstrate three typical convergence curves of the compared metaheuristics on different instances. As can be seen, with the increment of computation complexity, the differences between metaheuristics become larger.

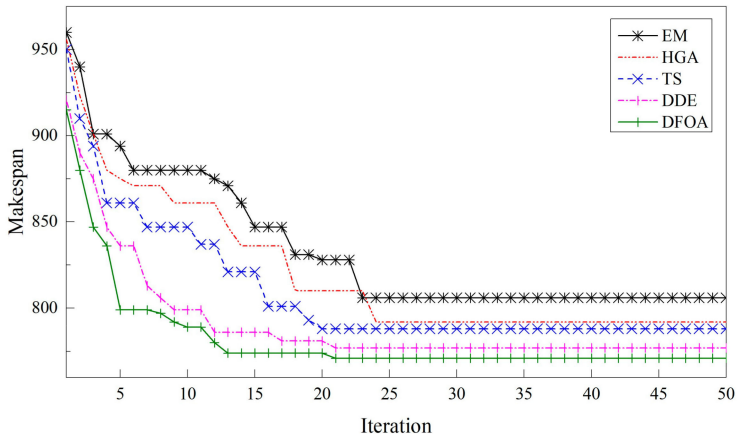

Figure 9. Convergence curves of instance Ta001_2.

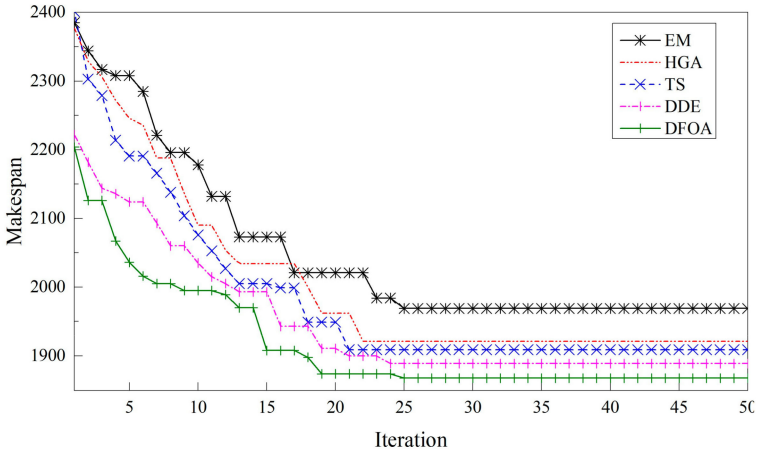

Figure 10. Convergence curves of instance Ta101_6.

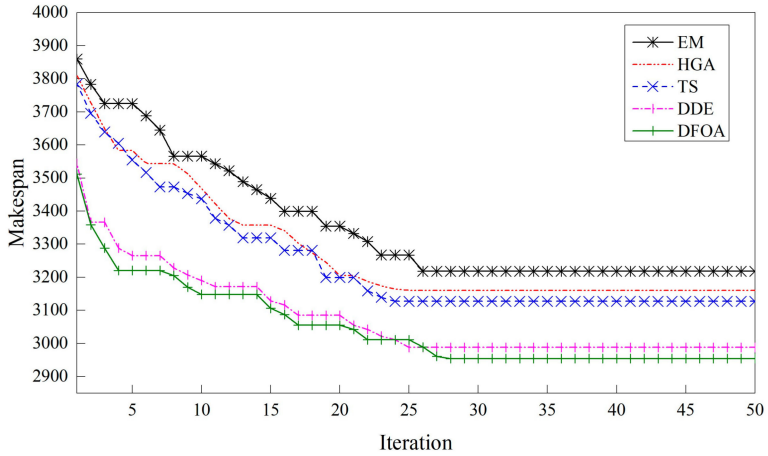

Figure 11. Convergence curves of instance Ta110_7.

Furthermore, Holm's multi-test [45] was adopted to evaluate the comparison between different metaheuristics. In Holm's method, $H_1, \ldots, H_m$ represent m hypotheses and $P_1, \ldots, P_m$ are the corresponding p-values, which denote the probabilities of observing the given results by chance. The p-values are ordered from lowest to highest by $p_{(1)} \leq p_{(2)} \leq \ldots \leq p_{(m)}$. For a given significance level α, let k be the minimal index so that $p_k \geq \alpha/(m+1-k)$. As a result, the null hypotheses from $H_{(1)}$ to $H_{(k-1)}$ will be rejected and the hypotheses from $H_{(k)}$ to $H_{(m)}$ will be accepted. As seen in Table 7, the returned p-values for all groups of hypotheses were zero. According to Holm's method, all the hypotheses were rejected. This confirms the statistical differences in the favor of DFOA over other metaheuristics.

Table 7. Holm's multi-test for all compared metaheuristics.

H_0	p-Value	$\alpha/(m+1-k)$	H_i
DFOA = EM	0	0.0095	Reject
DFOA = HGA	0	0.0184	Reject
DFOA = TS	0	0.0236	Reject
DFOA = DDE	0	0.3971	Reject

In the above paragraphs, the effectiveness of the proposed DFOA was tested and compared with the known existing metaheuristics. It can be concluded that DFOA is more effective than other metaheuristics when solving DBFSP with makespan criterion. The advantages are attributed to the delicate-designed algorithmic components according to the problem-specific characteristics as well as their suitable hybridization.

6. Conclusion and Future Works

In this study, a discrete fruit fly optimization algorithm (DFOA) is proposed to solve the blocking flowshop scheduling problem (DBFSP) in a distributed manufacturing system. Firstly, the problem description of DBFSP and mathematical model are presented. To solve this problem, a problem-specific initialization strategy was designed to generate an initial fruit fly population with quality and diversity. In the smell-based search phase of DFOA, four neighborhood structures for each individual were designed and the one with the best performance was selected to generate a new location in a deterministic way. Later, a VND-based local search scheme was designed. This mechanism implements an exhaustive search, which is competent for enhancing the exploitation ability in the promising space. In the vision-based search phase of DFOA, an effective update criterion was applied. The experiments were conducted by comparing the proposed DFOA with well-known constructive heuristics and metaheuristics on small-scale and large-scale instances. The experimental results indicate that the

proposed multiple-neighborhood strategy is conducive to enlarge the global search space. The performance of the proposed VND is proven to be more effective than single local search technology. Overall, the experiment results have shown that the proposed DFOA can handle DBFSP with a better search and optimization ability.

FOA has few parameters to be implemented, which can reduce the lengthy procedure of optimization. However, its simple structure still suffers from common problems (e.g., convergence precision) like other nature-inspired algorithms. Therefore, to overcome such deficiencies, future work will focus on the hybridization of the searching idea with other algorithms or strategies, such as PSO or knowledge-based systems.

On the other side, a multiobjective DBFSP is also considered, with the view of reaching a trade-off between energy consumption or carbon footprint and makespan.

Author Contributions: Conceptualization, S.T. and Z.L.; Methodology, X.Z., X.L., and G.K.; Software, X.Z and X.L.; Validation, X.Z., X.L., and Z.L.; Formal analysis, X.Z., X.L., and G.K.; Investigation, X.L. and S.T.; Writing—original draft preparation, X.Z. and X.L.; Writing—review and editing, G.K. and Z.L.; Supervision, S.T.; Project administration, X.L. and Z.L.

Funding: This research was funded by the Science and Technology Plan of Lianyungang (No. CG1615), Fundamental Research project of Central Universities (No. 201941008), Priority Academic Program Development of Jiangsu Higher Education Institutions (PAPD) and Australia ARC DECRA (No. DE190100931).

Acknowledgments: The authors would like to thank the anonymous reviewers for their contribution to this paper.

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

References

1. Zhao, F.; Xue, F.; Zhang, Y.; Ma, W.; Zhang, C.; Song, H. A discrete gravitational search algorithm for the blocking flow shop problem with total flow time minimization. *Appl. Intell.* **2019**, *49*, 3362–3382. [CrossRef]
2. Leisten, R. Flowshop sequencing problems with limited buffer storage. *Int. J. Prod. Res.* **1990**, *28*, 2085–2100. [CrossRef]
3. Graham, R.L.; Lawler, E.L.; Lenstra, J.K.; Kan, A.R. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discret. Math.* **1979**, *5*, 287–326. [CrossRef]
4. Riahi, V.; Newton, M.H.; Su, K.; Sattar, A. Constraint guided accelerated search for mixed blocking permutation flowshop scheduling. *Comput. Oper. Res.* **2019**, *102*, 102–120. [CrossRef]
5. Nagano, M.S.; Komesu, A.S.; Miyata, H.H. An evolutionary clustering search for the total tardiness blocking flow shop problem. *J. Intell. Manuf.* **2019**, *30*, 1843–1857. [CrossRef]
6. Leiras, A.; Hamacher, S.; Elkamel, A. Petroleum refinery operational planning using robust optimization. *Eng. Optim.* **2010**, *42*, 1119–1131. [CrossRef]
7. Zhu, Q.; Wu, N.; Yan, Q.; Zhou, M. Optimal scheduling of complex multi-cluster tools based on timed resource-oriented petri nets. *IEEE Access* **2017**, *4*, 2096–2109. [CrossRef]
8. Pan, D.; Yang, Y. Localized independent packet scheduling for buffered crossbar switches. *IEEE Trans. Comput.* **2009**, *58*, 260–274. [CrossRef]
9. Fernandez-Viagas, V.; Leisten, R.; Framinan, J.M. A computational evaluation of constructive and improvement heuristics for the blocking flow shop to minimise total flowtime. *Expert Syst. Appl.* **2016**, *61*, 290–301. [CrossRef]
10. Zhang, Y.; Gong, D.W.; Sun, J.Y.; Qu, B.Y. A decomposition-based archiving approach for multi-objective evolutionary optimization. *Inf. Sci.* **2018**, *430*, 397–413. [CrossRef]
11. Caraffa, V.; Ianes, S.; Bagchi, T.P.; Sriskandarajah, C. Minimizing Makespan in a Blocking Flowshop using Genetic Algorithms. *Int. J. Prod. Econ.* **2001**, *70*, 101–115. [CrossRef]
12. Grabowski, J.; Pempera, J. The permutation flow shop problem with blocking. A tabu search approach. *Omega* **2007**, *35*, 302–311. [CrossRef]
13. Wang, L.; Pan, Q.K.; Suganthan, P.N.; Wang, W.H.; Wang, Y.M. A novel hybrid discrete differential evolution algorithm for blocking flowshop scheduling problems. *Comput. Oper. Res.* **2010**, *37*, 509–520. [CrossRef]
14. Ribas, I.; Companys, R.; Tort-Martorell, X. An iterated greedy algorithm for the flowshop scheduling problem with blocking. *Omega* **2011**, *39*, 293–301. [CrossRef]

15. 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]
16. Wang, X.; Tang, L. A discrete particle swarm optimization algorithm with self-adaptive diversity control for the permutation flowshop problem with blocking. *Appl. Soft Comput.* **2012**, *12*, 652–662. [CrossRef]
17. Han, Y.Y.; Pan, Q.K.; Li, J.Q.; Sang, H.Y. An improved artificial bee colony algorithm for the blocking flowshop scheduling problem. *Int. J. Adv. Manuf. Technol.* **2012**, *60*, 1149–1159. [CrossRef]
18. Han, Y.Y.; Gong, D.W.; Sun, X.Y.; Pan, Q.K. An improved NSGA-II algorithm for multi-objective lot-streaming flow shop scheduling problem. *Int. J. Prod. Res.* **2014**, *52*, 2211–2231. [CrossRef]
19. Han, Y.Y.; Gong, D.W.; Li, J.Q.; Zhang, Y. Solving the blocking flowshop scheduling problem with makespan using a modified fruit fly optimisation algorithm. *Int. J. Prod. Res.* **2016**, *54*, 6782–6797. [CrossRef]
20. Taillard, E. Benchmarks for basic scheduling problems. *Eur. J. Oper. Res.* **1993**, *64*, 278–285. [CrossRef]
21. Shao, Z.; Pi, D.; Shao, W.; Yuan, P. An efficient discrete invasive weed optimization for blocking flow-shop scheduling problem. *Eng. Appl. Artif. Intell.* **2019**, *78*, 124–141. [CrossRef]
22. Naderi, B.; Ruiz, R. The distributed permutation flowshop scheduling problem. *Comput. Oper. Res.* **2010**, *37*, 754–768. [CrossRef]
23. Peng, K.; Pan, Q.K.; Gao, L.; Li, X.; Das, S.; Zhang, B. A multi-start variable neighbourhood descent algorithm for hybrid flowshop rescheduling. *Swarm Evolut. Comput.* **2019**, *45*, 92–112. [CrossRef]
24. Liu, H.; Gao, L. A discrete electromagnetism-like mechanism algorithm for solving distributed permutation flowshop scheduling problem. In Proceedings of the International Conference on Manufacturing Automation, Hong Kong, China, 13–15 December 2010; pp. 156–163.
25. Gao, J.; Chen, R. A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem. *Int. J. Comput. Int. Syst.* **2011**, *4*, 497–508. [CrossRef]
26. Gao, J.; Chen, R.; Deng, W. An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem. *Int. J. Prod. Res.* **2013**, *51*, 641–651. [CrossRef]
27. Wang, S.Y.; Wang, L.; Liu, M.; Xu, Y. An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem. *Int. J. Prod. Econ.* **2013**, *145*, 387–396. [CrossRef]
28. Naderi, B.; Ruiz, R. A scatter search algorithm for the distributed permutation flowshop scheduling problem. *Eur. J. Oper. Res.* **2014**, *239*, 323–334. [CrossRef]
29. Xu, Y.; Wang, L.; Wang, S.; Liu, M. An effective hybrid immune algorithm for solving the distributed permutation flow-shop scheduling problem. *Eng. Optim.* **2013**, *46*, 1269–1283. [CrossRef]
30. Bargaoui, H.; Driss, O.B.; Ghedira, K. A novel chemical reaction optimization for the distributed permutation flowshop scheduling problem with makespan criterion. *Comput. Ind. Eng.* **2017**, *111*, 239–250. [CrossRef]
31. Pan, Q.K.; Gao, L.; Wang, L.; Liang, J.; Li, X.Y. Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem. *Expert Syst. Appl.* **2019**, *124*, 309–324. [CrossRef]
32. Ruiz, R.; Pan, Q.K.; Naderi, B. Iterated Greedy methods for the distributed permutation flowshop scheduling problem. *Omega* **2019**, *83*, 213–222. [CrossRef]
33. Fernandez-Viagas, V.; Framinan, J.M. A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem. *Int. J. Prod. Res.* **2015**, *53*, 1111–1123. [CrossRef]
34. Zhang, G.; Xing, K.; Cao, F. Discrete differential evolution algorithm for distributed blocking flowshop scheduling with makespan criterion. *Eng. Appl. Artif. Intell.* **2018**, *76*, 96–107. [CrossRef]
35. Shao, W.; Pi, D.; Shao, Z. Optimization of makespan for the distributed no-wait flow shop scheduling problem with iterated greedy algorithms. *Knowl. Based Syst.* **2017**, *137*, 163–181. [CrossRef]
36. Pan, W.T. A new fruit fly optimization algorithm: Taking the financial distress model as an example. *Knowl. Based Syst.* **2012**, *26*, 69–74. [CrossRef]
37. Darvish, A.; Ebrahimzadeh, A. Improved fruit-fly optimization algorithm and its applications in antenna arrays synthesis. *IEEE Trans. Antennas Propag.* **2018**, *66*, 1756–1766. [CrossRef]
38. Cong, Y.; Wang, J.; Li, X. Traffic flow forecasting by a least squares support vector machine with a fruit fly optimization algorithm. *Procedia Eng.* **2016**, *137*, 59–68. [CrossRef]
39. Lin, S.M. Analysis of service satisfaction in web auction logistics service using a combination of fruit fly optimization algorithm and general regression neural network. *Neural Comput. Appl.* **2013**, *22*, 783–791. [CrossRef]
40. Meng, T.; Pan, Q.K. An improved fruit fly optimization algorithm for solving the multidimensional knapsack problem. *Appl. Soft Comput.* **2017**, *50*, 79–93. [CrossRef]

41. Zheng, X.L.; Wang, L.; Wang, S.Y. A novel fruit fly optimization algorithm for the semiconductor final testing scheduling problem. *Knowl. Based Syst.* **2014**, *57*, 95–103. [CrossRef]
42. Zheng, X.; Wang, L. A knowledge-guided fruit fly optimization algorithm for dual resource constrained flexible job-shop scheduling problem. *Int. J. Prod. Res.* **2018**, *54*, 1–13. [CrossRef]
43. Li, J.Q.; Pan, Q.K.; Mao, K. A hybrid fruit fly optimization algorithm for the realistic hybrid flowshop rescheduling problem in steelmaking systems. *IEEE Trans. Autom. Sci. Eng.* **2016**, *13*, 932–949. [CrossRef]
44. Deng, G.L.; Yu, H.Y.; Zheng, S.M. An enhanced discrete artificial bee colony algorithm to minimize the total flow time in permutation flowshop scheduling with limited buffers. *Math. Probl. Eng.* **2016**, *2016*, 1–11. [CrossRef]
45. Holm, S. A simple sequentially rejective multiple test procedure. *Scand. J. Stat.* **1979**, *6*, 65–70. [CrossRef]