An evolutionary algorithm for a hybrid flowshop scheduling problem with consistent sublots

Xinli Zhang

School of Mathematic Science,
Liaocheng University,
Liaocheng, 25200, China
Email: zhangxinli@lcu.edu.cn

Biao Zhang* and Leilei Meng

School of Computer Science,
Liaocheng University,
Liaocheng, 25200, China
Email: zhangbiao1218@gmail.com
Email: mengleilei@hust.edu.cn
*Corresponding author

Yaping Ren

School of Intelligent Systems Science and Engineering,
Jinan University,
Zhuhai Campus,
Zhuhai, 519000, China
Email: renyp1@163.com

Ronghua Meng

Hubei Key Laboratory of
Hydroelectric Machinery Design & Maintenance,
China Three Gorges University,
Yichang 443002, China
Email: mengrh@ctgu.edu.cn

Junqing Li

School of Computer Science,
Liaocheng University,
Liaocheng, 25200, China
and
School of Information Science and Engineering,
Shandong Normal University,
Jinan, 250000, China
Email: lijunqing@lcu-cs.com

Abstract: Lot streaming is the most often used technique to support the time-based strategy in the modern manufacturing system, which can split the jobs (or lots) with larger size into several sublots with smaller size. With this manufacturing technique, this paper studies a hybrid flowshop scheduling problem with consistent sublots (HFSP_CS). With the consideration of the integrated optimisation of lot sequencing and lot splitting, a mixed-integer linear programming (MILP) model is established with the objective of minimising the total flowtime. Since the NP-hard property of the problem, a solution method integrating the migrating birds optimisation (MBO) and variable neighbourhood descent (VND) algorithms is developed. Moreover, by taking into account the problem-special characteristics, the two-layer coding mechanism and a corresponding initialisation method are designed. Some heuristic methods are also presented in the decoding process. In the computational study, the effectiveness of the proposed algorithm is evaluated by comparing with CPLEX solver and other state-of-the-art algorithms.

Keywords: hybrid flowshop; lot streaming; consistent sublots; migrating birds optimisation; MBO.

Reference to this paper should be made as follows: Zhang, X., Zhang, B., Meng, L., Ren, Y., Meng, R. and Li, J. (2022) 'An evolutionary algorithm for a hybrid flowshop scheduling problem with consistent sublots', *Int. J. Automation and Control*, Vol. 16, No. 1, pp.19–44.

Biographical notes: Xinli Zhang received her BS and PhD from the Xi'an Jiaotong University, Xi'an, China in 2006 and 2009, respectively, and finished her postdoctoral study in 2012 at the Nanjing Normal University. Since 2012, she has been with the School of Mathematical Sciences.

Biao Zhang received his BS and MS from the Shandong University of Technology, Zibo, and Liaocheng University, Liaocheng, China in 2012 and 2015, respectively. Since 2015, he has been working toward to his PhD in Industrial Engineering at the Huazhong University of Science and Technology. His current research interests include discrete optimisation and scheduling.

Leilei Meng received his BS in Mechanical Engineering from the Chang'an University, Xian, China in 2014. He is currently working toward to his PhD in the School of Mechanical Science and Engineering at the Huazhong University of Science & Technology, China. His research mainly focuses on modelling, optimisation of scheduling problems, tool wear prediction and sustainable manufacturing.

Yaping Ren received his BS in Communications and Transportation Engineering from Liaocheng University, Liaocheng, China in 2014, and MS in Transportation College of Northeast Forestry University, China in 2016. He received his PhD degree in the School of Mechanical Science and Engineering from Huazhong University of Science and Technology (HUST), China in 2019. He is currently an Associate Professor in the School of Intelligent Systems Science and Engineering, Jinan University (Zhuhai Campus). His research mainly focuses on industrial engineering, disassembly planning, transportation planning, decision making and optimisation methods.

Ronghua Meng received his PhD degree from Huazhong University of Science and Technology, Wu Han, China in 2019. His work unit is China Three Gorges University. His research directions include manufacture system optimisation, production scheduling and so on.

Junqing Li received his MS degree of Computer Science and Technology from Shandong Economic University, Shandong, China in 2004, and PhD degree from System Engineering at Northeastern University, Shenyang, China. Since 2004, he has been with School of Computer Science Department, Liaocheng University, Liaocheng, China, where he became a Professor in 2018. His current research interests include intelligent optimisation and scheduling.

1 Introduction

The hybrid flowshop scheduling problem (HFSP), also known as flexible flowshop scheduling problem (FSP), is an extension of the FSP (Öztop et al., 2019; Zhang et al., 2018, 2019a, 2019b). Compared with the workshop layout of FSP, the workshop layout of HFSP has the advantages of increasing workshop throughput, balancing production capacity of each stage and reducing the impact of bottleneck stages, so it has been widely used in chemical industry, textile, paper making and other fields (Ruiz and Vázquez-Rodríguez, 2010; Lei and Wang, 2019; Chen et al., 2019). In the traditional HFSP research, the basic scheduling unit is a single job, which cannot be split. That is to say, only after the completion of a job at a certain stage, it can be released to the subsequent stage. With the diversification of customer demand and the increasingly fierce global competition, the production mode of multiple varieties and small batch has occupied the dominant position in the manufacturing system. In this mode, the lot streaming technique is widely used, which was first introduced by Reiter in 1966 and systematically described by Lundrigan in 1986. As an effective technology to improve the efficiency of the manufacturing system, lot streaming has gradually attracted the researchers' attention (Chang and Chiu, 2005; Wang et al., 2019; Novas, 2019). Lot streaming divides a larger lot, containing a number of identical items, into several smaller sublots. When a sublot is processed completely at a certain process, it can be transferred to the downstream stage immediately. In this way, a lot can be processed in parallel at different stages at the same time. Thus, it has the advantages of shortening the machine idle time, reducing work-in-process inventory and shortening the production cycle.

A relatively few studies on the HFSP with lot streaming can be found when compared to that on traditional production scheduling. Aiming to solve the small-sized problems, some papers considered the HFSP with lot streaming (Kim et al., 1997; Tsubone et al., 1996; Zhang et al., 2005). However, large-sized problems are often addressed in the real-world scenarios. To the best of our knowledge, there are two published papers that considered lot streaming in the hybrid flowshop with large-sized problems. Defersha and Chen (2012) introduced a parallel genetic algorithm for hybrid flexible flowshop lot streaming problem, aiming to minimise the makespan of the schedule. Mohsen and Iraj (2014) applied the genetic algorithm and the simulated annealing to solve the multi-job lot streaming in a HFSP to minimise 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. Zhang et al. (2017) addressed the HFSP with lot streaming, in which the sublots from different lots cannot be intermingled, and they proposed an effective migrating birds optimisation (MBO) algorithm to solve the problem. In their computational study, the effectiveness of the proposed algorithm is demonstrated by comparing with several state-of-the-art algorithms

in the literature. However, in their problem, the sublot sizes of the sublots from the same lot are the same, so the obtained scheduling might not be optimal and not satisfy some real-world scenarios.

Given above, in this paper, a HFSP with consistent sublots (HFSP_CS) is studied, in which not only the lot sequencing is considered but also the lot splitting. And a mixed-integer linear programming (MILP) model is developed. This considered problem is much more complex than regular HFSP, so it is apparently NP-hard. To this end, an emerging meta-heuristic, namely MBO, is introduced to solve the addressed problem, which was recently presented by Duman et al. (2012) for the quadratic assignment problem. Since then, the MBO has been successfully applied in other fields, such as scheduling problem (Zhang et al., 2017; Tongur and Ülker, 2014), closed loop layout (Niroomand et al., 2015), travelling salesman problem (Tongur and Ülker, 2016), sea freight transportation (Ruiz et al., 2015) and machine-part cell formation problem (Soto et al., 2015). In the EMBO framework proposed in our previous study (Zhang et al., 2017), we propose an effective version of the MBO by integrating the improved variable neighbourhood descent (VND) strategy, proposing some heuristics and modifying some operators with the consideration of the problem-specific characteristics. 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 organised as follows. In Section 2, the HFSP_CS is detailed and a MILP model is presented. In Section 3, the basic MBO algorithm is introduced. In Section 4, the proposed VMBO is developed. The experimental results and analysis are reported in Section 5. Finally, conclusions are given in Section 6.

2 Problem description

HFSP_CS is described as below. There exist n lots that need to be processed continuously on a hybrid flowshop configuration consisting of m stages. Each stage k contains $\sigma_k \geq 1$ identical machines, and at least one stage contains more than one machine. Each job cannot skip any stage and can arbitrarily select one machine at any stage. The lot size of job j ($j = 1, 2, \ldots, n$) is T_j, i.e., job j contains T_j identical items. The job j can be split into $l_j \geq 1$ sublots. And the sublot sizes of job j can be different, but must belongs to a reasonable range $[S_j^{\min}, S_j^{\max}]$, in which $S_j^{\min}$ is the minimum sublot size and $S_j^{\max}$ is the maximum sublot size. Any sublot can be immediately transferred to the next stage after its completion at the previous stage. In HFSP_CS, the processing time of each sublot at a certain stage is the product of the processing time of one item and the sublot size. With the objective of minimising the total flowtime, HFSP_CS needs to solve the integrated optimisation of the lot sequence and lot splitting. The lot sequence needs to determine the processing order of the processed lot, while the lot splitting needs to determine the sublot size of each sublot. The objective selected in this paper is the total flowtime, which can reduce the production cycle, reduce the work-in-process inventory and improve the customer service (Yimer and Demirli, 2009).

Several assumptions or constraints are summarised as follows.

- All the sublots of a lot should be processed continuously on the same machine at a certain stage. That is, the sublots that belong to different lots cannot be intermingled.

- The processing of any sublot at any stage cannot be interrupted, and no preemption is allowed.
- All machines are available at time zero and all lots are released at time zero.
- At any time, a machine can only process one sublot and one sublot can only be processed on one machine.
- The machine is allowed to be idle, each sublot can wait between two adjacent stages. Note that the buffer has no capacity limit.
- Setup and transport times are included in the processing time.

The notations used in this paper are summarised as follows.

2.1 Parameters and sets

M The set of lots.

J The set of stages.

m The number of stages, and $m = |M|$.

n The number of jobs, and $n = |J|$.

k The stage index, and $k \in \{1, 2, \ldots, m\}$.

j The lot index, and $j \in \{1, 2, \ldots, n\}$.

σ_k The number of stages at stage k.

i The machine index.

T_j The lot size of lot j.

l_j The quantity of the sublots that belong to lot j.

$S_j^{\min}$ The minimum sublot size of sublot j.

$S_j^{\max}$ The maximum sublot size of sublot j.

e The sublot index.

$P_{k,j}$ The processing time of the item of lot j at stage j.

C_j The maximum completion time of lot j.

Q A very large positive number.

2.2 Decision variables

$ST_{k,j,e}$ The starting time of the sublot e of lot j at stage k.

$CT_{k,j,e}$ The completion time of sublot e of lot j at stage k.

$LS_{j,e}$ The sublot size of sublot e of lot j.

$D_{k,j,i}$ A binary variable that equals to 1 when the lot j is assigned to machine i at stage k and 0 otherwise.

$Y_{k,j,j'}$ A binary variable that equals to 1 when lot j is processed before lot j' on the same machine at stage k and 0 otherwise.

With the notations presented above, the MILP model is formulated as follows:

- Objective:

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

- Constraints:

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

$$\sum_{e=1}^{l_j} LS_{j,e} = T_j \qquad \forall j \in J \tag{3}$$

$$ST_{k,j,e} \geq 0 \qquad \forall j \in J, k \in M, e = [1, \ldots, l_j] \tag{4}$$

$$CT_{k,j,e} - ST_{k,j,e} = p_{k,j} \times LS_{j,e} \qquad \forall j \in J, k \in M, e = [1, \ldots, l_j] \tag{5}$$

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

$$ST_{k,j,e+1} - CT_{k,j,e} \geq 0 \qquad \forall j \in J, k \in M, e = [1, \ldots, l_j] \tag{7}$$

$$Y_{k,j,j'} + Y_{k,j',j} \leq 1 \qquad \forall j, j' \in J, k \in M \tag{8}$$

$$Y_{k,j,j'} + Y_{k,j',j} \leq D_{k,j,i} + D_{k,j',i} \qquad \forall j, j' \in J, k \in M, i \in [1, \ldots, \sigma_k] \tag{9}$$

$$D_{k,j,i} + D_{k,j',i} - 1 \leq Y_{k,j,j'} + Y_{k,j',j}, \qquad \forall j, j' \in J, k \in M, i \in [1, \ldots, \sigma_k] \tag{10}$$

$$\begin{aligned} &ST_{k,j,1} - CT_{k,j',l_{j'}} + Q \\ &\times \left(3 - Y_{k,j',j} - D_{k,j,i} - D_{k,j',i}\right) \geq 0 \end{aligned} \qquad \forall j, j' \in J, k \in M, i \in \{1, 2, \ldots, \sigma_k\} \tag{11}$$

$$D_{k,j,i} \in \{0, 1\} \qquad \forall j \in J, k \in M, i \in \{1, 2, \ldots, \sigma_k\} \tag{12}$$

$$Y_{k,j,j'} \in \{0, 1\} \qquad \forall j, j' \in J, k \in M \tag{13}$$

$$LS_{j,e} - S_j^{\min} \geq 0 \qquad \forall j \in J, e = [1, \ldots, l_j] \tag{14}$$

$$LS_{j,e} - S_j^{\max} \leq 0 \qquad \forall j \in J, e = [1, \ldots, l_j] \tag{15}$$

Objective (1) states that the optimisation goal of the addressed problem is the total flow time. Constraint (2) determines that any lot must pass through all stages and can only be processed by only one machine at any stage. Constraint (3) determines that for any lot, the sum of their sublot sizes must equal to the lot size. Constraint (4) determines that the

starting time of any sublot at any stage should be positive. Constraint (5) defines that the processing of any sublot at any stage cannot be interrupted. Constraints (6) determines that the processing of any sublot at a certain stage cannot start until it is completed at the previous stage. Constraint (7) determines that at any stage, the sublot can be processed only after the completion of the its previous sublot from the same lot. Constraints (8)–(11) together define the machine processing capability, that is, at any stage the first sublot of any lot can be processed only after the completion of the last sublot of the previous adjacent lot assigned to the same machine. Constraint (12) and Constraint (13) define the value ranges of the decision variables $D_{k,j,i}$ and $Y_{k,j,j'}$ respectively. Constraint (14) and Constraint (15) define the value rages of the decision variable $LS_{j,e}$.

Table 1 The example data with the first lot splitting plan

Lot index	*Lot size*	*Sublot quantity*	*Sublot size*				*Item processing time*		
			1	*2*	*3*	*4*	*1 stage*	*2 stage*	*3 stage*
1	35	3	10	10	15	--	2	2	2
2	40	3	10	20	10	--	2	4	3
3	60	4	20	10	10	20	2	3	3
4	80	4	30	20	10	20	1	3	2

Table 2 The example data with the second lot splitting plan

Lot index	*Lot size*	*Sublot quantity*	*Sublot size*				*Item processing time*		
			1	*2*	*3*	*4*	*1 stage*	*2 stage*	*3 stage*
1	35	3	20	10	5	--	2	2	2
2	40	3	30	5	5	--	2	4	3
3	60	4	30	5	20	5	2	3	3
4	80	4	40	10	20	10	1	3	2

Figure 1 The Gantt chart with the first lot splitting plan

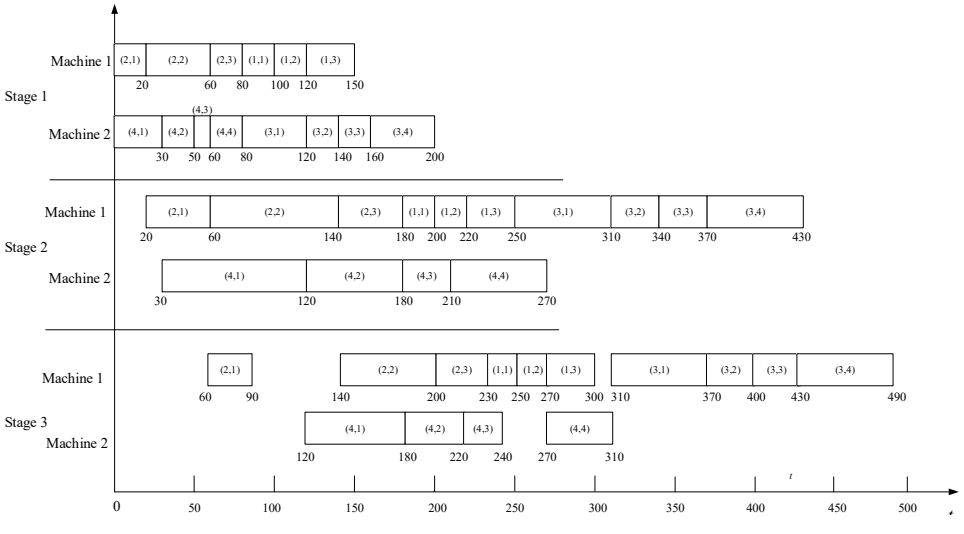
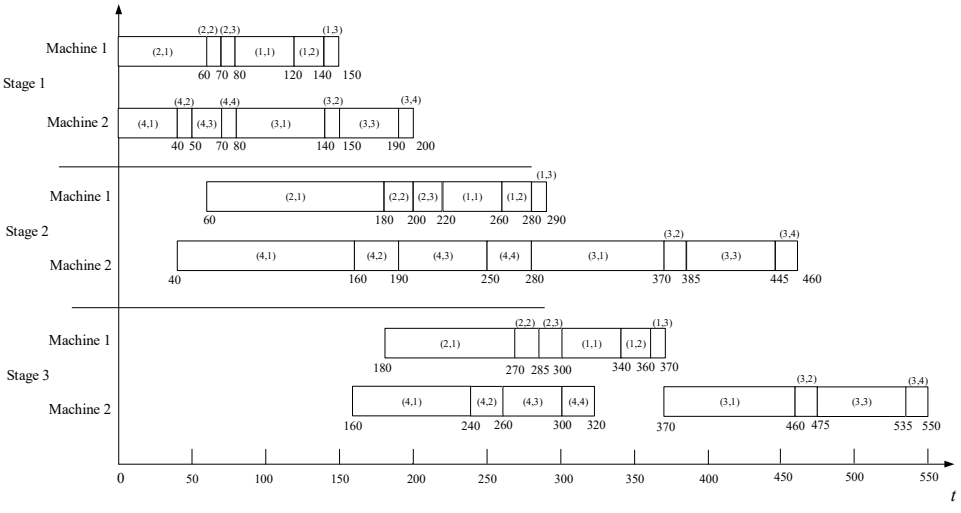


Figure 2 The Gantt chart with the second lot splitting plan



In order to illustrate the problem, consider a simple example with two lot splitting plans. This example has four lots and three phases, and two identical parallel machines on each phase. Tables 1 and 2, respectively show the example data with two different job splitting plans, including the lot number, lot size, sublot quantity of each lot, sublot sizes of each lot and item processing time of each lot at each stage. It should be noted that the lot splitting in Table 1 is relatively balanced, while the plan in Table 2 is quite different. The total flow time of the scheduling shown in Figure 1 is 1,330, while that in Figure 2 is 1,540. As can be seen from Figure 2, the unbalanced lot splitting can result in the machine staying in the idle stage for a long time or the sublot waiting between stages for a long time, thereby reducing the production efficiency and increasing the total flow time. Thus, tt can be seen that the lot splitting has an important impact on the scheduling results.

3 The basic MBO algorithm

3.1 Algorithm initialisation

Algorithmic parameters and population should be initialised. 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 neighbours 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 (Duman et al., 2012).

Another aspect is to initialise 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 neighbourhood, and k neighbours are generated in a predefined rule. Then, if the best neighbour among them has a better fitness, the leader is replaced by it; otherwise, the leader remains unchanged. And excluding the best one, the remaining $k - 1$ neighbours are collected to create two shared neighbour 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 minimisation optimisation problems, these $k - 1$ neighbours 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 $k - s$ neighbours and attempt to improve themselves by evaluating not only these k-s neighbours 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.

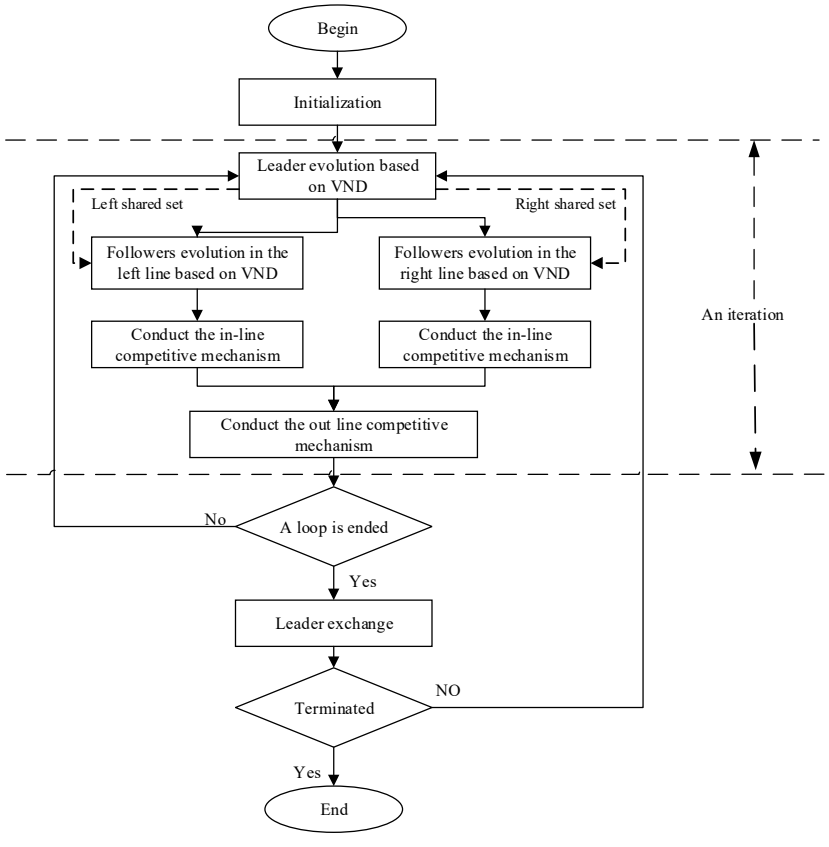
3.4 Leader change

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 $P_L(P_R)$. Thus, every solution in the population can have the chance to become the leader.

4 The proposed VMBO algorithm

In this section, to solve the HFSP_CS, the VND is embedded into the MBO algorithm, resulting in a new version VMBO. The VMBO algorithm is based on an effective framework, namely EMBO, developed in our previous study, the effectiveness of which in solving the HFSP with lot streaming has been evaluated. In view of the problem characteristics, a two-layer encoding and a population initialisation mechanism are proposed. And a heuristic-based decoding mechanism and an improvement strategy are developed. Moreover, five neighbourhood structures are developed and the VND is integrated into EMBO to fully explore the solution space in a systematic way. We first give the whole algorithm framework displayed in Figure 3.

Figure 3 The VMBO framework



4.1 Solution encoding and population initialisation

It is significant to ensure the solution encoding, which should bear comprehensive solution space information. As described above, to solve ILS_HFSP, two sub-problems should be solved simultaneously, i.e., lot sequencing and lot splitting. To this end, a two-layer encoding mechanism is adopted. The first layer is the *n*-dimensional lot sequence $\boldsymbol{\pi}_n = \{\pi_1, \ldots, \pi_i, \ldots, \pi_n\}$ corresponding to the lot scheduling order at the first stage, where π_i represents the lot index and n represents the number of lots. The second layer is the n-dimensional lot-sublot size matrix $v_j = \{LS_{j,1}, \ldots, LS_{j,e}, \ldots, LS_{j,l_i}\}$ corresponding to the lot splitting, where $LS_{j,e}$ is the sublot size of the sublot e from lot j. Take the schedule shown in Figure 1 as an example, its solution encoding can be represented as $\langle \boldsymbol{\pi}_4, \mathbf{v}_4 \rangle$. The first layer is $\boldsymbol{\pi}_4 = \{2, 4, 1, 3\}$, which indicates that the scheduling order at the first stage is lot 2, lot 4, lot 1 and lot 3.The second layer is $\mathbf{v}_4$, as shown in Figure 4, which indicates that lot 1 is divided into three sublots and their sublot sizes are 10, 10 and 15 respectively, and lot 2, lot 3 and lot 4 are the same with lot 1.

Figure 4 The second layer representation in the example

Lot 1	10	10	15	
Lot 2	10	20	10	
Lot 3	20	10	10	20
Lot 4	30	20	10	20

With regards to the population initialisation, in order to ensure the global exploration in the solution space, *ps* solutions are generated randomly, and they are randomly placed on the V-shaped population structure. Regarding the $\mathbf{v}_n$ initialisation, to obtain the feasible solution in a random way, the following progress is developed for lot j ($j = 1, \ldots, n$).

Step 1 Lot splitting in an even way.

Step 1.1 Firstly, the total lot is evenly distributed to several sublots. The sublot size of each sublot is $LS_{j,e} = \left\lfloor T_j / l_j \right\rfloor$ where $\lfloor\ \rfloor$ returns a nearest integer that is smaller than T_j / l_j.

Step 1.2 Then the remaining size r_j is obtained, i.e., $r_j = T_j - \sum_{e=1}^{l_j} LS_{j,e}$. Three case are arisen.

Case 1 If $r_j = 0$, step 1 is terminated;

Case 2 If $r_j < l_j$, then add r_j to any sublot randomly and terminate step 1.

Case 3 If $r_j > l_j$, then repeat steps 1.1–1.2.

Step 2 Randomness. Since it is evenly distributed through step 1, the sublot size of each sublot is in the range $[LS_j^{\min}, LS_j^{\max}]$. Conduct the following steps for the sublots $\left(e = 1, \ldots, \left\lfloor l_j / 2 \right\rfloor\right)$ in a sequential way.

Step 2.1 Obtain a random integer within the range $[LS_j^{\min}, LS_{j,e}]$ randomly, then let $LS_{j,e} = LS_{j,e} + Random$ or $LS_{j,e} = LS_{j,e} - Random$ with a 50% probability.

Step 2.2 Let $LS_{j,l_j+1-e} = LS_{j,l_j+1-e} - Random$ or $LS_{j,e} = LS_{j,e} + Random$ for the sublot $l_j + 1 - e$ accordingly.

Step 3 Scramble the sublots. After the above steps, each sublot size can be determined. In order to further ensure the randomness, the sublot size sequence is shuffled and rearranged in a random way.

4.2 Solution decoding mechanism

In order to decoding the solution into a feasible schedule, two sub-problems need to be considered: the first one is lot sequence at each stage; the second one is the machine assignment for each lot at each stage. Since the optimisation goal of the problem is the total flow time, two heuristic rules are considered. According to the 'first-come-first-served', the lot completed earlier at a certain stage is given priority to be scheduled at the following stage. And the 'first idle' rule is used for machine assignment, that is the machine that has the earlier available time has the priority to be assigned. For the HFSP problem with the time-based objectives, the validity of these two rules has been proved (Zhang et al., 2018). Specially, regarding to the 'first-come-first-served' rule, two methodologies are proposed with the consideration of the lot splitting characteristics: one is the 'sublot preemption', i.e., the lot, the first sublot of which is completed earlier at the previous stage has the priority at the following stage; another one is the 'lot preemption', i.e., the lot, which is completed earlier at the previous stage has the priority to be processed at the following stage.

Using the 'first-come-first-served' rule, the sequence of the lots that arrive at a certain stage simultaneously cannot be determined. To tackle this problem, the 'shortest waiting rule' is employed to determine their sequence, the effectiveness of which is demonstrated in (). First, at stage k ($k > 1$), these lots are arranged in an ascending order according to the value obtained by equation (16). Second, if they have the same value again, the lot that has smaller $p_{k,j}$ is located at the front.

$$\delta_j = \begin{cases} p_{k-1,j} - p_{k,j}, & \text{if } p_{k-1,j} > p_{k,j} \\ 0, & \text{else} \end{cases} \tag{16}$$

In view of the above description, the decoding processed is detailed as below. At the first stage, i.e., $k = 1$, take π_i from $\boldsymbol{\pi}_n$ in turn. First, find the earliest available machine, and then assign π_i to the machine; Regarding the lot splitting, the processing time of each sublot $(e = 1, \ldots, I_{\pi_i'})$ shall be calculated according to v_π, and the sublots can be scheduled on the selected machine in accordance with the production constraints. Then, the starting time and completion time of each sublot can be determined, and the idle time of the selected machine also can be updated according to the completion time of the last sublot. At the following stages, i.e., $k = \{2, \ldots, m-1\}$ first obtain a new lot sequence $\boldsymbol{\pi}_n' = (\pi_1', \pi_2', \ldots, \pi_n')$ by using above described methods, then take π_i' from $\boldsymbol{\pi}_n'$ in turn. Find the earliest available machine, and then assign π_i' to the machine. Regarding the lot splitting, the processing time of each sublot $(e = 1, \ldots, I_{\pi_i'})$ shall be calculated according to $v_{\pi_i'}$, and the sublots can be scheduled on the selected machine in accordance with the production constraints. Then, the starting time and completion time of each sublot can be determined, and the idle time of the selected machine also can be updated according to the completion time of the last sublot. At the last stage, i.e., $k = m$, the right shift and insert mechanism described below is executed to further improve the solution quality.

Figure 5 The Gantt chart before executing the right-shift and insert mechanisms (see online version for colours)

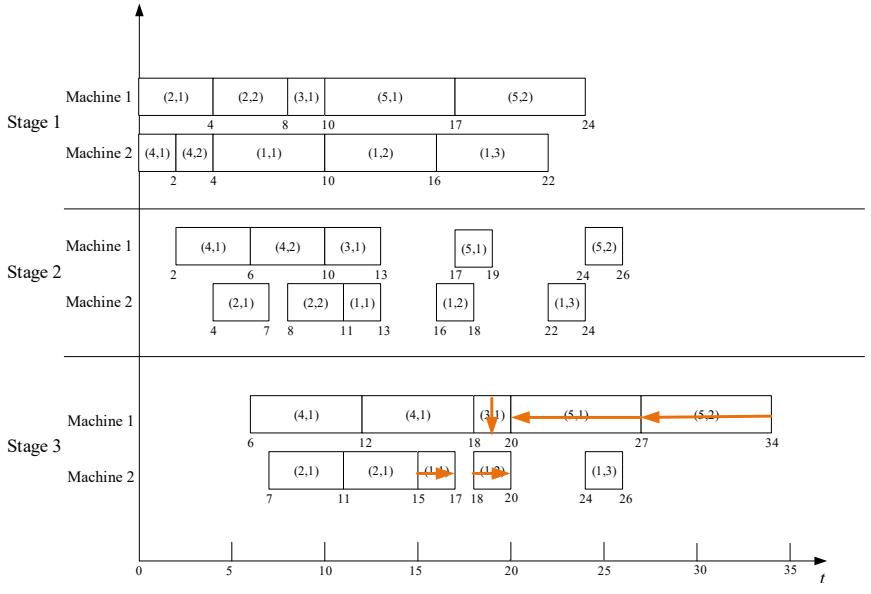
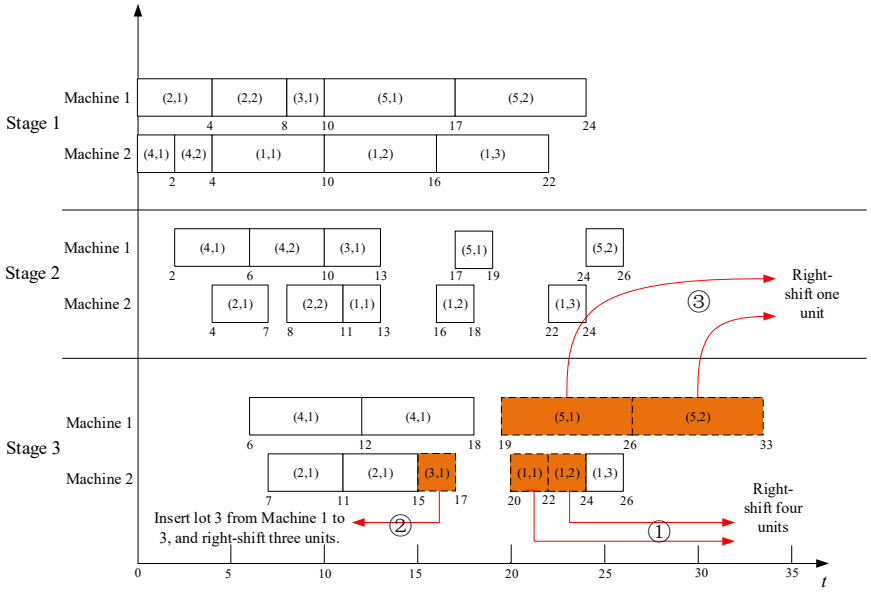


Figure 6 The Gantt chart after executing the right-shift and insert mechanism (see online version for colours)



In HFSP_CS, the sublots of different lots cannot be intermingled. Thus, if the processing times of the sublots are not balanced among stages, a large amount of machine idle time can be generated, resulting in a low machine utilisation rate. Thus, the following right shift and insertion mechanism are proposed to further improve the solution quality. In the

right-shift mechanism, after a lot is scheduled at the last stage, on the premise that the completion time of the lot is not affected (i.e., the completion time of the last sublot remains unchanged), we right-shift its sublots from the last second one to the first one to reduce the idle time occupied by this lot. In the insert mechanism, if there exists a machine, among its idle time, it can process a unscheduled lot, we insert the unscheduled lot into the idle time of this machine. To illustrate the two mechanisms, Figure 4 shows a Gannt char that has not been conducted the right-shift and insert mechanism, and also shows the operators that can be conducted in using the right-shift and insert mechanism, and Figure 6 shows the improved scheduling Gantt chart after using the right-shift and insert mechanism. The detailed process can be described as follows:

1 After lot 1 completes its scheduling and executes the right shift strategy, and them machine 2 will have five idle time units.

2 when executing the lot 3 scheduling, machine 1 should be selected using the 'first idle' rule. Applying the insertion operation, lot 3 can be inserted into the idle time of machine 2. Therefore, it is moved three units to the left when comparing with the original scheduling.

3 Lot 5 is moved left by one unit due to the movement of lot 3. Therefore, after the right-shift and insert mechanism, the total flow time is reduced by four units.

4.3 VND strategy

According to the encoding mechanism of HFSP_CS, five neighbourhood structures are designed in this section to enable the algorithm to explore the solution space thoroughly. For the lot sequence, two neighbourhood structures are widely used in the literature: insert and swap structures. For the lot splitting matrix, an size adjustment structure is designed. Moreover, two mixed neighbourhood structures are also designed. The five neighbourhood structures are as follows.

1 Insert structure: For the lot sequence, a lot is randomly selected and inserted into another randomly selected location. See Figure 7 for example.

2 Swap structure: For the lot sequence, two lots are randomly selected and their positions swapped. See Figure 7 for example.

3 Size adjustment structure: For the lot splitting matrix, randomly select a lot with more than two sublots and randomly select two sublots. Within the range [1, 5], randomly select an adjustment size and randomly select a sublot and transfer its sublot size to another one according to the adjustment size. If the maximum and minimum size constraint is not satisfied after adjustment, it will remain unchanged. See Figure 8 for example.

4 Mixed structure 1: First conduct the insert operator, then conduct the size adjustment operator.

5 Mixed structure 2: First conduct the swap operator, then conduct the size adjustment operator.

Figure 7 Insert and swap operators (see online version for colours)

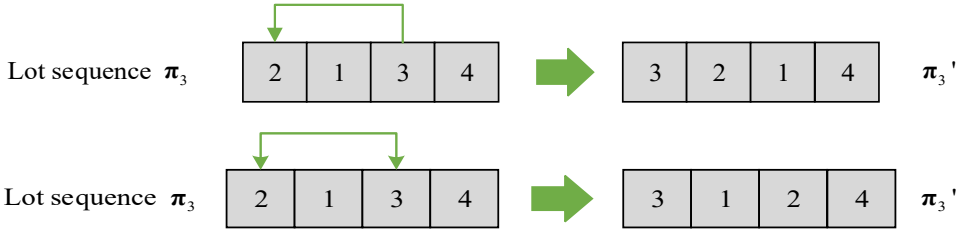
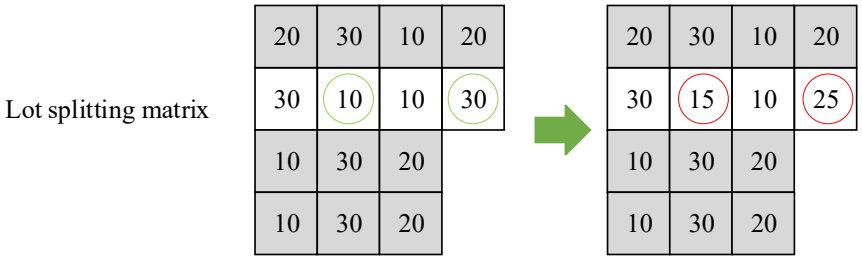


Figure 8 Size adjustment operator (see online version for colours)



Considering the limitation of the local search, in order to effectively utilise the five different neighbourhood structures mentioned above, the VND search strategy is introduced (Hansen et al., 2010). The core idea of the VND strategy is to promote the cooperation among various neighbourhood structures by systematically switching the neighbourhood structures, so as to realise the full exploration in solution space. In basic VND, different neighbourhood structures form a neighbourhood structure set. With the current neighbourhood structure, if the solution is improved, the current neighbourhood structure is to be switched to the first one in the set. If it is not promoted, the current neighbourhood structure is to be switched to the next one in the set. The disadvantages of this strategy can be reflected in the following two aspects. On the one hand, when the current solution produces a good neighbouring solution by using its current neighbourhood structure, the current neighbourhood structure may still have the potential for further exploration. Thus, the structure switching will affect the further use of the current structure, which is not conducive to the local search ability of the algorithm; on the other hand, when the current solution used its current neighbourhood structure to produce a good solution, which may not be updated due to the existence of sharing and benefit mechanism. In this case, the current neighbourhood structure cannot be correctly judged.

In order to tackle the above problem and make better use of the current neighbourhood structure, a control parameter g is introduced to control the transformation of neighbourhood structure (Zhang et al., 2018), which is defined as the maximum consecutive evaluation times of the current solution using its current neighbourhood structure. The neighbourhood structure switching strategy is improved as follows.

1 If the consecutive evaluation times reach at g and the current solution is updated in this process, the current neighbourhood structure is converted to the first one in the set.

2 If the consecutive evaluation times reach at g and the current solution is not updated during this process, the current neighbourhood structure is converted to the next one in the set.

Because the evolutionary processes of the leader and followers are different, the evolutionary process of the leader and followers by applying the VND strategy are respectively displayed in following two sections.

4.4 Dynamic acceptance criterion

In MBO algorithm, the followers evaluate both the neighbouring solutions generated by itself and the solutions in the shared set. Among them, if the best solution is better than the current one, no matter the best solution comes from itself or from the shared set, it will replace the current one. This sharing and benefit process is efficient when the population diversity is good, and can significantly increase the convergence speed. However, with the evolutionary process proceeding, the solutions in the population gradually converge, and this will leads to the low efficiency of the sharing process. For this reason, the dynamic acceptance criterion (Zhang et al., 2017) is also employed here.

$$p_{Iter} = \begin{cases} p_o - Iter \times \dfrac{p_o - p_f}{L} & \text{If the shared soltuion performs best} \\ 0 & \text{Otherwise} \end{cases} \tag{17}$$

In the dynamic acceptance criterion, if the neighbouring solution generated by itself is the best one, the current solution will be updated. And if the solution from the shared set is the best one, the current solution will be updated with a probability p. The probability p changes linearly with the number of iterations increase, as shown in equation (17). P_0 is the upper limit, and P_f is the lower limit. $Iter$ is the current iteration number, P_{Iter} is the probability at the current iteration, and P_{Iter} is the constant value used to define the step size. In this way, in the early stage of the algorithm, the population diversity is good and is large, which is conducive to the execution of the sharing process. In the late stage of the algorithm, the population diversity is reduced and P_{Iter} is small, which is beneficial to the exploration of each individual.

4.5 Leader evolution

Assume the leader solution is $\mathbf{X}_{Leader}$, N_{Leader} ($N_{Leader} \in [1, 5]$) is the neighbourhood structure index, and R_{Leader} records the consecutive evaluation times of the current neighbourhood structure. In an iteration, the leader evolutionary process is shown in Algorithm 1, where $Neighbourhood(\mathbf{X}_{Leader}, N_{Leader})$ represents that the current leader solution uses the current neighbourhood structure to generate a neighbouring solution, and store it in the set δ. The leading solution will be updated according to the best solution in the set. If it is better than the current solution, the current one will be replaced. Otherwise, the current solution remains unchanged. After adopting the improved VND strategy, if the consecutive evaluation times of the current leader solution using the current neighbourhood structure reach at g, then judge whether the leading solution has been updated in this process. If so, the current neighbourhood structure will be turned into the first one in the set. Otherwise the current neighbourhood structure will be switched to the next one in the set.

Algorithm 1 Leader evolution

Input: $\mathbf{X}_{Leader}$, N_{Leader} and R_{Leader}
Output: Updated $\mathbf{X}_{Leader}$, N_{Leader}, R_{Leader} and two shared solutions set.
For i = 1 to k **do**
    $neighbour_i \leftarrow Neighbourhood(\mathbf{X}_{Leader}, N_{Leader})$
    Put $neighbour_i$ into the set δ.
Endfor
Find the best solution $neighbour_{best}$ in the δ.
If $f(neighbour_{best}) < f(\mathbf{X}_{Leader})$ **then**
    $\mathbf{X}_{Leader} \leftarrow neighbour_{best}$; $R_{Leader} \leftarrow R_{Leader} + 1$; $impor \leftarrow true$;
Else
    $\mathbf{X}_{Leader} \leftarrow neighbour_{best}$; $R_{Leader} \leftarrow R_{Leader} + 1$;
Endif
If $R_{Leader} = g$ **then**
    If $impor = true$ **then**
        $N_{Leader} \leftarrow 0$; $impro \leftarrow false$; $R_{Leader} \leftarrow 0$;
    Else
        $N_{Leader} \leftarrow N_{Leader} + 1$; $impro \leftarrow false$; $R_{Leader} \leftarrow 0$;
    Endif
    If $N_{Leader} > 5$ then
        $N_{Leader} \leftarrow 1$;
    Endif
Endif
Use the solutions in δ except $neighbour_{best}$ to compose the two shared sets.

4.6 Followers evolution

Regarding the followers evolution in the left and right lines, the evolutionary process is shown in Algorithm 2 at each iteration, where $\mathbf{X}_j^L$ and $\mathbf{X}_j^R$ respectively represent the j^{th} individual in the left and right lines, and N_j^L and N_j^R respectively represent the neighbourhood structure for the j^{th} individual, and R_j^L and R_j^R are respectively the consecutive evaluation times of the jth individual using the current neighbourhood structure, and $impro_j^L$ and $impro_j^R$ indicate whether the j^{th} individual is updated successfully in the process. The difference between leader evolution and followers evolution can be reflected in the following two aspects: one lies in the dynamic acceptance criterion; another one lies in that the followers evaluate both the neighbouring solutions generated by itself and solutions from the shared set. Therefore, in order to evaluate the current neighbourhood structure more effectively, it is necessary to determine whether the promising solutions are generated by themselves or come from the shared set. If the promising solutions come from the shared set, the corresponding $impro_j^L$ or $impro_j^R$ cannot be reset to true if the current solution is updated successfully.

Algorithm 2 Followers evolution

Input: P_L, P_R, Λ_L and Λ_R

Output: Updated P_L and P_R

For $j = 1$ to $|P_L|$ **do**

  For $i = 1$ to k-s **do**

    $neighbour_i \leftarrow Neighbourhood(\mathbf{X}_j^L, N_j^L)$.

    Put $neighbour_i$ into δ.

  Endfor

  Find the best solution $neighbour_{best}$ in δ and Λ_L.

  If $neighbour_{best}$ comes from δ **then**

    If $f(neighbour_i) < f(\mathbf{X}_j^L)$ **then**

      $\mathbf{X}_j^L \leftarrow neighbour_{best}$; $R_j^L \leftarrow R_j^L + 1$; $impro_j^L \leftarrow true$.

    Else

      $\mathbf{X}_j^L \leftarrow \mathbf{X}_j^L$; $R_j^L \leftarrow R_j^L + 1$.

    Endif

  Else if $neighbour_{best}$ comes from Λ_L

    If $f(neighbour_{best}) < f(\mathbf{X}_j^L)$ **then**

      Update $\mathbf{X}_j^L$ using the dynamic acceptance criterion; $R_j^L \leftarrow R_j^L + 1$;

    Else

      $\mathbf{X}_j^L \leftarrow \mathbf{X}_j^L$; $R_j^L \leftarrow R_j^L + 1$;

      Combine δ and Λ_L into δ, and Λ_L clear Λ_L and use δ to reset Λ_L.

  Endif

  If $R_j^L = g$ **then**

    If $impro_j^L = true$ **then**

      $N_j^L \leftarrow 1$; $impro_j^L \leftarrow false$; $R_j^L \leftarrow 0$;

    Else

      $N_j^L \leftarrow N_j^L + 1$; $impro_j^L \leftarrow false$; $R_L^j \leftarrow 0$;

    Endif

  If $N_j^L > 5$ **then**

    $N_j^L \leftarrow 1$;

  Endif

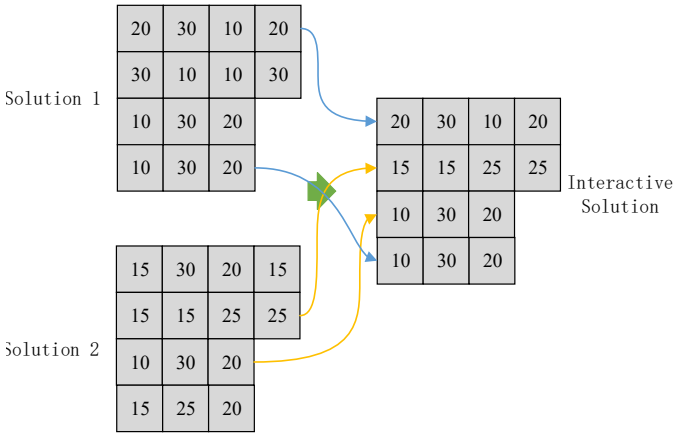
EndFor

Conduct 1–29 lines for the followers in P_R

4.7 Competitive and scout mechanisms

In the EMBO framework, the competitive mechanism is used to promote the cooperation of the solutions in the two lines, including the in-line competitive and out-line competitive mechanisms. For the lot sequence, we conduct the same process as described in Zhang et al. (2017). But for the lot splitting matrix, the cooperation should be designed according to the problem characteristic. Since the constraint of the total lot size, if the traditional crossover operation is still adopted, the unfeasible solution will be produced. In order to effectively utilise the information in the two solutions and produce an effective feasible solution, the matrix selection operator shown in Figure 9 is adopted. Moreover, the size adjustment operator mentioned above is used to mutate the solution. For the matrix selection operator, each splitting information for each lot is selected from the two solutions with a probability of 50%.

Figure 9 The matrix selection operator (see online version for colours)



In the EMBO framework, the scout mechanism is used to help the algorithm jump out of the local optima. In this study, for the lot sequence, the Glover operator is also adopted. But for the lot spitting matrix, since it contains no sequencing information, so the Glover operator is no longer applicable. Thus, here we present a perturbation operator, which can not only guarantee a large perturbation, but can also inherit some information from the evolutionary process. The detailed process is as below.

Step 1 For the lot with only one sublot, no action is taken. For the lot with more than two sublots, conduct the following three steps on the sublots ($e = 1, \ldots, l_j$) in turn.

- Step 1.1 Generate a value *random* within the range $[S_j^{\min}, LS_{j,e}]$ in a random way. Then select another sublot $f(f \neq e)$, conduct step 1.2.
- Step 1.2 Add or minus *random* to the sublot size of sublot e with the probability of 50%. Correspondingly, minus or add *random* to the sublot size of sublot *f*.
- Step 1.3 For the sublots *f* and *e*, if they satisfy the size constraint, they remain unchanged; otherwise, they return to their original state.

Step 2 For the lot with only one sublot, no action is taken. For the lot with more than two sublots, randomly shuffle their size sequence.

In the EMBO framework, the local search method is used to improve the quality of the solution generated in the scout phase. For HFSP_CS, since the two sub-problems are directly coupled, so after conducting the insertion operator on the lot sequence, if the solution is improved, we conduct the size adjustment operator on the lot splitting matrix based on the same process. The whole process of the local search is displayed in Algorithm 3.

Algorithm 3 The local search algorithm

```
Input: the current solution X
Output: the improved solution X
bool impro ← true;
While impro = true do
    impro ← false;
    For i = 1 to n do
        Conduct the insert operator on X to obtain X′
        If f(X′) < f(X) then
            X ← X′;
            impro ← true;
            bool impro2 ← true;
            While impro2 = true; do
                impro2 ← false;
                For i = 1 to n do
                    Conduct the size adjustment operator on X to obtain X′;
                    If f(X′) < f(X) then
                        X ← X′;
                        impro2 ← true;
                        break;
                    Endif
                Endfor
            Endwhile
            break;
        Endif
    Endfor
Endwhile
```

5 Computational study

In this section, we discuss the computational experiments that are used to evaluate the proposed VMBO algorithm using the maximum CPU elapsed time $n \times m \times 60$

milliseconds as the stopping criterion. Because we cannot find the benchmarks for our considered problem in the literature, a set of benchmark instances is generated in a random way. By comparing with the CPLEX solver, 16 problem scales can be obtained where $n \in \{5, 18, 10, 12\}$ and $m \in \{3, 5, 8, 10\}$. For each problem scale, only one instance is generated randomly. By comparing with the other evolutionary algorithms, 20 problem scales can be obtained where $n \in \{20, 40, 60, 80, 100\}$ and $m = \{5, 10\}$. For each problem scale, ten instances are randomly generated. In these instances, the total lot size can be yield in the interval [100, 300] randomly. The processing time of the item of each lot at each stage can be obtained in integer interval [1, 10] randomly. The sublot quantity and the number of machines at each stage can be respectively obtained in the range [2, 6] and [1, 5] randomly. Moreover, $S_j^{\min} = \left\lceil T_j \middle/ l_j \times 2 \right\rceil$, $S_j^{\max} = T_j$. The proposed algorithm is coded in C++ and run on a 2.3 GHZ Intel Core i7 processor in a WIN 10 operation system. VMBO algorithm need to set seven parameters, including the *ps*, *k*, *s*, *t*, *r*, *L* and *g*. By using the Taguchi method, we set $k = 3$, $s = 1$, $ps = 81$, $t = 10$, $L = 1{,}500$, $r = 80$ and $g = 8$. For comparisons, the relative percentage increase (RPI) values over the obtained best result are calculated as the performance measure as follows.

$$RPI(i) = (c_i - c_{best}) \big/ c_{best} \times 100\% \tag{18}$$

5.1 Comparing with the CPLEX solver

In order to test the efficiency of VMBO algorithm, for the small-sized problems, we compare the VMBO with the IBM ILOG CPLEX solver (version 12.7.1). Regarding CPLEX solver, the time limit is set as 3,600 seconds. While for the VMBO, it is run 20 times independently on each instance under the termination $n \times m \times 60$. The results are given in Table 3. From Table 3, we can observe that CPLEX can obtain the optimal solutions on 5 × 3, 5 × 5, 5 × 8 and 8 × 3. With the number of stages and lots increasing, CPLEX cannot get the optimal solution with 3,600 seconds. By comparing with VMBO, CPLEX only can get better solutions on 5 × 10, 8 × 5, 8 × 8, 10 × 3. But for the other instances, VMBO performs better than CPLEX. And with the number of lots and stages getting bigger and bigger, the advantage of the proposed VMBO becomes more and more obvious. Meanwhile, we can observe that the running time of VMBO is much smaller than CPLEX. Thus, the efficiency of the proposed VMBO can be demonstrated.

5.2 Comparing with the other algorithms

To demonstrated the effectiveness of VMBO in solving large-sized problems, we compare it with several meta-heuristics, including MBO (Duman et al., 2012), IMBO (Soto et al., 2015), MMBO (Niroomand et al., 2015), GA (Schiavinotto and Stützle, 2007), GAR (Ruiz and Maroto, 2006), DPSO (Tseng and Liao, 2008) and DABC (Pan et al., 2014). Table 4 shows the comparative results including the RPI average and standard deviation values. Meanwhile, the ANOVA test is used to analyse the statistical analysis. From Table 2, we can observe that comparing with MBO (2.35), IMBO (1.45), MMBO (1.62), GA (2.30), GA_R (2.56), DPSO (2.95) and DABC (6.69), VMBO yields the smallest overall average value 0.75 and can obtain the small average value for all the problems. Moreover, for almost the problems, VMBO can yield the smallest values, and

this validate that VMBO has the good robustness. From Figure 10, it can be observed that there is no overlap between VMBO and the other algorithms. This concludes that the proposed algorithm performs significantly better than other algorithms. Above all, the effectiveness of the proposed algorithm can be demonstrated.

Table 3 The results obtained by CPLEX and VMBO on the small sized problems

Problem	*Minimum value*	*CPLEX*		*VMBO*	
		RPI	*Time*	*RPI*	*Time*
5 × 3	7,064	*0*	95 s	0.31	0.9 s
5 × 5	9,077	*0*	1,800 s	0.40	1.5 s
5 × 8	11,825	*0*	3,402 s	0.25	2.4 s
5 × 10	20,279	*0.17*	3,600 s	0.24	3 s
8 × 3	16,415	*0*	2,509 s	0.28	1.44 s
8 × 5	27,961	*0.20*	3,600 s	0.25	2.4 s
8 × 8	41,741	*0.27*	3,600 s	0.32	3.84 s
8 × 10	38,540	0.25	3,600 s	*0.24*	4.8 s
10 × 3	18,857	*0.28*	3,600 s	0.30	1.8 s
10 × 5	33,396	0.30	3,600 s	*0.26*	3 s
10 × 8	50,884	0.23	3,600 s	*0.15*	4.8 s
10 × 10	53,514	0.35	3,600 s	*0.29*	6 s
12 × 3	55,203	0.57	3,600 s	*0.39*	2.16 s
12 × 5	39,596	0.66	3,600 s	*0.34*	3.6 s
12 × 8	71,911	0.90	3,600 s	*0.30*	5.76 s
12 × 10	58,849	0.97	3,600 s	*0.29*	7.20 s

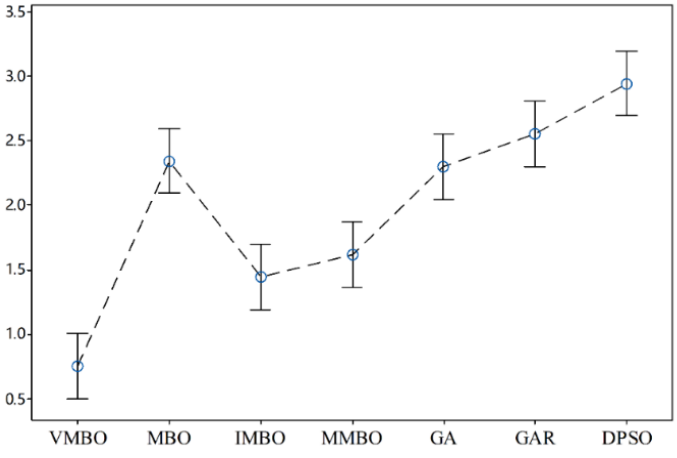
Table 4 The results obtained by CPLEX and VMBO on the small sized problems

Problem	*VMBO*	*MBO*	*IMBO*	*MMBO*	*GA*	*GAR*	*DPSO*	*DABC*
20 × 3	*0.44* (0.23)	1.56 (0.30)	1.49 (0.21)	1.49 (0.29)	1.28 (0.37)	1.69 (0.54)	2.25 (0.59)	3.20 (0.44)
20 × 5	*0.78* *(0.31)*	2.31 (0.55)	1.64 (0.60)	2.23 (0.54)	1.96 (0.62)	2.46 (0.71)	4.91 (1.60)	3.92 (0.69)
20 × 8	*0.69* *(0.23)*	1.76 (0.41)	1.65 (0.37)	1.97 (0.40)	1.22 (0.42)	2.01 (0.69)	3.31 (1.15)	2.96 (0.67)
20 × 10	*0.96* (0.43)	2.18 (0.45)	1.85 (*0.42*)	2.30 (0.54)	1.70 (0.58)	2.30 (0.65)	3.30 (1.19)	3.80 (0.69)
40 × 3	*0.52* (0.34)	1.71 (0.34)	1.10 (*0.29*)	1.34 (0.43)	1.85 (0.47)	2.12 (0.55)	2.80 (0.70)	6.40 (0.91)
40 × 5	*1.15* *(0.27)*	2.58 (0.42)	1.55 (0.38)	1.73 (0.46)	2.78 (0.47)	2.01 (0.72)	2.31 (0.71)	5.29 (0.96)
40 × 8	*0.77* *(0.29)*	1.76 (0.36)	1.18 (0.31)	1.63 (0.41)	1.55 (0.43)	2.18 (0.91)	2.68 (0.79)	5.67 (0.99)
40 × 10	*0.81* *(0.26)*	2.21 (0.34)	1.68 (0.40)	1.83 (0.47)	2.32 (0.52)	2.06 (0.68)	2.62 (0.58)	4.21 (0.75)

Table 4 The results obtained by CPLEX and VMBO on the small sized problems (continued)

Problem	*VMBO*	*MBO*	*IMBO*	*MMBO*	*GA*	*GAR*	*DPSO*	*DABC*
60 × 3	*0.46* *(0.25)*	2.57 (0.31)	1.56 (0.28)	1.46 (0.42)	2.67 (0.45)	2.95 (0.66)	2.51 (0.50)	8.46 (0.77)
60 × 5	*0.85* *(0.33)*	2.59 (0.40)	1.30 (0.35)	1.32 (0.62)	2.86 (0.56)	2.62 (0.95)	2.78 (0.79)	8.49 (0.99)
60 × 8	*0.70* *(0.26)*	2.27 (0.29)	1.33 (0.32)	1.40 (0.40)	2.36 (0.42)	2.15 (0.82)	1.77 (0.65)	4.69 (0.58)
60 × 10	*0.96* *(0.37)*	2.00 (0.62)	1.18 (0.38)	1.55 (0.49)	1.84 (0.61)	2.06 (1.03)	3.48 (0.85)	6.37 (0.88)
80 × 3	*0.41* *(0.25)*	2.75 (0.28)	1.38 (0.30)	1.59 (0.59)	2.93 (0.42)	3.40 (0.75)	3.17 (0.84)	10.19 (1.01)
80 × 5	*0.51* *(0.28)*	2.34 (0.29)	1.19 (0.39)	1.14 (0.52)	2.45 (0.40)	2.51 (0.92)	2.64 (0.65)	9.54 (0.96)
80 × 8	*0.94* (0.52)	2.95 (0.45)	1.51 (*0.39*)	1.54 (0.54)	3.17 (0.52)	3.14 (0.95)	2.72 (0.57)	7.49 (0.78)
80 × 10	*0.59* *(0.30)*	1.79 (0.32)	1.03 (0.32)	1.09 (0.41)	1.93 (0.34)	1.79 (0.54)	2.07 (0.55)	4.19 (0.63)
100 × 3	*0.76* *(0.31)*	3.97 (0.46)	2.16 (0.46)	3.03 (0.99)	3.58 (0.66)	5.31 (0.87)	4.68 (0.78)	15.50 (1.30)
100 × 5	*0.62* *(0.35)*	2.84 (0.42)	1.64 (0.41)	1.40 (0.52)	2.77 (0.39)	3.28 (0.56)	3.02 (0.49)	8.28 (0.82)
100 × 8	*1.14* *(0.22)*	2.33 (0.37)	1.24 (0.45)	1.48 (0.37)	2.38 (0.43)	2.39 (0.84)	3.00 (0.57)	7.25 (0.98)
100 × 10	*0.99* *(0.41)*	2.48 (0.50)	1.32 (0.43)	0.84 (0.44)	2.46 (0.46)	2.75 (0.95)	2.94 (0.66)	7.91 (1.04)
Avg	*0.75* *(0.31)*	2.35 (0.39)	1.45 (0.37)	1.62 (0.49)	2.30 (0.48)	2.56 (0.76)	2.95 (0.76)	6.69 (0.84)

Figure 10 95% confidence intervals (see online version for colours)



6 Conclusions

This paper studies a HFSP_CS. First, a MILP model is developed. And then the VMBO algorithm integrating the EMBO with VND is proposed to solve the problem. The main contributions of this paper is the following:

1 a MILP model is developed

2 put forward a two-layer encoding and decoding process

3 some heuristics considering the problem-specific characteristics are developed

4 the VND strategy is successfully integrated into the EMBO.

Finally, the effectiveness of the proposed VMBO is demonstrated by comparing with CPLEX solver and the other state-of-the-art meta-heuristics in the literature.

Regarding the HFSP with consistent sublots, our future research includes developing more effective encoding and decoding strategies or problem-specific characteristics to help improve the results and evaluating some other objectives (e.g., total flow time, total earliness and tardiness, machine utilisation). In addition, dynamic and uncertain events could be introduced into the HFSP with consistent sublots. Accordingly, some useful properties (e.g., dynamic and rescheduling strategies) should be derived. Regarding the algorithm, it could be interesting to investigate the performance of the proposed algorithm in other MOPs to further demonstrate its effectiveness.

Acknowledgements

This work was supported by National Natural Science Foundation of China (61773192, 61603169, 61803192), and the Open Project of Henan Key Laboratory of Intelligent Manufacturing of Mechanical Equipment, Zhengzhou University of Light Industry (IM201906).

References

Chang, J.H. and Chiu, H.N. (2005) 'A comprehensive review of lot streaming', *International Journal of Production Research*, Vol. 43, No. 8, pp.1515–1536.

Chen, J., Wang, M., Kong, X.T.R. et al. (2019) 'Manufacturing synchronization in a hybrid flowshop with dynamic order arrivals', *Journal of Intelligent Manufacturing*, Vol. 30, No. 7, pp.2659–2668.

Defersha, F.M. and Chen, M. (2012) 'Mathematical model and parallel genetic algorithm for hybrid flexible flowshop lot streaming problem', *The International Journal of Advanced Manufacturing Technology*, Vol. 62, Nos. 1–4, pp.249–265.

Duman, E., Uysal, M. and Alkaya, A.F. (2012) 'Migrating birds optimization: a new metaheuristic approach and its performance on quadratic assignment problem', *Information Sciences*, Vol. 217, No. 24, pp.65–77.

Hansen, P., Mladenović, N. and Pérez, J.A.M. (2010) 'Variable neighbourhood search: methods and applications', *Annals of Operations Research*, Vol. 175, No. 1, pp.367–407.

Kim, J., Kang, S. and Lee, S.M. (1997) 'Transfer batch scheduling for a two-stage flowshop with identical parallel machines at each stage', *Omega*, Vol. 25, No. 5, pp.547–555.

Lei, D. and Wang, T. (2019) 'Solving distributed two-stage hybrid flowshop scheduling using a shuffled frog-leaping algorithm with memeplex grouping', *Engineering Optimization*, DOI: 10.1080/0305215X.2019.1674295.

Lundrigan, R. (1986) 'What is this thing called OPT?', *Prod. Inventory Manage.*, Vol. 27, No. 2, pp.2–11.

Mohsen, N. and Iraj, M. (2014) '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*, Vol. 70, Nos. 1–4, pp.501–514.

Niroomand, S., Vencheh, A.H. and Sahin, R. (2015) 'Modified migrating birds optimization algorithm for closed loop layout with exact distances in flexible manufacturing systems', *Expert Systems with Applications*, Vol. 42, No. 19, pp.6586–6597.

Novas, J.M. (2019) 'Production scheduling and lot streaming at flexible job-shops environments using constraint programming', *Computers & Industrial Engineering*, Vol. 136, pp.252–264.

Öztop, H., Tasgetiren, M.F., Eliiyi, D.T. et al. (2019) 'Metaheuristic algorithms for the hybrid flowshop scheduling problem', *Computers & Operations Research*, Vol. 111, pp.177–196.

Pan, Q.K., Wang, L., Li, J.Q., and Duan, J.H. (2014) 'A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimisation', *Omega*, Vol. 45, pp.42–56.

Reiter, S. (1966) 'A system for managing job-shop production', *The Journal of Business*, Vol. 39, No. 3, pp.371–393.

Ruiz, E.L., Lzquierdo, C.E. and Armas, J.D. (2015) 'Migrating birds optimization for the seaside problems at maritime container terminals', *Journal of Applied Mathematics*, DOI: 10.1155/2015/781907.

Ruiz, R. and Maroto, C. (2006) 'A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility', *European Journal of Operational Research*, Vol. 169, No. 3, pp.781–800.

Ruiz, R. and Vázquez-Rodríguez, J.A. (2010) 'The hybrid flow shop scheduling problem', *European Journal of Operational Research*, Vol. 205, No. 1, pp.1–18.

Schiavinotto, T. and Stützle, T. (2007) 'A review of metrics on permutations for search landscape analysis', *Computers & Operations Research*, Vol. 34, No. 10, pp.3143–3153.

Soto, R., Crawford, B. and Almonacid, B. et al. (2015) *A Migrating Birds Optimization Algorithm for Machine-Part Cell Formation Problems*, Springer International Publishing, Cham.

Tongur, V. and Ülker, E. (2014) 'Migrating birds optimization for flow shop sequencing problem', *Journal of Computer & Communicating*, Vol. 2, No. 4, pp.142–147.

Tongur, V. and Ülker, E. (2016) 'The analysis of migrating birds optimization algorithm with neighborhood operator on travelling salesman problem', *Intelligent and Evolutionary Systems*, pp.227–237, Springer International Publishing, Cham, Switzerland.

Tseng, C. and Liao, C. (2008) 'A discrete particle swarm optimization for lot-streaming flowshop scheduling problem', *European Journal of Operational Research*, Vol. 191, No. 2, pp.360–373.

Tsubone, H., Ohba, M. and Uetake, T. (1996) 'The impact of lot sizing and sequencing on manufacturing performance in a two-stage hybrid flow shop', *International Journal of Production Research*, Vol. 34, No. 11, pp.3037–3053.

Wang, S., Kurz, M., Mason, S.J. and Rashidi, E. (2019) 'Two-stage hybrid flow shop batching and lot streaming with variable sublots and sequence-dependent setups', *International Journal of Production Research*, Vol. 57, No. 22, pp.6893–6907.

Yimer, A.D. and Demirli, K. (2009) 'Fuzzy scheduling of job orders in a two-stage flowshop with batch-processing machines', *International Journal of Approximate Reasoning*, Vol. 50, No. 1, pp.117–137.

Zhang, B., Pan, Q.K., Gao, L. et al. (2017) 'An effective modified migrating birds optimization for hybrid flowshop scheduling problem with lot streaming', *Applied Soft Computing*, Vol. 52, pp.14–27.

Zhang, B., Pan, Q.K., Gao, L. et al. (2018) 'A hybrid variable neighborhood search algorithm for the hot rolling batch scheduling problem in compact strip production', *Computers & Industrial Engineering*, Vol. 116, pp.22–36.

Zhang, B., Pan, Q.K., Gao, L. et al. (2019a) 'A multi-objective migrating birds optimization algorithm for the hybrid flowshop rescheduling problem', *Soft Computing*, Vol. 23, No. 17, pp.8101–8129.

Zhang, B., Pan, Q.K., Gao, L. et al. (2019b) 'A three-stage multiobjective approach based on decomposition for an energy-efficient hybrid flow shop scheduling problem', *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, DOI: 10.1109/TSMC.2019.2916088.

Zhang, W., Yin, C., Liu, J. et al. (2005) 'Multi-job lot streaming to minimize the mean completion time in m-1 hybrid flowshops', *International Journal of Production Economics*, Vol. 96, No. 2, pp.189–200.