An Effective Modified Migrating Birds Optimization for Hybrid Flowshop Scheduling Problem with Lot Streaming

Biao Zhang[a], Quan-ke Pan[a,*], Liang Gao[a], Xin-li Zhang[b], Hong-yan Sang[c], Jun-qing Li[c]

[a]State Key Lab. of Digital Manufacturing Equipment & Technology in Huazhong University of Science & Technology, Wuhan, 430074, P. R. China

[b]College of Mathematic Science, Liaocheng University, Liaocheng, 25200, P. R. China

[c]College of Computer Science, Liaocheng University, Liaocheng, 25200, P. R. China

Abstract In this paper, the problem of hybrid flowshop hybridizing with lot streaming (HLFS) with the objective of minimizing the total flow time is addressed. We propose a mathematical model and an effective modified migrating birds optimization (EMBO) to solve this problem within an acceptable computational time. A so-called shortest waiting time rule (SWT) is introduced to schedule the jobs concurrently arriving at stages more reasonably. A combined neighborhood search strategy is developed that unites two different neighborhood operators during evolution, not only taking full advantage of their specializations but also promoting their joint efforts. Two competitive mechanisms are respectively used to increase the probability of locating better solutions at the front of the flock and enhance the interaction between two lines. The scout phase on the basis of the Glover operator and a well-designed local search is applied to the individuals trapped into local optimums and helps the algorithm explore potential promising domains. The dynamic solution acceptance criteria is developed to strike a compromise between intensification and diversification mechanisms. The performance of our proposed algorithm is evaluated by comparisons with seven other efficient algorithms in the literature. And the extensive numerical illustrations demonstrate that the proposed algorithm performs much more effectively for the addressed problem.

Keywords: scheduling problem; hybrid flowshop; lot streaming; meta-heuristics; migrating bird optimization

1 Introduction

As a branch of flowshop scheduling, hybrid flowshop (HFS) has played a crucial rule in modern manufacturing and production systems, e.g., in electronics [1,2], textile [3], paper [4], and petrochemical and pharmaceutical industries [5]. In this problem, a set of *n* jobs have to be processed through a series of *m* stages, each of which has one or more identical machines in parallel. Certain stages may have only one machine, but at least one stage that has several machines exists. All jobs follow the same production route from the first stage to the final stage, and cannot skip any stage. According to [6,7], this shop configuration has the advantages of increasing the throughput of

* Corresponding author: panquanke@shu.edu.cn

production floors, balancing the speed of stages, and reducing the impact of bottleneck stages.

Given its superiority and application to real world industries, the HFS has become one of the hot topics and many different scheduling approaches have been developed, such as exact solutions, heuristics and meta-heuristics. Two mostly used exact solution procedures are the branch-and-bound (B&B) [8,9] and mixed-integer programming (MIP) [10], which can produce optimal solutions for solving problems that have a small scale or very simple scenario. However, due to the NP-hardness property of the HFS [11], it becomes much more difficult or evenly impossible to deal with large scale problems using these exact methods. Some heuristics have also presented to find optimal or near optimal solutions when solving small problems (see [11-15] for example), but to find satisfactory solutions for large problems is very computationally expensive. In recent years, a great amount of efforts have been dedicated to meta-heuristics and have achieved satisfactory results in solving large scale HFS problems, such as the simulated annealing method [16,17], tabu search algorithm [18], ant colony optimization [19,20], genetic algorithm [7,21], quantum-inspired immune algorithm [22], artificial immune approach [23], migrating birds optimization [24], artificial bee colony algorithm [25,26], and particle swarm optimization [27].

In the above mentioned HFS studies, each job is not allowed to be moved to the downstream stage before its whole operation is completed. Nevertheless, in today's global competitive environment, such flow may negatively affect the scheduling efficiency and cannot cater to many real world production systems, such as the electronics, fasteners and ceramic tiles industries in which jobs are composed of many identical items or sublots. Thus, lot streaming, first proffered in [28], is now widely applied to implement time-based strategies (see [29-35] for example) because of its potential benefits, including reductions in the production lead time, interim storage and space requirements. Lot streaming is a technique of splitting the given jobs consisting of identical items into a number of sublots to allow their operations between successive stages. That is, when a job sublot is completed, it can be immediately transferred to the following adjacent stage. However, most studies of lot streaming focus on the pure flowshop environment, and seldom research on lot streaming in the hybrid flowshop has been done. To the best of our knowledge, there are two published papers that considered lot streaming in the hybrid flowshop with large-sized problems. Defersha et al. [36] introduced a parallel genetic algorithm for hybrid flexible flowshop lot streaming problem, aiming to minimize the makespan of the schedule. Mohsen et al. [37] applied the genetic algorithm and the simulated annealing to solve the multi-job lot streaming in a hybrid flow shop scheduling problem to minimize the weighted completion time. A main feature of their addressed problems is that the sublots of different jobs can be intermingled and their scheduling objective is to determine the sequencing of the sublots.

In this paper, we strive to study a hybrid flowshop hybridizing with lot streaming, namely HLFS, in which the sublots of different jobs cannot be intermingled. This different assumption agrees with many real word production shops where the jobs are composed of the same items with different types. For

example, in the modern iron and steel production systems [38], the process of integrative production consists of three stages: steelmaking, refining and continuous casting. Each stage has multiple parallel identical machines, which can be selected by any charge that flows through the stage. The charge here is referred to as the basic unit in steelmaking process. And in the casting stage, a sequence of charges with identical or similar steel grade, which must be consecutively processed on the same continuous caster in the form of a batch, forms a cast. That means a charge can be processed only when the its adjacent previous charge in the same cast is completed. Thus, this problem is an HFS problem with batch production at certain stage, which can be seen as a special case of our considered problem detailed in the next section. This considered problem is much more complex than regular HFS, so it is apparently NP-hard. To this end, an emerging meta-heuristic, namely migrating birds optimization (MBO), is introduced to solve the addressed problem, which was recently presented by Duman et al. [39] for the quadratic assignment problem. Since then, the MBO has been successfully applied in other fields, such as scheduling problem [24, 41], closed loop layout [40], travelling salesman problem [42], sea freight transportation [43] and machine-part cell formation problem [44]. Following [24], we propose an effective version of the MBO with several efficient modifications, including a combined neighborhood search strategy, two competitive mechanisms, the scout phase and the dynamic solution acceptance criteria. The high performance of the proposed algorithm is demonstrated using extensive numerical comparisons with seven other efficient algorithms in the literature.

The remainder of this paper is organized as follows. In Section 2, the HLFS problem is stated in detail and a mathematical model is presented. In Section 3, the basic MBO algorithm is introduced. Section 4 describes our proposed EMBO algorithm for solving the HLFS problem. The experimental design and numerical comparisons are reported in Section 5. Finally, Section 6 provides the concluding remarks and possible future studies.

2 Problem statement

Our addressed HLFS contains a series of m processing stages, and each stage k has $\sigma_k \geq 1$ identical machines in parallel. Additionally, there exists at least one stage that possesses more than one machine. A collection of n jobs are to be processed on the shop following the same route from stage one to stage m consecutively. Also, each job j can be split into $l_j \geq 1$ sublots with equal size to allow overlapping of successive operations. That is, the sublots of a job have the same processing time at a stage, and once a sublot is completed at the current stage, it can be transferred to the next stage immediately. The scheduling problem is to determine the job sequence at the beginning of each stage and the assignment of jobs to machines at each stage, so as to minimize the total flow time. The assumptions for the addressed HLFS are given as follows.

- Each machine can process at most one sublot at a time, and each sublot can be processed by at most one machine at a time.

- All sublots of the same job must be processed continuously by the same machine at any stage or they are not allowed to be intermingled, i.e., once the first sublot of a job arrives at a machine, the other sublots of different jobs cannot be assigned to this machine until all of the sublots are processed.
- Preemption is not allowed, i.e., any sublot cannot be interrupted until the completion of its operation.
- Idle time is allowed for machines, and sublots can wait between successive stages with infinite buffer capacity.
- The machine setup times and sublot transportation times are negligible or included in the sublot processing times, which are determined in advance.

To present a mathematical model for the problem, we first define the following notations according to the above descriptions.

M	Set of stages where the stages are indexed by k and $m = \|M\|$.
J	Set of jobs where the jobs are indexed by j and $n = \|J\|$.
σ_k	The number of machines at stage k where the machines are indexed by i .
l_j	The number of sublots of job j where the sublots are indexed by e .
$p_{k,j}$	The processing time of each sublot of job j at stage k.
$ST_{k,j,e}$	The starting time of the eth sublot of job j at stage k.
$CT_{k,j,e}$	The completion time of the eth sublot of job j at stage k.
C_j	The completion time of job j through the shop.
$D_{k,j,i}$	A binary data equal to 1 if job j is assigned to machine i at stage k; 0 otherwise.
$Y_{k,j,j'}$	A binary data equal to 1 if job j precedes job j' to be assigned to the same machine at stage k; 0 otherwise.
Q	A very large positive number

Due to that all the relevant values are integers, the problem can be formulated as a mixed integer program as follows.

$$\textbf{Minimize} \quad Z = \sum_{j=1}^{n} C_j = \sum_{j=1}^{n} CT_{m,j,l_j} \tag{1}$$

Subject to:

$$\sum_{i=1}^{\sigma_k} D_{k,j,i} = 1 \quad \forall j \in J, k \in M \tag{2}$$

$$ST_{k,j,e} \geq 0 \quad \forall j \in J, k \in M, e = [1,...,l_j] \tag{3}$$

$$CT_{k,j,e} - ST_{k,j,e} = p_{k,j} \quad \forall j \in J, k \in M, e = [1,\ldots,l_j] \tag{4}$$

$$ST_{k+1,j,e} - CT_{k,j,e} \geq 0 \quad \forall j \in J, k \in M, e = [1,\ldots,l_j] \tag{5}$$

$$ST_{k,j,e+1} - CT_{k,j,e} \geq 0 \quad \forall j \in J, k \in M, e = [1,\ldots,l_j] \tag{6}$$

$$Y_{k,j,j'} + Y_{k,j',j} \leq 1 \quad \forall j, j' \in J, k \in M \tag{7}$$

$$ST_{k,j,1} - CT_{k,j',l_{j'}} + Q \times (3 - Y_{k,j',j} - D_{k,j,i} - D_{k,j',i}) \geq 0 \quad \forall j, j' \in J, k \in M, i \in \{1,2,\ldots,\sigma_k\} \tag{8}$$

$$D_{k,j,i} \in \{0,1\} \quad \forall j \in J, k \in M, i \in \{1,2,\ldots,\sigma_k\} \tag{9}$$

$$Y_{k,j,j'} \in \{0,1\} \quad \forall j, j' \in J, k \in M \tag{10}$$

The objective function in Eq. 1 considers the minimization of the total flow time, which is equal to the sum of the completion times of the last sublots of all the *n* jobs at the last stage *m*. The constraint in Eq. 2 requires that each job has to traverse through all the stages and be processed by exactly one machine at each stage. Eq. 3 states that the starting time of the sublots at any stage must be positive and Eq. 4 corresponds to the computation of the completion times of the sublots at any stage. Eq. 5 forces the processing of any sublot to be started only after it has been finished at the preceding stage, while Eq. 6 requires that any sublot can be processed until the completion of the previous sublot at the same stage. The constraints in Eqs. 7 and 8 guarantee the machine capacity restriction, that is, the starting time of the first sublot of a job must be larger than the completion time of the last sublot of its preceding job assigned to the same machine at any stage. And Eqs. 4-8 together ensure that there is no interruption of any sublot processing. Constraints (9) and (10) define the value ranges for the decision variables.

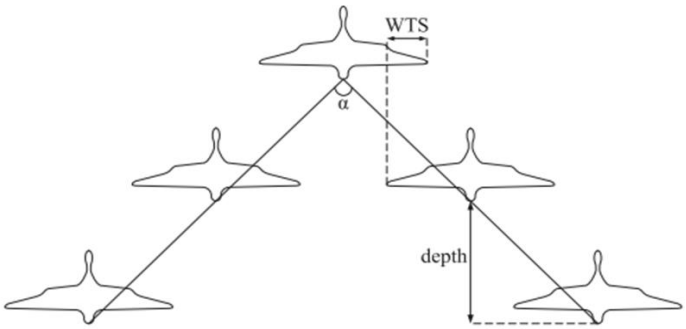


Fig. 1. The V flight formation of migrating birds

3 The Basic MBO algorithm

The V flight formation of natural migration of birds (see Fig. 1 [39]) has proven to be saving energy based on the positive correlation of wing-tip spacing (WTS), constant angle (α) and depth. About this, the readers can see the reference [39] for more information. Inspired by this special formation, the MBO algorithm does not employ the concepts of the constant angle and depth, but has a hypothetical V shaped population formed by a leader solution and other solutions in left and right lines following the leader and introduces a benefit mechanism corresponding to the WTS. The MBO starts with a number of solutions placed on the V population arbitrarily. Then, an evolving loop involving a number of tours or iterations proceeds, and each tour evolves beginning with the leader and progressing along the left and right lines in parallel by exploring their neighborhood. Particularly, the benefit mechanism that the solutions may share their best unused neighbors with the following solutions through a special neighbor shared set is applied in the evolutionary process. The unused neighbors are referred to the neighbors that are not used to update its current solution. Finally, when a loop is finished, the leader is to be changed, and another loop starts. The above procedure is conducted repeatedly until a termination condition is met. Generally, the framework of the MBO algorithm is displayed in Fig. 2 and there are four main operators [24,39]: algorithm initialization, leader evolution, followers evolution and leader change. These will be described in detail below.

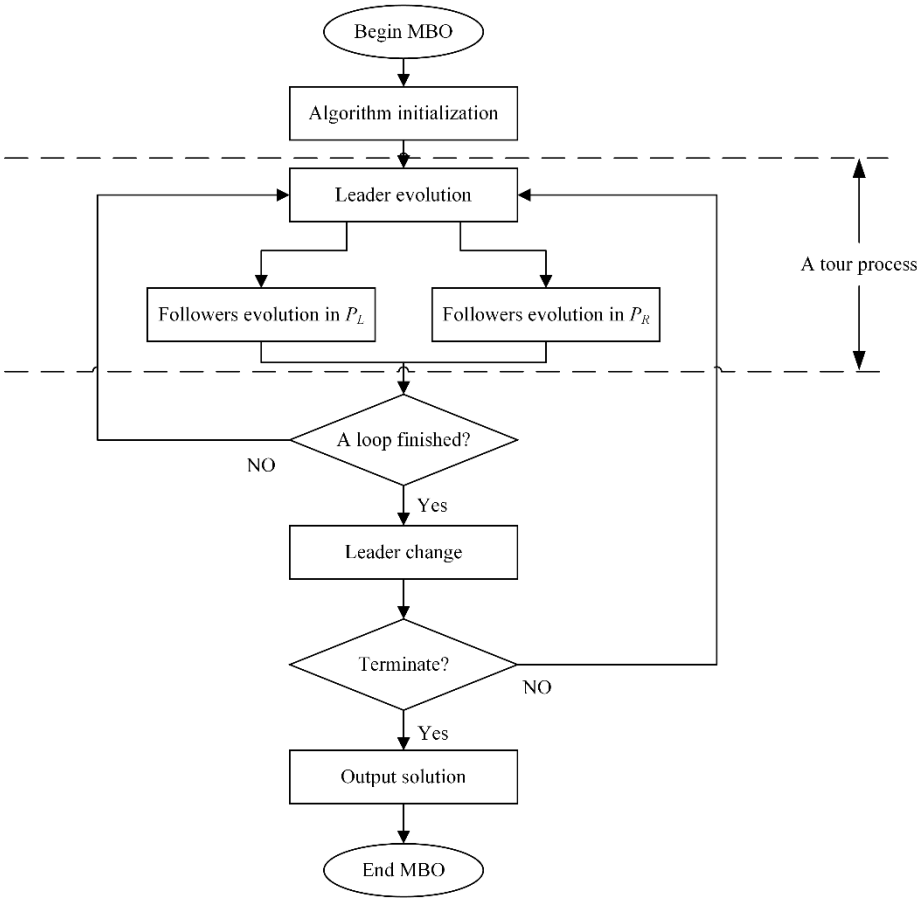


Fig. 2. The framework of the basic MBO algorithm

3.1 Algorithm initialization

Algorithmic parameters and population should be initialized. In the basic MBO algorithm, there are five key parameters to be set, including the number of solutions (*ps*) in the population, the number of neighbors to be explored (*k*), the number of solutions to be shared with the following solutions(*s*), the number of tours (*t*) contained in a loop and the termination limit (*K*). The first four parameters correspond to the number of birds in the flock, the flight speed, the wing-tip spacing (WTS) and the number of wing flaps, respectively [39].

Another aspect is to initialize the population. First, *ps* solutions are generated in the feasible solution space in a random manner. Then, the solutions are arbitrarily placed on a hypothetical V formation containing one leader solution, $(ps-1)/2$ solutions in the left line P_L and $(ps-1)/2$ solutions in the right line P_R.

3.2 Leader evolution

The leader solution attempts to seek improvement by exploring its own neighborhood, and *k* neighbors are generated in a predefined rule. Then, if the best neighbor among them has a better fitness,

the leader is replaced by it; otherwise, the leader remains unchanged. And excluding the best one, the remaining $k-1$ neighbors are collected to create two shared neighbor sets full of s members, namely the set Λ_L for the left line and the set Λ_R for the right line. For the minimization optimization problems, these k-1 neighbors are sorted in a non-descending order in terms of their objective values and then enter into two sets in turn until both are filled with s solutions, i.e., the first one enters into Λ_L, the second into Λ_R, the third into Λ_L, the fourth into Λ_R, and so on.

3.3 Followers evolution

The followers in $P_L(P_R)$ generate their own $k-s$ neighbors and attempt to improve themselves by evaluating not only these k-s neighbors but also the s solutions coming from $\Lambda_L(\Lambda_R)$. Each follower in $P_L(P_R)$ is replaced by the best one among these k solutions if it has better fitness; otherwise, stays unchanged. After that, the remaining $k-1$ solutions are sorted in a non-descending order and $\Lambda_L(\Lambda_R)$ is reset to be null. Finally, the first s solutions from the remaining $k-1$ solutions are added to $\Lambda_L(\Lambda_R)$ that is to be shared by the next follower. The above procedure progresses along P_L and P_R from the first solution to the last one in parallel. Such benefit mechanism is totally unique to the MBO, which can help the algorithm explore the domains around more promising solutions in greater detail.

3.4 Leader change

After an evolving loop is finished, the leader solution will be changed. The leader moves to the tail of P_L or P_R alternately, then the first solution following it in the corresponding line is forwarded to the leader position. That is, if at a time, the first solution from $P_L(P_R)$ becomes the new leader, then at the next time, the leader will come from $P_R(P_L)$. Thus, every solution in the population can have the chance to become the leader.

4 The Proposed EMBO for HLFS

In this section, an effective modified migrating birds optimization (EMBO) for solving the HLFS is proposed, aiming to minimize the total flow time of the schedule. We first determine the solution representation and present a so-called SWT decoding rule. Then, we make certain modifications to enhance the performance of the MBO, including a combined neighborhood search strategy, two competitive mechanisms, the scout phase and the dynamic solution acceptance criteria. With these modifications, the proposed algorithm is expected to capture the balance between the exploration and exploitation abilities and perform well in solving the HLFS. Finally, the procedure of the proposed algorithm is provided.

4.1 Encoding and population initialization

For the HLFS, the widely applied permutation-based representation is used for solution encoding, in which each integer denotes a job number. Consider a simple example of a HLFS problem with four jobs and two stages, each of which contains two identical parallel machines. Table 1 summarizes the

relevant data, where the first column denotes the four given jobs and the number of their sublots, and the other four columns denote the processing times of these jobs at the two stages. If one solution is represented by $\pi = (2, 4, 1, 3)$, it means the scheduling order in the first stage is first job 2, then job 4, then job 1, and finally job 3. Our proposed algorithm starts with *ps* randomly generated solutions organized in a V formation, including one leader and $(ps-1)/2$ members in P_L and P_R. These solutions have the same jobs, but their orders are randomly shuffled.

Table 1
Relevant data of the given example

Job(number of sublots)	Stage 1		Stage 2	
	M1	M2	M1	M2
Job 1(3)	6	6	2	2
Job 2(2)	4	4	3	3
Job 3(1)	2	2	3	3
Job 4(2)	2	2	4	4

4.2 Solution evaluation

To evaluate or decode solutions, we should take into account two important decisions: determining the job sequence at the beginning of each stage and assigning jobs to machines at each stage. Considering the characteristics of the HLFS, a common approach in the literature for solving the HFS is adjusted appropriately. For the first stage $k = 1$, each job is scheduled according to its occurrence order in the solution that is determined by the algorithm and is assigned to the first available machine (FAM) with the earliest release time. For the following stages $k > 1$, the job sequence is determined based on a non-decreasing order of the completion times of their first sublots at the previous stage $k-1$, and then, the jobs also use the FAM rule to select a machine.

Besides, the scheduling of jobs concurrently arriving at the stages, except the first, i.e., the completion times of their first sublots at the previous stage are same, also have to be considered, since there usually exist such jobs in the hybrid flowshop environment and they can have an important impact on the schedule. The solution dependent rule, referred to as SDR, is widely used [6] to arrange these concurrently arriving jobs according to their occurrence order in the solution representation. Supposing a job permutation $\pi = (2, 4, 1, 3)$ to launch on the aforementioned shop environment and adopting the SDR rule, a scheduling Gantt chart is plotted in Fig. 3, where each pair of numbers denotes the job and its sublot order. As is shown in Fig. 3, at stage one, the completion times of their first sublots are 10 of job 1, 4 of job 2, 10 of job 3 and 2 of job 4. Obviously, job 1 and job 3 will arrive at stage two concurrently. Because job 1 appears before job 3 in the solution, the job sequence at stage two is (2, 4, 1, 3). Other than the SDR rule, a random rule [45], referred to as RR, schedules the concurrently arriving jobs in a random manner. Regarding the given example, job 1 can be scheduled at stage two before job 3 (see Fig. 3) or behind job 3 (see Fig. 4) with 50% probability. And the RR rule in [45] shows higher performance than the SDR rule.

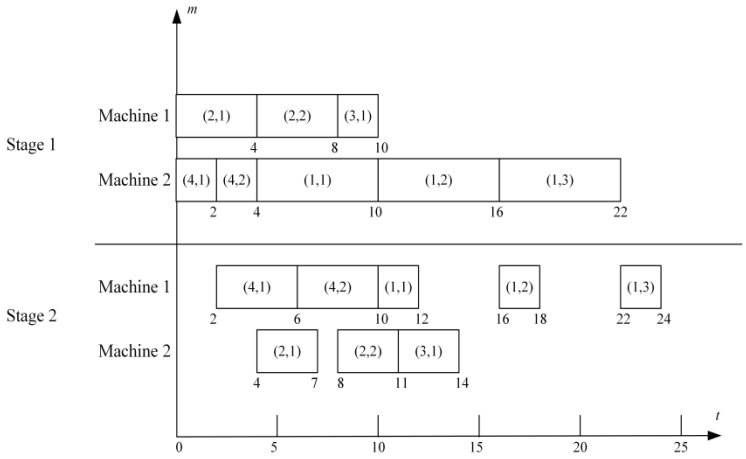


Fig. 3. Gantt chart for scheduling using the SDR rule

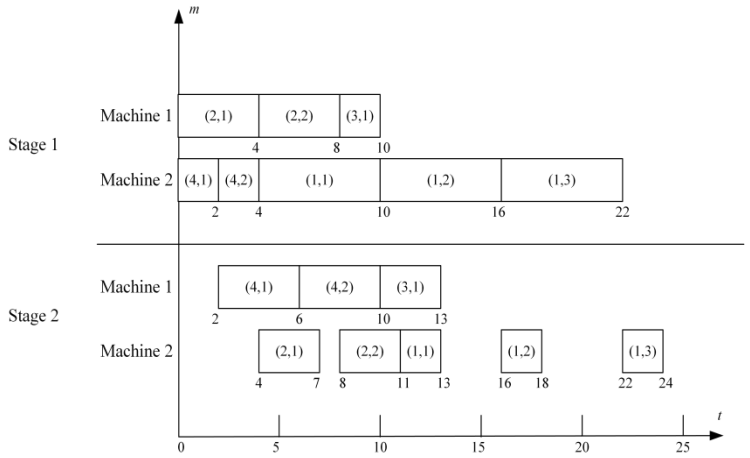


Fig. 4. Gantt chart for scheduling using SWT rule

However, when considering lot streaming in our problem, the SDR and RR rules are very unlikely to be effective because of the assumption that sublots from different jobs are not allowed to be intermingled. The reason behind this is that if the processing time of a previously scheduled job at stage k is much smaller than at stage $k-1$, there may exist a situation that the selected machine at stage k that has completed one sublot will wait for a long time for the arrival of the next sublot. This waiting may result in low efficiency under the total flow time criterion. For that reason, we present the shortest waiting time rule, referred to as SWT, and the procedure of which is described below. First, at each stage k, except the first, we sort the concurrently arriving jobs in non-decreasing order according to a relative value δ calculated in Eq. 11, which determines the arriving order at stage k.

$$\delta_j = \begin{cases} PT_{k-1,j} - PT_{k,j}, & \text{if } PT_{k-1,j} > PT_{k,j} \\ 0, & \text{else} \end{cases} \tag{11}$$

where k is larger or equal to 2. Second, if there are jobs that once again have the same value of δ, then the jobs having smaller $PT_{k,j}$ are assumed to be scheduled in prior. For the given example, the δ values of job 1 and job 3 are, respectively, 4 and 0, therefore job 3 is scheduled ahead of job 1. Fig. 4 shows the Gantt chart using the SWT rule. These three decoding methods will be evaluated later in Section 5.2.

4.3 Combined neighborhood strategy

Since the MBO algorithm belongs to neighborhood search methodologies, it is very necessary to determine the neighborhood search strategy for solutions to improve themselves. For the permutation-based encoding, a number of neighborhood operators can be found in the literature [46]. Each of them has their own specialization areas, and thus, employing the single method may make it difficult to find the optimal or near optimal solutions for solving scheduling problems [37,47,48], which generally cover an extremely large solution space with an excessive quantity of local optimums. Therefore, in view of the special V shaped population that the P_L and P_R are like two subpopulations in the evolutionary process, we propose a combined neighborhood search strategy. In this strategy, two different neighborhood operators are simultaneously employed in our algorithm: one is used for the individuals in P_L, whereas another is used for the individuals in P_R. As for the leader at the beginning of the algorithm, any operator can be chosen because this is executed only once and has very little impact on the experimental results. But for its following evolutions, the leader conducts the same operator with its source line to avoid destroying the neighborhood structure. In this way, on the one hand, we can take full advantage of their specializations. And on the other hand, the aforementioned leader change and the competitive mechanism among two lines presented in the following section can promote their joint efforts. To be specific, when an individual coming from one line becomes the leader, it has evolved with the corresponding neighborhood operator for a number of iterations and through the benefit mechanism, it will have the chance to continue to evolve by using another operator. The solution obtained through the joint efforts of two operators is not easily achieved by any single one, so that this combined neighborhood search strategy has, to an extent, global search capability.

Various neighborhood operators are tested for the MBO algorithm addressing TSP problems in [42], including Insertion (I), Insertion Greedy (IG), Insertion Best (IB), Swap (S), Swap Greedy (SG) and Swap Best (SB). The Insertion removes the randomly selected job and reinserts it into another randomly position to obtain a neighbor. The Insertion Greedy reinserts the randomly selected job into all other positions in sequence and this will generate $n-1$ solutions where n is the number of jobs, and then the best one in terms of the objective function is selected to be the neighbor. While for the Insertion Best, there will be n solutions generated using the Insertion, and then the best one is chosen as the neighbor. In the Swap, two randomly selected jobs exchange their positions. In the Swap Greedy, a randomly selected job in turn exchanges the position with all other jobs, and the best one of the

generated $n-1$ solutions is to be the neighbor. While in the Swap Best, n solutions are produced using the Swap, and the best one is selected as the neighbor. To maintain different neighborhood structures and balance the evolution speed in two lines, the above six operators are formed into five neighborhood combinations, which are referred to as IS, IGSG, IBSB, IGSB and IBSG. The experiments in Section 5.2 show that the combined neighborhood strategy is more effective than applying only the single neighborhood and that the Insertion and Swap constituting the strategy works much better than the other neighborhood combinations.

4.4 Competitive mechanisms

The benefit mechanism plays a very important rule in the MBO algorithm, the philosophy of which is that the solutions have the chance to benefit from the ones in front of them. However, because of the arbitrarily arranged population, if promising solutions appear in the rear, they may have few opportunities to share their neighbors with others. To improve this situation, we develop a competitive mechanism in the internal P_L and P_R, which is executed when each loop is finished. First, for each line, two individuals are picked randomly and compared to each other in terms of their fitness. Next, if the better individual is behind the worse one, their positions are exchanged; otherwise, they remain unchanged. This procedure is repeated the value of t times where t is defined as the number of tours contained in a loop. Consequently, more promising individuals can be given more opportunities to be located in the front of the lines and further exploited in more detail. This mechanism is analogous to the natural phenomenon that, when migrating, stronger migrating birds always attempt to fly in front of the flock.

Excluding the leader change, P_L and P_R mostly seem like two parallel sub-populations with no interactions. Due to the benefit mechanism, they may be quickly grouped by the copies of a few solutions, resulting in poor population diversity. According to [36], the frequent exchanges of individuals among the different populations can alleviate this problem. Hence, we also introduce a competitive mechanism between two lines. This mechanism includes two steps and is as well carried out after each loop is finished. The first step is to randomly select t pairs of individuals at the same positions of two lines and exchange the individuals in each pair. This step can also improve the joint efforts of the two neighborhood operators. The second step is to conduct a two point crossover (TPOX) [34,36] for the individuals in each pair, and the worse one is replaced by the generated solution if it has better fitness. Such attained offspring inherit good features from their parents and may lead the algorithm to potential promising areas.

As is known, the global exploration and the local exploitation dominate the design of the global search method. The above two competitive mechanisms indeed play different roles in the evolutionary process. The first one can be seen as an intensification mechanism, while the second one as well as the scout phase detailed in Section 4.5 can be referred to as diversification mechanisms. They respectively

correspond to the local exploitation and global exploration abilities of the algorithm. And the convergence curves presented in Section 5.4 demonstrate that they are well balanced in our algorithm that is these modifications not only satisfy the search efficiency but also enhance the capacity of the algorithm to converge to the global optimal solution.

4.5 Scout phase

In our algorithm, its two main myopic behaviors (the neighborhood-based searching and the benefit mechanism) focus excessively on the local exploitation ability. Hence, inspired by the artificial bee colony algorithm (ABC) [31,49], we apply the scout phase to highlight the important requirement of balancing between global exploration and local exploitation, which is triggered after r unsuccessful tours for any solution have been conducted. Considering the basic scout phase, if a solution has not been improved through predefined r tours, it is assumed to be a local optimum, and then, it is substituted by a solution generated at random in the feasible solution space.

Instead of the completely randomized method, we introduce the Glover operator [50] to produce a substitute with the abandoned solution as a seed. This substitute can spread out from the abandoned solution, leading the algorithm move to the domain that has not been explored yet. Also, this substitute can inherit certain location information of jobs from the evolutionary process. Supposing an abandoned solution $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, the process of the Glover operator is detailed as follows. First, several job sub-permutations $\pi_{sub}(h:w) = (\pi_w, \pi_{w+h}, \pi_{w+2h}, \ldots, \pi_{w+v\times h})$ are extracted from π, where h is a random integer from the uniform distribution $[1, n/2]$, and w takes turns to get a value from 1 to n and v increases one by one until $w+v\times h$ achieves the largest value that is less than or equal to n. Then, a new solution is re-constructed as $\pi_{new}(h) = (\pi_{sub}(h:h), \pi_{sub}(h:h-1), \ldots, \pi_{sub}(h:1))$. Fig. 5 shows a schematic example to illustrate the Glover operator, where h is set as 3.

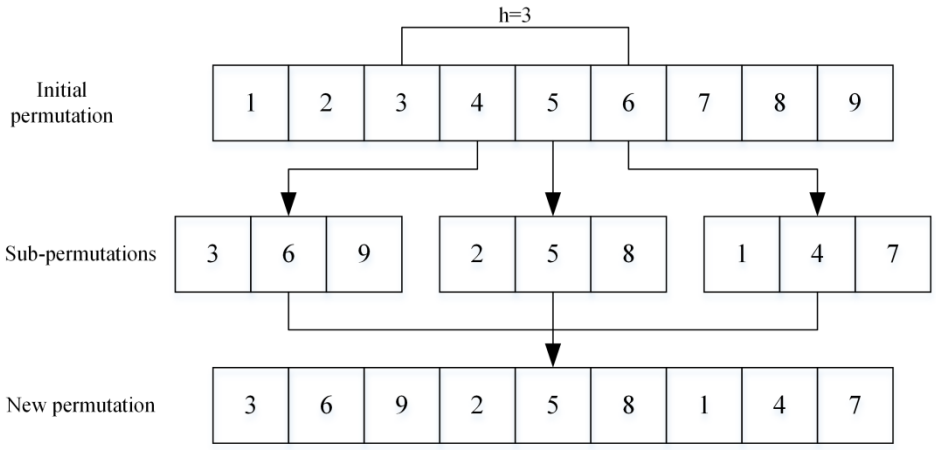


Fig. 5. The Glover operator

Given the above, the Glover operator is indeed a stochastic method and would most likely lead to poor solutions, which negatively affect the search efficiency. Hence, a local search procedure [51] is

conducted to enhance their qualities to improve this situation, the pseudo code of which is depicted in Fig. 6. As shown in this figure, a modified solution δ is obtained by relocating the job in the first position of the current solution c to a different randomly selected position. If $f(\delta) < f(c)$, the solution c is replaced by the solution δ. This procedure is repeated for all positions sequentially until no improvement is found.

```
Procedure for local search
  improvement = true
  while improvement = true do
    improvement = false
    for i = 1 to n do
        δ = insert the job in position i into another new randomly selected position of current c
      if f(δ) < f(c) then
        c = δ
        improvement = true
        break
      endif
    endfor
  endwhile
```

Fig. 6. The pseudo code of the local search

4.6 Dynamic solution acceptance criteria

Except the leader, each solution can be improved by the best neighbor from its own neighbors or the best neighbor from the shared neighbor set. And no matter where the neighbor performing best comes, it always replace the current solution. Apparently, this performs efficiently when the population has a high level of diversity. Nonetheless, with the evolutionary process proceeding, the population diversity would decrease and a number of solutions would be likely to converge, so the benefit mechanism may work with low efficiency in the latter stage. In addition, the leaped solutions obtained through the scout phase are generally poor, such that they may be rapidly replaced by the ones from the shared neighbor set and cannot be used well. Hence, in order to strike a compromise between intensification and diversification mechanisms, we develop a dynamic solution acceptance criteria, where a candidate solution φ' from the shared neighbor set is accepted as the current solution depending on a dynamic acceptance probability p, which decreases linearly as in Eq. 12. And Fig. 7 gives the complete procedure of the solution acceptance.

$$p_g = \begin{cases} p_o - g \times \dfrac{p_o - p_f}{L} & \text{if } \varphi' \text{ is the best among all the evaluated solutions} \\ 0 & \text{else} \end{cases} \tag{12}$$

where p_o, p_f and p_g are the initial probability, the final probability and the certain probability in the

iteration g, respectively. L is the desired number of probabilities between p_o and p_f. The value of p should be large in the early stage to favor the intensification mechanism and be small in the latter stage to favor the diversification mechanism. Here, we set $p_o = 0.99$ and $p_f = 0.1$. The value of L will be studied in Section 5.1.

Procedure for solution acceptance in the iteration g
  c is the current solution
  φ is the best one among the neighbors of c
  φ' is the best one in the shared neighbor set
  if $f(\varphi') < f(\varphi) \&\& f(\varphi') < f(c)$
    random is a random number between 0 and 1
    if $random < p_g$
      $c = \varphi'$
    else if $f(\varphi) < f(c)$
      $c = \varphi$
    endif
  else if $f(\varphi) \le f(\varphi') \&\& f(\varphi) < f(c)$
    $c = \varphi$
  else
    c remains unchanged
  endif

Fig. 7. The pseudo code of the solution acceptance

4.7 Procedure of the presented algorithm

Having described the details of the EMBO algorithm, we provide its complete procedure as follows and the framework of this algorithm is displayed in Fig. 8.

Step1: Algorithm initialization.

Step 1.1: Initialize parameters. Set parameters *ps*, *k*, *s*, *t*, *r*, *L*and the stopping criterion *K*.

Step 1.2: Initialize population. Randomly generate *ps* solutions in the feasible solution space and arbitrarily place them on a hypothetical V formation containing one leader solution, $(ps-1)/2$ solutions in P_L and $(ps-1)/2$ solutions in P_R.

Step 2: A loop evolution. Repeat the tour process including steps 2.1-2.3 for *t* times.

Step 2.1: Evolve the leader. The leader solution is to be improved by randomly generating *k* neighbors, and excluding the best one, the remaining k-1 neighbors are adopted to create two shared neighbor sets Λ_L and Λ_R, each of which contains *s* members.

Step 2.2: Evolve the solutions in P_L and P_R in parallel. Each solution in $P_L(P_R)$ is to be improved by randomly generating *k*-*s* its own neighbors and *s* neighbors from $\Lambda_L(\Lambda_R)$ by using the dynamic solution acceptance criteria. Then excluding the best one among these *k* solutions, the remaining $k-1$ solutions are used to be added to $\Lambda_L(\Lambda_R)$ after they are reset to be null. This

process is conducted for the solutions in P_L (P_R) sequentially from the first to the last.

Step 2.3: Check each solution and perform the scout phase for the solutions that meet the limit *r*.

Step 3: Select a new leader alternately from P_L and P_R.

Step 4: Modify the solutions order using the competitive mechanisms.

Step 5: Check the termination condition. If the stopping condition *K* is satisfied, output the best solution; otherwise, return to step 2.

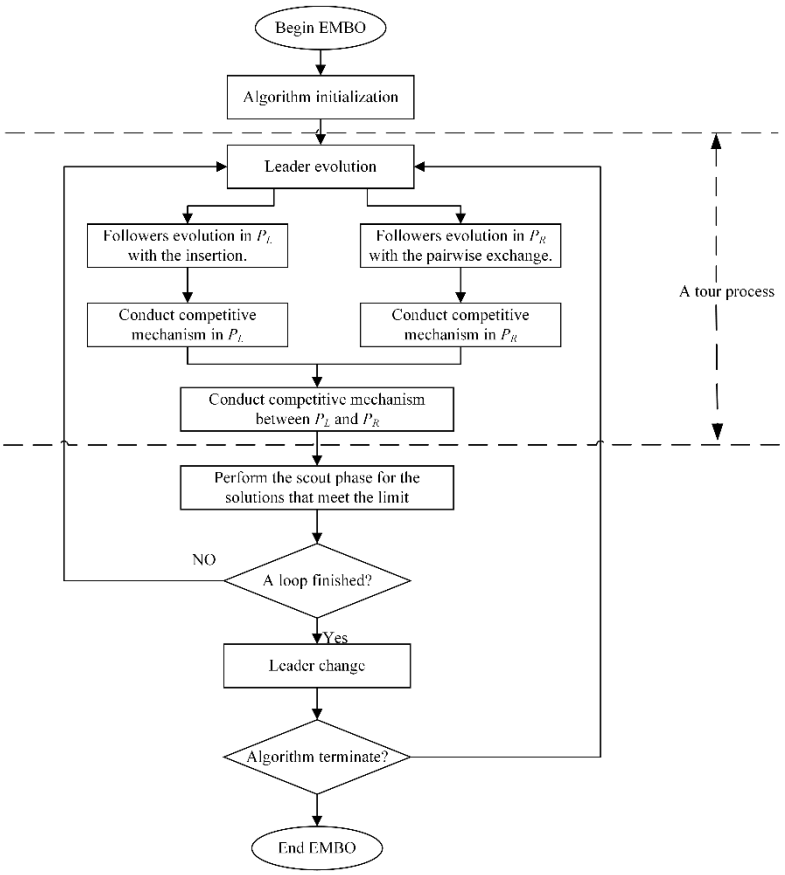


Fig. 8. The framework of the EMBO algorithm

5 Experimental study

In this section, we discuss the computational experiments that are used to evaluate the proposed EMBO algorithm using the maximum CPU elapsed time as the stopping criterion. The capability of obtaining satisfactory solutions within an acceptable time has the practical significance. Because we cannot find the benchmarks for our considered problem in the literature, we generate a collection of 100 test instances in a random way to provide detailed comparisons between the algorithms. Ten different problems with different sizes can be grouped by different combinations of *n* and *m*, where $n = \{20, 40, 60, 80, 100\}$ and $m = \{5, 10\}$. For each $n \times m$ combination, ten instances are generated at random: the processing times $p_{k,j}$ are obtained from a uniform distribution [1, 31], whereas the number of sublots l_j and the number of parallel machines σ_k are sampled from the uniform

distributions [1, 6] and [1, 5], respectively. These settings are common in the scheduling literature and the test data in this paper can be found from http://pan.baidu.com/s/1qX4jIJq. The proposed algorithm is coded in C++ and run on a 2.3 GHZ Intel Core i5 processor in a WIN7 Operation System.

5.1 Parameter settings

The appropriate setting of parameters has a significant influence on the effectiveness and efficiency of stochastic algorithms. To begin with, the neighborhood combination IS and the decoding rule SWT are selected to constitute our algorithm and their effectiveness will be demonstrated in the following two sections. The proposed EMBO contains six control factors, including *ps*, *k*, *s*, *t*, *L* and *r*. Among these, parameters *k* and *s* should be set under a limit that the value of *k* must be greater than or at least equal to $(2\times s+1)$ such that the Taguchi method mentioned below is not suitable for their settings. This limit ensures that the leader solution has a sufficient number of neighbors except for the best one to fill two shared neighbor sets with s members. We set the parameter *k* to 3, which is recommended in several studies [24,39,52,53] considering similar neighborhood-based evolutionary methods for solving combinatorial optimization problems. Then, the value of *s* can only be set to 1. And in our preliminary experiments, these two settings overall behave very well.

With regard to setting the other four parameters efficiently under $k=3$ and $s=1$, the Taguchi method of design of experiment (DOE) [54] with a randomly generated 60×10 instance is applied. We determine four reasonable levels for each of the four parameters listed in Table 2 and utilize the orthogonal array L_{16} to conduct the experiment. Then, setting the maximum elapsed CPU time of $60\times10\times20$ milliseconds as the termination criterion, the proposed algorithm EMBO is independently run 20 times for each combination of parameters, and the average value (AVG) is collected as the response variable. The selected orthogonal array and the corresponding AVG values are summarized in Table 3, and thus, we can obtain the factor level trend shown in Fig. 9 and the response values indicating the significance rank of each parameter given in Table 4. From Table 4, it can be observed that the delta of response values for the parameter *ps* is highest, but for the other three parameters the delta is much lower. This can indicate that the parameter *ps* has the important significance, whereas the other three parameters are less critical. And as is seen from Fig. 9, the proposed algorithm yields better performance with *ps*=51, *t*=10, *L*=1000 and *r*=50. Thus, we will fix these values to conduct the following experiments.

Table 2
Levels of the parameters for the proposed algorithm

Parameters	Parameter Level			
	1	2	3	4
ps	25	51	81	101
t	2	5	10	20
L	500	800	1000	1500
r	30	50	80	100

Table 3
Orthogonal array and AVG values

Experiment number	Parameter				AVG
	ps	*t*	*L*	*r*	
1	25	2	500	30	51876.6
2	25	5	800	50	51803.7
3	25	10	1000	80	51589.1
4	25	20	1500	100	51958.9
5	51	2	800	80	51893.0
6	51	5	500	100	51602.6
7	51	10	1500	30	51836.3
8	51	20	1000	50	51523.9
9	81	2	1000	100	51774.4
10	81	5	1500	80	51725.1
11	81	10	500	50	51849.1
12	81	20	800	30	51934.6
13	101	2	1500	50	52164.7
14	101	5	1000	30	52388.3
15	101	10	800	100	52155.8
16	101	20	500	80	52207.9

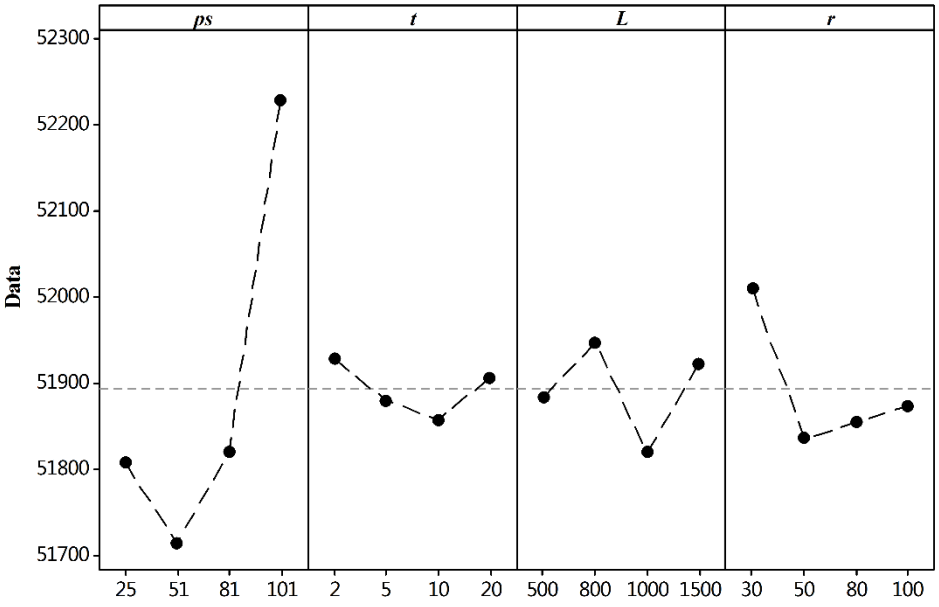


Fig. 9. The trend of the factor level

Table 4
The mean response values

Level	*ps*	*t*	*L*	*r*
1	51807.1	51927.2	51884.1	52009.0
2	51713.9	51879.9	51946.8	51835.4
3	51820.8	51857.6	51818.9	51853.8
4	52229.2	51906.3	51921.3	51872.9
Delta	515.3	69.6	127.9	173.6
Rank	1	4	3	2

5.2 Evaluation of different decoding rules

To evaluate the three different decoding rules' effectiveness, they are separately embedded in the proposed algorithm, namely, $EMBO_{SWT}$, $EMBO_{SDR}$ and $EMBO_{RR}$, and they are tested on all of the 100 randomly generated instances. It cannot be guaranteed to obtain the optimum solution value due to the NP-hardness of the addressed problem. Therefore, the commonly used percentage deviation of the solution from the lower bound in the scheduling literature (see [7,55,56] for example) is not considered here. Instead, for comparisons, the relative percentage increase (RPI) values [17,57] over the obtained best result are calculated as the performance measure as follows.

$$RPI(i) = (c_i - c_{best}) / c_{best} \times 100\% \quad (13)$$

where c_i is the total flow time obtained by a given algorithm i, and c_{best} is the minimum value among the results generated by all of the algorithms. Clearly, lower RPI values are preferred in comparison. We set the maximum CPU elapsed time of $n \times m \times 20$ milliseconds as the stopping criterion, where n is the number of jobs and m is the number of stages in the tested instances. For the tested instances, this termination allows for more time as the number of jobs or stages increases, and has been used in [7, 17, 55-57]. Naderi et al. [7] pointed out that if fixing the same CPU time for all instances, the effect of the instance size cannot be easily studied. To be specific, if the same time is given to all instances, one cannot attribute the worse results for large instance to only instance size due to insufficient CPU time. Moreover, setting a time limit like this allows for a much better statistical analysis. To obtain more reliable experimental results, 20 independent replications are run for each of the 100 instances. The average RPI values, grouped by ten instances with the same problem size $n \times m$, are reported in Table 5, where the overall RPI values are presented in the last row and the best values of each row are marked in bold.

It can be observed from Table 5 that $EMBO_{SWT}$ performs best for all of the ten problems and generates a lowest overall RPI value of 0.66% compared with $EMBO_{SDR}$ (0.96%) and $EMBO_{RR}$ (0.80%). We can also find that $EMBO_{SWT}$ performs much better than $EMBO_{SDR}$ when the problem sizes are large. The reason behind this can be explained that with the increasing of the problem size, there will be more

opportunities for the jobs arriving at stages concurrently. These findings suggest that the decoding rule SWT is more effective than the other two rules.

Table 5
Computational results for the RPI values of the different decoding methods

Problem	$EMBO_{SWT}$	$EMBO_{SDR}$	$EMBO_{RR}$
20×5	**0.34**	0.37	**0.34**
20×10	**0.52**	0.53	0.54
40×5	**0.89**	0.99	0.87
40×10	**0.70**	0.86	0.65
60×5	**0.51**	0.67	0.62
60×10	**0.94**	1.29	1.21
80×5	**0.48**	0.98	0.79
80×10	**0.92**	1.34	1.03
100×5	**0.45**	1.15	0.81
100×10	**0.86**	1.46	1.15
Mean	**0.66**	0.96	0.80

5.3 Evaluation of the combined neighborhood strategy

We proceed now is to investigate the effectiveness of the combined neighborhood strategy by comparing $EMBO_I$, $EMBO_S$ and $EMBO_{IS}$, which are obtained by respectively applying single Insertion, single Swap and neighborhood combination IS in the proposed algorithm. Then, to evaluate the five different neighborhood combinations detailed in Section 4.3, the other four neighborhood combinations are also separately included in the proposed algorithm and four algorithms are obtained, namely $EMBO_{IGSG}$, $EMBO_{IBSB}$, $EMBO_{IGSB}$ and $EMBO_{IBSG}$. We also compare these algorithms with the RPI values and the maximum CPU elapsed time of $n \times m \times 20$ as the stopping criterion. And the results are given in Table 6.

Table 6
Computational results for the RPI values of different neighborhood methodologies

Problem	$EMBO_I$	$EMBO_S$	$EMBO_{IS}$	$EMBO_{IGSG}$	$EMBO_{IBSB}$	$EMBO_{IGSB}$	$EMBO_{IBSG}$
20×5	0.38	0.39	**0.36**	0.53	0.39	0.52	0.43
20×10	0.66	0.56	**0.47**	0.78	0.54	0.51	0.41
40×5	1.18	1.03	**0.80**	4.46	3.66	3.82	4.08
40×10	0.83	0.74	**0.55**	4.28	3.42	3.59	3.86
60×5	0.50	0.55	**0.42**	9.61	5.40	7.11	6.98
60×10	1.27	1.14	**0.70**	13.34	8.64	10.05	11.55
80×5	0.90	0.68	**0.49**	21.02	17.26	17.39	20.81
80×10	1.25	1.10	**0.87**	19.95	17.07	17.10	19.68
100×5	0.73	0.59	**0.45**	26.06	21.39	21.46	24.89
100×10	1.35	1.28	**0.87**	22.80	19.63	19.66	22.33
Mean	0.91	0.81	**0.60**	12.28	9.74	10.12	11.50

From Table 6 it can be seen that the $EMBO_{IS}$ performs best for all of the ten problems and generates a lower RPI value of 0.60% when comparing with $EMBO_I$ (0.91%) and $EMBO_S$ (0.81%). Thus, we can conclude that the combined neighborhood search strategy is more effective than only applying the single neighborhood. It also can be observed that the IS outperforms the other four combinations when comparing $EMBO_{IS}$ with the last four algorithms. As the number of jobs increases, the last four algorithms degrade noticeably. Their poor performance can be explained that their explorations in the feasible space may not be enough under the limited condition of CPU elapsed time. As we have stated before, a neighbor is generated by evaluating n or $n-1$ solutions using the members of the last four combinations and this is computationally expensive for searching the neighborhood when n is large.

5.4 Evaluation of the co-evolutionary strategies

In the previous section, we have demonstrated the effectiveness of the combined neighborhood strategy. As applying the combined neighborhood strategy in our algorithm, the left and right lines can be viewed as two different sub-populations, which have different neighborhood structures and have different specialization searching areas. Thus, the leader change strategy presented in Section 3.4 and the second competitive mechanism proposed in Section 4.4 can be seen as the co-evolutionary strategies to help implement the population-collaboration between two lines [58]. Through immigrating of the solutions, these two strategies can realize the searching information sharing and interaction between two lines to avoid the premature convergence in a single line. And this section will study the effects of the co-evolutionary strategies by comparing four algorithms embedding different strategies with the RPI values. They are the proposed EMBO, the $EMBO_{R1}$ that removes the leader change strategy, the $EMBO_{R2}$ that removes the competitive mechanism, and $EMBO_{R3}$ that removes all of them. The stopping criterion for all these four algorithms is set as the maximum CPU elapsed time of $n \times m \times 20$ and Table 7 reports the results.

Table 7
Computational results for the RPI values of the evolutionary strategies

Problem	**EMBO**	**$EMBO_{R1}$**	**$EMBO_{R2}$**	**$EMBO_{R3}$**
20×5	**0.32**	0.33	0.39	0.39
20×10	**0.43**	**0.43**	0.45	0.48
40×5	**0.79**	0.80	1.27	1.27
40×10	**0.55**	0.58	0.95	0.96
60×5	**0.46**	0.49	0.82	0.84
60×10	**0.72**	0.74	1.24	1.26
80×5	**0.50**	0.54	0.95	1.00
80×10	**0.85**	0.90	1.44	1.49
100×5	**0.41**	0.43	0.91	0.94
100×10	**0.81**	**0.81**	1.42	1.44
Mean	**0.58**	0.61	0.98	1.01

As is seen from Table 7, among these four algorithms, the EMBO generates the best results for all of the ten problems and this can indicate the effectiveness of the two strategies. To be specific, the performance of EMBO ($EMBO_{R2}$) is slightly better than that of $EMBO_{R1}$ ($EMBO_{R3}$) when comparing EMBO with $EMBO_{R1}$ and $EMBO_{R2}$ with $EMBO_{R3}$. However, comparing EMBO with $EMBO_{R2}$ and $EMBO_{R1}$ with $EMBO_{R3}$, the EMBO and $EMBO_{R1}$ all yield much lower RPI values than their competitors. From the above observations, we can conclude that the competitive mechanism between two lines can improve the performance of the algorithm much more significantly than the leader change strategy. This can be explained by their different solution immigrating behaviors. In the leader change strategy, the leader is to be exchanged with a follower from one line after each loop is finished, and then it only can affect another line in the next loop through the benefit mechanism. Whereas in the competitive mechanism, when an evolving loop ends, t pairs of individuals at the same position of the two lines are exchanged and conduct a TPOX in each pair. Obviously, the interaction between two lines through the competitive mechanism is more intensive than that through the leader change. However, exchanging solutions between two lines too frequently may lose their differences and this can also reflect the importance of the fine tuning of the parameter t.

5.5 Effectiveness of the proposed EMBO algorithm

This section intends to investigate the effectiveness of the proposed EMBO and the pure EMBO algorithm, namely $EMBO_P$ that removes the local search strategy, by comparing them with seven other existing meta-heuristics in the literature. They are MBO (Duman et al. [39]), IMBO (Pan et al. [24]), MMBO (Niroomand et al. [40]), GA (Mohsen et al. [37]), GA_R (Ruiz et al. [59]), DPSO (Tseng et al. [35]), and DABC (Pan et al. [25]), which were proposed for other combinatorial optimization problems and have shown high performance. To compare against MBO, IMBO and MMBO, which are variants of the migrating birds algorithm, we can demonstrate the effectiveness of our modifications to the basic MBO. And by comparing with the other four algorithms, which are variants of well-known meta-heuristics and were proposed for various flowshop problems, we can further demonstrate the validity of our proposed algorithm. Since these seven algorithms were not dealing with the HLFS problem addressed in this paper, necessary adaptions should be made to make comparisons possible. To be specific, the IMBO was presented for the flowshop problem and also used the permutation-based representation, so the adaption is carried out by changing the objective evaluation. However, for another two MBO variants, which were dealing with quadratic assignment problem and closed loop layout problem respectively, they are adapted by modifying the solution encoding and the objective evaluation. Moreover, their neighborhood structures are all set the same as the IMBO, namely mixed neighborhood that was illustrated to be more effective than a single neighborhood search in [24]. As for the other four meta-heuristics, similar with the IMBO, they are also changed with the objective evaluation. All of these compared algorithms generate an initial population randomly and their parameters are directly taken from the literature. We have fully re-implemented and coded them in

C++.

For each of the 100 instances, the average RPI values as well as the standard deviations (SD) are gained across 20 independent replications, and they are also grouped under the same problem size. Moreover, in order to conduct more comprehensive comparisons, we consider three different levels for the maximum elapsed CPU time of $n \times m \times \mu$ with μ set as 10, 20 and 30. The choice of this stopping criterion is motivated by the fact that all the compared algorithms have been coded in the same programming language (C++)) using many common functions and structures, and are tested on the same aforementioned PC. The computational results under three different stopping criteria are reported in Tables 8-10, respectively. Besides, to verify the statistical validity of the results presented, they are analyzed by one-factor analysis of variance (ANOVA) where the type of algorithms is considered to be a single factor. Figs. 10-12 display the means plots with Tukey HSD (honestly significant difference) intervals at the 95% confidence level for the compared algorithms under different stopping criteria, respectively. It should be noted that if the confidence intervals of any two algorithms are overlapped, there is no statistically significant difference between them.

It can be seen from Tables 8-10 that the EMBO performs best among all algorithms in terms of the overall RPI values, which is followed by the EMBO_P. Comparing with the EMBO_P in more detail, the EMBO yields better results for 9 out of 10 problems except for 20×10 when $\mu = 10$ and $\mu = 20$, and for all of the problems when $\mu = 30$. These observations can indicate the effectiveness of the local search strategy. As is depicted in Figs. 10-12, although the EMBO_P performs less well than the EMBO but still keeps competitive. Whereas comparing with the other seven algorithm, the EMBO_P generally works better under different stopping criteria, and statistically better than all the other algorithms when $\mu = 10$ and 6 out of 7 algorithms except for the GA when $\mu = 20$ and $\mu = 30$.

Next, we will compare the whole proposed algorithm EMBO with other seven algorithms more detailedly. When $\mu = 10$, it can be seen from Table 8 that the EMBO algorithm obtains the minimum RPI values for all of the ten problems and generates a much lower overall RPI value of 0.75% when compared with MBO (3.74%), IMBO (2.97%), MMBO (4.75%), GA (2.83%), GA_R (3.52%), DPSO (4.86%) and DABC (4.85%). When μ achieves the values of 20 and 30, Tables 9 and 10 show that our algorithm yields the best results for all the problems and produces much lower overall RPI values once again. For the three different stopping criteria, considering the overall RPI values, the performance of the EMBO is obviously better than the other algorithms. And regarding the standard deviations, the EMBO algorithm also generates smallest values on the whole, which suggest that our algorithm is robust. Furthermore, Figs. 10-12 show that the EMBO performs statistically better than the others at different elapsed CPU times. In conclusion, the above experimental results allow us to conclude that the proposed EMBO is much more effective and robust than the other compared algorithms for the HLFS problem with the objective of minimizing the total flow time.

Table 8
The computational RPI (SD) values when $\mu = 10$

Instance	EMBO	$EMBO_P$	MBO	IMBO	MMBO	GA	GA_R	DPSO	DABC
20×5	**0.43(0.26)**	0.45(0.27)	0.91(0.30)	0.65(0.20)	2.18(0.79)	1.55(0.66)	1.95(0.99)	3.46(1.98)	1.06(0.92)
20×10	0.58(**0.25**)	**0.57(0.25)**	1.15(0.49)	0.71(0.33)	3.36(0.89)	2.23(0.92)	2.35(1.11)	3.68(1.89)	1.74(0.56)
40×5	**0.97(0.41)**	1.03(0.46)	3.70(0.70)	2.76(0.54)	4.05(1.36)	3.63(0.84)	3.03(1.70)	4.97(2.78)	4.83(2.45)
40×10	**0.76(0.33)**	0.81(0.35)	3.38(0.71)	2.67(0.49)	5.03(1.59)	2.99(0.91)	3.44(1.44)	4.65(3.36)	5.27(2.60)
60×5	**0.55(0.26)**	0.67(0.33)	4.02(0.77)	3.26(0.56)	2.66(1.87)	2.53(0.70)	2.40(1.52)	3.53(2.54)	5.06(2.12)
60×10	**1.06(0.38)**	1.15(0.43)	4.13(0.80)	3.47(0.55)	4.85(1.48)	2.83(0.85)	3.91(1.95)	4.09(2.00)	3.50(2.90)
80×5	**0.57(0.30)**	0.68(0.34)	4.17(0.73)	3.44(0.55)	5.08(1.89)	2.71(0.66)	3.98(1.83)	6.56(3.52)	7.89(2.89)
80×10	**1.00(0.45)**	1.13(**0.45**)	4.85(0.97)	4.24(0.87)	6.41(1.68)	3.13(0.87)	3.02(2.06)	6.09(4.18)	6.64(3.29)
100×5	**0.50(0.34)**	0.64(0.39)	4.98(0.85)	3.81(0.69)	6.31(1.85)	3.00(0.85)	5.18(2.02)	6.97(3.93)	5.55(3.24)
100×10	**1.08(0.61)**	1.31(0.71)	6.09(0.96)	4.66(0.69)	7.60(1.51)	3.65(0.96)	5.97(1.86)	4.59(3.64)	6.95(2.86)
Mean	**0.75**(**0.36**)	0.84(0.40)	3.74(0.73)	2.97(0.55)	4.75(1.49)	2.83(0.82)	3.52(1.65)	4.86(2.98)	4.85(2.38)

Table 9
The computational RPI (SD) values when $\mu = 20$

Instance	EMBO	$EMBO_P$	MBO	IMBO	MMBO	GA	GA_R	DPSO	DABC
20×5	**0.34(0.18)**	0.35(**0.18**)	0.96(0.23)	0.76(0.18)	2.18(0.51)	1.81(0.66)	1.42(0.86)	0.66(0.45)	0.87(0.44)
20×10	**0.47(0.16)**	**0.47**(0.17)	1.40(0.41)	0.69(0.31)	2.99(0.65)	1.95(0.93)	1.84(1.03)	0.78(0.53)	1.54(0.52)
40×5	**0.87(0.31)**	0.91(0.33)	3.06(0.56)	3.27(0.54)	4.19(0.95)	3.04(0.76)	4.00(0.74)	5.55(1.22)	2.61(0.36)
40×10	**0.69(0.37)**	0.78(0.39)	3.01(0.53)	2.61(0.35)	4.32(1.05)	2.38(0.79)	2.80(0.55)	4.66(0.87)	3.67(1.04)
60×5	**0.50(0.26)**	0.69(0.27)	2.78(0.48)	2.75(0.47)	5.65(1.51)	2.59(0.58)	1.42(0.59)	5.43(0.94)	4.15(0.65)
60×10	**0.90(0.34)**	1.15(0.39)	3.49(0.62)	2.81(0.47)	4.28(1.27)	2.52(0.74)	3.22(1.09)	7.74(1.24)	8.47(1.16)
80×5	**0.48(0.22)**	0.88(**0.22**)	2.67(0.52)	2.87(0.44)	5.64(1.42)	2.22(0.52)	2.94(0.61)	5.21(0.99)	4.23(1.29)
80×10	**0.91**(0.45)	1.34(**0.44**)	4.19(0.83)	3.66(0.57)	6.29(1.40)	3.17(0.76)	3.27(0.70)	8.22(1.21)	4.56(0.89)
100×5	**0.40(0.22)**	0.83(0.24)	3.45(0.61)	3.11(0.52)	5.18(1.48)	3.03(0.51)	4.10(0.77)	7.86(1.16)	3.20(0.41)
100×10	**0.86(0.36)**	1.38(0.38**)**	4.35(0.70)	4.17(0.53)	4.51(1.19)	3.26(0.73)	5.68(1.09)	4.52(0.97)	4.11(1.18)
Mean	**0.64(0.29)**	0.88(0.30)	2.94(0.55)	2.67(0.44)	4.52(1.14)	2.60(0.70)	3.07(0.80)	5.06(0.96)	3.74(0.79)

Table 10
The computational RPI (SD) values when $\mu = 30$

Instance	EMBO	$EMBO_P$	MBO	IMBO	MMBO	GA	GA_R	DPSO	DABC
20×5	**0.29(0.15)**	0.31(0.16)	1.97(0.24)	1.31(0.18)	1.03(0.31)	1.33(0.66)	1.21(0.70)	0.39(0.13)	0.80(0.78)
20×10	**0.40**(0.24)	0.44(0.24)	1.60(0.41)	1.40(**0.20**)	1.08(0.39)	1.96(0.93)	1.66(0.98)	0.54(0.15)	1.41(1.24)
40×5	**0.80**(0.42**)**	0.95(0.49)	2.60(0.53)	2.69(**0.38**)	5.01(0.66)	2.39(0.78)	2.36(1.25)	1.99(0.99)	4.12(1.81)
40×10	**0.65(0.24)**	0.83(0.27)	2.58(0.43)	2.13(0.26)	5.37(0.62)	3.10(0.80)	3.56(1.05)	3.24(2.25)	4.12(1.77)
60×5	**0.46(0.15)**	0.72(0.16)	3.66(0.35)	3.37(0.33)	7.07(0.78)	3.29(0.82)	4.25(0.88)	5.27(1.71)	3.92(1.60)
60×10	**0.82**(0.39)	1.11(**0.38**)	3.33(0.52)	3.11(0.42)	6.01(0.81)	2.40(0.72)	3.32(1.63)	4.22(1.91)	7.62(2.29)
80×5	**0.43(0.18)**	0.68(0.20)	2.37(0.42)	2.17(0.35)	4.72(0.86)	2.36(0.48)	2.11(1.24)	4.40(2.26)	6.66(1.99)
80×10	**0.86(0.31)**	1.25(0.33)	4.05(0.69)	3.99(0.42)	4.45(1.00)	2.72(0.72)	3.61(1.65)	6.80(2.44)	4.47(2.84)
100×5	**0.39(0.18)**	0.68(0.20)	3.22(0.48)	2.53(0.29)	5.41(1.01)	2.36(0.51)	4.97(1.23)	4.05(1.81)	3.74(1.86)
100×10	**0.79(0.30)**	1.19(0.32)	3.51(0.64)	3.94(0.44)	5.85(1.96)	3.07(0.72)	5.01(1.44)	4.58(1.26)	4.72(1.97)
Mean	**0.59(0.26)**	0.82(0.28)	2.89(0.47)	2.66(0.33)	4.60(0.84)	2.50(0.71)	3.21(1.21)	3.55(1.49)	4.16(1.82)

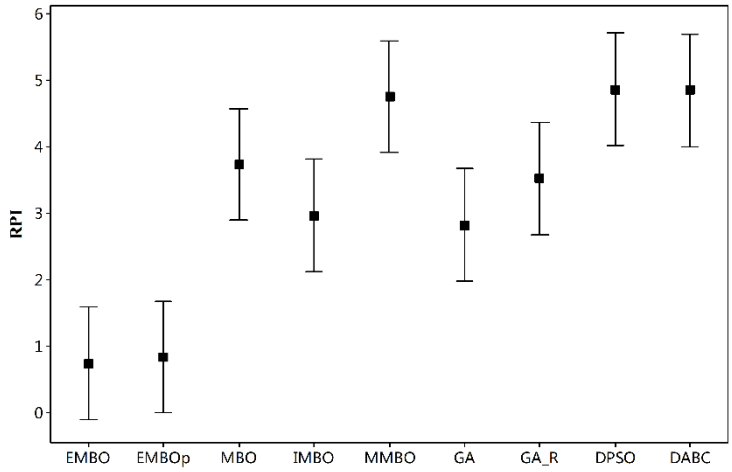


Fig. 10. The mean plot and 95% Tukey HSD confidence intervals between the algorithms when $\mu = 10$

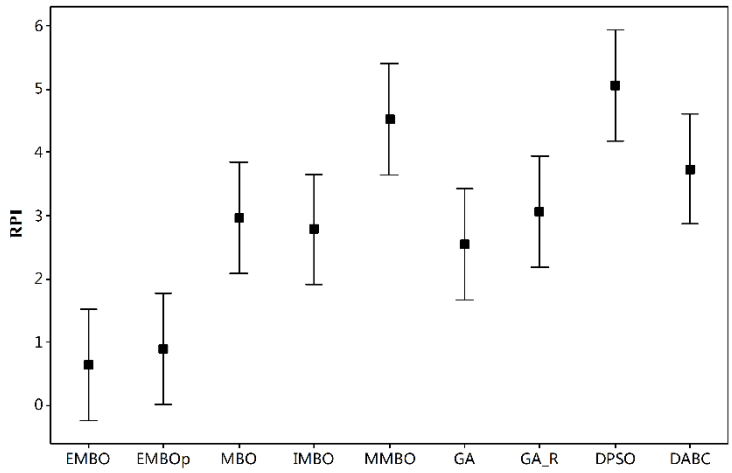


Fig. 11. The mean plot and 95% Tukey HSD confidence intervals between the algorithms when $\mu = 20$

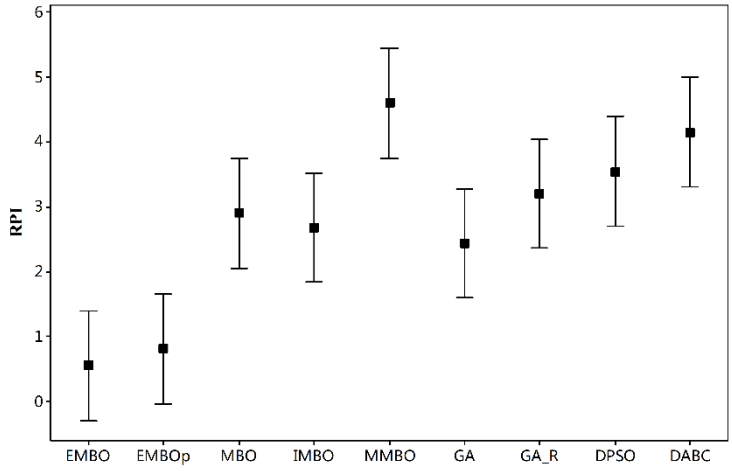


Fig. 12. The mean plot and 95% Tukey HSD confidence intervals between the algorithms when $\mu = 30$

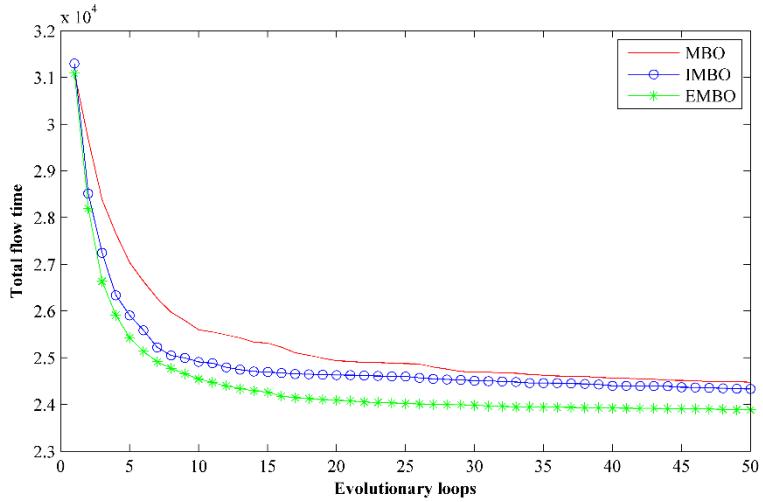


Fig.13. The convergence curve for one instance of 40×10

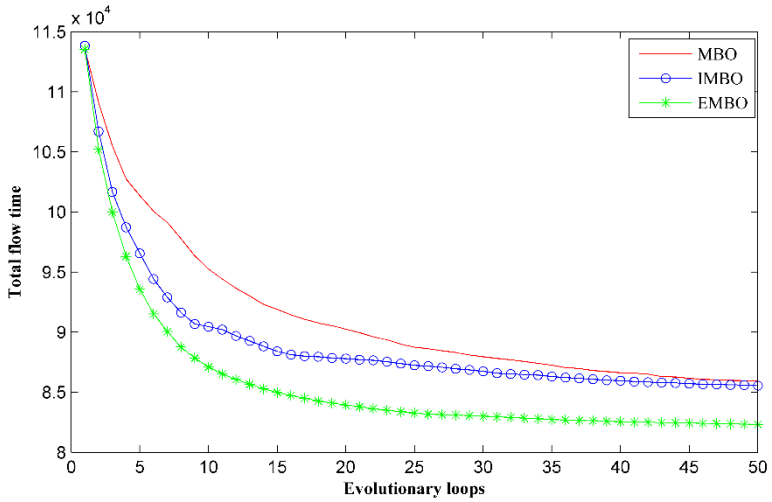


Fig. 14. The convergence curve for one instance of 80×10

To verify the convergence ability of the proposed algorithm, Figs. 13-14 respectively present the convergence curves of three MBO variants, including MBO, IMBO and EMBO for two instances randomly selected from the groups of 40×10 and 80×10. Note that the MMBO is removed because this algorithm is great difference from the basic MBO algorithm. It can be observed from the two figures that when comparing with the MBO and IMBO algorithms, the EMBO shows better convergence performance and can also converge to a better optimal result. Thus, it can be further demonstrated that the EMBO algorithm can obtain a good balance between the exploration and exploitation abilities.

6 Conclusions

This study proposes an effective modified migrating birds optimization (EMBO) for hybrid flow shop hybridizing with lot streaming (HLFS). The objective is to minimize the total flow time. We first present a mathematical model, and to the best of our knowledge, there are no published papers

addressing this problem. For this problem, we employ the commonly used permutation-based solution representation and introduce a new decoding rule, namely SWT to schedule the jobs arriving at stages concurrently more reasonably, which is demonstrated to be more effective than the other two rules in the literature. Then, we present several effective modifications to enhance the performance of the MBO algorithm, including a combined neighborhood search strategy, two competitive mechanisms, the scout phase and the dynamic solution acceptance criteria. Finally, computational simulations and comparisons based on 100 randomly generated instances demonstrate the effectiveness and efficiency of the EMBO algorithm.

Our future research could focus on exploration of problem-specific characteristics to develop more effective heuristics for the HLFS problem. Another clue is to analyze the effectiveness of the EMBO with the other optimization objectives, such as maximum tardiness and multi-objective cases, or to consider the problem with realistic and practical assumptions, such as machine setup time, machine eligibility and interim buffer constraints. Moreover, it could be interesting to investigate the performance of the EMBO in other combinatorial optimization problems, such as the blocking flowshop, job shop and the certain realistic scheduling problems.

Acknowledgments

This research is partially supported by National Science Foundation of China under Grants 51575212, 61503170, 61573178, Program for New Century Excellent Talents in University (NCET-13-0106), Doctoral Program Foundation of Institutions of Higher Education of China (20130042110035), Science Foundation of Hubei Province in China (2015CFB560), Key Laboratory Basic Research Foundation of Education Department of Liaoning Province (LZ2014014), College Science and Technology Program Project of Shandong Province (J13LI10) and Open Research Fund Program of the State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, China.

References

[1] R.J. Wittrock, Scheduling algorithms for flexible flow lines, IBM Journal of Research and Development 29 (4) (1985) 401-412.

[2] R.J. Wittrock, An adaptable scheduling algorithm for flexible flow lines, Operations Research 36 (3) (1988) 445-453.

[3] J. Grabowski, J. Pempera, Sequencing of jobs in some production system, European Journal of Operational Research 125 (3) (2000) 535-550.

[4] H.D. Sherali, S.C. Sarin, M.S. Kodialam, Models and algorithms for a two-stage production process, Production Planning and Control 1 (1) (1990) 27-39.

[5] A.G.P. Guinet, M. Solomon, Scheduling hybrid flowshops to minimize maximum tardiness or maximum completion time, International Journal of Production Research 34 (1996) 1643-1654.

[6] R. Ruiz, J.A. Vazquez Rodriguez, The hybrid flow shop scheduling problem, European Journal of Operations Research 205 (2010) 1-18.

[7] B. Naderi, R. Ruiz, M. Zandieh, Algorithms for a realistic variant of flowshop scheduling, Computers & Operations Research 37 (2010) 236-246.

[8] S.A. Brah, J.L. Hunsucker, Branch and bound algorithm for the flow shop with multiple processors, European Journal of Operations Research 51 (1991) 88-89.

[9] J. Carlier, E. Neron, An exact algorithm for solving the multiprocessor flowshop, RAIRO-Operations Research 34 (2000) 1-25.

[10] E. Neron, P. Baptiste, J.N.D. Gupta, Solving hybrid flow shop using energetic reasoning and global operations, OMEGA 29 (2001) 501-511.

[11] J.N. Gupta, Two-stage, hybrid flow shop scheduling problem, Journal of the Operational Research Society, 39 (1988) 359-364.

[12] H.T. Lin, C.J. Liao, A case study in a two-stage hybrid flow shop with setup time and dedicated machines, International Journal of Production Economics 86 (2003) 133-143.

[13] C. Kahraman, O. Enginb, I. Kaya, Multiprocessor task scheduling in multistage hybrid flow-shops: a parallel greedy algorithm approach, Applied Soft Computing 10 (2010) 1293-1300.

[14] C.D. Paternina, J.R. Montoya-Torres, Scheduling jobs on a k-stage flexible flow-shop, Annals of Operations Research 164 (2008) 29-40.

[15] J. Junwattanakit, M. Reodecha, D. Chaovalit, A comparison of scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times and dual criteria, Computers & Operations Research 36 (2) 2009 358-378.

[16] Z. Jin, Z. Yang, T.Ito, Metaheuristic algorithms for the multistage hybrid flowshop scheduling problem, International Journal of Production Economics 100 (2) (2006) 322-334.

[17] B. Naderi, M. Zandieh, A.K.G. Balagh, V. Roshanaei, An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion and total tardiness, Expert Systems with Applications 36 (6) (2009) 9625-9633.

[18] E. Nowicki, C. Smutnick, The flow shop with parallel machines: a tabu search approach, European Journal of Operations Research 106 (1998) 226-253.

[19] K. Alaykyran, O. Engin, A. Doyen, Using ant colony optimization to solve hybrid flow shop scheduling problems, International Journal of Advanced Manufacturing Technology 35 (2007) 541-550.

[20] S. Khaloui, F. Ghedjati, A. Hamzaoui, A meta-heuristic approach to solve a JIT scheduling problem in hybrid flowshop, Engineering Application of Artificial Intelligence 23 (2010) 765-771.

[21] C. Kahraman, O. Engin, I. Kaya, M.K. Yilmaz, An application of effective genetic algorithms for solving hybrid flow shop scheduling problems, International Journal of Computational Intelligence Systems 1 (2) (2008) 134-147.

[22] Q. Niu, T. Zhou, S. Ma, A quantum-inspired immune algorithm for hybrid flow shop with makespan criterion, Journal of Universal Computerence, 15 (4) (2009) 765-785.

[23] O. Engin, A. Doyen, A new approach to solve hybrid flow shop scheduling problems by artificial immune system, Future Generation Computer Systems, 20 (2004) 1083-1095.

[24] Q.K. Pan, Y. Dong, An improved migrating birds optimisation for a hybrid flowshop scheduling with total flowtime minimisation, Information Sciences, 277 (2014) 643-655.

[25] Q.K. Pan, L. Wang, J.Q. Li, A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimization, OMEGA 45 (2014) 42-56.

[26] J.Q. Li, Q.K. Pan, Solving the large-scale hybrid flow shop scheduling problem with limited buffers by a hybrid artificial befe colony algorithm, Information Sciences 316 (2015) 487-502.

[27] C.J. Liao, E. Tjandradjaj, T.P. Chung, An approach using particle swarm optimization and bottleneck heuristic to solve flowshop scheduling problem, Applied Soft Computing 12 (2012) 1755-1764.

[28] S. Reiter, A system for managing job shop production, Journal of Business, 39 (1965) 371-393.

[29] W. G. Truscott, Production scheduling with capacity constrained transportation activities, Journal of Operations Management 6 (1986) 333-348.

[30] Q.K. Pan, R. Ruiz, An estimation of distribution algorithm for lot-streaming flow shop problems with setup times, OMEGA 40 (2012) 166-180.

[31] Q.K. Pan, M.F. Tasgetiren, P.N. Suganthan, A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem, Information Sciences, 181 (12) 2011 2455-2468.

[32] M. Feldmann, D. Biskup, Lot streaming in a multiple product permutation flow shop with intermingling, International Journal of Production Research 46 (1) (2008) 197-216.

[33] S. Marimuthu, S.G. Ponnambalam, A.N. Jawahar, Evolutionary algorithms for scheduling m-machine flow shop with lot streaming, Robotics and Computer-Integrated Manufacturing 24 (2008) 125-139.

[34] C.H. Martin, A hybrid genetic algorithm/mathematical programming approach to the multi-family flowshop scheduling problem with log streaming, OMEGA, 37 (1) (2009) 126-137.

[35] C.T. Tseng, C.J. Liao, A discrete particle swarm optimization for lot-streaming flowshop scheduling problem, European Journal of Operations Research, 191 (2008) 360-373.

[36] F.M. Defersha, M. Chen, Mathematical model and parallel genetic algorithm for hybrid flexible flowshop lot streaming problem, International Journal of Advanced Manufacturing Technology, 62 (1-4) (2012) 249-265.

[37] N. Mohsen, M. Iraj, Multi-job lot streaming to minimize the weighted completion time in a hybrid flow shop scheduling problem with work shift constraint, International Journal of Advanced Manufacturing Technology, 70 (2014) 501-514.

[38] L.X. Tang, Y. Zhao, J.Y. Liu, An improved differential evolution algorithm for practical dynamic scheduling in steelmaking-continuous casting production, IEEE Transaction on evolutionary computation, 2 (18) (2014) 209-225.

[39] E. Duman, M. Uysal, A.F. Alkaya, Migrating Birds Optimization: A new metaheuristic approach and its performance on quadratic assignment problem, Information Sciences, 217 (2012) 65-77.

[40] S. Niroomand, A.H. Vencheh, R. Sahin, Modified migrating birds optimization algorithm for closed loop layout with exact distances in flexible manufacturing systems, Expert Systems with Applications, 42 (2015) 6586-6597.

[41] V. Tongur, E. Ülker, Migrating birds optimization for flow shop sequencing problem, Journal of Computer & Communicating, 2 (2014) 142-147.

[42] V. Tongur, E. Ülker, The analysis of migrating birds optimization algorithm with neighborhood operator on travelling salesman problem, Intelligent and Evolutionary Systems, Springer International Publishing, (2016) 227-237.

[43] E.L. Ruiz, C.E. Lzquierdo, J.D. Armas, Migrating birds optimization for the seaside problems at maritime container terminals, Journal of Applied Mathematics, 5 (2015).

[44] R. Soto, B. Crawford, B. Almonacid, A migrating birds optimization algorithm for machine-part cell formation problems, MICAI 2015, Part I, LNAI 9413, pp.1-12, 2015.

[45] J.Q. Li, Q.K. Pan, F.T. Wang, A hybrid variable neighborhood search for solving the hybrid flow shop scheduling problem, Applied Soft Computing, 24 (2014) 63-77.

[46] T. Schiavinotto, T. Stutzle, A review of metrics on permutations for search landscape analysis, Computers & Operations Research, 34 (2007) 3143-3153.

[47] A.D. Yimer, K. Demirli, Fuzzy scheduling of job orders in a two-stages flow shop with batch processing machines, International Journal of Approximate Reasoning, 50 (2009) 117-137.

[48] S.H. Chen, P.C. Cang, C.H. Liu, Sub-population genetic algorithm with mining gene structures for multi objective flow shop scheduling problems, Expert Systems with Applications, 33 (2007) 762-771.

[49] D. Karaboga. An idea based on honey bee swarm for numerical optimization. Technical report TR06, Erciyes University, Engineering Faulty, Computer Engineering Department, 2005.

[50] F. Glover, A template for scatter search and path relinking, in: J.K. Kao, E. Lutton, E. Ronald, M. Shoenauer, D. Snyers (Eds.), Artificial Evolution, Lecture Notes in Computer Science, vol. 1386, Springer, Berlin Heidelberg New York, (1998) 13-54.

[51] B. Naderi, M. Zandieh, A. Khaleghi Ghoshe Balagh, An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness, Expert Systems with Applications, 36 (2009) 9625-9633.

[52] L. Wang, X.L. Zheng, S.Y. Wang, A novel binary fruit fly optimization algorithm for solving the multidimensional knapsack problem, Knowledge-Based Systems, 48 (2013) 17-23.

[53] B. Zhang, Q.K. Pan, X.L. Zhang, An effective hybrid harmony search-based algorithm for solving multidimensional knapsack problems, Applied Soft Computing, 29 (2015) 288-297.

[54] D.C. Montgomery, Design and Analysis of Experiments, John Wiley & Sons, Arizona, (2005).

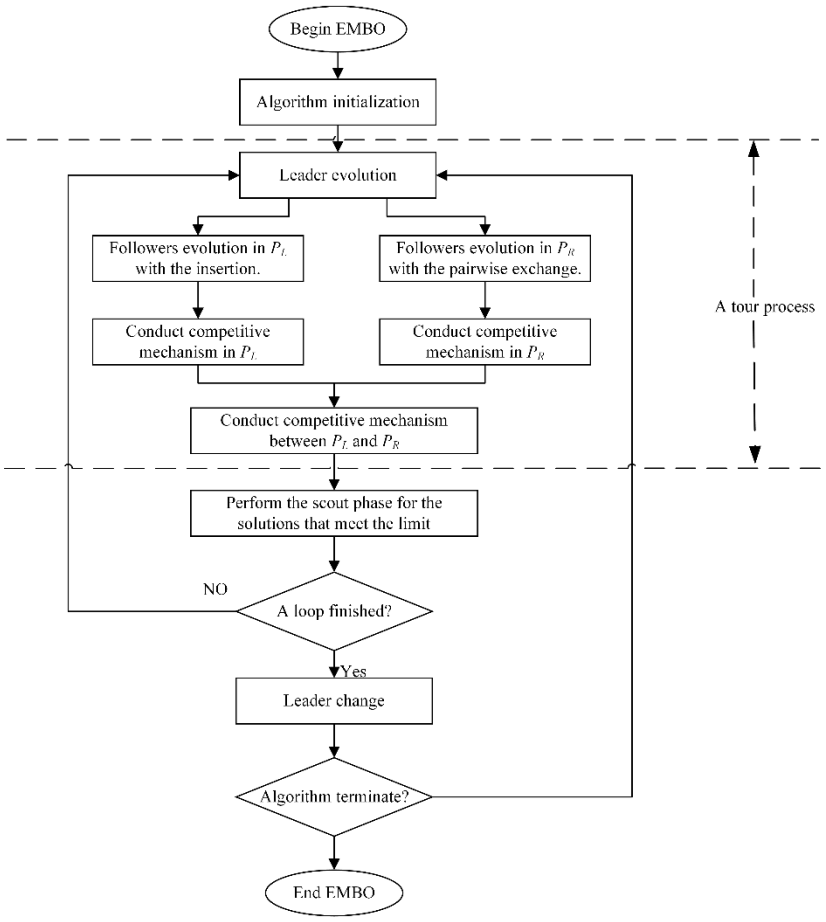
[55] C.L. Chen, T.R. Tzeng, C.L. Chen, A new heuristic based on local best solution for permutation flow shop scheduling, Applied Soft Computing, 29 (2015)75-81.

[56] R. Ruiz, T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, European Journal of Operational Research, 177(3) (2007) 2033-2049.

[57] E. Vallada, R. Ruiz, Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem, Omega, 38(1–2) (2010), 57-67.

[58] E. Alba, M. Tomassini, Parallelism and evolutionary algorithms[J], IEEE Trans on Evolutionary Computation, 6 (5) (2002) 443-462.

[59] R. Ruiz, C. Marota, A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility, European Journal of Operational Research, 169 (2006) 781-800.



This above figure illustrates the flowchart of our proposed algorithm (EMBO). The proposed algorithm starts with a number of initial solutions randomly generated in the solution space, including a leader solution and the other members in left and right lines. Then, an evolving loop involving a number of tours proceeds, and each tour evolves beginning with the leader and processing along the left and right lines in parallel by exploring their neighborhood and using the dynamic solution acceptance criteria. The insertion and the pairwise exchange neighborhood operators are respectively applied for the individuals in P_L and P_R. And two competitive mechanisms are used to modify the solutions order when a tour is finished. Finally, when a loop is finished, the scout phase for the solutions is conducted, the leader is to be changed, and another loop starts.