# A DISCRETE BUTTERFLY-INSPIRED OPTIMIZATION ALGORITHM FOR SOLVING PERMUTATION FLOW-SHOP SCHEDULING PROBLEMS

*X. Qi*[*], *Z. Yuan*[*], *X. Han*[*], *S. Liu*[†*]

**Abstract:** Permutation Flow-Shop Scheduling Problem (PFSP) which exists in many manufacturing systems is a classic combinatorial optimization problem. Studies have shown that the PFSP including more than three machines belongs to the NP-hard problems and is difficult to solve. Based on a new bio-inspired algorithm – Artificial Butterfly Optimization (ABO) algorithm, this paper presents a Discrete Artificial Butterfly Optimization (DABO) algorithm to find the permutation that gives the smallest completion time or the smallest total flow time. The performance of the proposed algorithm is tested on well-known benchmark suites of Car, Reeves and Taillard. The experimental results show that the proposed algorithm is able to provide very promising and competitive results on most benchmark functions. The DABO algorithm is then employed for one production optimization problem.

Key words: *Artificial Butterfly Optimization, artificial bee colony algorithm, particle swarm optimization, Permutation Flow-Shop Scheduling Problem*

*Received: June 18, 2019* **DOI:** 10.14311/NNW.2020.30.015
*Revised and accepted: August 30, 2020*

## 1. Introduction

Permutation Flow-Shop Scheduling Problem (PFSP) is a classic combinatorial optimization problem. The typical PFSP is described below [8]: A set of $n$ jobs must be performed on $m$ machines. All jobs have the same order on each machine. One job is processed by only one machine at any time. One machine processes only one job at a time. The processing time on any machine is known and deterministic. The processing time of individual jobs on different machines could be and usually is different. Travel time between two consecutive machines is negligible. The objective is to find an optimal permutation for the minimum completion time or the minimum total flow time.

[*]Xiangbo Qi – Corresponding author; Zhonghu Yuan; Xiaowei Han; School of Mechanical Engineering, Shenyang University, Shenyang 110044, China, E-mail: ustcdragon@126.com, drqixiangbo@gmail.com

[†]Shixin Liu; School of Information Science and Engineering, Northeastern University, Shenyang 110004, China.

 

Suppose $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ is a permutation of $n$ jobs. $P(\pi_i, j)$ denotes the processing time of job $\pi_i$ on machine $j$. The completion time of job $\pi_i$ on machine $j$ is denoted by $C(\pi_i, j)$. The objective function can be the minimization of the makespan ($C_{\max}$) or the total flow time ($TFT$), and so on. The completion time for the $n$-job and $m$-machine problem is computed as follows:

$$C(\pi_1, 1) = P(\pi_1, 1), \tag{1}$$

$$C(\pi_i, 1) = C(\pi_{i-1}, 1) + P(\pi_i, 1), i = 2, \ldots, n, \tag{2}$$

$$C(\pi_1, j) = C(\pi_1, j-1) + P(\pi_1, j), j = 2, \ldots, m, \tag{3}$$

$$C(\pi_i, j) = \max\{C(\pi_{i-1}, j), C(\pi_i, j-1)\} + P(\pi_i, j), i = 2, \ldots, n, j = 2, \ldots, m. \tag{4}$$

The objective is to find the optimal permutation $\pi^*$

$$C_{\max}(\pi^*) \leq C(\pi_n, m) \tag{5}$$

or

$$TFT = \sum_{i=1}^{n} C(\pi_i, m). \tag{6}$$

$C_{\max}$ and $TFT$ are widely used for measuring the quality of a schedule. Makespan is important for effective utilization of resources. Makespan is defined as the completion time of the last job to leave the system. $TFT$ is defined as the total time spent by the jobs in the production system. Total flow time is one of the most important performance measures. $TFT$ can help to effective utilization of resources, rapid turn-around of jobs, and minimization of work-in-process inventory costs [3]. A little improvement for makespan or $TFT$ can save much processing time and enhance production efficiency. Just because of this, PFSP has been studied in optimization theoretic research and engineering application field [10, 20, 25].

Studies have shown that the PFSP including more than three machines belongs to the NP-hard problems and is difficult to solve. PFSP with the objective of minimizing makespan has been proven to be a NP-complete problem [1, 7, 17]. PFSP with the objective of minimizing total flow time has been proven to be a NP-complete problem [4]. Lots of different algorithms have been proposed to solve the PFSP. The specific algorithms are sensible to the number of machines [22]. So, lots of heuristic algorithms for PFSPs are proposed [4, 5, 10, 11]. Among these existing heuristics, the Nawaz-Enscore-Ham (NEH) algorithm [10] is a well-known constructive method. The NEH algorithm was claimed to be good for minimizing the makespan, but not good for minimizing the total flow time. However, meta-heuristic algorithms can obtain effective results. In recent decade, an increasing number of research papers focusing on meta-heuristics for PFSP have been published [12, 20, 23, 25]. For example, particle swarm optimization (PSO) algorithm was inspired by the swarm behavior of birds and fish [6]. Tasgetiren employed PSO to solve PFSP [20]. In the literature, a very efficient local search, called variable neighborhood search (VNS), was embedded in the PSO algorithm to solve the well known benchmark suites. Artificial bee colony (ABC) algorithm is a popular swarm intelligence based algorithm by modeling foraging behaviors of honeybee colony. In order to find the permutation that gives the smallest total flow time, a

discrete artificial bee colony algorithm hybridized with a variant of iterated greedy algorithms is proposed [23]. Teaching learning based optimization (TLBO) proposed by Rao [14, 15] is a new optimization method simulating a classical school learning process. Tasgetiren combined a variable neighborhood search method for fast solution improvement and TLBO for solution evolution and proposed a hybrid TLBO algorithm (HTLBO) [25]. Qi in [13] introduced an Artificial Butterfly Optimization (ABO) algorithm to optimize continuous functions. Numerical comparisons demonstrated that the ABO algorithm is able to provide very promising and competitive results on most benchmark functions. As there is no detailed work that describes the use of the ABO algorithm to deal with the PFSP, we present a novel discrete ABO (DABO) algorithm for solving the PFSP in this paper.

The remainder of the article is organized as follows. Section 2 introduces the original ABO algorithm. Section 3 proposes a novel discrete artificial butterfly optimization algorithm, gives the pseudo code and the details of the new algorithm. Section 4 gives experiment details of the new algorithm on three well-known test suites and discusses the computational results over the test suites. A real application is also presented in Section 4. Finally, section 5 gives the conclusions.

## 2. Original artificial butterfly algorithm

Artificial Butterfly Optimization (ABO) algorithm is one of recently developed meta-heuristic algorithms developed by Qi [13]. The ABO algorithm is based on the mate-finding strategy of some butterfly species. It classifies the artificial butterflies into two groups. Two groups of artificial butterflies including sunspot group and canopy group are employed for simulating the flight strategies. The sunspot group has better fitness and the canopy group has worse fitness. The purpose of all the artificial butterflies is to find better locations.

There are three flight modes including sunspot flight mode, canopy flight mode and free flight mode. The sunspot group employs sunspot flight mode. The canopy group employs canopy flight mode and free flight mode. These three modes can be given different flight strategies. That is to say, if the flight strategies of artificial butterflies are redefined, ABO represents a new algorithm. The pseudo code of the ABO algorithm is given in Algorithm 1.

## 3. Discrete artificial butterfly algorithm

In order to solve PFSP problems, we redefine the flight strategies of artificial butterflies and a discrete artificial butterfly algorithm (DABO) is proposed.

A. Solution representation Firstly, the solution representation is given. The solution is represented by a permutation of jobs $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$. For example, the solution $[6, 5, 3, 2, 4, 1]$ for PFSP with six jobs means the processing sequence $6, 5, 3, 2, 4, 1$. This solution representation is simple.

B. Population initialization The initial solutions are constructed according to the smallest position value (SPV) rule [4]. $X = (X_1, X_2, \ldots, X_n)$ corresponds to

**Algorithm 1** Pseudo code of the ABO algorithm.

**Step 1: Initialization phase**
Initialize the locations of butterfly population
Evaluate the fitness of every butterfly
**Step 2: Division phase**
Sort all butterflies by their fitness
Select some butterflies with better fitness to form sunspot group, the rest form canopy group
**Step 3: Sunspot group phase**
**for** For each butterfly in sunspot group **do**
  Fly to one new location according to **sunspot flight mode**
  Evaluate the fitness of the new sunspot
  Apply greedy selection on the original location and the new one
**end for**
**Step 4: Canopy group phase**
**for** For each butterfly in canopy group **do**
  Fly to one randomly selected sunspot butterfly according to **canopy flight mode**
  Evaluate the fitness
  **if** better fitness **then**
    Apply greedy selection on the original location and the new one
  **else**
    Fly to new location according to **free flight mode**
  **end if**
**end for**
**Step 5:** If meet termination condition, stop the procedure; otherwise, go to step 2
**Step 6: Post processing phase**

a $n$-dimensional real-valued vector. $X$ is produced randomly according to the following formula:

$$X_i = X_{\min} + (X_{\max} - X_{\min}) * rand, i = 1, 2, \ldots, n, \tag{7}$$

where $X_{\min} = -1$, $X_{\max} = 1, rand$ is a uniform random number between 0 and 1.

$\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ corresponds to a permutation of $n$ jobs. SPV represents a direct relationship between $X$ and $\pi$. Tab. I gives an example of converting $X$

| Dimension,$j$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $X$ | 0.28 | −0.92 | 0.98 | −0.52 | 0.69 | −0.99 |
| $\pi$ | 6 | 2 | 4 | 1 | 5 | 3 |

**Tab. I** *SPV based solution representation of individual.*

to $\pi$. According to the SPV rule, the smallest position value is $X_6 = -0.99$, so the dimension $j = 6$ is assigned to be the first job $\pi_1 = 6$ in the permutation $\pi$. The second smallest position value is $X_2 = -0.92$, so the dimension $j = 2$ is assigned to be the second job $\pi_2 = 2$ in the permutation $\pi$, and so on.

C. Sunspot flight mode

According to the original ABO algorithm, the sunspot butterflies have better fitness and the sunspot butterflies are responsible for exploiting the better region. According to this, we propose some strategies to enhance the local search ability of sunspot butterflies. First, each sunspot butterfly generates a new solution using a crossover operator. Fig. 1 gives an example of crossover

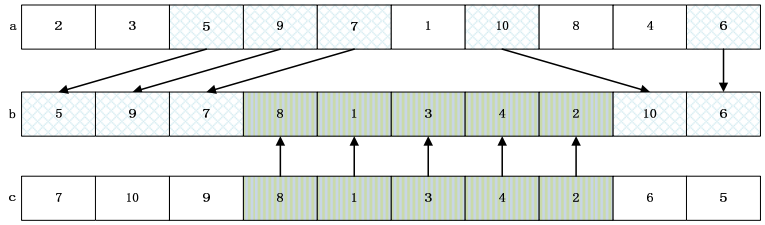


**Fig. 1** *Crossover operator.*

learning. Sunspot butterfly $a$ flies to a new position $b$ towards a neighbor sunspot butterfly $c$. Second, two discrete algorithms including inserting algorithm [23] and local search algorithm [25] are applied to the new solution. Algorithm 2 gives the pseudo code of inserting algorithm. Algorithm 3 gives

**Algorithm 2** Pseudo code of inserting algorithm.

Input an initial solution $\pi_0$
Set $i = 1$ do
$j = i + 1$ do
$\pi_1 = \pi_0$
Insert the job in the position $i$ to the position $j$ in $\pi_1$
**if** $(fitness(\pi_1) > fitness(\pi_0))$ **then**
  $\pi_0 = \pi_1$
  $i = 1$
  $j = i + 1$
**else**
  $j = j + 1$
**end if**
**while** $(j < n)$ **do**
  $i = i + 1$
**end while**
**while** $(i < n)$ **do**
  return $\pi_0$
**end while**

**Algorithm 3** Pseudo code of local search algorithm.

```
input an initial solution π_0
Set initial parameters: T_0, T_f, C_r, K_max, k = 0
π_1 = π_0
while (k < K_max) do
  while (T_0 > T_f) do
    Produce a new solution π_2 from the neighborhood of π_1
    if (fitness(π_2) > fitness(π_1)) then
      π_1 = π_2
    end if
    T_0 = C_r × T_0
  end while
  if (fitness(π_1) > fitness(π_0)) then
    π_0 = π_1
  else
    k = k + 1
  end if
end while
return π_0
```

the pseudo code of the local search algorithm. It is worth noting that the simulating annealing method (SA) [9, 21] is employed in the local search algorithm. SA has several parameters: the initial temperature $T_0$, the final temperature $T_f$ and the cooling rate $C_r$. Readers can refer to the literature we have mentioned for the details of the algorithm.

In the original ABO algorithm, a sunspot butterfly flies to a new position towards a neighbor sunspot butterfly. For PFSP, a sunspot butterfly employs a crossover operator to generate a new solution. The crossing process is as follows:

Step 1: Randomly select two different jobs from the neighbor solution (Sunspot butterfly)

Step 2: Copy jobs between the two jobs to the new solution

Step 3: Copy the rest of the original solution to the rest position of the new solution according to the order in the original solution

D. Canopy flight mode

According to the original ABO algorithm, the canopy butterflies have worse fitness and the canopy butterflies are responsible for exploring new regions. The canopy butterflies gain experience from the sunspot butterflies. This process also uses the method as shown in Fig. 1.

E. Free flight mode

A mutation operator is employed. Randomly select two different jobs from a solution and swap them. Fig. 2 illustrates the mutation process.

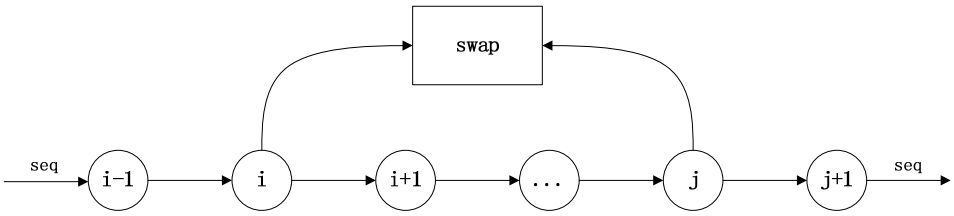


**Fig. 2** *Mutation operator.*

Algorithm 4 gives the pseudo code of the DABO algorithm. The flowchart of the DABO algorithm is shown in Fig. 3. The proposed algorithm is composed of five phases: initialization phase, division phase, sunspot group phase, canopy group phase and post processing phase. The initialization phase initializes each solution of the population and evaluates the fitness of each solu-

**Algorithm 4** Pseudo code of the DABO algorithm.

**Step 1: Initialization phase**
Initialize the butterfly population using SPV rule
Evaluate the fitness of every butterfly
**Step 2: Division phase**
Sort all butterflies by their fitness
Select some butterflies with better fitness to form sunspot group, the rest form canopy group
**Step 3: Sunspot group phase**
**for** butterfly in sunspot group **do**
  Generate a new solution using crossover operator
  Compute the fitness of the solution
  Apply greedy selection on the original solution and the new one
  Generate a new solution using inserting algorithm
  Generate a new solution using local search algorithm
**end for**
**Step 4: Canopy group phase**
**for** each butterfly in canopy group **do**
  Generate a new solution using crossover operator
  Compute the fitness of the solution
  **if** (better) **then**
    Apply greedy selection on the original solution and the new one
  **else**
    Generate a new solution using mutation operator
  **end if**
**end for**
**Step 5:** If meet termination condition, stop the procedure; otherwise, go to step 2
**Step 6: Post processing phase**

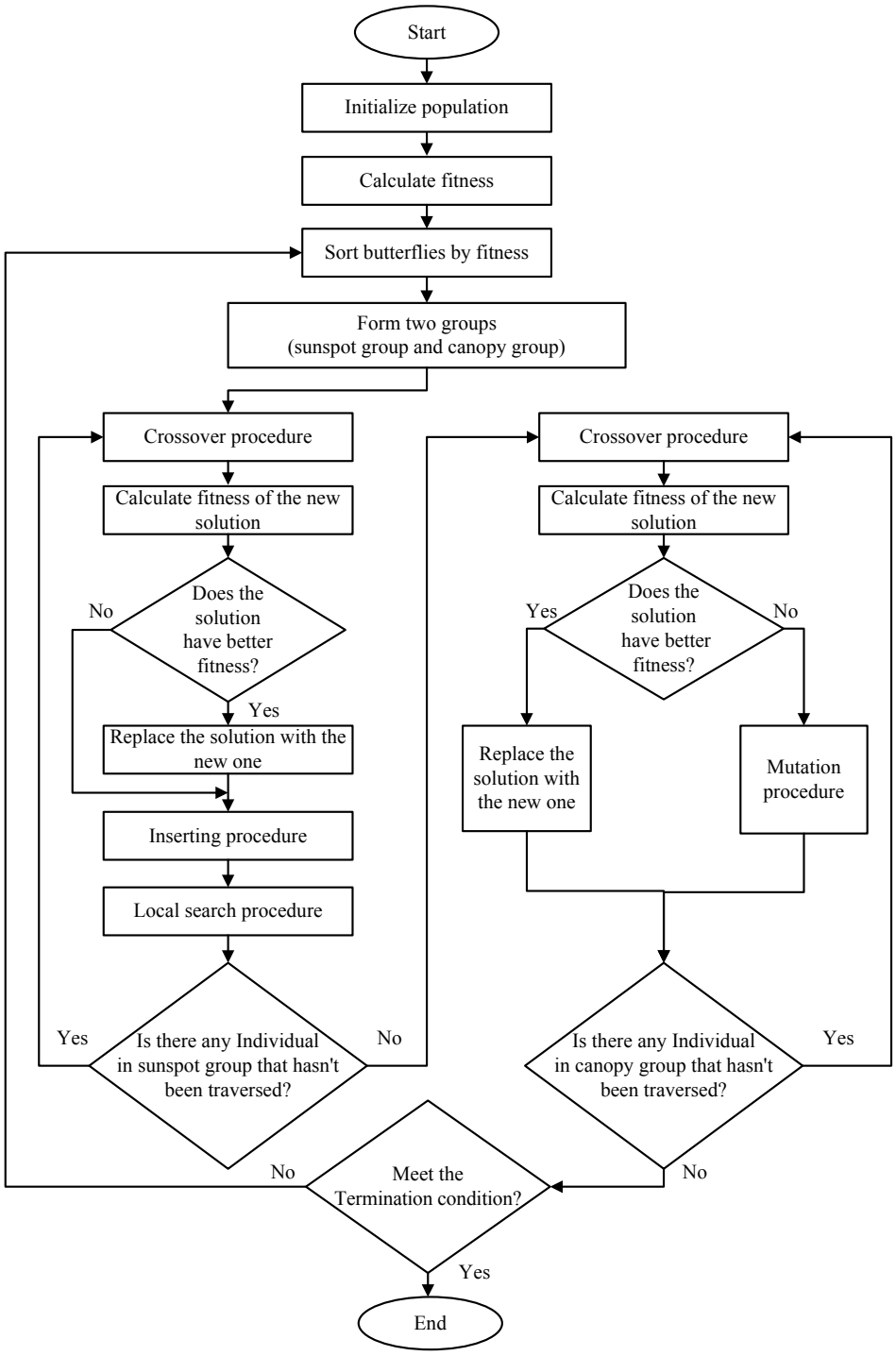


**Fig. 3** *Flowchart of the DABO Algorithm.*

tion. In the division phase, the population is divided into two subpopulations according to the fitness of the solution. The subpopulation with better fitness form sunspot group, while the rest form canopy group. The sunspot group is responsible for the exploitation of solutions while the canopy group is responsible for the exploration of solutions. Exploitation and exploration are two important aspects of meta-heuristic algorithms. There is a the balance between the right amount of exploration and the right degree of exploitation.

Exploitation nature of the sunspot group phase is its distinguished feature. Because the sunspot group is composed of individuals with good fitness, this phase can concentrate on the local regions or neighborhood of these individuals. So, the solution is generated by a variety of search strategies (crossover, inserting and local search). These search strategies start from a promising solution respectively and progressively improve it by applying a series of local modifications.

In the canopy group phase, sunspot butterflies provide information sharing for canopy butterflies, in which canopy butterflies can be optimized. Meanwhile, a mutation operator is introduced to enhance the diversity of solutions. By using mutation operator, the canopy butterfly which is not improved constructs a solution to the problem. Mutation operator makes this stage have the characteristics of exploration.

With the iteration, the division phase, the sunspot group phase and the canopy group phase are repeated. The use of combination of these three phases leads to high quality solutions.

# 4. Validation and Comparison

## 4.1 Benchmark Problems

To test the performance of the DABO algorithm, computational simulation is carried out with 69 well-known benchmark problems of PFSPs. The benchmark suites include Car benchmark suite [2], Reeves benchmark suite [16] and Taillard benchmark suite [19]. The Car benchmark suite includes 8 problems which are denoted by car1, car2, till car8. The Reeves benchmark suite includes 21 problems which are denoted by rec1, rec3, rec5 through rec41. The Taillard benchmark suite includes 40 problems which are denoted by ta001, ta002, till ta040. Reeves instances and Taillard instances are widely adopted by other researchers to test their algorithms for solving middle-sized and large-sized problems in many works [24, 26].

Reeves benchmark suite includes 7 groups which are denoted by $20 \times 5, 20 \times 10, 20 \times 15, 30 \times 10, 30 \times 15, 50 \times 10$ and $75 \times 20$. Each Reeves group includes 3 instances. Taillard benchmark suite include 4 groups which are denoted by $20 \times 5, 20 \times 10, 20 \times 20$ and $50 \times 5$. Each Taillard group includes 10 instances.

In order to compare the merits of the proposed algorithm with the other algorithms, some parameters are chosen as the reference standard, as shown as follows.

Best Relative Deviation ($BRD$):

$$BRD = \frac{\min(C_{\max} - C^*)}{C^*} \times 100\,\%, \tag{8}$$

Average Relative Deviation ($ARD$):

$$ARD = \frac{avg(C_{\max} - C^*)}{C^*} \times 100\,\%, \tag{9}$$

Worst Relative Deviation ($WRD$):

$$WRD = \frac{\max(C_{\max} - C^*)}{C^*} \times 100\,\%, \tag{10}$$

Best Average Relative Percentage Deviation ($BARPD$):

$$BARPD = \left( \sum_{i=1}^{k} \frac{\min\left(C_{\max}\right) - C^*}{C^*} \right) / k, \tag{11}$$

Average Average Relative Percentage Deviation ($AARPD$):

$$AARPD = \left( \sum_{i=1}^{k} \frac{avg\left(C_{\max}\right) - C^*}{C^*} \right) / k. \tag{12}$$

In Eq. (8), Eq. (9), Eq. (10), Eq. (11) and Eq. (12), $C^*$ denotes the best known makespan for the instance and $C_{\max}$ denotes the makespan of a solution generated by a given algorithm. It is worth noting that $C^*$ is replaced with $TFT^*$ and $C_{\max}$ is replaced with $TFT$ when the objective is to compute the total flow time. $k$ denotes the number of instances in different groups in a test set. The lower $BARPD$ is, the higher performance is. The lower $AARPD$ is, the higher performance is.

## 4.2 Experiment Sets

The population size is $10 \times m$. The termination condition is $0.1 \times n \times m$ seconds. In order to do meaningful statistical analysis, each algorithm runs for 10 times and takes the mean value and the best value as the final result. PSOvns [20], HTLBO [25] and NEH [10] were employed for comparison. The control parameters for the comparison algorithms were set according to the original literatures. For DABO, $T_0 = 1000, T_f = 1, C_r = 0.9$.

In this paper, all algorithms were implemented in Java using computer with Intel Core i5-2450M CPU, 2.5 GHz processor and 2 GB RAM. The operating system of the computer is Windows 7.

## 4.3 Experiment Results and Analysis

1) Results for the Car instances with makespan criterion

   Both the DABO algorithm and the HTLBO algorithm generate 8 best $BRD$ values. The PSOvns algorithm generates 7 best $BRD$ values. The DABO algorithm generates 8 best $ARD$ values. The HTLBO algorithm generates 6 best $ARD$ values.

   Fig. 4 gives the line chart of $BARPD$ with different algorithms on Car instances with makespan criterion. Fig. 5 gives the line chart of $AARPD$ with different algorithms on Car instances with makespan criterion. It is clear from Figs. 4 and 5 that the DABO is the winner.

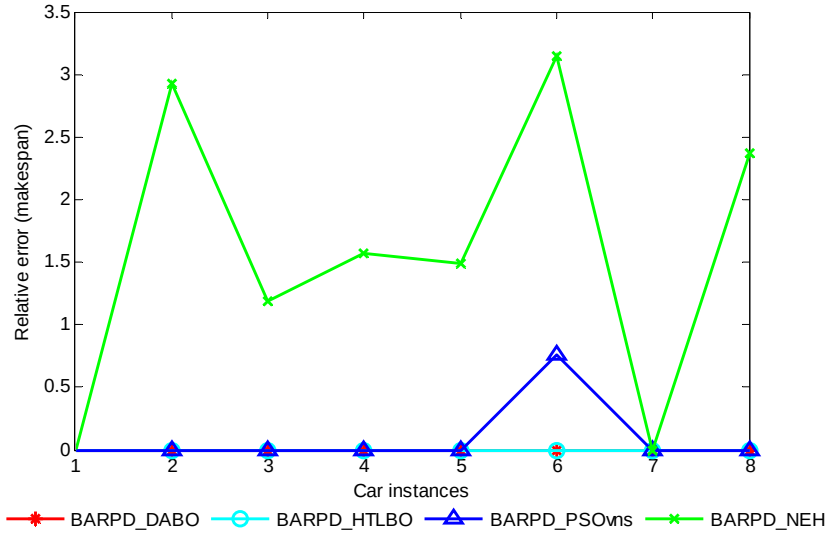


**Fig. 4** *Line chart of BARPD with different algorithms on Car instances. Here, 1–8 correspond to* $11\times5, 13\times4, 12\times5, 14\times4, 10\times6, 8\times9, 7\times7$ *and* $8\times8$*, respectively.*

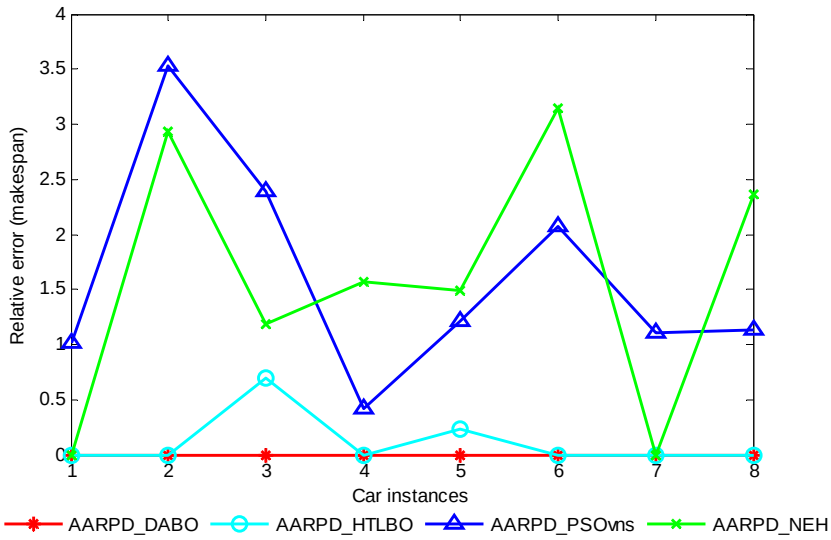


**Fig. 5** *Line chart of AARPD with different algorithms on Car instances. Here, 1–8 correspond to* $11\times5, 13\times4, 12\times5, 14\times4, 10\times6, 8\times9, 7\times7$ *and* $8\times8$*, respectively.*

2) Results for the Reeves instances with makespan criterion

The DABO algorithm generates 20 best *BRD* values. The HTLBO algorithm generates 8 best *BRD* values. The DABO algorithm generates 19 best *ARD* values. The HTLBO algorithm generates 3 best *ARD* values.

Fig. 6 gives the line chart of $BARPD$ with different algorithms on Reeves instances with makespan criterion. Fig. 7 gives the line chart of $AARPD$ with different algorithms on Reeves instances with makespan criterion. It is clear from Figs. 6 and 7 that the DABO is the winner.

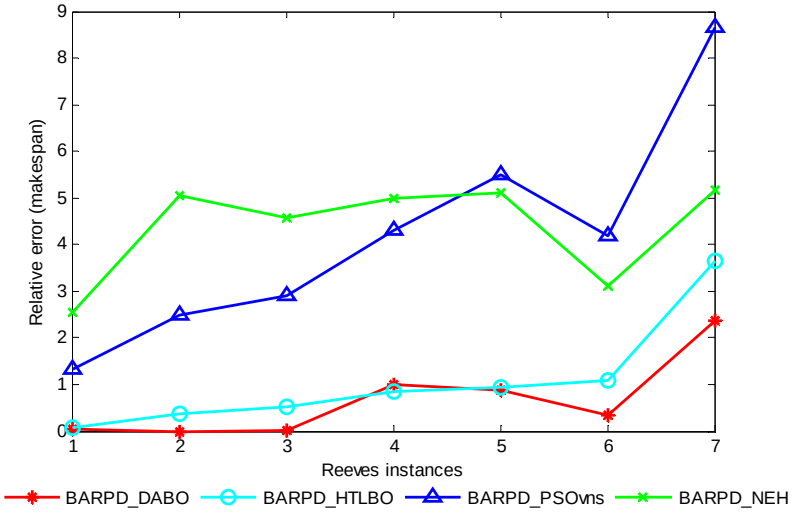


**Fig. 6** *Line chart of $BARPD$ with different algorithms on Reeves instances. Here, 1–7 correspond to $20 \times 5, 20 \times 10, 20 \times 15, 30 \times 10, 30 \times 15, 50 \times 10$ and $75 \times 20$, respectively.*

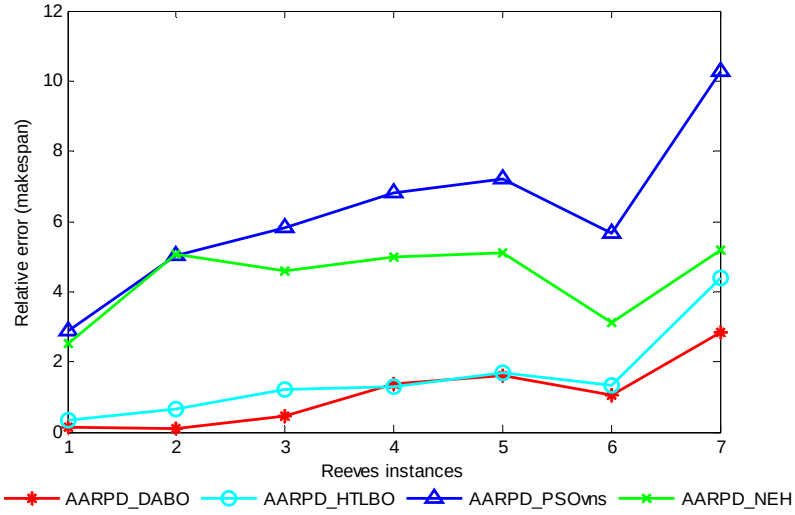


**Fig. 7** *Line chart of $AARPD$ with different algorithms on Reeves instances. Here, 1–7 correspond to $20 \times 5, 20 \times 10, 20 \times 15, 30 \times 10, 30 \times 15, 50 \times 10$ and $75 \times 20$, respectively.*

3) Results for the Taillard instances with makespan criterion

The DABO algorithm generates 34 best $BRD$ values. The HTLBO algorithm generates 17 best $BRD$ values. The DABO algorithm generates 36 best $ARD$ values. The HTLBO algorithm generates 5 best $ARD$ values.

Fig. 8 gives the line chart of $BARPD$ with different algorithms on Taillard instances with makespan criterion. Fig. 9 gives the line chart of $AARPD$

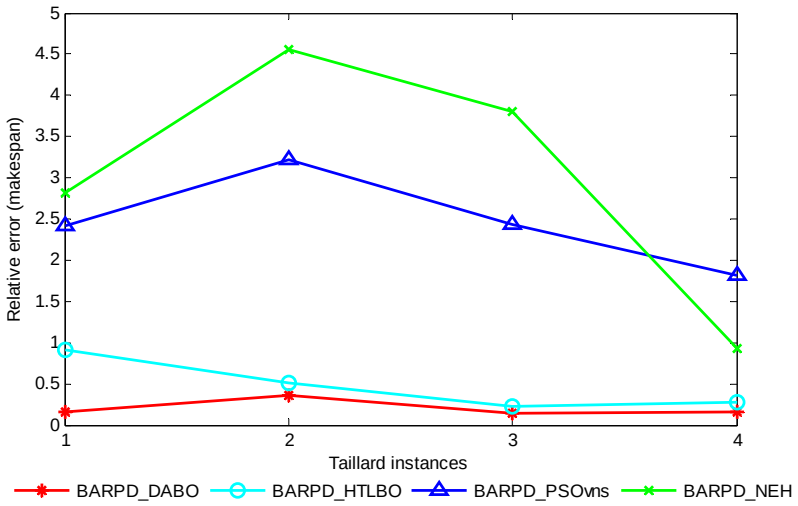


**Fig. 8** *Line chart of $BARPD$ with different algorithms on Taillard instances. Here, 1–4 correspond to $20 \times 5, 20 \times 10, 20 \times 20$ and $50 \times 5$, respectively.*

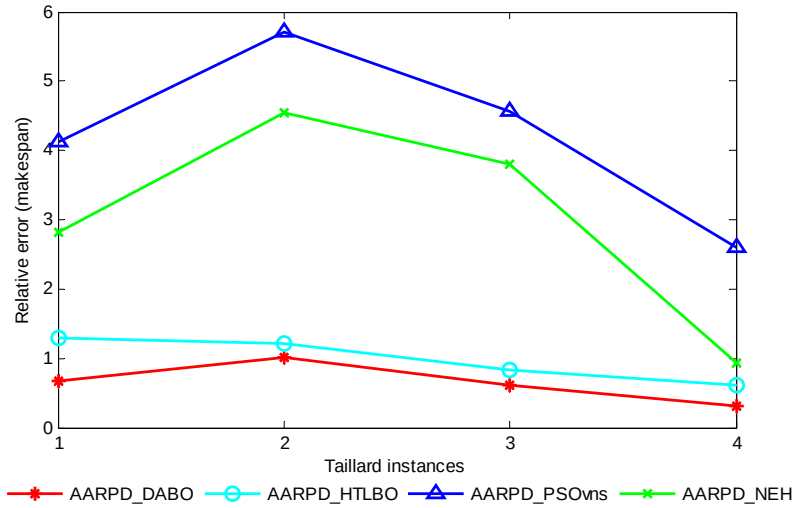


**Fig. 9** *Line chart of $AARPD$ with different algorithms on Taillard instances. Here, 1–4 correspond to $20 \times 5, 20 \times 10, 20 \times 20$ and $50 \times 5$, respectively.*

with different algorithms on Taillard instances with makespan criterion. It is clear from Figs. 8 and 9 that the DABO is the winner.

4) Results for the Taillard instances with $TFT$ criterion

The DABO algorithm generates 33 best $BRD$ values. The HTLBO algorithm generates 14 best $BRD$ values. The DABO algorithm generates 39 best $ARD$ values. The HTLBO algorithm generates 1 best $ARD$ values.

Fig. 10 gives the line chart of $BARPD$ with different algorithms on Taillard instances with $TFT$ criterion. Fig. 11 gives the line chart of $AARPD$ with

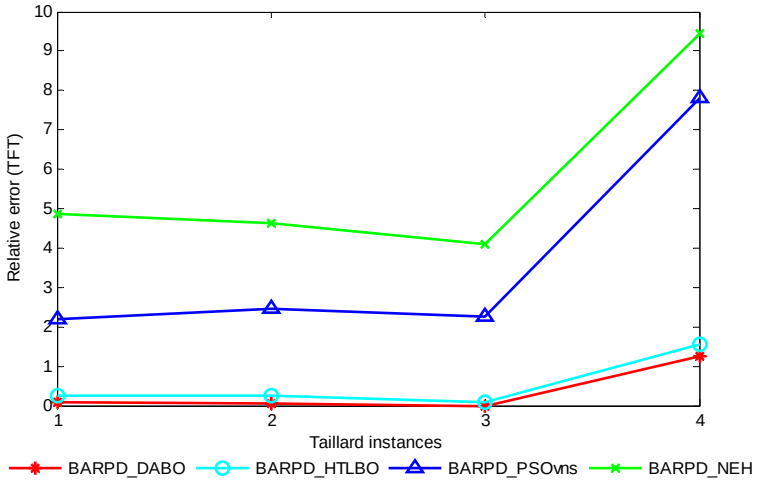


**Fig. 10** *Line chart of $BARPD$ with different algorithms on Taillard instances ($TFT$). Here, 1–4 correspond to $20 \times 5, 20 \times 10, 20 \times 20$ and $50 \times 5$, respectively.*

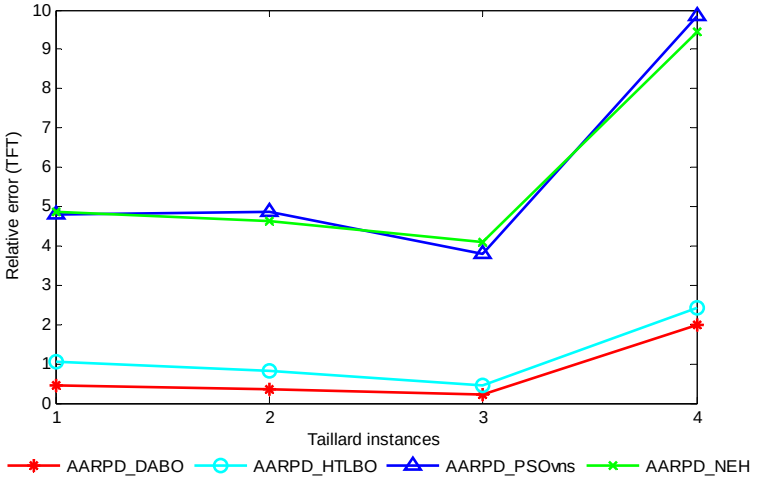


**Fig. 11** *Line chart of $AARPD$ with different algorithms on Taillard instances ($TFT$). Here, 1–4 correspond to $20 \times 5, 20 \times 10, 20 \times 20$ and $50 \times 5$, respectively.*

different algorithms on Taillard instances with $TFT$ criterion. It is clear from Figs. 10 and 11 that the DABO is the winner.

### 4.3.1 Engineering optimization example

The aluminum extrusion products are widely applied in transportation industry and civil construction industry recently because they have the advantages of light weight, high strength and attractive appearance. The optimization of extrusion production line can bring good economic benefits. During a production period, a variety of extrusion products passes through extruding machine, drawing machine, sawing machine, shaping machine, finished product saw and packing machine in turn. Tab. II gives the processing time of 10 kinds of extrusion products. According to the results of the computational evaluation given by Ruiz and Maroto [18], the NEH algorithm has been proved one of the most successful constructive heuristics which can obtain comparable results against some modern constructive methods. In this section, the NEH algorithm is employed as comparison.

| Batch | Flag | Extruding machine | Drawing machine | Sawing machine | Shaping machine | Finished product saw | Packing machine |
|---|---|---|---|---|---|---|---|
| PC15061902-1 | 1 | 174 | 121 | 183 | 102 | 190 | 150 |
| PC15061903-1 | 2 | 120 | 176 | 174 | 199 | 160 | 115 |
| PC15061904-1 | 3 | 167 | 148 | 160 | 166 | 138 | 130 |
| PC15061905-1 | 4 | 97 | 76 | 64 | 110 | 181 | 173 |
| PC15061906-1 | 5 | 87 | 86 | 80 | 90 | 83 | 75 |
| PC15061907-1 | 6 | 80 | 42 | 99 | 70 | 90 | 93 |
| PC15061908-1 | 7 | 69 | 72 | 54 | 80 | 90 | 160 |
| PC15061909-1 | 8 | 69 | 120 | 24 | 96 | 89 | 62 |
| PC15061910-1 | 9 | 50 | 88 | 46 | 63 | 76 | 150 |
| PC15061911-1 | 10 | 95 | 61 | 53 | 82 | 74 | 97 |

**Tab. II** *The processing time for 10 kinds of extrusion products.*

Fig. 12 shows the optimal job sequence with makespan criterion obtained by DABO over 10 independent runs. The job sequence is $4, 9, 6, 1, 7, 2, 10, 3, 8, 5$. The makespan of the job sequence is 1796. Fig. 13 shows the optimal job sequence with makespan criterion obtained by NEH over 10 independent runs. The job sequence is $9, 6, 7, 1, 4, 2, 10, 3, 5, 8$. The makespan of the job sequence is 1802.

Fig. 14 shows the optimal job sequence with $TFT$ criterion obtained by DABO over 10 independent runs. The job sequence is $9, 8, 10, 6, 5, 7, 4, 1, 3, 2$. The $TFT$ of the job sequence is 10299. Fig. 15 shows the optimal job sequence with $TFT$ criterion obtained by NEH over 10 independent runs. The job sequence is $9, 6, 8, 10, 7, 5, 4, 1, 3, 2$. The $TFT$ of the job sequence is 10366.

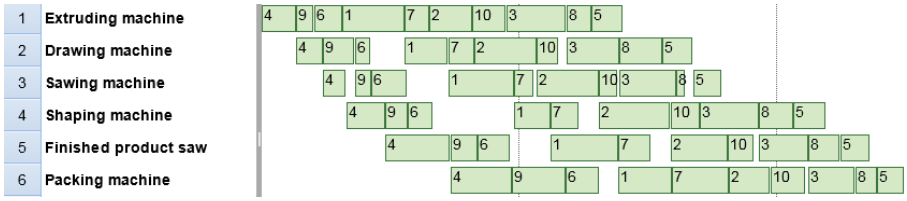


**Fig. 12** *Gantt chart using DABO under makespan criterion.*

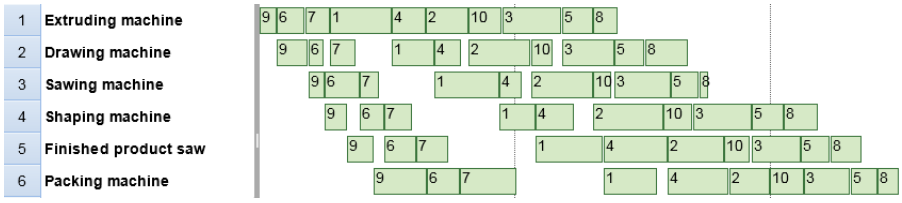


**Fig. 13** *Gantt chart using NEH under makespan criterion.*

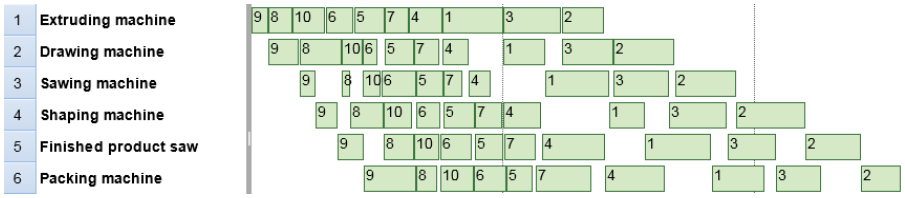


**Fig. 14** *Gantt chart using DABO under TFT criterion.*

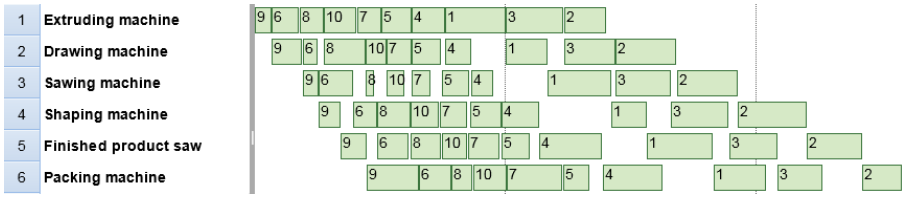


**Fig. 15** *Gantt chart using NEH under TFT criterion.*

# 5. Conclusion

This paper proposes a new Discrete Artificial Butterfly Optimization (DABO) algorithm to address the Permutation Flow-Shop Scheduling Problem (PFSP) with the minimization of makespan and total flow time. In order to solve PFSPs, the flight strategies of artificial butterflies are redefined. The sunspot butterflies employed three strategies based on crossover, inserting and local searching. The canopy butterflies employed two strategies based on crossover and mutation.

Exploitation and exploration are two important aspects of meta-heuristic algorithms. There is a the balance between the right amount of exploration and

the right degree of exploitation. Firstly, with the iteration, the DABO algorithm has the structure of dynamically dividing two subpopulations. Moreover, the two subpopulations are divided according to individual fitnesses. Secondly, the proposed algorithm let the high fitness subpopulation concentrate on the exploitation of solutions and the low fitness subpopulation concentrate on the exploration of solutions. In the sunspot group phase, the crossover procedure produces better solutions between solutions with better fitness. The inserting procedure is an enhanced search for the high-quality solutions found previously. The local searching is reintensification of search around previously encountered high-quality solutions. In the canopy group phase, each individual achieves diversification of search by crossing with randomly selected high fitness individuals and mutation operation. With these strategies, DABO produce better solutions under the same evaluation times.

For makespan criterion, the DABO algorithm has been tested against the other 3 algorithms based on Car suite, Reeves suite and Taillard suite. For total flow time criterion, the DABO algorithm has been tested against the other 3 algorithms based on Taillard suite. All experimental results show the effectiveness of the DABO algorithm. Then DABO is employed to solve a practical engineering problem. The proposed algorithm can be applied to independent scheduling software or job scheduling module of manufacturing execution system (MES) software. The actual application environment is more complex than the model in this paper, but as far as the algorithm itself is concerned, it does not need to be modified much when the algorithm is applied to the optimization problem. For the PFSP, we should find out the processing time of different jobs on different machines, which can be used to calculate the objective function. It should be noted that using the knowledge of experienced personnel can improve the quality of the initial solutions.

Future research efforts will be focused on finding new flight strategies to build more effective algorithms for multi-objective flow shop scheduling problems and applying the proposed algorithm to solve practical engineering problems.

## Acknowledgement

This work is supported by the National Key R&D Program of China (2017YFB0306-401) and National Natural Science Foundation of China (61573089). And the authors are very grateful to the anonymous reviewers for their valuable suggestions and comments to improve the quality of this paper.

# References

[1] BLAZEWICAZ J., LENSTRA J.K., KAN A. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*. 1983, 5(1), pp. 11–24, doi: 10.1016/0166-218x(83)90012-4.

[2] CARLIER J. Ordonnancements a contraintes disjunctives. *RAIRO – Operations Research*. 1978, 12(4), pp. 333–350, doi: 10.1051/ro/1978120403331.

[3] FRAMINAN J.M., LEISTEN R. An efficient constructive heuristic for flowtime minimisation in permutation flow shops. *Omega-International Journal of Management Science*. 2003, 31(4), pp. 311–317, doi: 10.1016/S0305-0483(03)00047-1.

[4] GAREY M.R., JOHNSON D.S., SETHI R. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*. 1976, 1(2), pp. 117–129, doi: 10.2307/3689278.

[5] GUPTA J. A functional heuristic algorithm for the flow shop scheduling problem. *Journal of the Operational Research Society*. 1971, 22(1), pp. 39–47, doi: 10.2307/3008015.

[6] KENNEDY J., EBERHART R. Particle swarm optimization. In: *Proceedings of IEEE Int. Conference on Neural Network*. 1995, pp. 1942–1948.

[7] LENSTRA J.K., Kan, RINNOOY KAN A.H.G., BRUCKER P. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*. 1977, 1, pp. 343–362, doi: 10.1016/S0167-5060(08)70743-X.

[8] LIU Y.F., LIU S.Y. A hybrid discrete artificial bee colony algorithm for permutation flowshop scheduling problem. *Applied Soft Computing*. 2013, 13, pp. 1459–1463, doi: 10.1109/CEC.2010.5586300.

[9] NADERI B., TAVAKKOLI-MOGHADDAM R., KHALILI M. Electromagnetism-like mechanism and simulated annealing algorithms for flowshop scheduling problems minimizing the total weighted tardiness and makespan. *Knowledge-Based Systems*. 2010, 23(2), pp. 77–85, doi: 10.1016/j.knosys.2009.06.002.

[10] NAWAZ M., ENSCORE E.E., HAM I.A. A heuristic algorithm for the m-machine, n-job flow shop sequencing problem. *OMEGA*. 1983, 11(1), pp. 91–95, doi: 10.1016/0305-0483(83)90088-9.

[11] PALMER D.S. Sequencing jobs through a multistage process in the minimum total time: Aquick method of obtaining a near optimum. *Journal of the Operational Research Society*. 1965, 16(1), pp. 101–107, doi: 10.1057/jors.1965.8.

[12] PAN Q ., TASGETIREN M.F., LIANG Y. A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Computers & Industrial Engineering*. 2008, 55(4), pp. 795–816, doi: 10.1016/j.cie.2008.03.003.

[13] QI X.B., ZHU Y.L., ZHANG H. A new meta-heuristic butterfly-inspired algorithm. *Journal of Computational Science*. 2017, 23, pp. 226–239, doi: 10.1016/j.jocs.2017.06.003.

[14] RAO R., SAVSANI V., VAKHARIA D. Teaching–learning-based optimization: a novel method for constrained mechanical design optimization problems, *Computer-Aided Design*. 2011, 43(3), pp. 303-315, doi: 10.1016/j.cad.2010.12.015.

[15] RAO R., SAVSANI V., VAKHARIA D. Teaching–learning-based optimization: an optimization method for continuous non-linear large scale problems. *Information Sciences*. 2012, 183(1), pp. 1–15, doi: 10.1016/j.ins.2011.08.006.

[16] REEVES C.R. A genetic algorithm for flowshop sequencing. *Computers & Operations Research*. 1995, 22(1), pp. 5–13, doi: 10.1016/0305-0548(93)E0014-K.

[17] RINNOOY KAN A.H.G. *Machine Scheduling Problems: Classification, Complexity, and Computations*. The Hague: Martinus Nijhoff, 1976.

[18] RUIZ R., MAROTO C. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*. 2005, 165(2), pp. 479–494, doi: 10.1016/j.ejor.2004.04.017.

[19] TAILLARDd E. Benchmarks for basic scheduling problems. *European Journal of Operational Research*. 1993, 64(2), pp. 278-285, doi: 10.1016/0377-2217(93)90182-m.

[20] TASGETIREN M.F., LIANG Y.C., SEVKLI M., GENCYILMAZ G. A particle swarm optimization algorithm for makespan and total flow time minimization in the permutation flow shop sequencing problem. *European Journal of Operational Research*. 2007, 177(3), pp. 1930–1947, doi: 10.1016/j.ejor.2005.12.024.

[21] TIAN P., MA J., ZHANG D.M. Application of the simulated annealing algorithm to the combinatorial optimization problem with permutation property: An investigation of generation mechanism. *European Journal of Operational Research*. 1999, 118(1), pp. 81–94, doi: 10.1016/s0377-2217(98)00308-7.

[22] TSENG F.T., STAFFORD E.F., GUPTA J.N. An empirical analysis of integer programming formulations for the permutation flowshop. *OMEGA*. 2004, 32(4), pp. 285–293, doi: 10.1016/j.omega.2003.12.001.

[23] TASGETIREN M.F., PAN Q.K., SUGANTHAN P.N., CHEN A.H.L. A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops. *Information Sciences*. 2011, 181(16), pp. 3459–3475, doi: 10.1016/j.ins.2011.04.018.

[24] WANG L., PAN Q.K., SUGANTHAN P.N. A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems. *Computers & Operations Research*. 2010, 37(3), pp. 509–520, doi: 10.1016/j.cor.2008.12.004.

[25] XIE Z.P., ZHANG C.Z., SHAO X.Y., LIN W.W., ZHU H.P. An effective hybrid teaching–learning-based optimization algorithm for permutation flow shop scheduling problem. *Advances in Engineering Software*. 2014, 77, pp. 35–47, doi: 10.1016/j.advengsoft.2014.07.006.

[26] ZHANG Y., LI X., WANG Q. Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization. *European Journal of Operational Research*. 2009, 196(3), pp. 869–876, doi: 10.1016/j.ejor.2008.04.033.