



Effective heuristics and metaheuristics to minimise total tardiness for the distributed permutation flowshop scheduling problem

Ankit Khare & Sunil Agrawal

To cite this article: Ankit Khare & Sunil Agrawal (2020): Effective heuristics and metaheuristics to minimise total tardiness for the distributed permutation flowshop scheduling problem, International Journal of Production Research, DOI: [10.1080/00207543.2020.1837982](https://doi.org/10.1080/00207543.2020.1837982)

To link to this article: <https://doi.org/10.1080/00207543.2020.1837982>



Published online: 02 Nov 2020.



Submit your article to this journal [↗](#)




View related articles [↗](#)



View Crossmark data [↗](#)



Effective heuristics and metaheuristics to minimise total tardiness for the distributed permutation flowshop scheduling problem

Ankit Khare  and Sunil Agrawal

PDPM, Indian Institute of Information Technology, Design and Manufacturing, Jabalpur, India

ABSTRACT

During recent years, the distributed permutation flowshop scheduling problem (DPFSP) has become a very active area of research. However, minimising total tardiness in DPFSP, a very essential and relevant objective for today's customer-orientated market, has not been studied much. In this paper, we address the DPFSP with the total tardiness criterion. We present a mixed-integer linear programming model, two heuristics, hybrid discrete Harris hawks optimisation and an enhanced variant of iterated greedy algorithm to solve the considered problem. Problem-specific knowledge is explored and effective technologies, such as path relinking and random sub-sequence/single-point local search, are employed to improve the presented algorithms. The operators and parameters of the algorithms are analysed and calibrated using the design of experiments. To evaluate the performance, the well-known benchmark problem set of Naderi and Ruiz for DPFSP is extended with due dates. We compare the presented algorithms against seven other well-known meta-heuristics from the literature. Statistically sound results demonstrate the effectiveness of the presented algorithms for the considered problem.

ARTICLE HISTORY

Received 15 February 2020
Accepted 7 October 2020

KEYWORDS

Distributed scheduling; total tardiness; mixed integer linear programming; Harris hawks optimisation algorithm; iterated greedy algorithm; path relinking

1. Introduction

The traditional permutation flowshop scheduling problem (PFSP) follows mono-factory production environment where all the manufacturing processes take place in the same factory. However, in today's decentralised and globalised economy, large enterprises usually have many manufacturing centres around the globe. This multi-factory production pattern not only enables companies to reduce the delivery and manufacturing costs, but also helps them in achieving better quality products (Naderi and Ruiz 2010; Fernandez-Viagas and Framinan 2015a; Olhager and Feldmann 2017). To tackle this multi-factory production scenario, Naderi and Ruiz (2010) proposed distributed permutation flowshop scheduling problem (DPFSP) which can be considered as an extension of the regular PFSP. DPFSP consists of F (more than one) identical factories, each one with m machines. The scheduling process in DPFSP can be divided into two parts: the assignment of jobs to different factories and sequencing the allotted jobs in each factory as per the given performance measure.

Since the initial work of Naderi and Ruiz (2010), where they presented assignment rules, mathematical models and heuristics to minimise the makespan of DPFSP,

distributed scheduling has become a very active research area in recent years. To minimise the makespan of DPFSP, numerous researchers have presented many approximate algorithms such as hybrid genetic algorithm (Gao and Chen 2011), an estimation of distribution algorithm (Wang et al. 2013), tabu search algorithm (Gao, Chen, and Deng 2013), modified iterative greedy algorithm (Lin, Ying, and Huang 2013), hybrid immune algorithm (Xu et al. 2014), scatter search algorithm (Naderi and Ruiz 2014), chemical reaction optimisation algorithm (Bargaoui, Driss, and Ghédira 2017) and iterative greedy algorithm (Ruiz, Pan, and Naderi 2018). Hamzadayi (2020) improved the existing mathematical model for minimising makespan of DPFSP. Rifai, Nguyen, and Dawal (2016) minimised makespan of DPFSP with re-entrant characteristic. Ribas, Companys, and Tort-Martorell (2017) presented a mathematical model along with some heuristics and Zhao et al. (2020) proposed an ensemble discrete differential evolution algorithm to minimise makespan of DPFSP with a blocking constraint. Different variants of the iterated greedy algorithm are utilised by Ying and Lin (2017) with blocking constraint, Ying et al. (2017) with no-idle constraint, Shao, Pi, and Shao (2017) with no-wait constraint, Cheng et al.

(2018) with mixed no-idle constraint and Huang, Pan, and Gao (2020) with sequence-dependent set-up times constraint to minimise makespan for DPFSP. Zhang and Xing (2019) proposed two constructive heuristics and a differential evolution algorithm to minimise makespan of DPFSP with limited-buffer constraint. Ying and Lin (2018) proposed a self-tuning iterated greedy (SIG) algorithm, Li et al. (2020) presented a discrete artificial bee colony algorithm and Cai, Lei, and Li (2020) proposed a new shuffled frog-leaping algorithm to tackle hybrid flowshop scheduling problem in distributed environment. Zhang, Xing, and Cao (2017) proposed a constructive heuristic along with two hybrid metaheuristics for makespan minimisation of DPFSP integrated with flexible assembly and set-up time. Fernandez-Viagas, Perez-Gonzalez, and Framinan (2018) proposed eighteen constructive heuristics along with an evolutionary algorithm and Pan et al. (2019) presented four metaheuristics along with three constructive heuristics to minimise the total flowtime of DPFSP. Meng, Pan, and Wang (2019) dealt with DPFSP with customer order constraint and presented three heuristics and three metaheuristics to minimise makespan. Li et al. (2019) presented the improved artificial bee colony algorithm to deal with a DPFSP with distance coefficient in a pre-fabricated system. Recently, Shao, Pi, and Shao (2020) presented a hybrid enhanced discrete fruit fly optimisation algorithm to minimise makespan of DPFSP with blocking constraint.

The preceding paragraph shows that most of the previous works for DPFSP acknowledged only makespan as the performance measure. However, in today's customer-oriented competitive market, on-time delivery of the product has become more critical than ever (Vallada, Ruiz, and Minella 2008). The minimisation of the total tardiness enables manufacturing industries in completing customer orders before their due dates (preventing either a reputation loss, or even the loss of customer). However, despite being a realistic performance measure, minimisation of total tardiness (TT) for DPFSP has not captured much attention so far. To address this drawback, in this paper, we investigate DPFSP with total tardiness criterion, which can be denoted as $DF|pmu| \sum T_j$ (as per Graham et al. 1979; Naderi and Ruiz 2010). The considered problem ($DF|pmu| \sum T_j$) is an NP-hard problem as it is an extension of regular PFSP with total tardiness criterion ($F|pmu| \sum T_j$), which is known to be an NP-hard in strong sense for $m \geq 2$ (Du and Leung 1990). Due to the complexity of the problem ($F|pmu| \sum T_j$), many approximate algorithms have been developed in the literature. In recent works, Kellegöz, Toklu, and Wilson (2010) proposed elite guided steady-state genetic algorithm and Vallada and Ruiz (2010) integrated the

path-relinking method with genetic algorithm. Chen and Li (2013) presented an integrated iterated local search algorithm based on the iterated local search method. Cura (2015) developed a new evolutionary algorithm approach with a problem-based mating procedure. Li et al. (2015) proposed several trajectory scheduling methods based on three perturbation methods and six different composite heuristics. Fernandez-Viagas and Framinan (2015b) proposed NEH-based heuristics along with several tie-breaking mechanisms. Karabulut (2016) presented a hybrid iterated greedy algorithm with a new temperature calculation formula and random local search algorithm. Finally, Fernandez-Viagas, Valente, and Framinan (2018) presented eight variations of an iterated greedy algorithm to minimise the total tardiness of PFSP.

In this paper, we present a mixed integer linear programming model (MILP), two heuristics and enhanced variant of Harris hawks optimisation (HHO) (Heidari et al. 2019) and iterated greedy (IG) algorithm (Ruiz and Stützle 2007) to solve the considered problem. HHO algorithm is a nature-inspired population-based meta-heuristic that is inspired by the foraging behaviour of Harris' hawks. Since it is recently proposed, it has been applied in very limited areas (Golilarz, Gao, and Demirel 2019; Moayedi, Nguyen, and Rashid 2019) and it is the first application of HHO in scheduling field. Contrary to HHO algorithm, IG algorithm has been successfully applied to several complex optimisation problems. IG algorithm starts with problem-specific heuristic solution, and then destruction phase is applied to this solution by removing d percentage of jobs from the sequence. After the destruction phase, the construction phase is carried out, where a greedy heuristic is applied in order to reinsert the destructed jobs into the solution resulting in a completely new sequence. After the construction phase, a local search algorithm is applied to improve the solution further. Finally, an acceptance criterion is employed to determine whether the new sequence replaces the incumbent one. The IG algorithm is simple yet very effective for the DPFSP, as shown in (Lin, Ying, and Huang 2013; Fernandez-Viagas and Framinan 2015a; Shao, Pi, and Shao 2017; Ruiz, Pan, and Naderi 2018).

The remainder of this paper is organised as follows. In Section 2, DPFSP with total tardiness criterion is described and formulated. In Section 3, two heuristics are proposed. An enhanced version of Harris hawks optimisation (HHO) algorithm is presented in Section 4. In Section 5, an enhanced iterated greedy (IG) algorithm to solve DPFSP with total tardiness criterion is presented. Section 6 contains the experimental design and parameter tuning of the presented algorithms. The numerical and statistical comparison of presented algorithms with other meta-heuristics is shown in Section 7. Finally, the

concluding remarks and suggestions for the future work are discussed in Section 8.

2. Problem description

This section explains the considered problem $DF|pmu|\sum T_j$ with the help of a numerical illustration and formulates the mathematical model along with the following assumptions:

- The problem considers more than one identical factory, i.e. $f > 1$, located in different areas, with the responsibility of processing a set of n jobs.
- All factories have the same set of m machines in the fixed permutation.
- All machines and jobs are available at time zero.
- Each machine can process only one job at a time and each job can be processed on one machine at a time and the processing time of a job at a specific machine is the same from factory to factory.

2.1. Mixed inter linear programming (MILP)

We present a mixed integer linear programming (MILP) model to describe the considered problem $DF|pmu|\sum T_j$. The aim is to allocate jobs to various factories and simultaneously determine their processing sequences in each factory to minimise the total tardiness (TT) objective.

Indices

j, k	Index for jobs and job positions; $j, k \in \{1, 2, \dots, n\}$
i	Index for machines; $i \in \{1, 2, \dots, m\}$
f	Index for factories; $f \in \{1, 2, \dots, F\}$
M	A sufficiently large positive number

Parameters

n	Number of jobs
m	Number of machines
F	Number of factories
$p_{i,j}$	Processing time of job j on machine i
d_j	Due date of job j

Decision Variable

$X_{j,k}$	1 if job j is assigned to position k ; 0 otherwise
$Y_{k,f}$	1 if job at position k is assigned to factory f ; 0 otherwise
$C_{k,i}$	Completion time of job at position k on machine i
U_k	Due date of job at position k
T_k	Tardiness of job j at position k
TT	Total Tardiness

$$MinTT = \sum_{k=1}^n T_k \quad (1)$$

$$\sum_{k=1}^n X_{j,k} = 1 \forall_j \quad (2)$$

$$\sum_{j=1}^n X_{j,k} = 1 \forall_k \quad (3)$$

$$\sum_{f=1}^F Y_{k,f} = 1 \forall_k \quad (4)$$

$$C_{k,i} \geq C_{k,i-1} + \sum_{j=1}^n X_{j,k} \cdot p_{i,j} \forall_{k,i} \quad (5)$$

$$C_{k,i} \geq C_{l,i} + \sum_{j=1}^n X_{j,k} \cdot p_{i,j} - M(1 - Y_{k,f}) - M(1 - Y_{l,f}) \forall_{k>l, l<k, k,i,f} \quad (6)$$

$$U_k = \sum_{j=1}^n X_{j,k} \cdot d_j \forall_k \quad (7)$$

$$T_k = \max(C_{k,i} - U_k, 0) \forall_{k,i=m} \quad (8)$$

$$T_k \geq 0 \forall_k \quad (9)$$

$$C_{k,i} \geq 0 \forall_{k,i} \quad (10)$$

$$X_{j,k} \in \{0, 1\} \forall_{j,k} \quad (11)$$

$$Y_{k,f} \in \{0, 1\} \forall_{k,f} \quad (12)$$

where $C_{k,0} = 0$. Equation (1) defines the objective function to minimise the total tardiness TT . Constraint sets (2) and (3) together ensure that every job must be allotted to exactly one position and each position must be assigned exactly one job. Constraint set (4) makes sure that each job is assigned to exactly one factory. Constraint set (5) forces that the operation on job at position k at each machine can only start when the operation of the same job is finished on the previous machine. Constraint set (6) makes sure that processing of each job can start only after the previous job allotted to the same machine is completed. Constraint set (7) finds the due date according to the position assigned to the jobs. Constraint (8) calculates tardiness of job at position k . Constraint sets (9)–(12) define the decision variables.

2.2. Numerical illustration

The solution representation for DPFSP proposed by Naderi and Ruiz (2010) is utilised in this paper. There is a set of F lists, one per factory and each list has a partial sequence (π_f) indicating the processing order of the jobs for the factory f , $f = \{1, 2, \dots, F\}$. Here, $\pi =$

$\{\pi_1, \pi_2, \dots, \pi_f, \dots, \pi_F\}$ represents the set of F lists. For example, consider a problem with 12 jobs and three factories, a possible solution is represented as $\pi = \{(1, 5, 8, 11, 12), (2, 4, 7, 10), (3, 6, 9)\}$. In this solution, the operation sequences of jobs in factory 1, 2 and 3 are 1-5-8-11-12, 2-4-7-10 and 3-6-9, respectively. For assigning jobs to different factories, Naderi and Ruiz (2010) show that the earliest completion time (ECT) rule gives a better result at a very small computational cost. Therefore, we too considered ECT rule in this study.

For a numerical illustration of the considered problem, let us assume a DPFSP problem where $n = 5$ jobs have to be scheduled through two identical factories with $m = 2$ at each factory. The processing time and due date are given in Table 1. Suppose $\pi = \{5, 4, 3, 2, 1\}$ is the sequence for which we want to find the total tardiness. When assigning first job 5, as both the factories are empty, there is no difference in assigning it to factory 1 or 2. So, job 5 is arbitrarily assigned to factory 1 with a completion time of 14 units. After scheduling job 5, the next job in sequence is 4. Completion time of job 4 in factory 1 is 20 units and in factory 2 is 15 units. Following ECT rule, job 4 is assigned to factory 2. The next job 3, when tested on factory 1, gives 18 units as completion time. However, assigning job 3 to factory 2 results in 20 units. Therefore, job 3 is assigned to factory 1 as per ECT rule. This process is repeated for the remaining jobs 2 and 1. Once the completion time of all the jobs is known, their tardiness can be calculated. A Gantt chart is shown in Figure 1. For example, the due date of

job 5 is given as 5 units and its completion time is 14 units, so its tardiness is $T_5 = 9$ units. In the same manner, the tardiness of job 1 is 16 units, job 2 is 17 units, job 3 is 0 units and job 4 is 5 units and the total tardiness is $TT = \sum_{j=1}^n T_j = 16 + 17 + 0 + 5 + 9 = 47$ units.

3. Heuristics methods

For the scheduling problems with due dates-based criteria, the earliest due date (EDD), the smallest overall slack time (OSL) and the smallest slack time on the last machine (LSL) rules are commonly used to generate initial sequences. The EDD sorts jobs in non-decreasing order of their due dates i.e. a permutation of n jobs $\pi = \{\pi_1, \pi_2, \pi_3, \dots, \pi_n\}$ is sorted such that $d_{\pi(j)} \leq d_{\pi(j+1)}, j = 1, 2, \dots, n-1$ where $\pi(j)$ represents job allotment in the j th position of the permutation. The OSL sorts jobs in non-decreasing order of their overall slack times such that $d_{\pi(j)} - \sum_{i=1}^m p_{i,\pi(j)} \leq d_{\pi(j+1)} - \sum_{i=1}^m p_{i,\pi(j+1)}$. The LSL rule sorts jobs in non-decreasing order of their slack times on the last machine such that $d_{\pi(j)} - p_{m,\pi(j)} \leq d_{\pi(j+1)} - p_{m,\pi(j+1)}$. In addition to these three dispatching rules, we present two more heuristics. The first one is called $NEHD_{edd}$. It is an extension of NEH_{edd} (Kim 1993) in the context of distributed environment. NEH_{edd} is the most widely used constructive heuristic to generate initial solution for other heuristic and metaheuristic algorithms for $F|pmdu|\sum T_j$ problem (Kellegöz, Toklu, and Wilson 2010; Vallada and Ruiz 2010; Li et al. 2015; Fernandez-Viagas and Framinan 2015b; Karabulut 2016). In $NEHD_{edd}$, first the jobs are sorted in non-decreasing order of their due dates. Then each job $j \in \{1, 2, \dots, n\}$ is taken and allotted to the factory with least partial total tardiness after evaluating all possible positions in all existing factories. The time complexity of $NEHD_{edd}$ is $O(n^2 m F)$. The pseudo-code of

Table 1. Processing times and due date for the example.

Machine	Jobs				
	J_1	J_2	J_3	J_4	J_5
M1	10	6	8	9	3
M2	5	7	4	6	11
Due date (d_j)	10	5	20	10	5

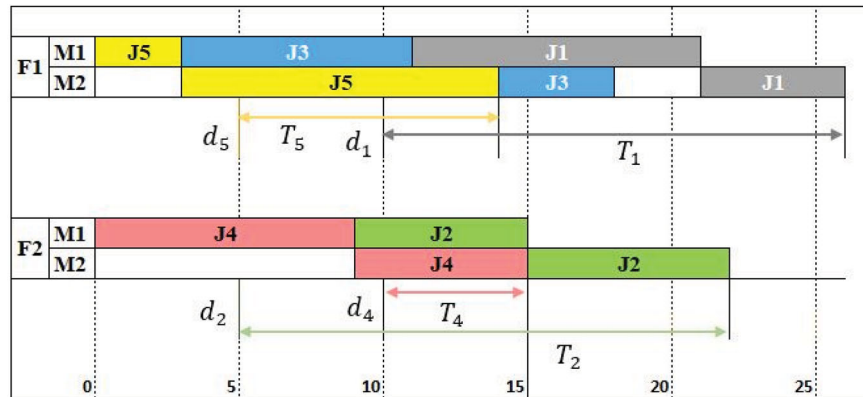


Figure 1. The Gantt Chart for the illustrated example.

Algorithm 1 Pseudo code of $NEHD_{edd}$ **Procedure** $NEHD_{edd}$

```

 $\pi^{edd} = \text{Sort } n \text{ jobs according to their due date in}$ 
 $\text{non-decreasing order}$ 
for  $f = 1$  to  $F$ 
     $\pi_f = \emptyset$  % (empty initial solution)
endfor
 $\pi = \{\pi_1, \pi_2, \dots, \pi_F\}$ 
for  $j = 1$  to  $n$ 
     $x = \pi^{edd}(j)$ 
    for  $f = 1$  to  $F$ 
        Test job  $x$  in all possible positions of  $\pi_f$ 
         $TT^f$  is the lowest  $TT$  obtained at position  $p^f$ 
    endfor
     $f_{min} = \arg(\min_{f=1}^F (TT^f))$ 
    Insert job  $x$  in  $\pi_{f_{min}}$  at position  $p^{f_{min}}$  resulting
    in lowest  $TT$ 
endfor
return  $\pi$ 

```

End

$NEHD_{edd}$ is shown in Algorithm 1. The second heuristic, the smallest slack time on every machine (ESL) is inspired by LSL dispatching rule. Instead of considering slack times only on the last machine as in LSL rule, in ESL, every machine is considered for sorting jobs in non-decreasing order of their slack times, resulting in total m sequences. The sequence with lowest total tardiness is then selected. The pseudo-code of ESL is shown in Algorithm 2. The comparison among these heuristics is given in the computational evaluation section.

Algorithm 2 Pseudo code of ESL**Procedure** ESL

```

for  $i = 1$  to  $m$ 
    for  $j = 1$  to  $n$ 
         $\text{slack}(i, j) = d_j - p_{i,j}$ 
    endfor
endfor
for  $i = 1$  to  $m$ 
     $\pi_i = \text{Sort } n \text{ jobs according to their slack on}$ 
     $\text{machine } i \text{ in non-decreasing order}$ 
    for  $j = 1$  to  $n$ 
        assign job  $\pi_i(j)$  to the factory with least
        completion time
    endfor
     $TT_i = \text{Evaluate } TT \text{ of sequence } \pi_i$ 
endfor
 $I_{min} = \arg(\min_{i=1}^m (TT_i))$ 
return  $\pi_{I_{min}}$ 

```

end**4. Hybrid discrete Harris hawks optimisation****4.1. Harris hawks optimisation**

Harris hawks optimisation (HHO) is a population-based nature-inspired meta-heuristic proposed by Heidari et al. (2019). HHO imitates the foraging behaviour of Harris' hawks found in the southern half of Arizona, USA. The whole foraging pattern of Harris' hawks can be divided into three phases; exploration phase, transition from exploration to exploitation, and exploitation phase.

4.1.1. Exploration phase

To detect the location of the prey, Harris' hawks wait and observe the desert site based on two strategies. In the first strategy, each Harris' hawk decides its location based on the position of the prey and the positions of other family members, whereas during the second strategy, they perch on the random locations. This phase is formulated in Equation (13)

$$X(t+1) = \begin{cases} X_{rand}(t) - r_1 |X_{rand}(t) - 2r_2 X(t)| & q \geq 0.5 \\ \begin{cases} X_{prey}(t) - X_m(t) \\ -r_3(LB + r_4(UB - LB)) \end{cases} & q < 0.5 \end{cases} \quad (13)$$

where $X(t+1)$ represents the position vector of the search agent in the next iteration, $X_{rand}(t)$ is the position vector of a randomly selected hawk from the current population, $X(t)$ is the position vector in the current iteration t , $X_{prey}(t)$ represents the position vector of the prey (best solution obtained till current iteration t), r_1, r_2, r_3, r_4 and q are random numbers between 0 and 1. UB and LB are the upper and lower bounds, respectively, and $X_m(t)$ is the average position based on the location of other family members:

$$X_m(t) = \left(\sum_{i=1}^N X_i(t) \right) / N \quad (14)$$

where $X_i(t)$ denotes the position vector of each search agent at iteration t and N is the total number of Harris' hawks (search agents).

4.1.2. Transition from exploration to exploitation phase

During the whole hunting and escaping process, the energy of prey continuously decreases. This step is modelled by Equation (15)

$$E = 2E_0 \left(1 - \frac{t}{T} \right) \quad (15)$$

where E indicates the escaping energy of the prey at iteration t , $E_0 = 2\text{rand}() - 1$ is the initial energy state of the

prey, T is the total number of iterations, and t represents the current iteration. The value of E keeps decreasing over the entire iterations and results in the transition of exploration into exploitation phase. When in starting stages $|E| \geq 1$, the exploration takes place, and when in later stages $|E| < 1$, exploitation occurs.

4.1.3. Exploitation phase

Instead of searching in different locations (exploration), in exploitation phase, the search agents (hawks) start exploiting the neighbourhood of the solutions detected during exploration phase. In HHO, $r(rand[0, 1])$ is considered to be the escaping probability of the prey. Depending on the value of r and $|E|$, the exploitation phase of HHO algorithm is divided into four following steps:

Step 1: When prey has some escaping energy ($E \geq 0.5$) but still gets trapped ($r \geq 0.5$). This scenario is known as soft besiege and formulated by Equation (16).

$$X(t+1) = \Delta X(t) - E|J.X_{prey}(t) - X(t)| \quad (16)$$

$$\Delta X(t) = X_{prey}(t) - X(t) \quad (17)$$

where $J = 2(1 - r_5)$ indicates the random jump strength of the prey over the entire hunting process. r_5 is a random number between 0 and 1. $\Delta X(t)$ is the difference between the position vector of best solution so far and current position at iteration t .

Step 2: When prey is completely exhausted ($E < 0.5$) and gets trapped ($r \geq 0.5$). This scenario is known as hard besiege and modelled by Equation (18).

$$X(t+1) = X_{prey}(t) - E|\Delta X(t)| \quad (18)$$

Step 3: When prey still has some energy ($E \geq 0.5$) with escaping probability ($r < 0.5$). This scenario is known as soft besiege with progressive rapid dive and demonstrated by Equation (19).

$$X(t+1) = \begin{cases} Y & \text{if } f(Y) < f(X(t)) \\ Z & \text{if } f(Z) < f(X(t)) \end{cases} \quad (19)$$

$$Y = X_{prey}(t) - E|J.X_{prey}(t) - X(t)| \quad (20)$$

$$Z = Y + S \times LF(D) \quad (21)$$

where S is the random vector of size $(1 \times D)$, D is the problem dimension, and $LF(D)$ is the Levy Flight function (Heidari et al. 2019).

Step 4: When prey is completely exhausted ($E < 0.5$) and still has less probability of getting trapped ($r < 0.5$). This scenario is known as hard besiege with progressive rapid dives and modelled by Equation (22).

$$X(t+1) = \begin{cases} A & \text{if } f(Y) < f(X(t)) \\ B & \text{if } f(Z) < f(X(t)) \end{cases} \quad (22)$$

$$A = X_{prey}(t) - E|J.X_{prey}(t) - X_m(t)| \quad (23)$$

$$B = A + S \times LF(D) \quad (24)$$

where $X_m(t)$ is obtained using Equation (14).

4.2. Hybrid discrete Harris hawks optimisation (HDHHO)

4.2.1. Initialisation

Population-based meta-heuristics, in general, initially start with random solutions which results in large convergence time to achieve global optimum. To ensure certain quality and diversity in initial population, EDD, OSL, LSL, ESL, and $NEHD_{edd}$ heuristics are employed to generate five initial solutions out of the N random population. The continuous positions of Harris hawk are mapped into the corresponding job permutation using the smallest position value (SPV) rule described by Tasgetiren et al. (2007). In the SPV rule, all elements of the position vector are sorted in non-decreasing order based on their position values. The SPV mapping mechanism is shown in Table 2 with a simple instance ($n = 5$) where X_i represents the position vector of i^{th} search agent and x_{ij} denotes j^{th} element (or dimension) of i^{th} search agent. Following the SPV rule, as x_{i5} is the smallest value of X_i , it is assigned first position in job permutation. Then x_{i3} is selected secondly and assigned second position in job permutation. In the same way, x_{i2} , x_{i1} , x_{i4} are assigned third, fourth, and fifth positions, respectively.

4.2.2. Path relinking

Over the last decade, many meta-heuristics (Vallada and Ruiz 2010; Costa, Goldbarg, and Goldbarg 2012; Pan et al. 2013) have been enhanced by utilising the path relinking (Glover and Laguna 1998) strategy. Path relinking is an effective search strategy used as a local search procedure to explore the search space between two given solutions. In this work, it is applied to generate and explore the solutions between the worst and best solutions in the population. The worst solution is considered as an *initial solution* (β) and the best solution as the *guiding solution* (γ). To transform β into γ , a series of moves (insert/swap) is performed and each time a move is carried out, an intermediate solution is generated. As the number of moves increases, the obtained intermediate solution looks more like γ and less like β . In the end,

Table 2. SPV mechanism.

Dimension	x_{i1}	x_{i2}	x_{i3}	x_{i4}	x_{i5}
X_i	0.824	0.695	0.317	0.950	0.034
Jobs	5	3	2	1	4

Algorithm 3 Pseudo code of proposed path relinking strategy

Procedure Path-relinking (β, γ, ϑ)

```

for  $i = 1$  to  $n$ 
  if  $\beta(i) \neq \gamma(i)$ 
    Find job  $k$  in  $\beta$  which is identical to  $\gamma(i)$ 
    if  $\vartheta > 0.5$ 
      Remove job  $k$  from its original position and
      insert it into position  $i$  of  $\beta$ .
    elseif
      Exchange the position of job  $k$  and  $\beta(i)$ .
    endif
    if the generated solution is not identical to  $\gamma$ ,
      put it into set  $S$ .
    endif
  endfor
  Evaluate all the solution in  $S$  and return the one
  with the least total tardiness value.
end

```

all the intermediate solutions are evaluated and the solution with the lowest total tardiness value is selected. The pseudo-code of the proposed path relinking strategy is shown in Algorithm 3. Let us consider an example with two solutions, $\beta = [1, 2, 3, 4, 5]$ and $\gamma = [5, 4, 3, 2, 1]$. By performing path relinking with insert move, three intermediate solutions are obtained $[5, 1, 2, 3, 4]$, $[5, 4, 1, 2, 3]$, and $[5, 4, 3, 1, 2]$. However, by performing path relinking with swap move, only one intermediate solution $[5, 2, 3, 4, 1]$ is obtained. Figure 2 illustrates the set of movements required to transform β into γ using insert and swap moves. The selection between insert and swap moves is done by generating a random number ϑ between 0 and 1. If $\vartheta > 0.5$ insert move is selected, otherwise swap move

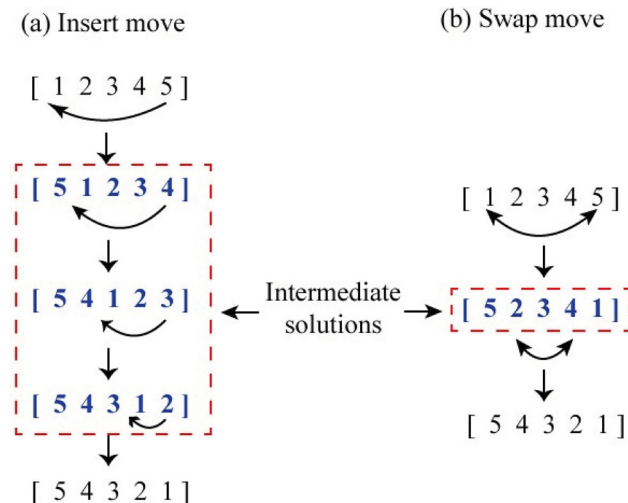


Figure 2. Illustration of path-relinking method.

Algorithm 4 Pseudo-code of HDHHO

```

Initialize first five solutions with  $EDD, LSL, OSL, ESL$ ,
and  $NEHD_{edd}$ 
Generate remaining  $(N - 5)$  population randomly
 $X_i (i = 6, 7, \dots, N)$ 
while (stopping condition is not met) do
  Apply SPV rule to convert continuous value into
  discrete job permutation.
  Calculate the fitness values of hawks.
  Set  $X_{prey}$  and  $X_{worst}$  as location of hawk with least and
  most fitness value, respectively.
  Generate a random number  $\vartheta$  between 0 and 1.
   $X' = \text{Path-relinking}(X_{worst}, X_{prey}, \vartheta)$  % Path-relinking
  if  $TT(X') < TT(X_{prey})$  then
     $X_{prey} = X'$ 
  elseif  $TT(X') < TT(X_{worst})$  then
     $X_{worst} = X'$ 
  endif
  for (each hawk ( $X_i$ )) do
    Update  $E$  using Equation (15)
    if ( $|E| \geq 1$ ) then % Exploration phase
      Update the location vector using Equation (13)
    elseif ( $|E| < 1$ ) then % Exploitation phase
      if ( $r \geq 0.5$  and  $|E| \geq 0.5$ ) then % Soft besiege
        Update the location vector using Equation (16)
      elseif ( $r \geq 0.5$  and  $|E| < 0.5$ ) then % Hard
        besiege
          Update the location vector using Equation (18)
        elseif ( $r < 0.5$  and  $|E| \geq 0.5$ ) then % Soft besiege
          with progressive rapid dives
            Update the location vector using Equation (19)
          elseif ( $r < 0.5$  and  $|E| < 0.5$ ) then % Hard
            besiege with progressive rapid dives
              Update the location vector using Equation (22)
          endif
        endif
      endif
    endif
  endfor
endwhile
return  $X_{prey}$ 

```

is chosen. The pseudo-code of the presented HDHHO algorithm is given in Algorithm 4.

5. Iterated greedy algorithm

The Iterated Greedy (IG) algorithm for flowshop scheduling problem was first proposed by Ruiz and Stützle (2007). As mentioned, IG algorithm has been successfully applied several times to minimise the makespan of DPFSP in the past. To the best of our knowledge, this is the first implementation of IG algorithm to minimise total tardiness of DPFSP. In the following sections, we describe each phase of IG algorithms in detail.

5.1. Initialise solution

For the initialisation procedure, two heuristics, NEHD_{edd} and ESL, described in Section 3 are considered. The final procedure is selected after calibration.

5.2. Destruction and construction phase

In the destruction phase, d percentage of total n jobs are randomly removed from the current solution regardless of the factories and placed into a list π_R . In this paper, instead of a fixed value for d , we considered it a problem-dependent variable. The percentage value of d is to be calibrated. In the construction phase, every job in π_R is tested in all possible positions in all the F factories. The factory and position with the lowest total tardiness is chosen.

5.3. Local search

In this study, we present two local search methods to enhance the performance of the IG algorithm for the considered problem. This local search is carried out after both the initialisation and construction phases.

5.3.1. Random subsequence local search

The random subsequence local search (RSLS) method has two modes: multi-factory (RSLS_m) and single-factory (RSLS_s). A random number v between 0 and 1 is generated to decide the selection between these two modes. In RSLS_s, insert/reverse operation occurs within a randomly chosen factory, whereas in RSLS_m, swap/(swap + reverse) operation takes place between two randomly selected factories. A random number w between 0 and 1 is generated to decide the selection between these operations. The pseudo-code of RSLS and its illustration are shown in Figures 3 and 4, respectively.

Figure 4(a) demonstrates an instance of RSLS_s with insert operation. The first step is to choose a factory f randomly, followed by selecting a starting point and the subsequence of random length g in the chosen factory. The subsequence length g must be less than n_f (number of jobs assigned to the f^{th} factory). Then, an insert point is selected randomly and the subsequence is inserted after the insert point. Figure 4(b) shows an example of RSLS_s with reverse operation. In this case, after choosing a factory f randomly, the reverse operation is applied on a subsequence of random length g within

```

Procedure RSLS ( $\pi$ )
 $v = \text{rand}()$ ,  $w = \text{rand}()$ 
while  $\text{flag} = \text{true}$ 
     $\text{flag} = \text{false};$ 
    if  $v < 0.5$            % single-factory mode (RSLSs)
        choose a factory  $\pi_f$  randomly;
        if  $w < 0.5$        % insert operation
            choose a starting point for subsequence (with random length  $g$ ) and an insert point
            randomly;
             $\pi' = \text{insert the subsequence after the insert point within the chosen factory } \pi_f;$ 
        else             % reverse operation
            select a subsequence of random length  $g$  in factory  $\pi_f$ ;
             $\pi' = \text{reverse the selected subsequence};$ 
        endif
    else                 % multi-factory mode (RSLSm)
        select two factories  $\pi_{f1}$  and  $\pi_{f2}$  randomly;
        randomly choose a same starting point for subsequence (with equal random length  $g$ ) at
        each factory;
        if  $w < 0.5$        % swap operation
             $\pi' = \text{swap the subsequences};$ 
        else             % swap + reverse operation
             $\pi' = \text{swap and reverse the subsequences};$ 
        endif
    endif
    if  $TT(\pi') < TT(\pi)$  then
         $\pi = \pi'$ 
         $\text{flag} = \text{true};$ 
    endif
endwhile
return  $\pi$ ;
end

```

Figure 3. Pseudo-code of RSLS.

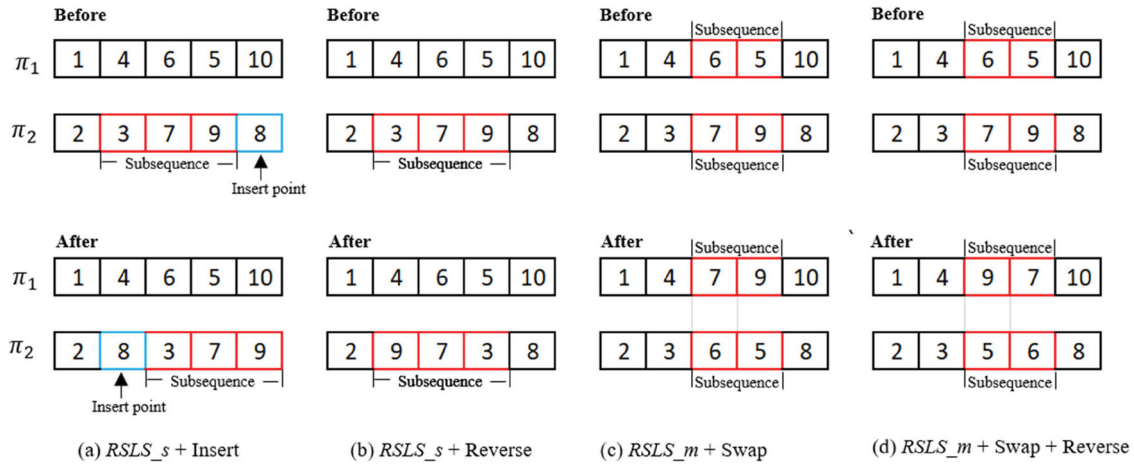


Figure 4. The example of random subsequence local search ($RSLs$) method.

the chosen factory. In $RSLs_m$ with swap operation, two random factories and a subsequence with equal random length g in each factory are chosen. Then, swap operation is executed. Lastly, in $RSLs_m$ with swap + reverse operation, two random factories and a subsequence with equal random length g in each factory are chosen. Then, both the sub-sequences are interchanged and reversed. Figure 4(c,d) demonstrates instances of $RSLs_m$ with swap and swap + reverse operations, respectively.

5.3.2. Random single-point local search

Similar to $RSLs$ method, random single-point local search ($RPLs$) has multi-factory ($RPLs_m$) and single-factory ($RPLs_s$) modes. A random number v between 0 and 1 is generated to decide between these two modes. If v is less than 0.5, $RPLs_s$ is chosen otherwise $RPLs_m$. In $RPLs_s$, insert/swap operation occurs within a randomly chosen factory, whereas in $RPLs_m$, these operations take place between two randomly selected factories. The decision of selection between insert and swap operation

is done by generating a random number w between 0 and 1. If w is less than 0.5, insert operation is chosen otherwise swap. The illustration of $RPLs$ is shown in Figure 5. An instance of $RPLs_s$ with insert operation is shown in Figure 5(a) where two different jobs are selected randomly in a randomly chosen factory and then the first job is inserted after the second one. Figure 5(b) shows an example of $RPLs_s$ with swap operation where both the selected jobs are swapped within the chosen factory. Figure 5(c,d) demonstrates the working of $RPLs_m$ with insert and swap operation, respectively. In these cases, two different factories are selected randomly with one random job at each factory. The insert/swap operation occurs between these two jobs.

5.4. Acceptance criterion

We consider a simulated annealing-like acceptance criterion proposed by Ruiz and Stützle (2007) with a constant

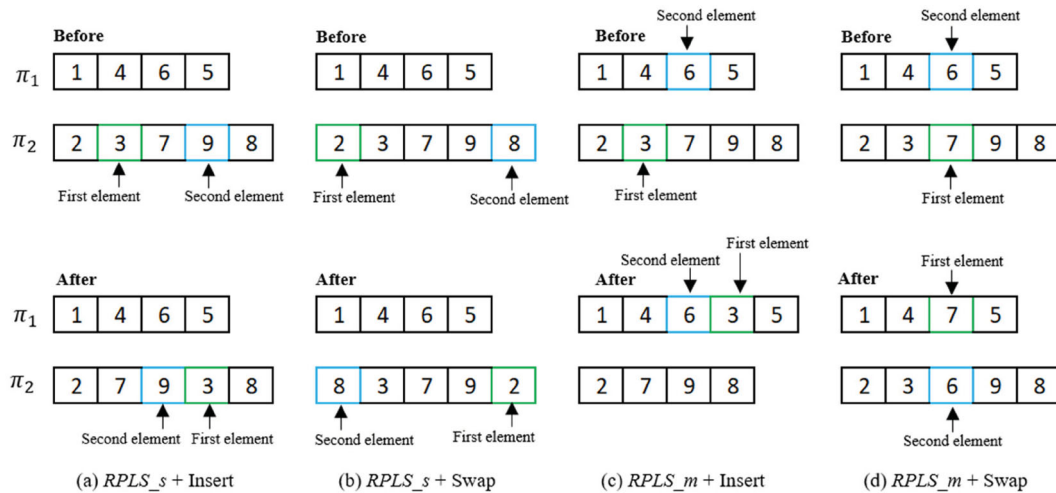


Figure 5. The example of random single-point local search ($RPLs$) method.

```

Procedure IG algorithm
   $\pi = \text{NEHD}_{edd}$  % Initialization
   $\pi = \text{local\_search}(\pi)$ 
   $\pi_b = \pi$ ;
  while (termination criterion not satisfied)
     $\pi' = \pi$ ;
    for  $y = 1$  to  $(d \times n)$  % Destruction phase
       $\pi_R = \text{remove one job at random from } \pi' \text{ regardless of the factories and insert it in } \pi_R$ ;
    endfor % Construction phase
    for  $y = 1$  to  $(d \times n)$ 
       $x = \pi_R(y)$ ;
      for  $f = 1$  to  $F$ 
        Test job  $x$  in all possible positions of  $\pi'_f$ ;
         $TT^f$  is the lowest  $TT$  obtained at position  $p^f$  for factory  $f$ ;
        ( $TT^f$  is the total tardiness obtained on assigning job  $x$  at factory  $f$ )
      endfor
       $f_{min} = \arg(\min_{f=1}^F(TT^f))$ ;
      Insert job  $x$  in  $\pi'_{f_{min}}$  at position  $p^{f_{min}}$  resulting in lowest  $TT$ ;
    endfor
     $\pi' = \text{local\_search}(\pi')$  % Local Search
    if  $TT(\pi') < TT(\pi)$  % Acceptance criterion
       $\pi = \pi'$ ;
      if  $TT(\pi) < TT(\pi_b)$ 
         $\pi_b = \pi$ ;
      endif
    elseif ( $\text{rand} \leq \exp\{-(TT(\pi') - TT(\pi))/T\}$ )
       $\pi = \pi'$ ;
    endif
  endwhile
  return  $\pi_b$ ;
end

```

Figure 6. Pseudo-code of the proposed IG algorithm.

temperature, shown as Equation (26).

$$T = T_0 \frac{\sum_{i=1}^n \sum_{j=1}^m P_{ij}}{10 \times n \times m} \quad (26)$$

where T_0 is a temperature parameter and is to be calibrated. This acceptance criterion considers the solution generated by the local search for the next iteration only in two scenarios; if the solution is better than the current one, or a random number $r[0, 1]$ is smaller than $\exp\{-(TT(\pi') - TT(\pi))/T\}$. Here, $TT(\pi')$ and $TT(\pi)$ denote total tardiness values for reconstructed and incumbent solutions, respectively. This procedure takes care of the diversification and prevents stagnation situation. The pseudo-code of the proposed IG algorithm is given in Figure 6.

6. Experimental design

6.1. Data generation

To tune the parameters of the proposed algorithms, we generate a set of 54 instances with number of jobs $n = \{20, 50, 100\}$, number of machines $m = \{5, 10, 20\}$, and

number of factories $F = \{2, 3, 4, 5, 6, 7\}$. Processing time of jobs at each machine is generated randomly in the range of $U[1, 99]$. To determine the due dates of jobs, we extended the method of Hasija and Rajendran (2004) in context of DPFSP as follows:

- (1) Calculate the total processing time of each job on all m machines, $P_j = \sum_{i=1}^m p_{ij} (j = 1, 2, \dots, n)$.
- (2) Calculate the due date for each job as follows:

$$d_j = P_j \times (1 + \text{rand} \times (1/f) \times \alpha) \quad (27)$$

where rand is the random number uniformly distributed in the range of $[0, 1]$, the term $(1/f)$ balances the tightness of due date according to the number of factories in the problem i.e. it tightens the due date for the problem with a greater number of factories and vice versa. Numerous experiments were carried out to determine the value of α ($= 0.5$) so that the due date obtained is neither too tight nor too loose for large- and small-sized problem sets.

6.2. Parameters tuning

Design of experiments (DOE) (Montgomery 2007) is utilised to test the effect of parameters of the proposed IG algorithm. It has two control parameters: d and T_0 . After initial tests, we consider d at three levels: 0.2, 0.3, and 0.4 and T_0 at three levels: 0.1, 0.2, and 0.4. As mentioned, we consider two initialisation procedures, $NEHD_{edd}$, and ESL and two local search methods, $RSLs$ and $RPLs$. Thus, there are total four parameters leading to $3 \times 3 \times 2 \times 2 = 36$ configurations. Each configuration is tested against all the 54 instances with three replications of each case. This full factorial design test is conducted on the Intel (R) Core (TM) i5-3470 CPU @ 3.40 GHz

Table 3. Results of ANOVA for parameter calibration of IG algorithm.

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Initial method	1	0.88920	0.889200	96.57	0.000
D	2	0.33909	0.169543	18.41	0.000
T	2	0.01394	0.006968	0.76	0.478
Local search	1	0.00754	0.007537	0.82	0.373
Error	29	0.26703	0.009208		
Total	35	1.51679			

with 8.00 GB RAM under a windows 10 operating system environment. The termination criterion is considered to be $n \times m \times 0.025$ seconds. Relative percentage deviation (RPD) over the best solution found in the experiment is considered as performance measure:

$$RPD = \frac{Alg_{sol} - Best_{sol}}{Best_{sol}} \times 100 \quad (28)$$

where Alg_{sol} is the TT obtained by a particular configuration and $Best_{sol}$ is the lowest TT obtained in the whole experiment. The analysis of variance (ANOVA) technique is frequently used in the scheduling literature (Naderi and Ruiz 2010; Ruiz, Pan, and Naderi 2018) to calibrate metaheuristics. In this study, we too utilised it to analyse the effect of the above-mentioned factors in order to calibrate IG algorithm. The results of ANOVA are summarised in Table 3. From Table 3, it can be seen that the parameter initial method has the largest F-ratio, indicating that it has the most significant effect on the performance of the algorithm. Figure 7 presents the main effects plot of all the parameters. From Figure 7(a), it can be seen that $NEHD_{edd}$ as initial method yields better result than ESL. Figure 7(b) shows that IG performs

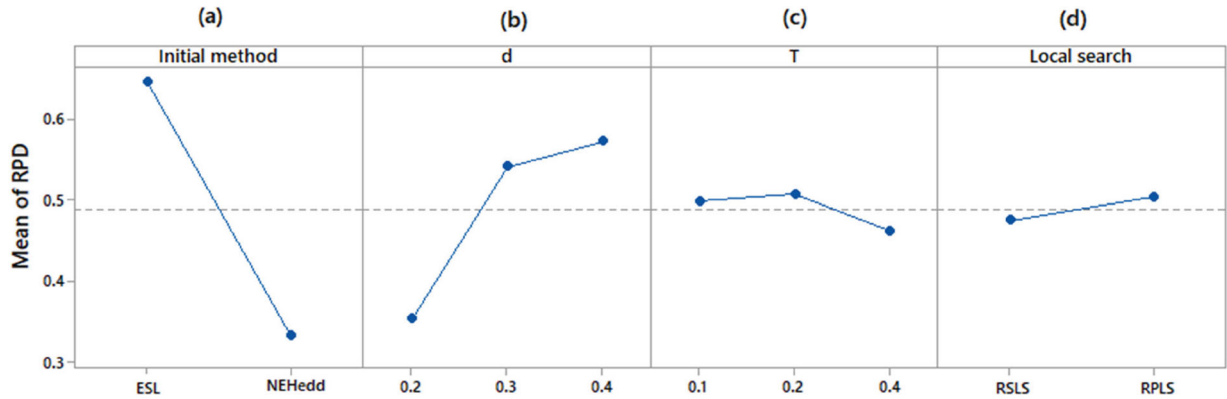


Figure 7. Main effect plots of IG parameters.

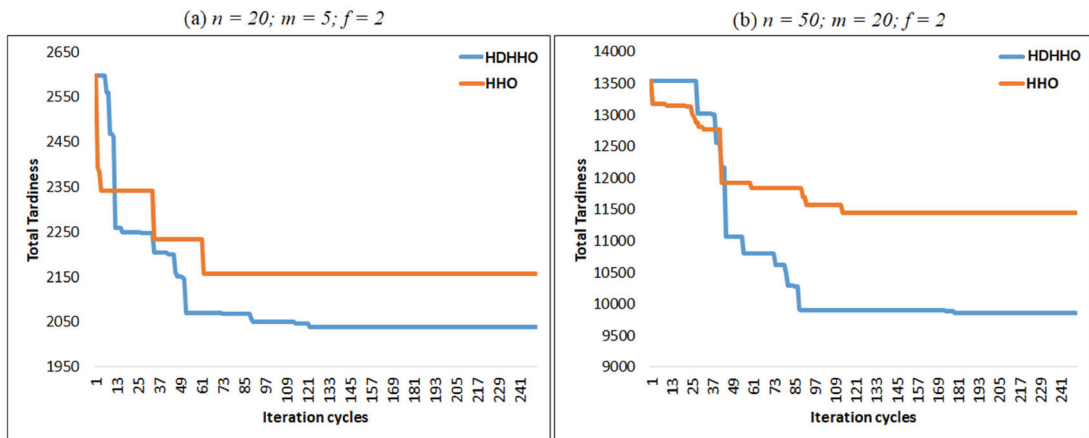


Figure 8. Comparative convergence behaviours of HHO and the proposed HDHHO algorithm.

better at $d = 20\%$ and the result becomes worse and worse as the value of d increases. Figure 7(c) indicates that the performance of IG algorithm is better at $T = 0.4$ than any other levels. Finally, Figure 7(d) shows that the local search, *RSLs* improves the proposed IG algorithm little better than *RPLs*. Hence, we set the parameters of IG algorithm as $d = 0.2$, $T = 0.1$, $NEHD_{edd}$ an initial method, and *RSLs* as local search method. As HHO is free from initialisation of input parameters (Heidari et al. 2019), its parameter tuning is not required. Figure 8 shows the variation of total tardiness with the iteration cycles of HHO and HDHHO for $(n, m, f) = \{20, 5, 2\}$ and $\{50, 20, 2\}$ problems. From Figure 8, it can be seen that the presented HDHHO algorithm has a better convergence rate than the native HHO algorithm.

7. Computational evaluation

Since in this paper the DPFSP with total tardiness criterion is addressed for the first time, there is no standard problem set for it. Each problem of $DF|pmu|\sum T_j$ is defined by four variables: the number of factories, the number of jobs, the number of machines at each factory, and due dates of every job. To test the presented algorithms, we extend the well-known Naderi and Ruiz (2010) benchmark set for DPFSP with due dates. The test problem set of Naderi and Ruiz contains two sets of instances. The first set consists of 420 small-sized problems with five instances for each combination of factories $F = \{2, 3, 4\}$, jobs $n = \{4, 6, 8, 10, 12, 14, 16\}$, and machines $m = \{2, 3, 4, 5\}$. The second set is based on Taillard's benchmark problems (Taillard 1993). It contains 720 large-sized instances with different combinations of jobs and machines, $\{n\} \times m : \{20, 50, 100\} \times 5, \{20, 50, 100, 200\} \times 10$ and $\{20, 50, 100, 200, 500\} \times 20$. With ten instances for each combination, these 120 instances are considered with six factories $F = \{2, 3, 4, 5, 6, 7\}$. In order to add due dates, the same procedure, as mentioned in Section 6.1, is utilised. The value of α for small- and large-sized instance is taken as 0.1 and 0.5, respectively. The extended Naderi and Ruiz's benchmark sets along with the best-known solutions are available at <https://doi.org/10.17632/4tkhdx4jjx>.

7.1. Evaluation of the MILP model

The proposed MILP model is tested on the small benchmark set consisting 420 instances using Gurobi 8.1.1 solver. Instances only with $n = \{4, 6, 8, 10\}$ are considered since the other instances are too large to find an optimal solution in reasonable CPU times. Table 4 summarises the computational results of the MILP model along with ESL, $NEHD_{edd}$, HDHHO, and IG algorithms

Table 4. Average RPD values of MILP, ESL, $NEHD_{edd}$, HDHHO, and IG algorithm.

Factory	Job	MILP	ESL	$NEHD_{edd}$	HDHHO	IG
2	4	0	10.276	12.552	0	0
	6	0	7.572	7.528	0.022	0
	8	0	14.052	8.131	0.978	0.022
	10	0	13.382	7.939	2.204	0
	12	N/A	14.192	10.040	2.941	0.120
	14	N/A	14.848	8.879	4.478	0.076
	16	N/A	18.971	8.672	4.555	0.166
3	4	0	1.163	21.221	0	0
	6	0	5.587	18.017	1.645	0
	8	0	10.802	12.621	0.708	0
	10	0	15.157	17.479	1.560	0.056
	12	N/A	13.887	13.895	2.691	0.035
	14	N/A	16.638	11.961	3.719	0.178
	16	N/A	14.690	11.031	3.885	0.367
4	4	0	0	0	0	0
	6	0	3.058	15.968	0	0
	8	0	11.040	22.882	0.656	0
	10	N/A	13.414	23.422	1.578	0
	12	N/A	12.474	15.973	2.807	0.024
	14	N/A	14.831	18.932	3.290	0.179
	16	N/A	16.170	15.609	3.096	0.499

grouped by number of factories and jobs. HDHHO and IG algorithms run three times with the termination criterion as $n \times m \times 0.25$ seconds. The MILP model obtains a total of 220 optimal solutions, while ESL, $NEHD_{edd}$, HDHHO, and IG algorithms obtain 100, 73, 207, and 216 out of 220 of the obtained optimal solutions, respectively. Notably, IG algorithm was able to rapidly converge and obtain (near-) optimal solutions for most test instances of the small problem set.

7.2. Evaluations of the proposed algorithms

The presented algorithms, HDHHO, and IG are tested on the 720 large instances. To evaluate the effectiveness of the presented algorithms, we compare them against the following seven other metaheuristics:

- (1) Hybrid genetic algorithm (HGA) (Gao and Chen 2011)
- (2) Hybrid immune algorithm (HIA) (Xu et al. 2014)
- (3) Scatter search algorithm (SSA) (Naderi and Ruiz 2014)
- (4) Hybrid Iterated Greedy Algorithm (HIG) (Karabulut 2016)
- (5) Chemical reaction optimisation algorithm (CRO) (Bargaoui, Driss, and Ghédira 2017)

Table 5. The parameters setting of compared algorithms.

Algorithms	Parameters setting
HGA	$pm = 0.4$
SSA	$b = 17, l = 7, \alpha = 15$
HIA	$\alpha = 0.4, \gamma = 0.1, t_1 = 50$
CRO	$KELossRate = 0.7, InitialKE = 2, \alpha = 165, \beta = 3$
DGWO	no parameters to be calibrated

Table 6. ARPD values of each algorithm grouped by n and m .

F	$n \times m$	HGA	HIA	SSA	HIG	CRO	DGWO	ILS	HDHHO	IG
2	20 × 5	10.3/1.4/2.9	5.7/1.1/2.4	1.8/2.5/1.4	2.2/1.5/1.1	5.0/3.9/1.8	3.8/3.1/1.7	1.4/1.6/1.5	1.1/1.6/1.8	1.7/1.2/1.1
	20 × 10	8.8/1.8/3.1	5.2/1.3/2.3	2.2/1.6/2.4	2.3/2.0/1.0	3.8/4.1/2.0	3.0/2.4/1.7	1.2/1.7/1.4	1.1/1.5/1.3	2.1/1.8/0.7
	20 × 20	3.3/0.7/1.4	2.7/0.5/1.1	1.1/1.1/1.2	1.2/1.1/0.6	2.6/2.4/1.0	1.2/1.9/0.7	1.0/0.9/0.9	0.4/0.7/1.1	1.2/1.1/0.8
	50 × 5	14.5/12.2/8.0	13.0/11.5/6.2	10.4/8.1/3.8	2.6/2.3/2.9	10.1/10.6/5.0	11.9/10.0/3.1	2.0/2.2/2.1	8.5/5.5/3.2	1.5/1.6/2.5
	50 × 10	22.1/20.2/10.3	17.6/18.8/8.8	17.9/11.4/4.8	4.3/4.5/4.2	19.1/16.7/7.4	19.9/12.4/4.5	3.6/3.4/4.4	14.6/7.8/5.7	3.4/3.0/3.0
	50 × 20	19.7/20.4/10.8	20.3/19.4/9.4	17.4/10.9/4.9	4.2/5.7/4.5	19.1/17.4/8.0	19.2/14.1/6.3	4.0/4.7/5.3	13.9/7.9/5.4	3.3/3.0/3.6
	100 × 5	5.1/5.1/4.4	5.8/4.6/4.0	3.8/4.2/2.0	1.9/1.9/1.5	5.3/5.6/3.1	4.3/4.9/2.2	1.5/1.2/1.5	2.9/3.1/1.8	0.9/0.9/1.0
	100 × 10	9.6/9.4/7.2	11.1/8.7/6.3	7.0/7.6/3.6	2.5/2.8/2.5	9.3/10.3/5.5	7.8/8.5/3.8	2.1/1.9/2.2	6.2/5.3/2.9	1.7/1.0/2.2
	100 × 20	16.4/20.2/12.2	17.6/20.1/10.1	14.9/13.2/6.5	4.0/4.8/4.3	14.8/16.5/8.3	16.2/15.2/6.3	4.1/3.6/2.9	14.1/9.4/5.1	1.9/2.0/3.2
	200 × 10	3.7/2.8/3.3	3.5/2.8/2.8	1.6/3.3/1.3	1.3/1.0/0.9	3.3/4.3/2.0	1.9/3.7/1.2	1.1/1.2/0.7	1.6/2.2/1.0	0.6/0.6/0.8
	200 × 20	12.4/7.6/9.9	10.6/7.4/8.3	4.5/8.7/4.2	3.5/3.1/2.7	9.6/12.0/5.7	5.6/10.0/4.8	3.2/2.5/2.4	4.5/5.6/3.5	1.3/1.5/2.5
	500 × 20	0.9/0.4/0.3	0.8/0.4/0.3	0.4/0.3/0.2	0.0/0.0/0.0	0.6/0.3/0.2	0.5/0.3/0.2	0.0/0.0/0.0	0.3/0.3/0.1	0.0/0.0/0.0
3	20 × 5	9.9/3.6/3.1	8.9/3.1/2.4	3.8/2.9/2.1	2.0/2.5/1.4	8.5/4.7/2.6	4.3/3.9/2.3	1.6/1.4/1.4	2.6/2.1/0.9	2.0/1.5/1.2
	20 × 10	11.5/3.0/3.0	13.1/3.0/2.6	3.3/2.4/1.9	2.6/2.6/1.7	8.8/6.2/2.3	5.0/3.6/2.3	1.8/2.5/2.2	3.4/1.8/2.3	2.2/1.1/1.1
	20 × 20	4.9/1.3/1.9	5.5/1.4/1.9	1.7/1.2/1.5	1.8/1.0/0.9	3.8/2.4/1.3	2.2/2.5/0.9	1.3/0.7/1.2	1.2/1.2/0.9	1.3/0.8/0.8
	50 × 5	18.0/15.3/9.9	18.7/15.2/7.5	12.0/10.8/4.0	3.4/3.5/3.0	16.1/15.1/6.3	12.4/12.3/3.7	2.7/4.2/2.5	11.4/7.4/3.8	2.1/2.4/2.0
	50 × 10	22.7/22.6/12.0	23.1/22.1/10.4	18.2/12.6/6.2	4.1/5.3/4.0	19.1/16.2/7.9	19.1/13.8/7.2	3.5/4.3/4.4	17.0/8.3/4.7	1.8/2.5/2.9
	50 × 20	28.6/22.8/12.2	28.5/23.1/11.3	21.2/14.4/6.2	6.7/5.8/5.3	25.5/20.6/8.5	24.6/17.0/6.9	6.8/5.7/6.9	21.3/9.3/7.0	4.0/4.1/4.4
	100 × 5	7.5/6.1/5.8	7.4/6.0/4.5	4.6/4.8/2.6	1.9/1.9/1.8	6.7/7.0/3.8	4.7/5.7/2.6	1.7/1.9/1.5	4.1/3.4/2.2	1.1/0.9/1.4
	100 × 10	11.7/9.7/8.0	12.0/9.8/6.1	7.6/7.7/3.0	2.8/2.9/2.0	9.1/10.5/5.7	7.8/8.7/3.7	2.4/2.6/2.2	7.4/5.9/3.0	2.2/1.5/1.8
	100 × 20	17.3/18.2/12.2	17.8/17.7/10.2	14.2/10.9/6.2	4.1/4.8/3.8	13.5/14.6/9.0	14.8/12.5/6.4	3.7/4.4/4.1	13.3/8.1/5.3	2.6/2.1/3.6
	200 × 10	4.3/2.9/2.9	4.0/2.9/2.5	2.1/3.5/1.4	1.2/1.4/0.8	3.2/4.0/2.3	2.4/3.8/1.5	1.1/1.1/0.8	1.7/2.5/1.1	0.9/0.6/0.7
	200 × 20	10.4/6.6/8.3	10.2/6.5/6.8	3.6/8.0/3.3	2.5/2.6/3.1	9.1/9.9/6.3	5.1/9.1/4.0	1.6/2.5/2.4	3.7/6.0/3.0	1.9/0.9/2.1
	500 × 20	0.9/0.4/0.2	0.7/0.3/0.2	0.5/0.2/0.1	0.0/0.0/0.0	0.6/0.3/0.2	0.5/0.2/0.2	0.0/0.0/0.0	0.5/0.2/0.1	0.0/0.0/0.0
4	20 × 5	13.5/4.8/3.4	14.6/4.5/2.6	4.6/2.0/1.8	1.7/1.7/1.0	9.8/4.3/1.8	5.3/3.6/1.5	1.2/1.7/1.6	4.1/1.8/1.7	0.9/1.0/1.4
	20 × 10	16.6/3.7/4.0	19.6/3.9/3.3	4.6/4.2/3.4	2.8/2.8/1.5	14.2/4.9/4.0	7.5/6.7/2.7	2.1/4.5/1.9	4.8/4.0/3.3	3.3/1.6/2.1
	20 × 20	8.6/1.4/2.4	10.0/1.1/2.0	1.7/1.2/1.7	1.8/0.7/0.5	5.8/3.0/0.8	2.2/2.7/0.6	1.5/1.0/0.7	1.4/0.6/1.0	0.8/0.5/0.7
	50 × 5	20.8/18.5/11.3	20.6/18.7/9.1	14.5/13.0/5.0	4.2/4.8/2.6	17.4/15.7/7.0	14.3/14.5/4.5	3.5/4.6/3.2	13.8/8.9/4.1	2.7/2.9/2.0
	50 × 10	30.2/29.5/15.8	32.2/28.8/12.4	23.6/18.7/8.4	7.8/6.5/3.6	27.9/24.1/8.8	25.0/21.9/6.4	6.9/7.0/5.2	23.9/12.2/6.0	4.4/4.4/3.0
	50 × 20	24.4/24.7/12.8	25.5/23.1/10.6	20.2/13.4/7.0	4.4/5.1/5.4	22.6/17.5/10.0	20.6/15.9/6.6	3.2/5.0/6.2	17.5/10.1/6.6	3.9/3.6/2.6
	100 × 5	7.0/6.3/5.2	7.0/6.2/4.8	4.2/5.6/2.5	2.1/2.1/1.4	6.4/6.7/3.4	4.9/6.6/2.0	2.0/2.0/2.1	3.8/3.9/1.8	0.8/1.3/0.7
	100 × 10	11.2/10.1/7.7	11.3/9.8/6.0	7.0/7.8/3.5	2.1/3.0/2.4	8.6/10.4/5.3	7.1/8.6/3.3	1.9/2.5/2.5	6.5/5.8/2.6	1.6/1.4/1.3
	100 × 20	16.8/16.4/12.3	14.5/16.5/10.8	12.2/11.3/5.0	3.9/3.2/3.8	13.5/12.8/8.4	13.3/12.8/5.8	3.4/3.6/5.1	11.8/8.0/4.4	1.6/2.9/2.1
	200 × 10	4.3/2.9/3.4	4.0/2.5/2.8	2.0/3.7/1.4	1.2/1.2/1.1	3.9/4.4/2.1	2.3/4.0/1.6	1.1/1.1/1.4	1.3/2.3/1.1	0.6/0.9/0.4
	200 × 20	11.1/6.7/8.5	8.9/6.8/7.1	3.7/7.9/3.5	2.7/2.3/2.8	9.3/8.9/5.7	4.6/8.7/3.3	2.4/2.2/3.0	3.8/5.3/3.1	1.2/2.0/1.4
	500 × 20	1.0/0.4/0.5	0.8/0.3/0.5	0.6/0.2/0.4	0.0/0.0/0.3	0.6/0.3/0.5	0.6/0.2/0.5	0.0/0.0/0.2	0.5/0.2/0.4	0.0/0.0/0.1
5	20 × 5	18.0/5.6/3.4	11.0/4.9/2.1	6.7/2.2/1.9	2.1/1.9/1.1	13.7/6.0/2.1	7.7/4.3/1.7	1.5/1.7/2.1	5.2/1.9/1.3	2.2/1.1/0.6
	20 × 10	15.8/3.9/5.4	11.4/3.7/4.2	6.2/2.9/1.7	3.9/3.3/1.5	13.3/6.6/3.1	7.3/4.8/2.9	2.7/2.5/2.3	5.2/2.7/2.9	2.7/2.5/0.4
	20 × 20	6.9/2.0/2.5	6.4/1.7/1.7	2.5/1.3/0.8	1.8/1.1/0.9	6.0/2.9/1.2	2.9/2.5/1.3	1.2/1.2/1.7	1.6/1.1/1.4	1.4/1.3/0.3
	50 × 5	23.4/20.2/13.9	19.5/19.5/12.0	15.4/16.1/5.6	5.7/5.1/4.2	20.5/19.6/9.7	16.7/17.6/6.2	4.9/4.3/5.3	14.6/10.0/5.1	3.6/3.8/1.8
	50 × 10	31.9/30.3/18.5	27.7/30.0/14.9	23.0/18.5/7.7	5.8/7.3/5.7	27.3/25.3/13.2	25.6/21.0/9.7	5.7/5.9/7.8	22.4/11.8/6.3	4.3/6.2/3.3
	50 × 20	31.0/27.4/16.5	31.5/27.2/13.4	23.5/18.0/7.4	6.5/6.2/5.0	27.1/22.5/9.8	25.3/20.6/6.6	5.2/6.0/7.5	22.3/10.8/6.1	3.5/5.1/2.8
	100 × 5	8.4/5.9/5.3	8.3/5.8/5.2	4.7/5.2/2.6	2.5/2.1/1.9	7.6/8.8/3.8	5.3/5.9/2.9	2.0/1.5/2.5	4.2/3.5/2.5	1.0/1.3/0.9
	100 × 10	11.2/10.1/9.7	12.1/9.8/8.2	7.8/7.4/5.0	2.7/2.5/3.9	10.3/9.7/6.3	8.3/8.5/5.1	2.5/2.4/4.0	7.2/5.1/3.9	1.2/1.9/1.6
	100 × 20	16.8/16.8/11.7	16.6/15.9/10.6	12.2/10.8/5.3	3.9/3.7/4.4	14.6/13.4/8.4	13.1/13.0/5.8	3.7/3.6/4.4	12.1/7.7/4.8	2.1/2.6/1.9
	200 × 10	4.8/3.0/3.8	4.2/2.8/3.4	2.0/3.5/1.7	1.7/1.4/1.4	4.2/4.0/2.6	2.2/4.1/1.9	1.4/1.0/1.6	1.8/2.4/1.4	0.8/0.8/0.7
	200 × 20	10.6/6.9/8.7	8.9/6.5/7.1	3.7/8.1/3.8	2.5/2.5/2.5	9.0/9.5/5.8	4.9/9.5/3.5	2.1/2.3/2.8	3.0/5.7/3.4	1.2/2.4/1.3
	500 × 20	1.4/0.6/0.4	1.0/0.6/0.4	0.6/0.4/0.3	0.0/0.0/0.0	0.9/0.5/0.4	0.7/0.4/0.3	0.0/0.1/0.0	0.6/0.4/0.3	0.0/0.0/0.0
6	20 × 5	8.5/2.4/1.2	10.0/2.2/0.6	2.5/0.9/0.3	0.6/0.4/0.1	7.4/2.5/0.8	4.6/2.2/0.4	0.6/0.4/0.3	2.9/0.9/0.2	0.8/0.7/0.1
	20 × 10	8.9/3.5/1.1	10.3/2.9/1.1	3.8/1.0/0.5	0.9/0.6/0.2	5.4/2.6/1.2	4.4/2.0/0.4	0.7/0.3/1.0	2.7/1.1/0.6	0.8/0.7/0.2
	20 × 20	6.2/1.7/1.1	7.1/1.0/0.9	2.2/0.9/0.7	1.0/0.7/0.2	3.6/1.3/0.5	2.6/1.7/0.7	0.8/0.2/0.3	1.2/0.6/0.4	0.7/0.6/0.1
	50 × 5	21.9/19.7/15.5	23.6/19.4/13.1	15.1/15.5/6.1	4.7/4.4/5.1	20.2/19.1/10.7	15.8/17.9/7.4	4.3/4.4/4.3	13.5/10.8/5.7	2.7/3.7/2.4
	50 × 10	33.1/29.5/17.9	32.8/29.3/15.9	23.4/21.3/6.6	7.8/8.4/5.0	29.9/25.8/11.4	25.3/24.5/9.4	7.8/7.5/5.4	21.9/13.6/6.3	3.6/4.9/3.4
	50 × 20	39.8/32.6/20.5	39.6/31.9/17.6	29.2/22.2/10.0	9.0/6.7/7.5	34.1/27.2/14.4	29.8/25.1/9.4	7.6/6.5/8.2	26.6/14.6/8.5	6.2/5.7/3.8
	100 × 5	9.4/6.9/6.0	8.9/6.7/4.5	4.7/6.2/2.4	1.8/2.2/1.6	8.2/8.1/3.9	5.0/7.1/2.2	1.7/2.2/1.6	4.2/4.4/2.0	1.2/1.4/0.8
	100 × 10	12.3/10.4/9.2	12.1/10.5/7.8	6.9/8.7/3.8	2.5/3.1/3.0	11.2/10.4/6.6	7.6/9.6/4.3	2.3/2.1/2.2	6.6/5.9/3.3	1.1/2.4/1.2
	100 × 20	15.8/16.6/12.1	15.9/15.7/10.8	12.3/11.5/5.9	4.3/4.7/4.3	13.0/13.8/8.4	12.5/13.2/5.9	4.1/3.5/4.1	11.0/8.0/5.0	1.8/2.6/1.9
	200 × 10	4.5/3.1/3.5	4.1/3.2/3.2	2.1/3.5/1.6	1.7/1.4/1.0	4.3/4.6/2.3	2.5/4.0/1.5	1.4/1.3/1.0	1.9/2.5/1.3	0.8/0.8/0.5
	200 × 20	10.3/5.9/7.7	9.3/5.7/6.5	3.6/7.3/3.8	2.6/2.5/2.3	8.8/9.4/5.2	5.0/8.1/3.3	2.3/2.3/2.4	3.4/5.1/3.0	1.6/1.4/1.2
	500 × 20	0.9/0.5/0.2	0.6/0.4/0.2	0.6/0.3/0.2	0.0/0.0/0.0	0.6/0.4/0.2	0.6/0.3/0.2	0.0/0.0/1.6	0.5/0.2/0.1	0.0/0.0/1.5
7	20 × 5	9.4/2.0/0.7	11.7/2.2/0.6	2.7/0.7/0.0	0.4/0.3/0.0	7.2/2.5/0.4	4.3/1.7/0.1	0.4/0.3/0.1	2.5/0.4/0.1	0.4/0.3/0.0
	20 × 10	8.1/3.8/0.8	9.6/2.9/0.8	4.0/0.5/0.3	0.7/0.5/0.3	6.3/2.1/0.5	4.4/1.4/0.5	0.3/0.4/0.3	2.2/0.5/0.4	0.3/0.4/0.1
	20 × 20	4.4/0.7/1.0	5.1/0.7/0.4	1.						

- (6) Discrete grey wolf optimisation (DGWO) (Khare and Agrawal 2019)
- (7) Iterated local search (ILS) (Pan et al. 2019)

All these metaheuristics are recent and competitive algorithms, representing the best performance against different scheduling problems. We carefully reimplemented them by considering our solution representation and objective function. Moreover, the values of adjustable parameters for the compared algorithms are tuned as the same in Section 6.2 and the results are given in Table 5. For an unbiased assessment, the values of common parameters, i.e. the termination criterion and population size are considered the same. All these methods are coded in MATLAB R2014a and ran on the same PC, as mentioned in Section 6.2. For each algorithm, we performed five independent runs with the termination criterion as $n \times m \times \rho$ seconds, where $\rho \in \{0.25, 0.30, 0.35\}$. Average relative percentage deviation (ARPD) considered as a performance measure is described as

follows:

$$ARPD = \frac{Alg_{avg} - Best_{avg}}{Best_{avg}} \times 100 \quad (29)$$

where Alg_{avg} is the average value of total tardiness for a particular algorithm over 50 results (ten problems replicate of the same combination and their five independent runs) and $Best_{avg}$ is the mean of the lowest total tardiness obtained in the whole experiment over the same combination for a particular ρ value. Table 6 shows the ARPD values of each algorithm grouped by n and m . The computational results with respect to the three ρ values are separated by a slash ($\rho = 0.25/0.30/0.35$). From Table 6, it can be observed that the presented algorithms perform much better than most of the compared metaheuristics in terms of ARPD values. The IG algorithm turned out to be the best performing algorithm that achieves the least ARPD values for most of the problem instances, regardless of the CPU time employed. ILS and HIG ranked second and third, respectively with slightly

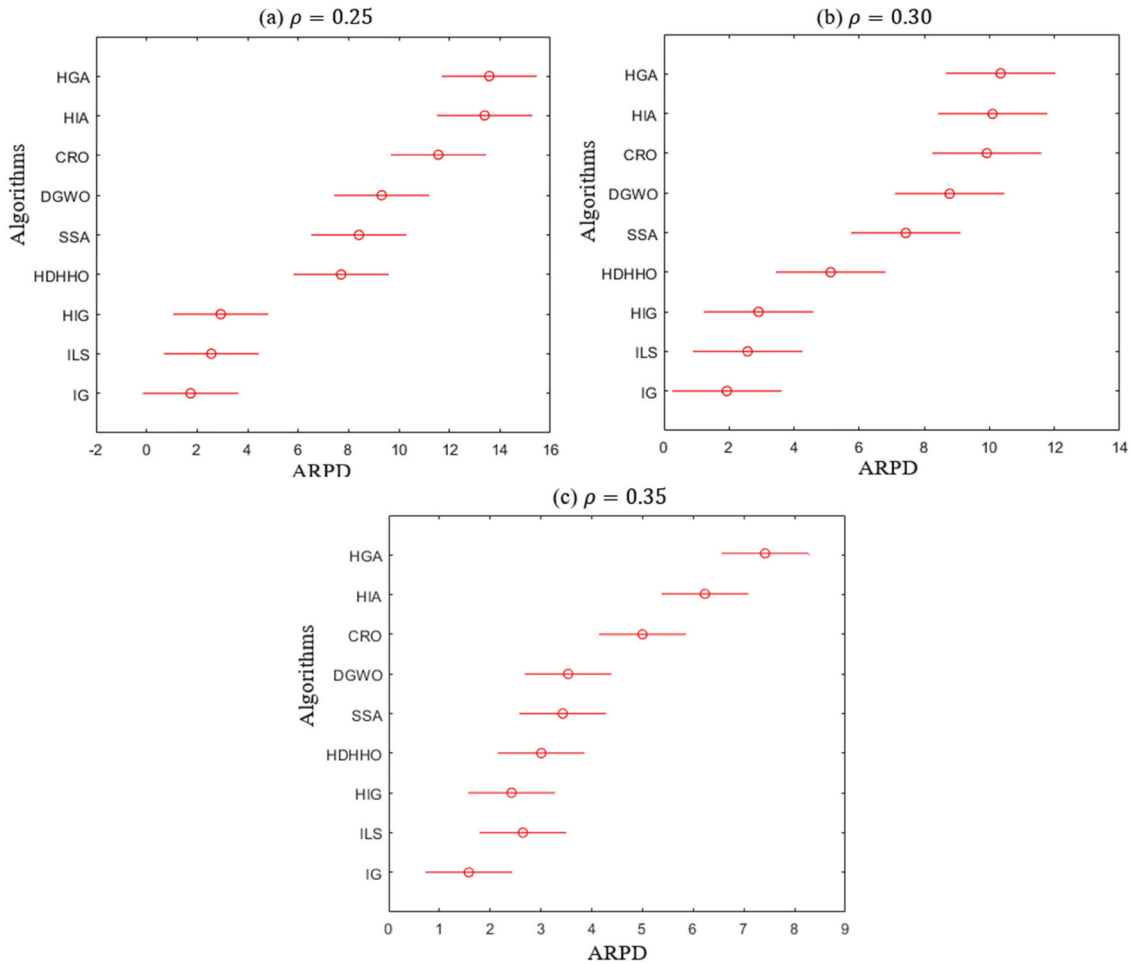


Figure 9. Mean plot with 95% Tukey-HSD confidence interval of ARPD obtained by all the algorithms.

worse overall ARPD value than IG algorithm. For the 720 benchmark instances, IG algorithm obtained 543 (75.42%), 441 (61.25%), and 469 (65.13%) best solutions for $\rho = 0.25, 0.30$, and 0.35 , respectively. The overall performance of the HDHHO algorithm is superior to HIA, HGA, CRO, DGWO, and SSA but inferior to HIG, ILS, and IG algorithms.

We conduct a multifactor ANOVA to investigate the statistical validity of the results listed in Table 6. Here, all the competing algorithms are considered as different factors. Figure 9 illustrates the means plots with 95% Tukey's Honest Significant Difference (HSD) confidence intervals of the interaction between compared algorithms for different CPU time. Note that the overlapping confidence intervals of two algorithms mean a statistical insignificant difference between them. Hence, all the competing metaheuristics can be arranged into separate groups, with insignificant statistically differences within each group. From Figure 9(a), it can be observed that the competing algorithms can be divided into three different groups, with no significant differences within each group. These groups, from worst to best, are {HGA, HIA, CRO} {CRO, DGWO, SSA, HDHHO} {HIG, ILS, IG}. Similarly, from Figure 9(b), {HGA, HIA, CRO, DGWO, SSA} {SSA, HDHHO} {HDHHO, HIG, ILS, IG} and finally from Figure 9(c), {HGA, HIA} {HIA, CRO} {CRO, DGWO, SSA} {DGWO, SSA, HDHHO, ILS, HIG} {HDHHO, HIG, ILS, IG}. It can be observed that most of the algorithms are improved progressively as more CPU time is allowed. Particularly, as the value of ρ is increased, the performance of the HDHHO algorithm also improved. This confirms that for the considered problem, the integration of path-relinking method with HHO algorithm enhanced its performance and guided it to better resolve local optima stagnation. Moreover, it is interesting to note that for the considered problem, the trajectory-based algorithms (HIG, ILS, and IG) outperform the population-based metaheuristics, especially for the shortest CPU time limit ($\rho = 0.25$). The excellent performance of the presented IG algorithm throughout the entire evaluation phase indicates that the selection of $NEHD_{edd}$ heuristic for initialisation, problem-based selection of jobs for destruction and construction phases and subsequent random local search enhanced its overall performance for the considered problem. The presented, IG and HDHHO along with HIG and ILS algorithms ranked among the top four out of all nine competing algorithms. Therefore, it can be safely established that these algorithms are very effective for solving the DPFSP with total tardiness criterion. Among them, the presented IG algorithm is the best one for $DF|prmu| \sum T_j$ problem that can provide better results, especially in the circumstances that require a faster solution.

8. Conclusion and future research

This paper studies the distributed permutation flow-shop scheduling problem (DPFSP) with total tardiness criterion. To the best of our knowledge, it is the first effort to minimise the total tardiness for DPFSP. We have presented five solution schemes to solve the considered problem with different computational effort and solution quality. These are a mixed integer linear programming model, two heuristics: ESL and $NEHD_{edd}$, a population-based hybrid discrete Harris hawk optimisation (HDHHO) and an enhanced iterative greedy (IG) algorithm. Design of experiment and ANOVA are employed to analyse and calibrate the parameters of the proposed algorithms. To evaluate the algorithmic performance, the well-known benchmark set of Naderi and Ruiz for DPFSP is extended with due dates. We compared the presented algorithms with seven other metaheuristics and statistically verified the result. The detailed computational evaluation shows that the presented metaheuristics, particularly IG algorithm produce the best result for the considered problem.

For future research, it could be interesting to study the considered problem with different objectives such as minimising tardy jobs, minimising total earliness, and tardiness. Other aspects could include more real-life constraints such as due windows instead of due dates and set-up times.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Notes on contributors



Ankit Khare is a Ph.D. student in the Department of Mechanical Engineering, PDPM, Indian Institute of Information Technology, Design and Manufacturing, Jabalpur, India. He received his B.E. degree in Production and Industrial Engineering from Jabalpur Engineering College, India, in 2012 and M.Tech degree in Computer Aided Manufacturing from Motilal Nehru National Institute of Technology, Prayagraj, India, in 2015. His current research interest includes computational intelligence with applications in scheduling, transportation, and manufacturing domains.



Sunil Agrawal is an associate professor at Indian Institute of Information Technology, Design and Manufacturing Jabalpur, India. He received his PhD in Industrial and Management Engineering from Indian Institute of Technology Kanpur, India in 2008. His research interests are in the areas of Supply Chain Management, Inventory Management, and Manufacturing Systems. His research works are Bullwhip Effect in Supply Chains,

Inventory Order Crossover, Agriculture Supply Chain Management, Flow Shop Scheduling, and Two-sided Assembly Line Balancing Problems. Currently he is working in the domain of use of cyber physical systems in managing Agriculture Supply Chains. His research results were disseminated in 15 international journals and more than 35 national and international conferences.

ORCID

Ankit Khare  <http://orcid.org/0000-0002-2021-2006>

References

- Bargaoui, H., O. B. Driss, and K. Ghédira. 2017. "A Novel Chemical Reaction Optimization for the Distributed Permutation Flowshop Scheduling Problem with Makespan Criterion." *Computers and Industrial Engineering* 111: 239–250. doi:10.1016/j.cie.2017.07.020.
- Cai, J., D. Lei, and M. Li. 2020. "A Shuffled Frog-Leaping Algorithm with Memplex Quality for Bi-Objective Distributed Scheduling in Hybrid Flow Shop." *International Journal of Production Research* (June 25, 2020), 1–18. doi:10.1080/00207543.2020.1780333.
- Chen, T., and X. Li. 2013. "Integrated Iterated Local Search for the Permutation Flowshop Problem with Tardiness Minimization." *Proceedings – 2013 IEEE international Conference on systems, Man, and Cybernetics, SMC 2013*. doi:10.1109/SMC.2013.478.
- Cheng, C.-Y., K.-C. Ying, H.-H. Chen, and H.-S. Lu. 2018. "Minimising Makespan in Distributed Mixed no-Idle Flowshops." *International Journal of Production Research* 57 (1): 48–60. doi:10.1080/00207543.2018.1457812.
- Costa, W. E., M. C. Goldbarg, and E. G. Goldbarg. 2012. "Hybridizing VNS and Path-Relinking on a Particle Swarm Framework to Minimize Total Flowtime." *Expert Systems with Applications* 39 (18): 13118–13126. doi:10.1016/j.eswa.2012.05.090.
- Cura, T. 2015. "An Evolutionary Algorithm for the Permutation Flowshop Scheduling Problem with Total Tardiness Criterion." *International Journal of Operational Research* 22 (3): 366–384. doi:10.1504/IJOR.2015.068287.
- Du, J., and J. Y.-T. Leung. 1990. "Minimizing Total Tardiness on One Machine is NP-Hard." *Mathematics of Operations Research* 15 (3): 483–495. doi:10.1287/moor.15.3.483.
- Fernandez-Viagas, V., and J. M. Framinan. 2015a. "A Bounded-Search Iterated Greedy Algorithm for the Distributed Permutation Flowshop Scheduling Problem." *International Journal of Production Research* 53 (4): 1111–1123. doi:10.1080/00207543.2014.948578.
- Fernandez-Viagas, V., and J. M. Framinan. 2015b. "NEH-Based Heuristics for the Permutation Flowshop Scheduling Problem to Minimise Total Tardiness." *Computers and Operations Research*. doi:10.1016/j.cor.2015.02.002.
- Fernandez-Viagas, V., P. Perez-Gonzalez, and J. M. Framinan. 2018. "The Distributed Permutation Flow Shop to Minimise the Total Flowtime." *Computers and Industrial Engineering* 118: 464–477. doi:10.1016/j.cie.2018.03.014.
- Fernandez-Viagas, V., J. M. Valente, and J. M. Framinan. 2018. "Iterated-Greedy-Based Algorithms with Beam Search Initialization for the Permutation Flowshop to Minimise Total Tardiness." *Expert Systems with Applications* 94: 58–69. doi:10.1016/j.eswa.2017.10.050.
- Gao, J., and R. Chen. 2011. "A Hybrid Genetic Algorithm for the Distributed Permutation Flowshop Scheduling Problem." *International Journal of Computational Intelligence Systems* 4 (4): 497–508. doi:10.1080/18756891.2011.9727808.
- Gao, J., R. Chen, and W. Deng. 2013. "An Efficient Tabu Search Algorithm for the Distributed Permutation Flowshop Scheduling Problem." *International Journal of Production Research* 51 (3): 641–651. doi:10.1080/00207543.2011.644819.
- Glover, F., and M. Laguna. 1998. "Tabu Search." *Handbook of Combinatorial Optimization*, 2093–2229. doi:10.1007/978-1-4613-0303-9_33.
- Golilarz, N. A., H. Gao, and H. Demirel. 2019. "Satellite Image De-Noiseing with Harris Hawks Meta Heuristic Optimization Algorithm and Improved Adaptive Generalized Gaussian Distribution Threshold Function." *IEEE Access* 7: 57459–57468. doi:10.1109/ACCESS.2019.2914101.
- Graham, R. L., E. L. Lawler, J. K. Lenstra, and A. R. Kan. 1979. "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey." *Annals of Discrete Mathematics*, 287–326. doi:10.1016/s0167-5060(08)70356-x.
- Hamzadayi, A. 2020. "An Effective Benders Decomposition Algorithm for Solving the Distributed Permutation Flowshop Scheduling Problem." *Computers & Operations Research*, 105006. doi:10.1016/j.cor.2020.105006.
- Hasija, S., and C. Rajendran. 2004. "Scheduling in Flowshops to Minimize Total Tardiness of Jobs." *International Journal of Production Research* 42 (11): 2289–2301. doi:10.1080/00207540310001657595.
- Heidari, A. A., S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, and H. Chen. 2019. "Harris Hawks Optimization: Algorithm and Applications." *Future Generation Computer Systems* 97: 849–872. doi:10.1016/j.future.2019.02.028.
- Huang, J. P., Q. K. Pan, and L. Gao. 2020. "An Effective Iterated Greedy Method for the Distributed Permutation Flowshop Scheduling Problem with Sequence-Dependent Setup Times." *Swarm and Evolutionary Computation* 59: 100742. doi:10.1016/j.swevo.2020.100742.
- Karabulut, K. 2016. "A Hybrid Iterated Greedy Algorithm for Total Tardiness Minimization in Permutation Flowshops." *Computers and Industrial Engineering* 98: 300–307. doi:10.1016/j.cie.2016.06.012.
- Kellegöz, T., B. Toklu, and J. Wilson. 2010. "Elite Guided Steady-State Genetic Algorithm for Minimizing Total Tardiness in Flowshops." *Computers and Industrial Engineering* 58 (2): 300–306. doi:10.1016/j.cie.2009.11.001.
- Khare, A., and S. Agrawal. 2019. "Scheduling Hybrid Flowshop with Sequence-Dependent Setup Times and Due Windows to Minimize Total Weighted Earliness and Tardiness." *Computers and Industrial Engineering* 135: 780–792. doi:10.1016/j.cie.2019.06.057.
- Kim, Y. D. 1993. "Heuristics for Flowshop Scheduling Problems Minimizing Mean Tardiness." *Journal of the Operational Research Society* 44 (1): 19–28. doi:10.1057/jors.1993.3.
- Li, J. Q., S. C. Bai, P. Y. Duan, H. Y. Sang, Y. Y. Han, and Z. X. Zheng. 2019. "An Improved Artificial Bee Colony Algorithm for Addressing Distributed Flow Shop with Distance Coefficient in a Prefabricated System." *International Journal of Production Research* 57 (22): 6922–6942. doi:10.1080/00207543.2019.1571687.
- Li, X., L. Chen, H. Xu, and J. N. Gupta. 2015. "Trajectory Scheduling Methods for Minimizing Total Tardiness

- in a Flowshop.” *Operations Research Perspectives* 2: 13–23. doi:10.1016/j.orp.2014.12.001.
- Li, Y., X. Li, L. Gao, B. Zhang, Q. K. Pan, M. F. Tasgetiren, and L. Meng. 2020. “A Discrete Artificial bee Colony Algorithm for Distributed Hybrid Flowshop Scheduling Problem with Sequence-Dependent Setup Times.” *International Journal of Production Research*, 1–20. doi:10.1080/00207543.2020.1753897.
- Lin, S. W., K. C. Ying, and C. Y. Huang. 2013. “Minimising Makespan in Distributed Permutation Flowshops Using a Modified Iterated Greedy Algorithm.” *International Journal of Production Research* 51 (16): 5029–5038. doi:10.1080/00207543.2013.790571.
- Meng, T., Q. K. Pan, and L. Wang. 2019. “A Distributed Permutation Flowshop Scheduling Problem with the Customer Order Constraint.” *Knowledge-Based Systems* 184: 104894. doi:10.1016/j.knsys.2019.104894.
- Moayedi, H., H. Nguyen, and A. S. A. Rashid. 2019. “Comparison of Dragonfly Algorithm and Harris Hawks Optimization Evolutionary Data Mining Techniques for the Assessment of Bearing Capacity of Footings Over Two-Layer Foundation Soils.” *Engineering with Computers*, 1–11. doi:10.1007/s00366-019-00834-w.
- Montgomery, D. C. 2007. *Design and Analysis of Experiments*. 5th ed. New York: John Wiley & Sons.
- Naderi, B., and R. Ruiz. 2010. “The Distributed Permutation Flowshop Scheduling Problem.” *Computers and Operations Research* 37 (4): 754–768. doi:10.1016/j.cor.2009.06.019.
- Naderi, B., and R. Ruiz. 2014. “A Scatter Search Algorithm for the Distributed Permutation Flowshop Scheduling Problem.” *European Journal of Operational Research* 239 (2): 323–334. doi:10.1016/j.ejor.2014.05.024.
- Olhager, J., and A. Feldmann. 2017. “Distribution of Manufacturing Strategy Decision-Making in Multi-Plant Networks.” *International Journal of Production Research* 56 (1–2): 692–708. doi:10.1080/00207543.2017.1401749.
- Pan, Q. K., L. Gao, L. Wang, J. Liang, and X. Y. Li. 2019. “Effective Heuristics and Metaheuristics to Minimize Total Flowtime for the Distributed Permutation Flowshop Problem.” *Expert Systems with Applications* 124: 309–324. doi:10.1016/j.eswa.2019.01.062.
- Pan, Q. K., L. Wang, H. Y. Sang, J. Q. Li, and M. Liu. 2013. “A High Performing Memetic Algorithm for the Flowshop Scheduling Problem with Blocking.” *IEEE Transactions on Automation Science and Engineering* 10 (3): 741–756. doi:10.1109/TASE.2012.2219860.
- Ribas, I., R. Companys, and X. Tort-Martorell. 2017. “Efficient Heuristics for the Parallel Blocking Flow Shop Scheduling Problem.” *Expert Systems with Applications* 74: 41–54. doi:10.1016/j.eswa.2017.01.006.
- Rifai, A. P., H. T. Nguyen, and S. Z. M. Dawal. 2016. “Multi-Objective Adaptive Large Neighborhood Search for Distributed Reentrant Permutation Flow Shop Scheduling.” *Applied Soft Computing Journal* 40: 42–57. doi:10.1016/j.asoc.2015.11.034.
- Ruiz, R., Q. K. Pan, and B. Naderi. 2018. “Iterated Greedy Methods for the Distributed Permutation Flowshop Scheduling Problem.” *Omega* 83: 213–222. doi:10.1016/j.omega.2018.03.004.
- Ruiz, R., and T. Stützle. 2007. “A Simple and Effective Iterated Greedy Algorithm for the Permutation Flowshop Scheduling Problem.” *European Journal of Operational Research* 177 (3): 2033–2049. doi:10.1016/j.ejor.2005.12.009.
- Shao, W., D. Pi, and Z. Shao. 2017. “Optimization of Makespan for the Distributed No-Wait Flow Shop Scheduling Problem with Iterated Greedy Algorithms.” *Knowledge-Based Systems* 137: 163–181. doi:10.1016/j.knsys.2017.09.026.
- Shao, Z., D. Pi, and W. Shao. 2020. “Hybrid Enhanced Discrete Fruit fly Optimization Algorithm for Scheduling Blocking Flow-Shop in Distributed Environment.” *Expert Systems with Applications* 145: 113147. doi:10.1016/j.eswa.2019.113147.
- Taillard, E. 1993. “Benchmarks for Basic Scheduling Problems.” *European Journal of Operational Research* 64 (2): 278–285. doi:10.1016/0377-2217(93)90182-M.
- Tasgetiren, M. F., Y. C. Liang, M. Sevkli, and G. Gencyilmaz. 2007. “A Particle Swarm Optimization Algorithm for Makespan and Total Flowtime Minimization in the Permutation Flowshop Sequencing Problem.” *European Journal of Operational Research* 177 (3): 1930–1947. doi:10.1016/j.ejor.2005.12.024.
- Vallada, E., and R. Ruiz. 2010. “Genetic Algorithms with Path Relinking for the Minimum Tardiness Permutation Flowshop Problem.” *Omega* 38 (1–2): 57–67. doi:10.1016/j.omega.2009.04.002.
- Vallada, E., R. Ruiz, and G. Minella. 2008. “Minimising Total Tardiness in the M-Machine Flowshop Problem: A Review and Evaluation of Heuristics and Metaheuristics.” *Computers and Operations Research* 35 (4): 1350–1373. doi:10.1016/j.cor.2006.08.016.
- Wang, S. Y., L. Wang, M. Liu, and Y. Xu. 2013. “An Effective Estimation of Distribution Algorithm for Solving the Distributed Permutation Flow-Shop Scheduling Problem.” *International Journal of Production Economics* 145 (1): 387–396. doi:10.1016/j.ijpe.2013.05.004.
- Xu, Y., L. Wang, S. Wang, and M. Liu. 2014. “An Effective Hybrid Immune Algorithm for Solving the Distributed Permutation Flow-Shop Scheduling Problem.” *Engineering Optimization* 46 (9): 1269–1283. doi:10.1080/0305215X.2013.827673.
- Ying, K. C., and S. W. Lin. 2017. “Minimizing Makespan in Distributed Blocking Flowshops Using Hybrid Iterated Greedy Algorithms.” *IEEE Access* 5: 15694–15705. doi:10.1109/ACCESS.2017.2732738.
- Ying, K. C., and S. W. Lin. 2018. “Minimizing Makespan for the Distributed Hybrid Flowshop Scheduling Problem with Multiprocessor Tasks.” *Expert Systems with Applications* 92: 132–141. doi:10.1016/j.eswa.2017.09.032.
- Ying, K. C., S. W. Lin, C. Y. Cheng, and C. D. He. 2017. “Iterated Reference Greedy Algorithm for Solving Distributed No-Idle Permutation Flowshop Scheduling Problems.” *Computers and Industrial Engineering* 110: 413–423. doi:10.1016/j.cie.2017.06.025.
- Zhang, G., and K. Xing. 2019. “Differential Evolution Metaheuristics for Distributed Limited-Buffer Flowshop Scheduling with Makespan Criterion.” *Computers and Operations Research* 108: 33–43. doi:10.1016/j.cor.2019.04.002.
- Zhang, G., K. Xing, and F. Cao. 2017. “Scheduling Distributed Flowshops with Flexible Assembly and set-up Time to Minimise Makespan.” *International Journal of Production Research* 56 (9): 3226–3244. doi:10.1080/00207543.2017.1401241.
- Zhao, F., L. Zhao, L. Wang, and H. Song. 2020. “An Ensemble Discrete Differential Evolution for the Distributed Blocking Flowshop Scheduling with Minimizing Makespan Criterion.” *Expert Systems with Applications* 160: 113678. doi:10.1016/j.eswa.2020.113678.