Scientia Iranica E (2020) 27(3), 1562–1571

SCIENTIA IRANICA

Sharif University of Technology
Scientia Iranica
Transactions E: Industrial Engineering
http://scientiairanica.sharif.edu

Mathematical models and an elephant herding optimization for multiprocessor-task flexible flow shop scheduling problems in the Manufacturing Resource Planning (MRPII) system

H.R. Gholami[a], E. Mehdizadeh[a,*], and B. Naderi[b]

a. *Faculty of Industrial and Mechanical Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran.*
b. *Department of Industrial Engineering, Faculty of Engineering, Kharazmi University, Tehran, Iran.*

Received 10 November 2017; received in revised form 9 September 2018; accepted 5 November 2018

KEYWORDS
Hybrid flow shop;
Multiprocessor jobs;
Mathematical modeling;
Elephant herding optimization;
Manufacturing Resource Planning (MRPII).

Abstract. Shop Floor Control (SFC) is one of the main concepts in Manufacturing Resource Planning (MRPII), and production scheduling is a key element in SFC. This paper studies the hybrid flow shop scheduling problem, where jobs are multiprocessors. The objective is to minimize total completion time. Although there are several papers considering the hybrid flow-shop scheduling problem with multiprocessor tasks, none has proposed a mathematical model for this problem. At first, the two problems (fixed and selective cases) are mathematically formulated by mixed integer linear programming models. By using commercial software, the model is used to solve the small instances of the problems. Moreover, an elephant herding optimization is developed to solve large instances of the problems. To numerically evaluate the proposed algorithm, it is compared with three available algorithms in the literature.

1. Introduction

Production is one of the most basic and important functions of human activities in modern industrial societies, and production planning is the selection of the future course of production actions. Manufacturing Resource Planning (MRPII) is a method for the effective planning of all resources of a manufacturing company. It is made up of a variety of interlinked functions. One of the main concepts in MRPII is Shop Floor Control (SFC). The execution of the manufacturing portion of the planning order release schedule whereby factory orders are released, monitored, and reported is called SFC. The SFC consists of Factory Coordination (FC) and Production Activity Control (PAC). In PAC, more detailed planning is performed. At this point, the top-level plans of the company have been broken down into specific tasks. In firms using MRPII systems, the execution of the detailed material and capacity plans involves the scheduling of machines and other work centers. Scheduling is a PAC function that is aimed at ensuring the right tasks are conducted at the right time to produce the output. In scheduling, we are looking to specify the start and finish times of the activities [1]. Scheduling is a decision-making process that is used on a regular basis in many manufacturing and services industries. It deals with the allocation of resources to tasks over given time periods, and its goal is to optimize one or

*. *Corresponding author.*
E-mail addresses: hr_ghlamy@yahoo.com (H.R. Gholami); emehdi@qiau.ac.ir (E. Mehdizadeh); bahman.naderi@aut.ac.ir (B. Naderi)

doi: 10.24200/sci.2018.5552.1343

more objectives; in addition, it plays an important role in the MRPII system [2]. Flow Shop (FS) is one of the important problems in scheduling. For nearly several years, FS scheduling problems have been studied as an important subject in manufacturing research pieces. Some novel useful definitions and properties about the FS scheduling problem were established by Aminnayeri and Naderi [3]. In the classical flow-shop problem, there are m stages in series with only one machine at each one [4]. In addition, there are n jobs, each of which must be processed by a single machine at all the m stages, starting from Stage 1 and Stage 2 until stage m [5].

There are some gaps between academic and practical aspects of this problem. One of these gaps in the FS is the assumption of one machine existing at each stage, while companies likely employ more than one machine at stages with more workload. In this case, they can reduce the impact of bottleneck stages or even more balance their production capacity. This generalized version of the problem is called Hybrid Flow Shops (HFS). The HFS has been well studied in the literature [6–8].

The HFS classically assumes that each job requires only one machine for each operation. Hence, it is assigned to only one machine among machines at each stage, while another extension inspired by the realistic applications is assuming multiprocessor tasks. That is, the operation of a job may simultaneously need more than one machine for its process. A subset of machines among the machines available at that stage is required. In scheduling problems with multiprocessor tasks, the subset of machines required for processing a job at a stage can be either fixed or selective. If the subset is fixed, the machines have already been already determined due to technical limitations or machine eligibilities. The process of a job at a stage begins as soon as all the necessary machines are available. If the subset is selective, the machines are not predetermined and the selection is a decision to make. As soon as the required number of machines for the processing of a job at a stage becomes idle, the process of that job can be started.

The applications of scheduling problems with multiprocessor tasks can be seen in some systems. For example, in semiconductor circuit design workforce planning [9], a design project is to be processed by m persons (a team of people). Other applications can be found in (i) the berth allocation problems, where a large vessel may occupy several berths for loading and/or unloading, (ii) diagnosable microprocessor systems, where a job must be performed on parallel processors in order to detect faults, (iii) manufacturing, where a job may need machines, tools, and people simultaneously, and (iv) scheduling a sequence of meetings, where each meeting requires a certain group of people. In all of the above examples, one job may need to be processed by several machines simultaneously [10–12].

By looking into the literature of multiprocessor tasks, the following papers can be found: Chen and Lee [10] considered the problem of parallel machines with multiprocessor tasks, where the subset of machines is selective. They proposed a pseudo polynomial algorithm for two- and three-machine problems. Jansen and Porkolab [13] studied the same problem; however, the processing time of a job was a function of the subset selected. They proposed a fully polynomial time approximation scheme for a preemptive version of the problem and a polynomial time approximation scheme for its non-preemptive one.

Many studies have applied heuristic and metaheuristic algorithms to solve scheduling problems [14]. Since the problem of Hybrid Flow Shops (HFSs) with Multiprocessor Tasks (HFS-MT) is strongly NP-hard, the most effective algorithms of this problem include metaheuristics. Serifoğlu and Ulusoy [15] proposed a genetic algorithm for the problem. Oğuz et al. [16] and Oğuz and Ercan [17] investigated HFS-MT, where the subset of machines required for processing a job at a stage is selective, and developed a tabu search and a genetic algorithm. Later, Ying and Lin [18] proposed an ant colony optimization algorithm. They compared the proposed algorithm with two available algorithms of genetic algorithm [17] and tabu search [16]. Next, Tseng and Liao [19] developed a particle swarm optimization and compared it with ant colony optimization, used in the referenced study [18] for performance. Kahraman et al. [20] proposed a parallel greedy algorithm, and showed that this algorithm outperformed genetic algorithm [17] and tabu search [16]. Engin et al. [21] proposed another genetic algorithm shown to be better than the parallel greedy algorithm used in [20]. Wang et al. [22] proposed a Simulated Annealing (SA) that shows high performance in the numerical experiments. Recently, Xu et al. [23] developed a Shuffled Frog-Leaping Algorithm (SFLA) and compared it with genetic algorithm used in [15], ant colony optimization in [18], particle swarm optimization in [19], and SA in [22]. Lahimer et al. [24] developed a Limited Discrepancy Search (LDS) heuristic to solve the problem and lower bound for the problem. An enhanced invasive weed optimization meta-heuristic algorithm was presented to solve the flexible FS scheduling problem with probable rework times, transportation times with a conveyor between two subsequent stages, different ready times, and anticipatory sequence-dependent setup times [25]. Very recently, Hidri et al. [26] developed another lower bound for HFS-MT.

Although there are several papers on the HFS scheduling problem with multiprocessor tasks, none of them has proposed a mathematical model for this particular problem. The HFS was already formulated by Ziaeifar et al. [27] by a non-linear model and linearly by Behnamian and Fatemi Ghomi [28]. However, they both failed to consider multiprocessor tasks. This paper studies the HFS scheduling problem with multiprocessor tasks. Both cases of fixed and selective subsets of machines are considered. At first, the two problems are mathematically formulated by mixed integer linear programming models. By using commercial software, the model is used to solve the small instances of the problems. Moreover, an effective Elephant Herding Optimization (EHO) is developed to solve large instances of the problems. EHO is a population-based algorithm for continuous search space [29]. To numerically evaluate the proposed algorithm, it is compared with three available algorithms in the literature, SA in [22], SFLA in [23], and LDS in [24].

The rest of the paper is arranged as follows: Section 2 presents the mathematical models of the problems under consideration. Section 3 develops an EHO. Section 4 numerically evaluates the proposed model and algorithms. Section 5 concludes the paper and introduces some future research directions.

2. Problem definition and formulation

The problem of scheduling hybrid flow shops with multiprocessor tasks can be described in the following. There is a set of n jobs in the form of multiprocessor tasks. Each job needs m operations for completion. For each operation, there is one work stage and, at each stage i, there is a set of m_i machines. The process of a job at a stage requires $d_{j,i}$ machines simultaneously. The $d_{j,i}$ machines can be either fixed (say Case 1) or selective (say Case 2). The objective is to schedule jobs at each stage so as to minimize makespan, the maximum completion time of jobs.

For each of the two cases, one mixed integer linear programming model is developed. Before presenting the models, the following parameters and indexes are established.

Parameters and indices

n	The number of jobs
m	The number of stages
j,k	Indices for jobs where $\{1,2,...,n\}$
i	Indices for stages where $\{1,2,...,m\}$
m_i	The number of machines at stage i
l	Indices for machines at stage i where $\{1,2,...,m_i\}$
$p_{j,i}$	Processing time of job j at stage i
$D_{j,i}$	The subset of machines required to process job j at stage $i(\lvert D_{j,i}\rvert = d_{j,i})$
M	A large positive number

Case 1. Multiprocessor tasks with a fixed subset of machines

Variables:

$Y_{j,i,k}$	Binary variable taking value 1 if job k proceeds job j at stage i, and 0 otherwise where $k>j$ and $D_{j,i}\cap D_{k,i}\neq\phi$)
$F_{j,i}$	Continuous variable for the completion time of job j at stage i.

The problem can be formulated as follows:

Min $\sum_{j=1}^{n} F_{j,m}$
Subject to:

$$F_{j,1}\geq p_{j,1} \qquad \forall_j, \tag{1}$$

$$F_{j,i}\geq F_{j,i-1}+p_{j,i} \qquad \forall_{j,i>1}, \tag{2}$$

$$F_{j,i}-F_{k,i}+M.(1-Y_{j,i,k})\geq p_{j,i}$$

$$\forall_{i,(j,k)|D_{j,i}\cap D_{k,i}\neq\phi}, \tag{3}$$

$$F_{k,i}-F_{j,i}+M.(Y_{j,i,k})\geq p_{k,i}$$

$$\forall_{i,(j,k)|D_{j,i}\cap D_{k,i}\neq\phi}, \tag{4}$$

$$F_{j,i}\geq 0 \qquad \forall_{j,i}, \tag{5}$$

$$Y_{j,i,k}\in\{0,1\} \qquad \forall_{i,(j,k)|D_{j,i}\cap D_{k,i}\neq\phi}. \tag{6}$$

Constraint set (1) assures the least possible completion time for each job at the first stage. Constraint set (2) specifies that each job can be processed at most at one stage at a time. Constraint sets (3) and (4) are a pair of constraints to ensure that each machine processes at most one job at a time. Finally, Constraint sets (5) and (6) define the decision variables.

Case 2. Multiprocessor tasks with a selective subset of machines

The following decision variables are defined for the model:

Continuous variables:

$Y_{j,i,k}$	Binary variable taking value 1 if job k proceeds job j at stage i, and 0 otherwise where $k>j$),

$Z_{j,i,l}$ Binary variable taking value 1 if job j is processed by machine l at stage i, and 0 otherwise.

The problem can be formulated as follows:

Min $\sum_{j=1}^{n} F_{j,m}$

Subject to:

Constraint sets (1), (2), and (5):

$$\sum_{l=1}^{m_i} Z_{j,i,l} = d_{j,i} \qquad \forall_{j,i}, \tag{7}$$

$$F_{j,i} \geq F_{k,i} + p_{j,i} - M.(3 - Y_{j,i,k} - Z_{j,i,l} - Z_{k,i,l}) \quad \forall_{j<k,i,l}, \tag{8}$$

$$F_{k,i} \geq F_{j,i} + p_{k,i} - M.(2 - Y_{j,i,k} - Z_{j,i,l} - Z_{k,i,l}) \quad \forall_{j<k,i,l}, \tag{9}$$

$$Y_{j,i,k} \in \{0,1\} \quad \forall_{j<k,i}, \tag{10}$$

$$Z_{j,i,l} \in \{0,1\} \quad \forall_{j,i,l}. \tag{11}$$

Constraint set (7) is to select machines to process each operation. Constraint sets (8) and (9) assure that the machine carries out at most one operation at a time. Constraint sets (10) and (11) define the decision variables.

3. Hybrid elephant herding optimization (EHO)

Elephant Herding Optimization (EHO) is a novel swarm-based meta-heuristic algorithm that was inspired by the herding behavior of an elephant group. This algorithm was proposed by Wang et al. [29] for solving continuous non-linear problems. In nature, elephants are social animals and live in several clans under the leadership of a matriarch, often the oldest cow, and the male elephants will leave their family group when they grow up. These two behaviors in EHO modeled into two phases: clan updating phase and separating phase.

As mentioned earlier, EHO is developed to solve the continuous function. However, HFS-MT is a combinatorial problem whose solution space is discrete. Therefore, this paper develops a discrete variant of EHO algorithm. This is done by defining a proper encoding scheme, crossing operators, and a moving operator in updating and separating phases. To further enhance EHO, a local search, based on SA, is applied. The stopping criterion is the computational time of n (the number of jobs). In this case, a longer computational time is given to the algorithm to solve larger sizes. Figure 1 shows the general outline of the proposed hybrid EHO.

```
Procedure: Hybrid EHO Algorithm

Initialization
While Computational_time< n do
        Simulated annealing
        clan updatingphase
        separatingphase
Endwhile
```

Figure 1. The general outline of hybrid Elephant Herding Optimization (EHO).

3.1. Encoding scheme and initialization

The first step in meta-heuristics is to determine the encoding scheme to make a solution recognizable for algorithms. The commonly used encoding scheme for HFS problems is job permutation representation [23]. In this representation, the permutation of job numbers shows the processing order of all jobs at the first stage. Assume a problem with 10 jobs. One possible permutation is {5, 2, 10, 4, 1, 9, 7, 3, 8, 6}.

We need a decoding method to obtain a complete schedule represented by an encoded solution. The following approach is used. The job sequence for the subsequent stages is based on "the earliest completion time of jobs at the previous stage". That is, the jobs are sorted according to the ascending order of completion times at the previous stage. The machine assignment depends on the problem case. In the case of a fixed subset of machines, the job at each stage starts as soon as all its corresponding machines are available. In the case of the selective subset of machines, the job starts as soon as a necessary number of machines are available. Note that EHO starts with initial population that is generated randomly.

3.2. Simulated Annealing (SA)

Enhancing meta-heuristics with the local search is an effective idea to have better performance [30]. This study uses SA as a local search. SA is a local-search-based meta-heuristic algorithm with the capability to leave the local optimum with conditional acceptance of worse solutions. SA is inspired by the mechanism in the annealing of solids. It has been successfully applied to many combinatorial optimization problems.

This method operates over one incumbent solution, x. Then, a new solution s is generated using a moving operator and its objective function is evaluated. If the new solution functions better, then it is accepted as the incumbent one; otherwise, it is accepted with probability given by $exp(\Delta/T)$, where Δ is the difference between objective values of the two solutions and T is a control parameter referred to as temperature. SA iterates until no improvement is found in 50 consecutive

Procedure: Simulated_annealing

While *stopping criterion is not met***do**
  move operator
  acceptance criterion
Endwhile

Figure 2. The general outline of Simulated Annealing (SA).

The current sequence	2	3	1	4	6	5
The new sequence	2	3	6	4	1	5

Figure 3. The numerical example for the swapping operator.

The current elephant	2	4	6	1	3	5
The new elephant	4	6	1	2	3	5

Figure 4. The numerical example for the insertion operator.

The current elephant	3	6	1	2	4	5
The new elephant	3	4	2	1	6	5

Figure 5. The numerical example for the inversion operator.

moves over the best solution found. Figure 2 shows the procedure of SA.

As for the moving operator, one of the four following mechanisms is randomly used:

1. *Swap.* Swapping operator is to swap the positions of two randomly selected job numbers. Figure 3 shows a numerical example of a problem with 6 jobs. Suppose that the selected positions are 3 and 5. Then, the part numbers in these two positions are swapped and a new elephant is generated;
2. *Insertion.* Insertion operation is to reallocate a job number into the sequence. That is, a job number of a randomly selected position is reinserted into another randomly selected position. Figure 4 shows a numerical example of the above problem. Suppose that the randomly selected position is position 1, and the randomly selected position for reinsertion is position 4. Then, job number 1 is moved to position 4;
3. *Inversion.* The inversion operator is to inverse the job numbers between two randomly selected positions. Figure 5 shows an example of the above problem. Suppose that the two randomly selected positions are 2 and 5. Then, the part numbers between these two positions are inversed;
4. *Or-opt.* The Or-opt operator is as follows: One position is randomly selected. Then, the job number in that position and the next position are moved to another pair of two consecutive randomly selected positions. Figure 6 shows the numerical example for this operator over the above problem. Suppose that the two selected neighboring job numbers are 3 and 2 at positions 2 and 3. They are both moved to positions 5 and 6.

The current elephant	5	3	2	4	6	1
The new elephant	5	4	6	1	3	2

Figure 6. The numerical example for the Or-opt operator.

The matriarch	4	3	6	5	1	2
The current elephant	2	6	4	5	1	3
The new elephant	4	3	6	2	5	1

Figure 7. The numerical example for the one-point crossover.

3.3. Clan updating phase

In this phase, the matriarch of clan C_i influences each elephant in clan C_i; in other words, the matriarch attempts to improve the elephants in its clan. Matriarch C_i is the fittest elephant in clan C_i. The new elephants in each clan are generated by a one-point crossover operator of each elephant and its matriarch. Note that the matriarch at each clan cannot be updated by this operator. In this case, it can be updated by the two-point crossover operator over the best elephant of all the clans. nC_i is the number of elephants in clan C_i. The one- and two-point crossovers are as follows:

1. One-point crossover: In this operator, the part numbers before a randomly selected position from the matriarch are copied into the new elephant. The remaining part numbers from the other elephant are copied into the new elephant according to their relative order in the other elephant. Figure 7 shows a numerical example of the above problem for this operator. Suppose that the selected cut point is position 4. The part numbers from the beginning to this position are copied. Then, the copied part numbers are removed from the other elephant from left to right. The remaining part numbers (3-2) are copied into the empty positions in the new elephant.
2. Two-point crossover: In this operator, two randomly selected cut positions from the best elephant are copied into the new elephant. Then, the remaining part numbers from the matriarch are copied into the new elephant according to their relative order in the matriarch. Figure 8 shows a numerical example of the above problem for this

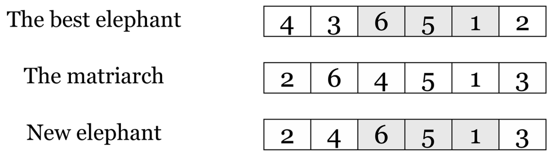

Figure 8. The numerical example for the two-point crossover.

operator. Suppose that the selected cut points are positions 3 and 5. The part numbers from position 3 to position 5 are copied. Then, the copied part numbers are removed from the matriarch from left to right. The remaining part numbers (1-3-1) are copied into the empty positions in the new elephant.

3.4. Separating phase

To further diversify EHO, it requires an operator like mutation in genetic algorithm. In this phase, the worst elephant in each clan is replaced with a new elephant using the insertion operator.

4. Numerical evaluation

This section evaluates the proposed models and algorithms in both cases. Let us remind that, in Case 1, the subsets of machines required by each job at each stage are fixed; yet, in Case 2, the subset is selective from all available machines at each stage. To this end, in each case, three experiments are conducted, one for parameter tuning, one for model evaluation, and one for algorithm evaluation.

To compare the algorithms, the Relative Percentage Deviation (RPD) is used. That is, the objective function of the solution obtained by the algorithm is normalized as follows:

$$\frac{TC - Min}{Min} \times 100,$$

where TC and Min are the total completion time of the solution of the algorithm for a given instance and the minimum makespan obtained for that instance, respectively. Note that this study also adds two effective algorithms to experiments to better position EHO in the literature. These three algorithms include SFLA in [23], SA in [22], and LDS in [24]. The stopping criterion is set to 3 nm seconds for all of the algorithms.

4.1. Parameter tuning

Different levels of the control factors are shown in Table 1. n_{clan} is the number of all clans, nC_i is the number of elephants in clan C_i, P_{MIE} is the probability to execute the multi-type individual enhancement scheme on each elephant in clan C_i, betta is the temperature reduction factor, Tf is the final temperature, *prob* is the probability of executing the swapping, insertion, inversion, and Or-opt movement scheme, respectively.

EHO algorithm has 6 factors and, for each one, 3 different levels are considered. The required number of tests for a full factorial experiment is 3^6; however, the Taguchi method uses orthogonal array L_{27} that includes only 27 tests. To implement the experiments, a set of 10 instances with different combinations of the number of orders, parts, and machines is considered. The results are evaluated using Minitab 16. According to the results, the parameters for the EHO algorithm are set as follows:

$$nclan = 10, \quad nci = 10 \quad pMIE = 0.3, \quad betta = 0.75,$$

$$tf = 0.3, \quad prob = [0.1\ 0.1\ 0.4\ 0.4].$$

4.2. Evaluation in Case 1

In this section, at first, the algorithm is tuned. Then, the model and algorithm are evaluated over small-sized instances. Finally, the algorithm is compared with two state-of-the-art algorithms (SFLA and SA) over large-sized instances.

Now, a set of 16 instances is generated with the following sizes (two instances for each size):

$$n = \{6, 8, 10, 12\}, \quad m = \{3, 5\}, \quad \text{and} \quad m_i = \{U[2, 5]\}.$$

The processing times are randomly distributed over (1, 99). The model is implemented in a maximum of 1000 seconds of the computational time.

Table 2 shows the results obtained by the proposed model and algorithm. As is clear, the model of Case 1 cannot optimally solve instances containing more than 12 jobs and 3 stages. All the instances with 8 jobs or less can be solved to achieve optimality in less than one second. Instances with 10 jobs can also be optimally solved in about 100 seconds. The proposed algorithm solves 10 instances out of 14 to optimality.

Now, the proposed algorithm is evaluated against SFLA and SA over large sizes. A set of 60 instances, 5 instances for each of the following 12 sizes, is given:

$$n = \{20, 50, 100\}, \quad m = \{3, 6\}, \quad \text{and} \quad m_i = \{2, U[1, 3]\}.$$

Table 1. Factors and their levels for Elephant Herding Optimization (EHO).

Level	n_{clan}	nC_i	P_{MIE}	*betta*	Tf	*prob*
1	5	5	0.1	0.55	0.1	[0.1 0.1 0.4 0.4]
2	10	10	0.3	0.75	0.2	[0.2 0.2 0.3 0.3]
3	15	20	0.5	0.9	0.3	[0.4 0.4 0.1 0.1]

Table 2. Model and algorithm results (computational time in seconds) in Case 2.

	Model			Algorithm (EHO)	
$n \times m$	TC	Time	Optimality gap	$C_{\max}$	Time
6 × 3	154	0.05	0	154*	54
6 × 3	290	0.06	0	290*	54
6 × 5	228	0.08	0	228*	90
6 × 5	360	0.09	0	364	90
8 × 3	270	0.36	0	270*	72
8 × 3	748	0.31	0	774	72
8 × 5	780	0.48	0	780*	120
8 × 5	1740	0.51	0	1754	120
10 × 3	858	143.83	0	858*	90
10 × 3	1422	26.86	0	1422*	90
10 × 5	1160	119.56	0	1160*	150
10 × 5	1814	74.08	0	1820	150
12 × 3	1540	178.92	0	1540*	108
12 × 3	1430	132.05	0	1430*	108
12 × 5	1306	1000	13.14%	1458	180
12 × 5	1458	1000	8.5%	1500	180

*: The optimal makespan.

Table 3. The average RPD of the algorithms over large instances for Case 1.

	Algorithms			
$n \times m$	SA	SFLA	EHO	LDS
20 × 3	1.43	0.51	0.00	0.12
20 × 6	1.53	0.70	0.00	0.24
50 × 3	1.05	0.65	0.32	0.45
50 × 6	1.33	1.02	0.16	0.53
100 × 3	1.82	1.32	0.09	0.95
100 × 6	1.57	1.16	0.00	0.88
Average	1.46	0.89	0.09	0.53

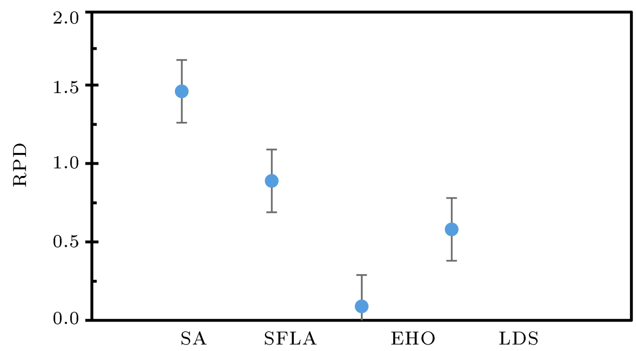

Figure 9. The average RPD and 95% confidence level of the tested algorithms in Case 1.

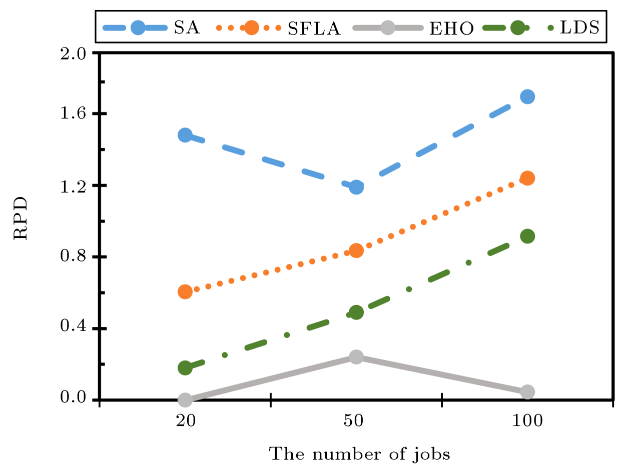

Figure 10. The average RPD of the tested algorithms versus the number of jobs in Case 1.

Table 3 shows the results over large instances. As can be seen, EHO outperforms the SFLA, SA, and LDS with the average RPF of 0.09%. The second best is LDS with an average RPD of 0.53%.

In this part, we also show the performance of the algorithms versus the problem sizes. In Figure 9, the average RPDs of these four tested algorithms are shown versus the number of jobs. As shown in this figure, in all three problem sizes, the proposed EHO keeps its better performance, especially for instances of larger size. Figure 10 shows the average RPDs of the algorithms versus the number of stages.

4.3. Evaluation for Case 2

In this section, like Case 1, at first, the algorithm is tuned. Then, the model and algorithm are evaluated over small-sized instances. Finally, the algorithm is

Table 4. Model and algorithm results (computational time in seconds) in Case 2.

	Model			Algorithm (EHO)	
$n \times m$	TC	Time	Optimality gap	$C_{\max}$	Time
6 × 3	130	1.25	0	130*	54
6 × 3	148	0.23	0	148*	54
6 × 5	200	231.08	0	204	90
6 × 5	250	413.87	0	250*	90
8 × 3	336	5.32	0	340	72
8 × 3	448	99.73	0	448*	72
8 × 5	636	1005.47	5.33%	636*	120
8 × 5	1156	1005.80	11.25%	1158	120
10 × 3	1186	1122.27	19.83%	1186	90
10 × 3	1214	1021.60	15.82%	1214	90
10 × 5	1568	1026.83	28.34%	1552	150
10 × 5	1394	1017.03	6.03%	1402	150

compared with two algorithms (SFLA and SA) in literature over large-sized instances.

Now, the MILP model and algorithm of Case 2 are evaluated to solve the problem. A set of 12 instances is generated as follows (2 instances for each size).

$$n = \{6, 8, 10\}, \quad m = \{3, 5\}, \quad \text{and} \quad m_i = \{U[2, 5]\}).$$

The processing times are randomly distributed over (1, 99). Table 4 shows the results obtained by the model (within 1000s of the computational time) and EHO. As can be seen, the model of Case 2 cannot optimally solve those instances with more than 8 jobs and 3 stages. It gives the average optimal gap of 14.43%. Concerning EHO for Case 2, it optimally solves 4 instances out of 6 ones, which have already been solved by the model.

The proposed algorithm is now compared with the three other algorithms from the literature (i.e., SFLA, SA, and LDS) over large sizes. A set of 60 instances, 5 instances for each of the following 12 sizes, is considered as follows:

$$n = \{20, 50, 100\}, \quad m = \{3, 6\}, \text{ and } m_i = \{2, U[1, 3]\}.$$

Table 5 shows the results over large instances. As can be seen, EHO outperforms SFLA and SA with average RPF of 0.16%. The second best is LDS with the average RPD of 0.67%.

In this part, we also show the performance of the algorithms versus the problem sizes. In Figure 11, the average RPDs and 95% confidence level of these four tested algorithms are shown. Moreover, Figure 12 shows the average RPDs of the algorithms versus the number of jobs. As shown earlier, in all three problem sizes, the proposed EHO keeps its better performance, especially for larger size instances.

Table 5. The average RPD of the algorithms over large instances for Case 2.

	Algorithms			
$n \times m$	SA	SFLA	EHO	LDS
20 × 3	1.82	0.55	0.21	0.19
20 × 6	1.66	0.79	0.04	0.39
50 × 3	1.95	0.66	0.07	0.46
50 × 6	2.75	1.03	0.62	0.88
100 × 3	2.67	1.16	0.02	1.16
100 × 6	2.14	1.56	0.01	0.92
Average	2.17	0.96	0.16	0.67

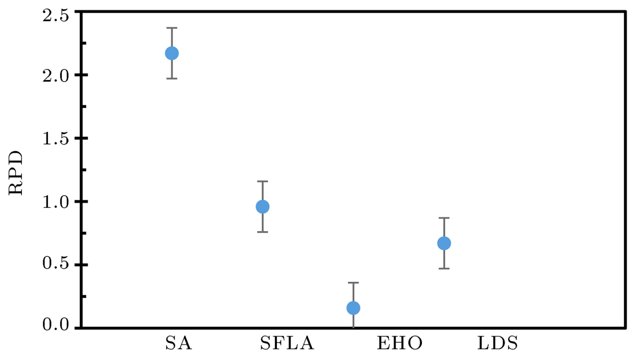

Figure 11. The average RPD and 95% confidence level of the tested algorithms in Case 2.

5. Conclusion and future research

The hybrid flow shop scheduling problem with multiprocessor tasks in the MRPII system was studied. Two cases of fixed and selective subsets of machines were considered. At first, the two problems were

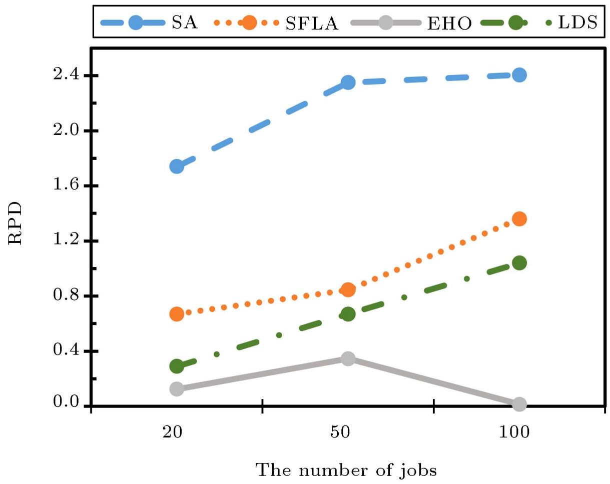

Figure 12. The average RPD of the tested algorithms versus the number of jobs in Case 2.

mathematically formulated by mixed integer linear programming models, one for each case. Then, by applying commercial software, the models were used to solve the small instances of the problems to optimality. The model of Case 1 solved instances with about 12 jobs, while the model of Case 2 solved instances with at most 10 jobs in 1000 seconds.

Moreover, a novel algorithm, called elephant herding optimization, was proposed to solve large instances of the problems. The proposed algorithm was compared with two available algorithms (simulated annealing and shuffled frog-leaping algorithm) presented in the literature for performance. In both cases, the proposed algorithm outperformed the other algorithms.

One interesting future research direction is to include backorder possibility as in [31], where the author predicts backorder aging and unfilled backorders. Another future research is to incorporate setup times into the model.

References

1. Sheikh, K., *Manufacturing Resource Planning (MRP II): With Introduction to ERP, SCM and CRM*, McGraw-Hill Professional Publishing (2003).
2. Pinedo, M.L., *Scheduling: Theory, Algorithms, and Systems*, Springer (2016).
3. Aminnayeri, M. and Naderi, B. "Novel properties along with solution methods for permutation flowshop scheduling", *Scientia Iranica. Transaction E, Industrial Engineering*, **23**(5), pp. 2261–2276 (2016).
4. Vasiljovic, D. and Danilovic, M. "Handling ties in heuristics for the permutation flow shop scheduling problem", *Journal of Manufacturing Systems*, **35**, pp. 1–9 (2015).
5. Ye, H., Li, W., and Miao, E. "An effective heuristic for no-wait flow shop production to minimize makespan", *Journal of Manufacturing Systems*, **40**, pp. 2–7 (2016).
6. Linn, R. and Zhang, W. "Hybrid flow shop scheduling: a survey", *Computers & Industrial Engineering*, **37**(1-2), pp. 57–61 (1999).
7. Naderi, B., Gohari, Sh., and Yazdani, M. "Hybrid flexible flowshop problems: Models and solution methods", *Applied Mathematical Modelling*, **38**(24), pp. 5767–5780 (2014).
8. Naderi, B. and Yazdani, M. "A model and imperialist competitive algorithm for hybrid flow shops with sublots and setup times", *Journal of Manufacturing Systems*, **33**(4), pp. 647–653 (2014).
9. Dal Cin, M. and Dilger, E. "On the diagnosability of self-testing multi-microprocessor systems", *Microprocessing and Microprogramming*, **7**(3), pp. 177-184 (1981).
10. Chen, J., and Lee, C.-Y. "General multiprocessor task scheduling", *Naval Research Logistics (NRL)*, **46**(1), pp. 57–74 (1999).
11. Bertel, S. and Billaut, J.-C. "A genetic algorithm for an industrial multiprocessor flow shop scheduling problem with recirculation", *European Journal of Operational Research*, **159**(3), pp. 651–662 (2004).
12. Ercan, M.F. and Fung, Y.F. "Real-time image interpretation on a multi-layer architecture", *TENCON 99, Proceedings of the IEEE Region 10 Conference*, **2**, IEEE (1999).
13. Janson, K. and Porkolab, L. "General multiprocessor task scheduling: Approximate solutions in linear time", *SIAM Journal on Computing*, **35**(3), pp. 519–530 (2005).
14. Mehdizadeh, E., Tavakkoli-Moghaddam, R., and Yazdani, M. "A vibration damping optimization algorithm for a parallel machines scheduling problem with sequence-independent family setup times", *Applied Mathematical Modelling*, **39**(22), pp. 6845–6859 (2015).
15. Serifoglu, F.S. and Ulusoy, G. "Multiprocessor task scheduling in multistage hybrid flow-shops: a genetic algorithm approach", *Journal of the Operational Research Society*, **55**(5), pp. 504–512 (2004).
16. Oğuz, Y., Zinder Y., Ha Do, V., Janiak A., and Lichtenstein, M. "Hybrid flow-shop scheduling problems with multiprocessor task systems", *European Journal of Operational Research*, **152**(1), pp. 115–131 (2004).
17. Oguz, C. and Ercan, M.F. "A genetic algorithm for hybrid flow-shop scheduling with multiprocessor tasks", *Journal of Scheduling*, **8**(4), pp. 323–351 (2005).
18. Ying, K.C. and Lin, S.W. "Multiprocessor task scheduling in multistage hybrid flow-shops: an ant colony system approach", *International Journal of Production Research*, **44**(16), pp. 3161–3177 (2006).
19. Tseng, C.T. and Liao, C.J. "A particle swarm optimization algorithm for hybrid flow-shop scheduling with multiprocessor tasks", *International Journal of Production Research*, **46**(17), pp. 4655–4670 (2008).

20. Kahraman, C., Engin, O, Kaya, I., and Ozturk, R.E. "Multiprocessor task scheduling in multistage hybrid flow-shops: A parallel greedy algorithm approach", *Applied Soft Computing*, **10**(4), pp. 1293–1300 (2010).

21. Engin, O., Coran, G., and Yilmaz, M.K. "An efficient genetic algorithm for hybrid flow shop scheduling with multiprocessor task problems", *Applied Soft Computing*, **11**(3), pp. 3056–3065 (2011).

22. Wang, H.M., Chou, F.D., and Wu, F.C. "A simulated annealing for hybrid flow shop scheduling with multiprocessor tasks to minimize makespan", *The International Journal of Advanced Manufacturing Technology*, **53**(5–8), pp. 761–776 (2011).

23. Xu, Y., Wang, L., Liu, M., and Wang, S.Y. "An effective shuffled frog-leaping algorithm for hybrid flowshop scheduling with multiprocessor tasks", *The International Journal of Advanced Manufacturing Technology*, **68**(5–8), pp. 1529–1537 (2013).

24. Lahimer, A., Lopez, P., and Haouar, M. "Improved bounds for hybrid flow shop scheduling with multiprocessor tasks", *Computers & Industrial Engineering*, **66**(4), pp. 1106–1114 (2013).

25. Jolai, F., Tavakkoli-Moghaddam, R., Rabiee, M., and Gheisariha, E. "An enhanced invasive weed optimization for makespan minimization in a flexible flowshop scheduling problem", *Scientia Iranica, Transactions E, Industrial Engineering*, **21**(3), p. 1007 (2014).

26. Hidri, L., Elkosantini, S., and Mabkhot, M.M. "Exact and heuristic procedures for the two-center hybrid flow shop scheduling problem with transportation times", *IEEE Access*, **6**, pp. 21788–21801 (2018).

27. Ziaeifar, A., Tavakkoli-Moghaddam, R., and Pichka, Kh. "Solving a new mathematical model for a hybrid flow shop scheduling problem with a processor assignment by a genetic algorithm", *The International Journal of Advanced Manufacturing Technology*, **61**(1–4), pp. 339–349 (2012).

28. Behnamian, J., and Fatemi Ghomi, S.M.T. "Multiobjective fuzzy multiprocessor flowshop scheduling", *Applied Soft Computing*, **21**, pp. 139–148 (2014).

29. Wang, G.G., Deb, S., and Coelho, L.dos S. "Elephant herding optimization", *Computational and Business Intelligence (ISCBI), 2015 3rd International Symposium on*, IEEE (2015).

30. Liao, C.J., Tseng, C.T., and Luarn, P. "A discrete version of particle swarm optimization for flowshop scheduling problems", *Computers & Operations Research*, **34**(10), pp. 3099–3111 (2007).

31. Rodger, J.A. "Application of a fuzzy feasibility Bayesian probabilistic estimation of supply chain backorder aging, unfilled backorders, and customer wait time using stochastic simulation with Markov blankets", *Expert Systems with Applications*, **41**(16), pp. 7005–7022 (2014).

Biographies

Habib Reza Gholami is currently a PhD student at the Department of Industrial Engineering, Islamic Azad University, Qazvin Branch, Qazvin, Iran. He obtained MSc degree from Islamic Azad University, South Tehran Branch in 2004 and his BSc degree in Computer science from Shiraz University in Iran (1991). His research interests are in the areas of operation research such as production planning and production scheduling.

Esmaeil Mehdizadeh is currently an Associate Professor at the Department of Industrial Engineering, Islamic Azad University, Qazvin Branch, Qazvin, Iran. He received his PhD degree from Islamic Azad University, Science and Research Branch, Tehran in 2009, MSc degree from Islamic Azad University, South Tehran Branch in 1999, and BSc degree from Islamic Azad University, Qazvin Branch in 1996 all in Industrial Engineering. His research interests are in the areas of operation research such as production planning, production scheduling, fuzzy sets, and meta-heuristic algorithms. He has several papers in journals and conference proceedings. Moreover, he is a Managing Editor of the International Journal of Optimization in Industrial Engineering.

Bahman Naderi completed his BSc degree in Industrial Engineering from Mazandaran University of Science and Technology (Iran), MSc and PhD degrees form Amirkabir University of Technology, Tehran, Iran, and a post-doctoral program at University of Windsor, Canada. Currently, he is an Assistant Professor at Kharazmi University, Tehran, Iran. His research interests are applied operations research (mathematical modeling and solution methods).