Memetic Algorithm With Meta-Lamarckian Learning and Simplex Search for Distributed Flexible Assembly Permutation Flowshop Scheduling Problem

GUANGHUI ZHANG,1,2 KEYI XING,2 GUANGYUN ZHANG,3 AND ZHENXUE HE1

1 School of Information Science and Technology, Hebei Agricultural University, Baoding 071001, China
2 State Key Laboratory for Manufacturing Systems Engineering, and Systems Engineering Institute, Xi’an Jiaotong University, Xi’an 710049, China
3 School of Geomatics Science and Technology, Nanjing Tech University, Nanjing 211816, China

Corresponding author: Guanghui Zhang (ghzhang@hebau.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61972132, Grant 61573278, and Grant 41601365, in part by the Research Project of Hebei Agricultural University under Grant KY201903, Grant YJ201953, and Grant YJ201829, and in part by the Hebei Youth Talents Support Project under Grant BJ2019008.

ABSTRACT This paper studies a novel and practical distributed flexible assembly permutation flowshop scheduling problem with makespan criterion, which has attracted wide attention due to important applications in modern manufacturing. The problem integrates two machine environments of distributed production and flexible assembly, which can process and assemble the jobs into customized products. We first present a mixed integer linear programming model to characterize the problem essence and to solve small-size problems. Due to the NP-hard, we further propose an efficient memetic algorithm, which consists of a global exploration optimizer designed based on improved social spider optimization and two local exploitation optimizers designed based on meta-Lamarckian learning and simplex search, respectively. To implement the algorithm, a problem-specific encoding scheme is presented. Algorithmic parameters are calibrated by a design of experiments, and a comprehensive computational campaign is conducted to evaluate the performance of the mathematical model and algorithms. Statistical results show that their problem-solving abilities are effective, and especially the proposed memetic algorithm outperforms the existing algorithms significantly.

INDEX TERMS Distributed flowshop scheduling, flexible assembly, memetic algorithm, meta-Lamarckian learning, simplex search.

I. INTRODUCTION Flowshop scheduling problem (FSP) has been extensively studied in operational research and computer science due to its theoretical significance and practical applications [1]–[5]. In flowshop layout, the jobs are processed on a set of machines disposed in a fixed order. The typical FSP assumes that there is only a manufacturing unit or factory to be used to complete all tasks. With the deepening of economic globalization, however, modern enterprises are more and more dragged into the market with cutthroat competitiveness. Therefore, it becomes popular to duplicate multiple manufacturing factories to add flexibility into the production campaign [6]. This leads to the manufacturing reform from typical single FSP to the cooperation of multiple FSP, i.e., distributed FSP (DFSP). DFSP can decompose total production tasks into distinct manufacturing units, which helps enterprises to harvest many advantages such as better corporate reputation, higher product quality, and lower production cost and period. Compared to FSP, the scheduling in distributed environment is more complicated considering two relevant decisions must be addressed at the same time: assign the jobs to candidate factories and solve a FSP at each factory. Naderi and Ruiz [7] handled such a distributed permutation flowshop scheduling problem (DPFSP) for the first time.

The economic trend towards customized manufacturing leads to increasing prevalence for assembly manufacturing system in recent decades. As a generalization of FSP, the assembly flowshop scheduling problem (AFSP) was proposed by Lee et al. [8], in which the layout is a flowshop for
job-processing plus a single machine for product-assembling. Due to the limitation of single machine assembly, it is detrimental to the concurrent and mass production. Therefore, more flexible or multiple machine assembly comes into view. Mozdghir et al. [9] and Navaei et al. [10] are the pioneers in this field. They considered a two stage assembly flowshop scheduling with unrelated second stage assembly machines. Recently, the integration scheduling problem for distributed flowshops plus an extra single-machine assembly stage was first handled by Hatami et al. [11], which is referred to as the distributed assembly permutation flowshop scheduling problem (DAPFSP). In addition to two decisions of factory assignment and job sequencing at each factory, DAPFSP must address the third decision: product sequencing on assembly machine. The disadvantage of single-machine assembly, however, still exists and even becomes more severe due to the higher productivity from the distributed production stage. For this reason, most enterprises adopt the multi-machine flexible assembly mode in actual production.

This paper adopts the flexible assembly layout to replace the single machine assembly in DAPFSP, which generates a more practical distributed scheduling problem with flexible assembly, termed distributed flexible assembly permutation flowshop scheduling problem (DFAPFSP). An example of DFAPFSP can be found in the manufacturing of automobile engines. Different parts such as bent axles, cylinder blocks and cylinder heads are first processed using one of multiple flowshops, and then assembled into the complete engine on one of some unrelated assembly machines. The DFAPFSP is NP-hard due to the NP-hard of its special cases such as FSP, AFSP, DFSP, and DAPFSP. Hence, the evolutionary algorithm (EA) is one of the best candidates for solving the DFAPFSP. A high-quality solution in optimization problem can be gained by EA mainly depending on the good balance of exploration and exploitation abilities. In this sense, EA is not competent due to its insufficient local search ability. So, researchers devised new optimization model, referred to as memetic algorithm (MA). Usually, MA consists of a global exploration optimizer and a group of local exploitation optimizers, which are constructed based on the problem-specific knowledge and implemented in a complementary way [12]. According to the unique design preferences and evolutionary framework of MA [13], this paper presents an efficient MA for the DFAPFSP, and its main contributions are summarized as follows:

a) Propose a global exploration optimizer by refining an efficient social spider optimization, which possesses the abilities of bi-population co-evolution and cross-population interaction;

b) Propose a meta-Lamarckian learning strategy-based local exploitation optimizer to refine the best solution in population;

c) Propose a simplex search-based local exploitation operator to refine the diversity and the overall quality of the population;

d) Formulate a novel MA to cope with the DFAPFSP based on the above global and local optimizers.

The rest of this paper proceeds as follows. Section II reviews the closely related literature. Section III formulates the problem under study. Section IV details the proposed memetic algorithm. Section V conducts a comprehensive computational experiment. Finally, Section VI concludes this paper with some discussions and future research topics.

II. LITERATURE REVIEW

The problem studied in the paper is a new and promising research topic. To the best of our knowledge, the work on this problem published at present is very limited although it has been focused on widely. We review the closely related work as follows.

A. DISTRIBUTED PRODUCTION SCHEDULING PROBLEM

In distributed production scheduling problems, DFSP has obtained the most attention and gained the most extensive research achievements. In the pioneer work of Naderi and Ruiz [7], they proposed multiple mathematical models and heuristics to solve the problem. Since then, a variety of the heuristic and metaheuristic methods have been devised for it, such as estimation of distributed algorithm (EDA) [14], [15], chemical reaction optimization [16], iterated greedy algorithm [17]–[20], scatter search algorithm [21], differential evolution (DE) algorithm [22], [23].

The first attempt that addressed the AFSP in distributed manufacturing setting, i.e., DAPFSP, was made by Hatami et al. [11], and a mathematical model and some heuristics and metaheuristics were proposed to solve it. For DAPFSP, Wang and Wang [24] constructed an EDA-based memetic algorithm. Lin and Zhang [25] established a biogeography-based optimization algorithm. Lin et al. [26] presented a backtracking search hyperheuristic method. Sang et al. [27] proposed an invasive weed optimization algorithm. Ferone et al. [28] devised a biased-randomized iterated local search algorithm. Ochi and Driss [29] presented a bounded-search iterated greedy algorithm. To add more flexibility, Zhang et al. [30] first addressed the distributed flexible assembly permutation flowshop scheduling problem, i.e., DFAPFSP under study. In this pioneer work, a constructive heuristic and two hybrid algorithms were designed based on variable neighborhood search and particle swarm optimization. In addition to DFSP, the distributed scheduling problems with non-flowshop layout are also extensively concerned [6], [31] in recent decades.

B. MEMETIC ALGORITHM AND RELATED STRATEGIES

Due to the remarkable optimization ability, MA has gained more and more attention along with successful applications in distinct fields [12]. Recently, a variety of MAs have been built to solve the production scheduling problems. Ishibuchi et al. [32] proposed a memetic genetic algorithm for solving the multiobjective permutation FSP. Wang and Wang [24] devised an EDA-based MA.
for DAPFSP. Wu and Che [33] constructed a memetic DE for the energy-efficient parallel machine scheduling problem. Liu et al. [34] established a PSO-based MA for FSP. Pan et al. [35] proposed a high-performing MA for FSP with blocking constraints. Qian et al. [36] devised a DE-based MA for multiobjective jobshop scheduling problem. Gonzalez and Vela [37] established an efficient MA for the single machine scheduling problem. Shao et al. [38] developed a MA based on node and edge histogram for no-idle flowshop scheduling problem. Deng and Wang [39] gave a competitive MA for multiobjective DPFSP.

The social spider optimization (SSO) is a population-based EA, which is inspired by the cooperative behavior among social spider colony [40]. At present, it has been successfully applied to a lot of optimization problems [31]. The key feature of SSO is that its population is divided into two subpopulations, which first evolves based on different mechanisms, and then interacts between them to keep the population diversity. With such bi-population co-evolution and cross-population interaction, it shows excellent global exploration ability. For this reason, we improve the basic SSO to perform global search in the proposed MA. One of the features in MA is the application of multiple strategies to be local search. But how to arrange them in the evolution significantly affects the efficiency of MA. Ong and Keane [41] pointed out that meta-Lamarckian learning strategy can realize such arrangement in an adaptive way. The simplex search is an effective local search technique, which is easy to implement and has been successfully applied to various domains [42]. Because the original simplex search is weak in global exploration and its performance depends on the initial value, it is usually used by the hybridization with EA. Inspired by these facts, we establish two local exploitation optimizers for the proposed MA based on the meta-Lamarckian learning strategy and simplex search method, respectively.

From the above literature review, it can be seen that DFSP has been a hot research topic, and a large number of achievements have been reported in recent years. However, the investigation on the DFAPFSP is still very preliminary although it has shown important and extensive applications in modern manufacturing. Therefore, we urgently need to present more effective and efficient algorithms to address this problem.

III. DISTRIBUTED FLEXIBLE ASSEMBLY PERMUTATION FLOWSHOP SCHEDULING PROBLEM
A. PROBLEM DESCRIPTION

Figure 1 provides the structure of DFAPFSP, and Table 1 lists all necessary notations throughout the paper. In DFAPFSP, the layout consists of two consecutive stages: distributed production and flexible assembly. The production stage has $f$ identical factories working in parallel, each of which is set $m$ production machines in the flowshop manner. The assembly stage can be viewed as a single factory with $g$ unrelated assembly machines. In the manufacturing process, there are $t$ different products to be manufactured by assembling $n$ jobs according to a predefined assembly plan. To be specific, $n$ jobs are first assigned to $f$ distributed factories and at the same time specified a processing order in each factory. After all jobs in the assembly plan of a product are finished at the production stage, they are assembled into the final product by any a machine at the assembly stage. The objective is to find a schedule for jobs and products to minimize makespan. In addition to the constraints in traditional FSP, this paper also assumes: each product has an assembly plan and each job belongs to a special product. In distributed factories, each job must exactly select one factory and each factory can operate all jobs. If a job is allotted to one factory, the transfer to other factories is not allowed. In flexible assembly, there are a few unrelated machines, where each product must exactly select
one machine and each machine can assemble all products. A product can start assembly only when its assembly plan is finished in distributed factories. There are infinite buffers between machines and between manufacturing stages.

B. MAKESPAN CALCULATION
Let $\lambda_k = [\lambda_k(1), \lambda_k(2), \ldots, \lambda_k(n_k)]$ be the job processing sequence in factory $k$ at the production stage, where $n_k$ is the number of jobs assigned to factory $k$. Then, the job schedule in distributed factories can be represented as $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_f\}$, i.e., a set of job sequences. Denote $P_{ij}$ and $CP_{ij}$ as the processing time and the completion time of job $i$ on processing machine $j$, respectively. It follows that:

$$CP_{\lambda_k(1), 1} = P_{\lambda_k(1), 1}, \quad k = 1, 2, \ldots, f$$

(1)

$$CP_{\lambda_k(i), 1} = CP_{\lambda_k(i-1), 1} + P_{\lambda_k(i), 1}$$

$$i = 2, 3, \ldots, n_k; \quad k = 1, 2, \ldots, f$$

(2)

$$CP_{\lambda_k(1), j} = CP_{\lambda_k(1), j-1} + P_{\lambda_k(1), j}$$

$$j = 2, 3, \ldots, m; \quad k = 1, 2, \ldots, f$$

(3)

$$CP_{\lambda_k(i), j} = \max\{CP_{\lambda_k(i-1), j}, CP_{\lambda_k(i-1), j} + P_{\lambda_k(i), j}\}$$

$$i = 2, 3, \ldots, n_k; \quad j = 2, 3, \ldots, m; \quad k = 1, 2, \ldots, f$$

(4)

Denote $C_l$ as the latest completion time of jobs in assembly plan $AP_l$ of product $l$ at the production stage. It is calculated as follows:

$$C_l = \max_{i \in AP_l} \{CP_{i,m}\}$$

(5)

Let $\xi_s = [\xi_s(1), \xi_s(2), \ldots, \xi_s(t_s)]$ be the assembling sequence of $t_s$ products assigned to assembly machine $s$. The product schedule on all assembly machines can be represented as $\xi = \{\xi_1, \xi_2, \ldots, \xi_g\}$, i.e., a set of product sequences. Denote $T_{s,t}$ and $CA_{s,t}$ as the assembly time and completion time of product $l$ on assembly machine $s$, respectively. Then, the following calculation can be available:

$$CA_{\xi_s(1), s} = \xi_s(1) + T_{\xi_s(1), s}, \quad s = 1, 2, \ldots, g$$

(6)

$$CA_{\xi_s(l), s} = \max\{CA_{\xi_s(l-1), s}, CA_{\xi_s(l-1), s} + T_{\xi_s(l), s}\}$$

$$l = 2, 3, \ldots, t_s; \quad s = 1, 2, \ldots, g$$

(7)

Finally, we combine the job schedule $\lambda$ and product schedule $\xi$ to gain a complete DFAPFSP schedule $\pi = (\lambda; \xi)$. This schedule consists of four consecutive decisions: assign jobs to candidate factories, sequence jobs in each of distributed factory, assign products to assembly machines and arrange products on each assembly machine. Denote the makespan of $\pi$ as $C_{\text{max}}(\pi)$. The objective is to find a schedule $\pi$ with minimum $C_{\text{max}}(\pi)$. It can be calculated by:

$$C_{\text{max}}(\pi) = \max_s \{CA_{\xi_s(t_s), s}\}$$

(8)

C. ILLUSTRATIVE EXAMPLE
To understand the problem in depth, an example is presented to illustrate it. It consists of $n = 9$, $t = 3$, $f = 3$, $m = 3$ and $g = 2$. The assembly plan is that $AP_1 = \{J_1, J_6, J_7\}$, $AP_2 = \{J_2, J_5, J_9\}$ and $AP_3 = \{J_3, J_4, J_8\}$ corresponding to products $P_1$, $P_2$ and $P_3$, respectively. Table 2 presents the operating times ($P_{ij}$) for jobs and products on different machines. Consider problem schedule $\pi = (\lambda; \xi)$, where $\lambda = \{[J_1, J_4, J_3], [J_6, J_7, J_9], [J_5, J_6, J_2]\}$ and $\xi = \{[P_1], [P_3, P_2]\}$. With the recursive calculation (1)-(7), it is easy to calculate the starting and completion time of each job and product in this schedule on corresponding machines. They are also provided in Table 2 by columns $st$, $CP_{ij}$ and $CA_{ij}$, respectively. Based on these data, the makespan is calculated as 24 by Eq. (8), and the Gantt chart is presented in Figure 2 to show the scheduling details.

| Product | AP_1 | AP_2 | AP_3 |
|---------|------|------|------|
| P_1     | 5,1823 | 6,~,,~ |      |
| P_2     | 4,~,,~ | 2,2224 |      |
| P_3     | 5,~,,~ | 2,1921 |      |

D. MATHEMATICAL MODEL
To formulate the mathematical description, a mixed integer linear programming (MILP) model is built in this section. It is inspired by the research in [11], which is a special case of
the work in this paper. The indices, parameters and variables used in this model see Table 1, and the MILP model can be described as follows:

$$\min C_{\text{max}}$$  \hspace{1cm} (9)

The objective function (9) shows the makespan to be minimized. In the distributed production stage, the model should subject to the constraints:

$$\sum_{i=0,l \neq r}^{n} X_{i,r} = 1 \quad \forall r$$  \hspace{1cm} (10)

$$\sum_{r=0, r \neq i}^{n} X_{i,r} = 1 \quad \forall i$$  \hspace{1cm} (11)

$$\sum_{r=1}^{n} X_{0,r} = f$$  \hspace{1cm} (12)

$$\sum_{i=1}^{n} X_{i,0} = f - 1$$  \hspace{1cm} (13)

Constraints (10) shows that each job has only one predecessor. Constraint (11) controls that each job has at most one successor. Constraint (12) assures that dummy job 0 must be a predecessor of another job $f$ times, while constraint (13) shows that dummy job 0 should be a successor $f - 1$ times. The existence of dummy job 0 in such way can divide the job sequence into $f$ parts, each of which represents a factory’s job schedule. Thus, there is no factory index to be used in the constraints. Constraint (14) points out that a job cannot be another job’s predecessor and successor at the same time. Constraints (15) and (16) indicate that at a time a job is processed by at most one machine and a machine can process at most one job.

In the flexible assembly stage, the model has to subject to the constraints:

$$\sum_{l=0, l \neq v}^{t} Y_{l,v} = 1 \quad \forall v$$  \hspace{1cm} (17)

$$\sum_{v=0, v \neq l}^{t} Y_{l,v} = 1 \quad \forall l$$  \hspace{1cm} (18)

$$\sum_{v=1}^{g} Y_{0,v} = g$$  \hspace{1cm} (19)

$$\sum_{l=1}^{t} Y_{l,0} = g - 1$$  \hspace{1cm} (20)

$$Y_{l,v} + Y_{v,l} \leq 1 \quad \forall v \in \{1, 2, \ldots, t - 1\}, \ l > v$$  \hspace{1cm} (21)

$$CA_{l,s} \geq C_{l} + T_{l,s} \quad \forall l, s$$  \hspace{1cm} (22)

$$CA_{l,s} \geq CA_{v,s} + T_{l,s} + (Y_{v,l} - 1) \cdot W \quad \forall l, v, s, l \neq v$$  \hspace{1cm} (23)

Constraints (17) and (18) indicate that each product has only one predecessor and at most one successor. Constraints (19) and (20) enforce that every product must be exactly assigned to one assembly machine. Constraint (21) specifies that a product cannot be at the same time another product’s predecessor and successor. Constraint (22) states that each product cannot start assembly before its assembly plan is finished in distributed factories. Constraint (23) assures that if product $l$ is assembled immediately after product $v$, its assembly on machine $s$ cannot start until machine $s$ finishes product $v$. Constraint (24) defines the makespan.

Finally, the domain of the decision variables is defined by the following constraints:

$$X_{i,r}, Y_{l,v} \in \{0, 1\} \quad \forall i, r, l, v, i \neq r, l \neq v$$  \hspace{1cm} (25)

$$CP_{i,j}, C_{i}, CA_{i,s} \geq 0 \quad \forall i, j, l, s$$  \hspace{1cm} (26)

$$CP_{i,0} = CA_{i,0} = 0 \quad \forall i, l$$  \hspace{1cm} (27)

IV. MEMETIC ALGORITHM WITH META-LAMARCKIAN LEARNING AND SIMPLEX SEARCH

Due to the NP-hardness, the above mathematical model is only able to solve some small-size problems effectively. Whereas for the large-size problems, it is hard to obtain the optimal solutions within a reasonable computational time. For solving the DFAPFSP with different sizes, we propose a memetic algorithm that mainly consists of a global exploration optimizer devised by improving social spider optimization and two local exploitation optimizers devised separately based on meta-Lamarckian learning and simplex search. It is denoted as MAGL in short and the details can be depicted as follows.

A. ENCODING AND DECODING SCHEMES

Encoding scheme is a key factor in the algorithm design [43]. First, it affects the number and distribution of solutions in the search space. Second, it affects the neighborhood structure and size of a solution such that affecting the time cost. Third, along with the decoding scheme, it affects the solution quality. Thus, the encoding and decoding schemes should be well designed based on the problem knowledge.

In this paper, a problem solution is represented as a vector $x = [x_1, x_2, \ldots, x_n]$ of $n$ real numbers, each of which are randomly generated in interval $[1, f + 1)$. To decode $x$ as a complete DFAPFSP schedule, the following two steps are performed:

Step 1: Construct the job schedule $\lambda$ in distributed factories. Let $x_i$ correspond to job $J_i$ and denote $|x_i|$ as the integer part of $x_i$. Assign job $J_i$ to factory $|x_i|$, and the jobs in the same factory are processed in ascending order of corresponding numbers in $x$. In this way, the job schedules in distributed factories are obtained. Consider $f = 2$ and an example $x = [1.26, 2.33, 1.67, 2.75, 2.14, 1.91, 1.41]$ to explain this step. Since $x_1 = 1.26, x_3 = 1.67, x_6 = 1.91$ and $x_7 = 1.41$ have the same integer part 1, the jobs $J_1, J_3, J_6$ and $J_7$ are assigned.

$$C_{\text{max}} \geq CA_{l,s} \quad \forall l, s$$  \hspace{1cm} (24)
to factory 1. According to the order of \( x_1 < x_7 < x_3 < x_6 \), the job processing sequence in factory 1 can be determined to be \( \lambda_1 = [J_1, J_7, J_3, J_6] \). Similarly, there is \( \lambda_2 = [J_5, J_2, J_4] \), and therefore the job schedule in distributed factories is \( \lambda = \{\lambda_1, \lambda_2\} \).

Step 2: Construct the product schedule \( \xi \) on flexible assembly machines. For any two products, the one whose assembly plan is completed earlier in the distributed factories is assigned first. That is to say, if \( C_l \) is less than \( C_h \), product \( l \) is assigned prior to product \( h \). For each product to be assigned first, it is assembled by any one assembly machine that completes this product at the earliest possible time. If there are more than one choice, we select one from the candidates surely.

As stated before, a complete DFAPFSP schedule should contain four interrelated decisions. If all these decisions are applied only in the global search module. For the minimization problem under study, it is formulated as follows:

\[
\text{Step 2: Randomly select } f \text{ variables from } x \text{ and denote them as } y_r, r = 1, 2, \ldots, f.
\]

Step 3: Replace \( y_r \) by the value randomly generated from the interval \([r, r+1], r = 1, 2, \ldots, f\), which ensures that each factory is assigned at least one job.

Step 4: Repeat the above steps until meeting the specified population size.

The acceptance criteria determine which of the old and new solutions can survive in next population. So, its choice affects an algorithmic performance to a certain extent. In this work, we give two acceptance criteria and denote them as \( AC_1 \) and \( AC_2 \), respectively. \( AC_1 \) is a biased acceptance criterion and is applied only in the global search module. For the minimization problem under study, it is formulated as follows:

\[
y = \begin{cases} 
  x_{\text{new}}, & \text{if } RE < 0 \text{ or } rand < \min \{\tau - RE, 0\} \\
  x_{\text{old}}, & \text{otherwise}
\end{cases}
\]

where \( y \) is the survivor, \( \tau \in (0, 1) \) is the bias acceptance constant, and \( RE \) is the relative fitness error of \( x_{\text{new}} \) regarding \( x_{\text{old}} \) and is defined by \( RE = (C_{\text{max}}(x_{\text{new}}) - C_{\text{max}}(x_{\text{old}}))/C_{\text{max}}(x_{\text{old}}) \). As can be seen, \( x_{\text{new}} \) surely survives, i.e., \( y = x_{\text{new}} \), if it is absolutely better than \( x_{\text{old}} \), i.e., \( RE < 0 \). When \( x_{\text{new}} \) is only slightly worse than \( x_{\text{old}} \), i.e., \( 0 < RE < \tau \), it may also survive by a small probability \( \tau - RE \).

In this way, \( \tau \) can control several solutions with relatively poor quality to survive in next population. The population diversity is maintained to some extent.

\( AC_2 \) is a simple greedy or descent acceptance criterion, i.e., accepting new solution only if it is absolutely better than the old one. This criterion is used only in the local search module.

B. INITIALIZATION AND ACCEPTANCE CRITERION

The initial population is generated according to the idea of avoiding any idle factories, and therefore the following steps are adopted:

\[
x_i = x_{\text{min}} + \text{rand} \cdot (x_{\text{max}} - x_{\text{min}})
\]

(28)

where \( \text{rand} \) is a random number in \([0, 1]\).

Step 2: Randomly select \( f \) variables from \( x \) and denote them as \( y_r, r = 1, 2, \ldots, f \).

Step 3: Replace \( y_r \) by the value randomly generated from the interval \([r, r+1], r = 1, 2, \ldots, f\), which ensures that each factory is assigned at least one job.

Step 4: Repeat the above steps until meeting the specified population size.

The acceptance criteria determine which of the old and new solutions can survive in next population. So, its choice affects an algorithmic performance to a certain extent. In this work, we give two acceptance criteria and denote them as \( AC_1 \) and \( AC_2 \), respectively. \( AC_1 \) is a biased acceptance criterion and is applied only in the global search module. For the minimization problem under study, it is formulated as follows:

\[
y = \begin{cases} 
  x_{\text{new}}, & \text{if } RE < 0 \text{ or } rand < \min \{\tau - RE, 0\} \\
  x_{\text{old}}, & \text{otherwise}
\end{cases}
\]

(29)

where \( y \) is the survivor, \( \tau \in (0, 1) \) is the bias acceptance constant, and \( RE \) is the relative fitness error of \( x_{\text{new}} \) regarding \( x_{\text{old}} \) and is defined by \( RE = (C_{\text{max}}(x_{\text{new}}) - C_{\text{max}}(x_{\text{old}}))/C_{\text{max}}(x_{\text{old}}) \). As can be seen, \( x_{\text{new}} \) surely survives, i.e., \( y = x_{\text{new}} \), if it is absolutely better than \( x_{\text{old}} \), i.e., \( RE < 0 \). When \( x_{\text{new}} \) is only slightly worse than \( x_{\text{old}} \), i.e., \( 0 < RE < \tau \), it may also survive by a small probability \( \tau - RE \).

In this way, \( \tau \) can control several solutions with relatively poor quality to survive in next population. The population diversity is maintained to some extent.

\( AC_2 \) is a simple greedy or descent acceptance criterion, i.e., accepting new solution only if it is absolutely better than the old one. This criterion is used only in the local search module.

C. GLOBAL EXPLORATION OPTIMIZER

1) IDEA OF BASIC SSO

Basic SSO models the search space as a spider web, where spiders interact with each other by vibrating the web. Each position the spiders go through corresponds to a problem solution. All spider positions make up the SSO population. Each spider is given a weight that is calculated by its related fitness function, and the greater the weight, the better the quality of the solution. During the search process, the SSO uses two search agents: male and female spiders. Accordingly, the population is divided into two subsets based on the spiders’ gender. The spider positions in these two subsets are updated by different operators, and then a mating operator is carried out between two subsets. The detailed steps of the basic SSO can refer to the work in [40]. Compared to traditional EAs, the SSO has good global search ability, which derived from its ingenious design: bi-population co-evolution and cross-population interaction.

2) IMPROVED SSO FOR MAGL GLOBAL OPTIMIZER

According to the template of the basic SSO, we propose an improved version of it, which has five modules: population division, weight calculation, updating female positions, updating male positions, and mating operation. The improved SSO is used for the MAGL global optimizer. Next, we detail each module as follows.

a: POPULATION DIVISION AND WEIGHT CALCULATION

In social spider colony, the number of females \( N_f \) accounts for 65%-90% of total population number \( N_p \), and of course the other is the number of males \( N_m \). They are calculated by:

\[
N_f = \text{floor} \left( (90\% - \text{rand} \cdot 25\%) \cdot N_p \right)
\]

(30)

\[
N_m = N_p - N_f
\]

(31)

where \( \text{floor}(\cdot) \) represents a function that maps real number to the integer. The first \( N_f \) members of total population \( S \) is divided into female subset \( F \), and the other members are the male subset \( M \). That is, \( S = F \cup M \).

Each spider \( x \) receives a weight \( w \) which reflects the solution quality that corresponds to this spider. To model the weight, the following formula is adopted:

\[
w = \frac{\max \{C_{\text{max}}(x)\} - C_{\text{max}}(x)}{\max \{C_{\text{max}}(x)\} - \min \{C_{\text{max}}(x)\} + c_1} + c_2
\]

(32)
where \( c_1 \) and \( c_2 \) are two constants and used to avoid zero denominator and zero weight. In this paper, we set \( c_1 = 0.01 \) and \( c_2 = 0.001 \).

\( b \): UPDATING FEMALE POSITIONS

The social spiders update their positions in the web by interacting with each other and learning the external stimulus. The spiders with different gender update the position in different way. The research in Zhang and Xing [31] showed that a female spider migrates its position depending on:

1. The cooperation with the closest and more weighted spider, \( v_{c,x} \),
2. The vibration perceived by the spider at \( x \), \( v_{b,x} \), from the spider at \( x \).

Representing the vibration perceived by the male spider at \( x \) from the female spider at \( x_{a} \), \( M \) is the subset of all male spiders, \( x_{\text{mean}} \) is the weighted mean of all male positions.

\( d \): IMPROVED ADAPTIVE MATING OPERATOR

After both female and male positions are updated, a mating operator will be executed on dominant male spiders in current colony. In the original mating operator, each dominant male spider has a mating radius and mates with the females within this radius to produce new baby spider. The position value of the baby spider at every dimension is randomly chosen from the corresponding dimensional values of the mating candidates by roulette-wheel technique based on their weights. If there are no female spiders within this radius, the mating operator is not performed. However, such mating operation has two limitations: a) there may be no female spiders in the specified mating radius for a long time; b) it is difficult to generate better baby spiders in the later search stage due to lack of population diversity. Therefore, we improve the original mating operator to avoid such two shortcomings, and the new version can be depicted as follows.

\[
\begin{align*}
z_i &= \begin{cases} 
x_i, & \text{if } rand < P_i \\
x_{\text{min}} + \text{rand} \cdot (x_{\text{max}} - x_{\text{min}}), & \text{otherwise}
\end{cases} \\
P_i &= P_{\text{min}} + (P_{\text{max}} - P_{\text{min}})/(1 + \exp (20 \cdot t/t_{\text{max}} - 10))
\end{align*}
\]

where \( i = 1, 2, \ldots, n \), \( t \) is the current iteration number, \( t_{\text{max}} \) is the allowable maximum iteration number, \( P_i \in [P_{\text{min}}, P_{\text{max}}] \) can be viewed as the inheritance rate for the mating parents. As can be seen from Figure 3, \( P_i \) dynamically drops as sigmoid curve, which make the baby spider inherit more genes of mating parents in the beginning search stage, and gradually increase variation with iteration. In this way, we can maintain the population diversity as much as possible during the evolutionary process. New baby spider is used to replace the worst individual in population, and inherit the same gender.

\[ y = x + \kappa \cdot r_1 \cdot v_{c,x} \cdot (x_c - x) + \kappa \cdot r_2 \cdot v_{b,x} \cdot (x_b - x) + r_3 \cdot (r_4 - 0.5) \]  
\[ v_{c,x} = w_c \cdot \exp \left( -\sqrt{\| x_c - x \|} \right) \]  
\[ v_{b,x} = w_b \cdot \exp \left( -\sqrt{\| x_b - x \|} \right) \]  

where \( r_1, r_2, r_3 \) and \( r_4 \) are four random numbers in \([0, 1]\), \( x_c \) and \( w_c \) are the position and weight of the closest and more weighted spider, \( x_b \) and \( w_b \) are the position and weight of the spider with the highest weight in the current population, notation \( || \cdot || \) calculates the Euclidean distance, \( v_{c,x} \) is the vibration perceived by the spider at \( x \) from the spider at \( x_c \), \( v_{b,x} \) is the vibration perceived by the spider at \( x \) from the spider at \( x_b \), the last random item in Eq. (33) produces an unbiased perturbation along any directions, and \( \kappa \) controls the move near or far away, which works in this paper according to the scheme: if \( \text{rand} < 0.7 \), then \( \kappa = 1 \), else \( \kappa = -1 \).

\( c \): UPDATING MALE POSITIONS

According to the role played in the colony, male spiders are divided into two categories: dominant and non-dominant male spiders. Dominant males have larger weight than the non-dominant. Dominant male spiders usually move towards the nearest female spider for reproduction, whereas non-dominant male spiders prefer to concentrate in the center of the male colony for making use of the resources wasted by dominant males. In this paper, half of the male spiders with greater weight are referred to as dominant and the other half are called non-dominant. For male spiders, the position migration from \( x \) to \( y \) is modeled by:

\[
y = \begin{cases} 
x + r_1 \cdot v_{a,x} \cdot (x_a - x) + r_2 \cdot (r_3 - 0.5), & \text{if dominant} \\
x + r_1 \cdot (x_{\text{mean}} - x), & \text{otherwise}
\end{cases}
\]

\[ v_{a,x} = w_a \cdot \exp \left( -\sqrt{\| x_a - x \|} \right) \]  
\[ x_{\text{mean}} = \frac{\sum_{x_i \in M} w_i \cdot x_i}{\sum_{x_i \in M} w_i} \]  

\[ P_i = P_{\text{min}} + (P_{\text{max}} - P_{\text{min}})/(1 + \exp (20 \cdot t/t_{\text{max}} - 10)) \]
D. LOCAL EXPLOITATION OPTIMIZERS

The global optimizer proposed above focuses more on the global exploration, but its local exploitation capacity is relatively insufficient. In order to balance them effectively, two local exploitation optimizers are further proposed as follows.

1) MLS
meta-Lamarckian learning-based local search. MLS is used to improve the best individual in population. This local search is not directly employed to the spider position \( x = [x_1, x_2, \ldots, x_n] \), i.e., continuous real-number vector, but to the related job schedule \( \lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_f\} \), i.e., a set of job permutations where \( \lambda_k \) is the job processing sequence in factory \( k \). For this reason, a decoding process from \( x \) to \( \lambda \) is needed before applying MLS. Four single job moves are first designed by:
- \( JM_1 \): Randomly select a factory and swap the position of two different jobs in this factory;
- \( JM_2 \): Randomly select a factory and insert one job to another position in this factory;
- \( JM_3 \): Swap the position of two jobs separately from different factories;
- \( JM_4 \): Remove a job from one factory and reinsert it to another factory.

Based on the four job moves, two compound job moves are further defined by an integration of them: \( CM_1 \) and \( CM_2 \) respectively, perform \( JM_1 \) and \( JM_3 \) simultaneously, and \( CM_2 \) is \( JM_2 + JM_4 \).

In the search of MLS, we apply \( JM_k \), \( k = 1, 2, 3, 4 \), to perform a slight perturbation on the search candidates for preventing falling into a local trap, and then utilize \( CM_k \), \( k = 1, 2 \), to conduct the local exploitation on the perturbation results. Inspired by the work in [34], we not only consider two distinct compound job moves in local exploitation, but also adopt an adaptive meta-Lamarckian learning technique [41] to decide which compound job move is more suitable for current local exploitation.

This meta-Lamarckian learning method consists of a training phase and a working phase. In the training phase, each of \( CM_k \), \( k = 1, 2 \), is executed \( Tim = n \times (n - 1) \) times on the best member within the initial or first-generation population of our proposed MAGL algorithm. Then for each move, the reward value and selection probability are calculated using Eq. (41) and Eq. (42), respectively.

\[
\eta_k = \frac{(C_{bef} - C_{aft})}{Tim} \tag{41}
\]

\[
\xi_k = \frac{\eta_k}{\sum_{i=1}^{2} \eta_i} \tag{42}
\]

where \( C_{bef} \) is the makespan value before performing the move, and \( C_{aft} \) is the best makespan value generated during \( Tim \) times executions by the AC2 acceptance criterion.

The working phase is performed on the best member in the non-initial MAGL population. For the search candidate, we first apply one of \( JM_k \), \( k = 1, 2, 3, 4 \), to perturb it, and then use the roulette-wheel strategy to decide which one of \( CM_1 \) and \( CM_2 \) to be selected for improving the perturbing solution based on selection probability. If \( CM_k \) is chosen, it will be carried out \( Tim \) times to update its reward value via \( \eta_k = \eta_k + \eta_{new} \), where \( \eta_{new} \) is the newly obtained reward also calculated according to Eq. (41). Accordingly, the selection probability of \( CM_1 \) and \( CM_2 \) is updated by Eq. (42).

It should be noted that the search result generated from MLS is job permutations, which cannot be evolved by the proposed MAGL global optimizer. So, a conversion from \( \lambda \) to \( x \) is needed after implementing MLS. The conversion is stated by Algorithm 1, and the process of MLS is presented by Algorithm 2.

Algorithm 1 Conversion From \( \lambda \) to \( x \)
1: input \( \lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_f\} \)
2: set seq1 = seq2 = ∅
3: for \( k = 1 \) to \( f \) do
4:    \( len \leftarrow \text{length}(\lambda_k) \)
5:    seq1 = [seq1, \lambda_k]
6:    seq2 = [seq2, sort(rand(1, \text{len})) + \text{k*ones}(1, \text{len})]
7: end
8: [~ index] = sort(seq1)
9: \( x \leftarrow \text{seq2(index)} \)

Algorithm 2 MLS
1: if iter = 0 then % (perform the training phase)
2:   a) find the best individual \( \lambda \) in initial population
3:   b) perform \( CM_1 \) and \( CM_2 \) \( n \times (n - 1) \) times on \( \lambda \) respectively
4:   c) calculate \( \xi_1 \) and \( \xi_2 \) by Eqs. (41) and (42)
5: else % (perform the working phase)
6:   find the best member \( \lambda \) in current population
7:   \( k = 1 \)
8: while \( k \leq 4 \) do
9:     \( \lambda' \leftarrow \text{perturb} \lambda \) by \( JM_k \)
10: \( l = 0 \)
11: while \( l < Tim \) do
12:   select \( CM_1 \) or \( CM_2 \) based on \( \xi_1 \) and \( \xi_2 \) by roulette wheel rule
13:   \( \lambda'' \leftarrow \text{perturb} \lambda' \) times on \( \lambda'' \)
14:   \( \lambda' \leftarrow \text{select the better of} \lambda' \text{ and} \lambda'' \)
15:   calculate \( \eta_{new} \) by Eq. (41)
16:   \( \eta_k \leftarrow \eta_k + \eta_{new} \)
17:   update \( \xi_1 \) and \( \xi_2 \) by Eq. (42)
18:   \( l = l + 1 \)
19: if \( \lambda \) is improved then \( k = 1 \)
20: else \( k = k + 1 \)

2) SLS
Simplex-based local search. During the process of iterating the above global and local optimizers, the optimal individual of each iteration
may fall into local optimum. So, it is necessary to find new solutions to jump out of the local optimum. Simplex method is a direct optimization method, which does not need to know gradient information and has strong local search ability [42]. Thus, a simplex-based local search (SLS) method is proposed herein. It first constructs a simplex using current population, and rescales it to generate a solution (spider position) with better fitness value based on four operators: reflection, expansion, contraction and shrink. Then, the worst solution in population is replaced by this solution generated newly. In this way, the population’s overall quality can be gradually improved to approximate the global optimum. One iteration of SLS can be described as follows:

Step 1: Reflection. Find the spiders with the global best and the worst position in current population. Denote them as \(x_{gb}\) and \(x_w\), respectively. Calculate \(x_{cent}\) as the average of the swarm positions excluding \(x_w\). Generate a new position \(x_{ref}\) by reflecting \(x_w\) according to Eq. (43).

If \(C_{\text{max}}(x_{ref}) < C_{\text{max}}(x_{gb})\), go to step 2.
If \(C_{\text{max}}(x_{ref}) > C_{\text{max}}(x_w)\), go to step 3.
If \(C_{\text{max}}(x_{gb}) \leq C_{\text{max}}(x_{ref}) \leq C_{\text{max}}(x_w)\), go to step 4.

Step 2: Expansion. Expand the reflection position \(x_{ref}\) along the same direction as the reflection. This operation is performed by Eq. (44). If \(C_{\text{max}}(x_{exp}) < C_{\text{max}}(x_{gb})\), the best one of \(x_{exp}\) and \(x_{ref}\) replaces \(x_w\), otherwise \(x_{ref}\) replaces \(x_w\).

Step 3: Contraction. Perform the contraction operation using Eq. (45). If \(C_{\text{max}}(x_{con}) < C_{\text{max}}(x_w)\), \(x_{con}\) replaces \(x_w\).

Step 4: Shrink. Perform the shrink operation by Eq. (46). If \(C_{\text{max}}(x_{shr}) < C_{\text{max}}(x_w)\), the best one of \(x_{shr}\) and \(x_{ref}\) replaces \(x_w\), otherwise \(x_{ref}\) replaces \(x_w\).

\[
\begin{align*}
x_{\text{ref}} &= x_{\text{cent}} + \alpha \cdot (x_{\text{cent}} - x_w) \\
x_{\text{exp}} &= x_{\text{cent}} + \beta \cdot (x_{\text{ref}} - x_{\text{cent}}) \\
x_{\text{con}} &= x_{\text{cent}} + \gamma \cdot (x_w - x_{\text{cent}}) \\
x_{\text{shr}} &= x_{\text{cent}} - \delta \cdot (x_w - x_{\text{cent}})
\end{align*}
\]

where \(\alpha = 1\), \(\beta = 2\), and \(\gamma = \delta = 0.5\). For a population with only three individuals \(x_{gb}\), \(x_w\) and \(x\), these four operators is shown in Figure 4.

E. OVERALL MAGL ALGORITHM

With the steps designed previously, the overall MAGL algorithm is shown in Figure 5. It should be noted that due to the position moves, partial spiders may move out of the specified range resulting in infeasible solutions. Once that happens, each illegal variable in solution will be reset by a random value in the encoding interval stipulated in Section IV-A.

V. NUMERICAL EXPERIMENTS

A. EXPERIMENTAL PREPARATIONS

To demonstrate the algorithmic performance, a variety of numerical experiments are carried out in this section. Some necessary preparations for carrying out experiments are first introduced as follows.

Benchmark problem. The studied DFAPFSP is a novel scheduling problem, and no benchmark is presented for it in current literature. In this section, we establish a benchmark set consisting of 96 different-size instances. To facilitate the further research, all benchmark data will be attached online in the supplementary materials together with the published paper.

The small-size instances are a set of 48 with the combination of the number of jobs (\(n\)), number of factories (\(f\)), number of machines in each factory (\(m\)), number of product (\(t\)), number of assembly machines (\(g\)), i.e., \(4f/5m/6t/g\), where \(n = \{20, 25, 30\}\), \(f = \{2, 3\}\), \(m = \{2, 4\}\), \(t = \{6, 8\}\), and \(g = \{2, 3\}\). For large-size instances, there are also a set of 48 with the same combination, where \(n = \{100, 150, 200\}\), \(f = \{6, 8\}\), \(m = \{6, 10\}\), \(t = \{20, 30\}\), and \(g = \{4, 6\}\). The
and $N_p$, and the suggested parameter values are $N_p = 30$, $\tau = 0.04$, $\text{Tim} = 200$, and $P_{\text{min}} - P_{\text{max}} = 0.2-0.9$; 2) For large-size instances, the influence priority on MAGL is that $P_{\text{min}} - P_{\text{max}}$ ranks first, followed by $\text{Tim}$, $\tau$ and $N_p$, and the suggested parameter values are $N_p = 40$, $\tau = 0.02$, $\text{Tim} = 200$, and $P_{\text{min}} - P_{\text{max}} = 0.1-1.0$.

Other preparations. To prepare more convergence time for those testing instances with larger size, the termination criterion used in algorithmic calibration and comparison is the maximum elapsed runtime $\rho \times n \times m \times t$ milliseconds, where $\rho$ is set at the value 10 for small-size instances and the value 3 for large-size instance. All algorithms in the experiments are coded in Matlab 2012a and implemented on the PC: Intel(R) Core(TM) i3-4150 CPU @ 3.50 GHz with 4.00 GB RAM in the 32-bit Windows 7 operating system. In order to measure the algorithmic performance, the relative percentage deviation (RPD) is applied and calculated by:

$$\text{RPD} = 100 \times \left(\frac{C_{\text{max}} - C_{\text{best}}}{C_{\text{best}}}\right)$$

where $C_{\text{max}}$ is the makespan of a schedule obtained by the given algorithm for an instance, and $C_{\text{best}}$ is the best known makespan found throughout the experiments.

Due to inherent randomness, the computational results of metaheuristic methods may vary with different executions. Thus, instead of directly applying RPD as the performance metric, the metaheuristics are run 20 times independently for each instance. Then, the best $\text{RPD}$ ($b\text{RPD}$) and the average $\text{RPD}$ ($a\text{RPD}$) among 20 runs are used for the final performance metrics.

### B. VALIDATION OF MILP MODEL AND MAGL ALGORITHM

To validate our MILP model and MAGL algorithm, we compare them against three algorithms [30]: TPHS, HVNS and HPSO. The three algorithms are originally proposed for the DFAPFSP but with setup constraints. So, we adapt them to the problem in this paper by the way of our makespan calculation. The MILP model and the TPHS method are deterministic, and so they are only implemented once for each instance to generate the metric RPD. Whereas for the algorithms HVNS, HPSO and MAGL, two metrics of $b\text{RPD}$ and $a\text{RPD}$ are calculated. The MILP model is solved by the CPLEX 12.2 with the time limits of $5 \times n \times m \times t$ seconds for small size instances and 3600 seconds for large size instances. If CPLEX cannot gain an optimal solution within the stipulated time, it will continue to be executed until finding a feasible solution (which may be not optimal). The computational results have been grouped in Tables 6 and 7 by problem parameters $n, f, m, t$ and $g$, and the best metrics are marked in bold.

For small size instances, Table 6 shows that the proposed MILP model and the MAGL algorithm are significantly better than the compared three competitors TPHS, HVNS and HPSO for each instance group. It can be concluded that the proposed MILP model and MAGL algorithm are very effective when solving the DFAPFSP with small size. From Table 6, we can also see that the performance of the MAGL algorithm is very close to that of the MILP model.
TABLE 5. Response variable statistics (× 1000).

| Parameter level | RV_1 |     | RV_2 |     |
|-----------------|------|-----|------|-----|
|                 | Np   | r   | Tim  | Pmax | Np   | r   | Tim  | Pmax |
| 1               | 0.9041 | 0.9047 | 0.9058 | 0.9058 | 1.1837 | 1.1814 | 1.1792 | 1.1734 |
| 2               | 0.9037 | 0.9065 | 0.9032 | 0.9039 | 1.1785 | 1.1762 | 1.1856 | 1.1805 |
| 3               | 0.9041 | 0.9037 | 0.9049 | 0.9022 | 1.177 | 1.1797 | 1.1772 | 1.1799 |
| 4               | 0.9039 | 0.9009 | 0.9019 | 0.9056 | 1.1796 | 1.1815 | 1.1768 | 1.1851 |
| Range           | 0.0004 | 0.0056 | 0.0039 | 0.0036 | 0.0067 | 0.0053 | 0.0088 | 0.0117 |
| Rank            | 4     | 1   | 2   | 3   | 4   | 1   | 2   | 1   |
| Value           | 30    | 0.04 | 200 | 0.2-0.9 | 40    | 0.02 | 200 | 0.1-1.0 |

TABLE 6. Computational results of MILP model and compared algorithms for small-size instances.

| Instance group | MILP | TPHS | HVNS | HPSO | MAGL | Time (sec.) |
|----------------|------|------|------|------|------|-------------|
| n              | 0.143 | 2100 |      |      |      |             |
| 25             | 0.229 | 2625 | 20.738 | 0.02 | 0.777 | 0.358 | 1.722 | 2.224 | 0.095 | 0.252 | 5.25 |
| 30             | 0.144 | 3150 | 18.123 | 0.05 | 0.949 | 0.399 | 1.407 | 1.965 | 0.179 | 0.174 | 6.3  |
| f              | 0.183 | 3975 | 18.061 | 0.05 | 0.844 | 0.199 | 1.179 | 1.526 | 0.212 | 0.239 | 7.95 |
| 3              | 0.238 | 2625 | 24.819 | 0.03 | 1.479 | 0.657 | 2.411 | 2.867 | 0.277 | 0.113 | 5.25 |
| 2              | 0.369 | 1750 | 17.951 | 0.02 | 0.891 | 0.436 | 1.332 | 1.713 | 0.046 | 0.064 | 3.5  |
| m              | 0.295 | 3500 | 24.929 | 0.05 | 1.431 | 0.421 | 2.293 | 2.679 | 0.441 | 0.288 | 7    |
| 4              | 0.329 | 2250 | 20.528 | 0.02 | 1.186 | 0.345 | 1.651 | 2.278 | 0.203 | 0.161 | 4.5  |
| 6              | 0.177 | 3000 | 22.352 | 0.03 | 1.137 | 0.521 | 1.965 | 2.113 | 0.283 | 0.191 | 6    |
| t              | 0.207 | 2625 | 20.672 | 0.02 | 1.031 | 0.456 | 1.643 | 2.239 | 0.173 | 0.168 | 5.25 |
| 8              | 0.319 | 2625 | 21.404 | 0.03 | 1.291 | 0.401 | 1.996 | 2.154 | 0.318 | 0.184 | 5.25 |
| g              | 0.239 | 2750 | 21.367 | 0.03 | 1.161 | 0.429 | 1.813 | 2.211 | 0.244 | 0.176 | 5.5  |
| Mean           |       |      |      |      |      |             |

TABLE 7. Computational results of MILP model and compared algorithms for large-size instances.

| Instance group | MILP | TPHS | HVNS | HPSO | MAGL | Time (sec.) |
|----------------|------|------|------|------|------|-------------|
| n              | 0.108 | 3600 | 20.303 | 0.11 | 2.138 | 1.561 | 3.367 | 3.863 | 0   | 0   | 60   |
| 150            | 0.902 | 3600 | 19.317 | 0.15 | 1.245 | 0.738 | 1.843 | 2.297 | 0.023 | 0.047 | 90   |
| 200            | 0.387 | 3600 | 17.366 | 0.24 | 1.224 | 0.613 | 1.833 | 2.186 | 0.045 | 0.084 | 120  |
| f              | 0.226 | 3600 | 20.165 | 0.18 | 1.461 | 0.872 | 2.118 | 2.511 | 0.033 | 0.018 | 90   |
| 6              | 18.897 | 3600 | 18.754 | 0.15 | 1.610 | 1.069 | 2.577 | 3.052 | 0.013 | 0.069 | 90   |
| m              | 0.117 | 3600 | 20.063 | 0.12 | 1.861 | 1.166 | 2.921 | 3.367 | 0.013 | 0.035 | 67.5 |
| 10             | 0.781 | 3600 | 18.856 | 0.21 | 1.211 | 0.775 | 1.774 | 2.196 | 0.032 | 0.052 | 112.5 |
| 20             | 18.886 | 3600 | 19.224 | 0.10 | 1.699 | 1.097 | 2.258 | 2.788 | 0.013 | 0.033 | 72   |
| t              | 0.197 | 3600 | 19.804 | 0.17 | 1.371 | 0.844 | 2.437 | 2.782 | 0.033 | 0.055 | 108  |
| 30             | 20.138 | 3600 | 19.895 | 0.25 | 1.284 | 0.884 | 2.231 | 2.598 | 0.031 | 0.056 | 90   |
| g              | 19.306 | 3600 | 19.024 | 0.15 | 1.704 | 1.057 | 2.464 | 2.965 | 0.015 | 0.031 | 90   |
| Mean           | 19.685 | 3600 | 19.261 | 0.17 | 1.528 | 0.971 | 2.348 | 2.782 | 0.023 | 0.044 | 90   |

TABLE 8. Statistical test for validation of MILP model and MAGL algorithm.

| Algorithm pairs | Small size | Large size |
|-----------------|------------|------------|
|                 | Z | p-value | Z | p-value |
| MAGL vs HVNS    | -2.981* | 0.003 | -3.059* | 0.002 |
| MAGL vs HPSO    | -3.059* | 0.002 | -3.059* | 0.002 |
| MAGL vs MILP    | -1.804* | 0.071 | -3.059* | 0.002 |

For large size instances, it can be seen from Table 7 that the MAGL algorithm significantly surpasses all competitors compared, which demonstrates its excellent ability to solve large-size problems. Whereas for the MILP model,
TABLE 9. Computational results of compared MAGL components.

| Instances | TPHS | MAGL$_{G0}$ | MAGL$_{G1}$ | MAGL$_{SS}$ | MAGL$_{ML}$ | MAGL | Time (sec.) |
|-----------|------|-------------|-------------|-------------|-------------|------|-------------|
| 20/24/6/2 | 20.513 | 5.877 | 5.305 | 5.609 | 4.501 | 4.327 | 4.471 | 2.564 | 0.563 | 0 | 4.8 |
| 20/32/8/2 | 15.167 | 6.304 | 6.743 | 5.913 | 5.404 | 5.398 | 4.672 | 1.285 | 0.707 | 0 | 3.2 |
| 25/24/6/2 | 24.197 | 6.677 | 7.368 | 6.281 | 6.719 | 4.947 | 4.856 | 1.738 | 1.286 | 1.069 | 0.814 | 6 |
| 25/32/8/2 | 27.069 | 7.799 | 7.987 | 6.896 | 6.714 | 7.566 | 6.714 | 2.414 | 3.103 | 0 | 8 |
| 30/24/6/2 | 19.667 | 6.543 | 7.018 | 5.961 | 6.437 | 6.498 | 6.188 | 1.263 | 0.559 | 0.179 | 0 | 3.6 |
| 30/32/8/2 | 23.234 | 8.324 | 8.798 | 7.751 | 8.338 | 7.163 | 8.278 | 2.711 | 1.318 | 0 | 9.6 |
| 100/66/20/6 | 22.352 | 16.788 | 18.001 | 16.134 | 17.092 | 11.848 | 14.941 | 0.756 | 1.445 | 0 | 36 |
| 100/810/30/4 | 22.288 | 14.013 | 14.899 | 13.201 | 14.027 | 13.036 | 13.757 | 0.413 | 0.459 | 0 | 90 |
| 150/66/20/6 | 18.568 | 15.979 | 15.025 | 14.885 | 14.714 | 13.952 | 13.964 | 0.571 | 0.637 | 0 | 90 |
| 150/86/20/4 | 25.983 | 20.002 | 20.197 | 19.301 | 19.152 | 19.936 | 19.167 | 1.509 | 1.524 | 0 | 54 |
| 200/610/20/6 | 15.548 | 16.031 | 15.895 | 14.297 | 13.437 | 12.672 | 12.406 | 0.917 | 0.289 | 0 | 120 |
| 200/86/30/6 | 18.765 | 20.031 | 18.768 | 18.024 | 17.359 | 17.469 | 16.788 | 1.975 | 1.678 | 0 | 705 | 108 |
| Mean | 21.105 | 12.029 | 12.167 | 11.188 | 11.158 | 10.403 | 10.517 | 1.508 | 1.131 | 0.104 | 0.127 | 44.4 |

Table 7 shows that the mean value of $RPD$ reaches 19.685 together with a time cost of 3600 seconds. This $RPD$ value is much larger than that generated from the algorithms compared. It suggests that the MILP model is not effective when solving the DFAPFSP with large size.

Furthermore, the above comparison results are analyzed by using the non-parametric Wilcoxon signed-rank test [45] with confidence level 0.05. It can be seen from Table 8 that these comparisons are statistically significant.

C. VALIDATION OF MAGL COMPONENTS

The MAGL proposed in this paper is a hybrid method consisting of different components. The problem-solving ability of the whole algorithm has been proved. Next, we analyze the effectiveness of its main components. For the purpose, the other four versions of MAGL algorithm are tested, i.e., MAGL$_{G0}$, MAGL$_{G1}$, MAGL$_{SS}$ and MAGL$_{ML}$. MAGL$_{G0}$ is MAGL without any local optimizers and with the original mating operation presented in [40]. MAGL$_{G1}$ is MAGL$_{G0}$ with the improved mating operator presented in this paper. MAGL$_{SS}$ is MAGL only with local search SLS, whereas MAGL$_{ML}$ is MAGL only with local search MLS. We compare them with TPHS and MAGL by 12 different-size instances from the proposed benchmark problems. The comparison results are shown in Table 9.

From Table 9, it can be seen that MAGL$_{G1}$ outperforms TPHS significantly for each instance and each performance metric, which proves that the global exploration optimizer we proposed is very effective. Also, MAGL$_{G1}$ gets smaller $bRPD$ and $aRPD$ values than MAGL$_{G0}$, demonstrating the effectiveness of our improved mating operator. With the application of local search strategies, the performance of MAGL$_{SS}$ and MAGL$_{ML}$ surpasses those of TPHS, MAGL$_{G0}$ and MAGL$_{G1}$ at a considerable margin. It suggests that the devised two local search strategies are of great significance to the MAGL algorithm. Furthermore, from the comparison of MAGL$_{SS}$ and MAGL$_{ML}$, we can draw a conclusion that the local search MLS is more critical than the local search SLS. As expected, MAGL algorithm performs best among all the candidates being compared, the success of which is mainly due to the effectiveness of the MAGL components and their organic hybridization in MAGL. To be specific, the global exploration optimizer is efficient due to the well-designed mechanism: bi-population co-evolution and cross-population interaction. The local search optimizer based on meta-Lamarckian learning strategy can effectively enhance the exploitation in the promising region, while the simplex search-based local search optimizer is helpful to refine the population’s overall quality. The hybridization of the global and local search optimizers within MA framework can well balance the exploration and exploitation abilities. Besides, Table 10 shows that the above comparison results are statistically significant.

TABLE 10. Statistical test for validation of MAGL components.

| Algorithm pairs | Z | p-value |
|-----------------|---|---------|
| MAGL$_{G1}$ vs TPHS | -3.180 | 0.001 |
| MAGL$_{G1}$ vs MAGL$_{G0}$ | -3.180 | 0.001 |
| MAGL$_{SS}$ vs MAGL$_{G1}$ | -2.981 | 0.003 |
| MAGL$_{ML}$ vs MAGL$_{G1}$ | -3.180 | 0.001 |
| MAGL$_{ML}$ vs MAGL$_{SS}$ | -3.180 | 0.001 |
| MAGL vs MAGL$_{ML}$ | -3.180 | 0.001 |

VI. CONCLUSIONS

This paper studies a novel distributed flexible assembly permutation flowshop scheduling problem (DFAPFSP) to minimize makespan criterion. It is a practical and promising research topic due to the successful applications in modern manufacturing. The configuration consists of the distributed flowshop factories for job-processing at the first stage and the flexible assembly machines for product-assembling at the second stage. To solve the DFAPFSP, we first construct a MILP model, and then put forward a memetic
algorithm (MAGL) that has been proved to be very effective based on a comprehensive computational campaign. For the MAGL, the contributions include four aspects: a) Present a global exploration optimizer with bi-population co-evolution and cross-population interaction; b) Present a meta-Lamarckian learning-based local exploitation optimizer to improve the best solution in population; c) Present a simplex search-based local exploitation optimizer to refine the population’s overall quality; and d) Formulate a novel and efficient MA for the DAFPS based on the encoding scheme designed by introducing problem-specific knowledge.

For future research, some aspects deserve our attention. The proposed MA algorithm is efficient to cope with the DAFPS, which inspires us to apply it to more complex scenarios such as restricted buffers, batch processing, multi-objective, heterogeneous factories, and so on. It is also very meaningful to design more sophisticated global and local optimizers based on problem-related knowledge and other excellent optimization technologies.

REFERENCES

[1] V. Santucci, M. Baiololeti, and A. Milani, “Algebraic differential evolution algorithm for the permutation flowshop scheduling problem with total flowtime criterion,” IEEE Trans. Evol. Comput., vol. 20, no. 5, pp. 682–694, Oct. 2016.

[2] J. Cebere, E. Iuruczki, A. Mendibilu, and J. A. Lozano, “A distance-based ranking model estimation of distribution algorithm for the flowshop scheduling problem,” IEEE Trans. Evol. Comput., vol. 18, no. 2, pp. 286–300, Apr. 2014.

[3] F. Zhao, L. Zhang, Y. Zhang, W. Ma, C. Zhang, and H. Song, “A hybrid discrete wave optimization algorithm for the no-idle flow shop scheduling problem with total tardiness criterion,” Expert Syst. Appl., vol. 146, May 2020, Art. no. 113166.

[4] F. Zhao, S. Qin, Y. Zhang, W. Ma, C. Zhang, and H. Song, “A hybrid biogeography-based optimization with variable neighborhood search mechanism for no-wait flow shop scheduling problem,” Expert Syst. Appl., vol. 126, pp. 321–339, Jul. 2019.

[5] F. Zhao, Y. Liu, Y. Zhang, W. Ma, and C. Zhang, “A hybrid harmony search algorithm with efficient job sequence scheme and variable neighborhood search for the permutation flow shop scheduling problems,” Eng. Appl. Artif. Intell., vol. 65, pp. 178–199, Oct. 2017.

[6] J. Behnamian and S. M. T. F. Ghami, “A survey of multi-factory scheduling,” Int. J. Manuf. Res., vol. 27, no. 1, pp. 231–249, Feb. 2016.

[7] B. Naderi and R. Ruiz, “The distributed permutation flowshop scheduling problem,” Comput. Oper. Res., vol. 37, no. 4, pp. 754–768, Apr. 2010.

[8] C.-Y. Lee, T. C. E. Cheng, and B. M. T. Lin, “Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem,” Manage. Sci., vol. 39, no. 5, pp. 616–625, May 1993.

[9] A. Mozdjar, S. M. T. Fatemi Ghomi, F. Jolai, and J. Navaei, “Two-stage flow-shop scheduling problem with non-identical assembly machines considering setup times,” Int. J. Prod. Res., vol. 51, no. 12, pp. 3625–3642, Jun. 2013.

[10] J. Navaei, S. M. T. F. Ghomi, F. Jolai, M. E. Shiraqai, and H. Hidaji, “Two-stage flow-shop scheduling problem with non-identical second stage assembly machines,” Int. J. Adv. Manuf. Technol., vol. 69, nos. 9–12, pp. 2215–2226, Dec. 2013.

[11] S. Hatami, R. Ruiz, and C. Andrés-Romano, “The distributed assembly permutation flowshop scheduling problem,” Int. J. Prod. Res., vol. 51, no. 17, pp. 5292–5308, Sep. 2013.

[12] X. Chen, Y.-S. Ong, M.-H. Lim, and K. C. Tan, “A multi-facet view on memetic computation,” IEEE Trans. Evol. Comput., vol. 15, no. 5, pp. 591–607, Oct. 2011.

[13] F. Neri and C. Cotta, “Memetic algorithms and memetic computing optimization: A literature review,” Swarm Evol. Comput., vol. 2, pp. 1–14, Feb. 2012.

[14] S.-Y. Wang, L. Wang, M. Liu, and Y. Xu, “An effective estimation of distribution algorithm for solving the distributed permutation flow shop scheduling problem,” Int. J. Prod. Econ., vol. 145, no. 1, pp. 387–396, Sep. 2013.

[15] W. Shao, D. Pi, and Z. Shao, “A Pareto-based estimation of distribution algorithm for solving multiobjective distributed no-wait flow shop scheduling problem with sequence-dependent setup time,” IEEE Trans. Autom. Sci. Eng., vol. 16, no. 3, pp. 1344–1360, Jul. 2019.

[16] H. Bargouei, O. Belkahla Driss, and K. Ghédira, “A novel chemical reaction optimization for the distributed permutation flowshop scheduling problem with makespan criterion,” Comput. Ind. Eng., vol. 111, pp. 239–250, Sep. 2017.

[17] I. Ribas, R. Company, and X. Tort-Martorell, “An iterated greedy algorithm for solving the total tardiness parallel blocking flow shop scheduling problem,” Expert Syst. Appl., vol. 121, pp. 347–361, May 2019.

[18] R. Ruiz, Q.-K. Pan, and B. Naderi, “Iterated greedy methods for the distributed permutation flowshop scheduling problem,” Omega, vol. 83, pp. 213–222, Mar. 2019.

[19] W. H. Li, J. Q. Li, and K. Z. Gao, “Solving robotic distributed flowshop problem using an improved iterated greedy algorithm,” Int. J. Adv. Robot. Syst., vol. 12, no. 9, Sep. 2015.

[20] C.-Y. Cheng, K.-C. Ying, H.-H. Chen, and H.-S. Lu, “Minimising makespan in distributed mixed no-idle flowshops,” Int. J. Prod. Res., vol. 57, no. 1, pp. 48–60, Jan. 2019.

[21] B. Naderi and R. Ruiz, “A scatter search algorithm for the distributed permutation flowshop scheduling problem,” Eur. J. Oper. Res., vol. 239, no. 2, pp. 323–334, Dec. 2014.

[22] G. Zhang and X. K. Ding, “Differential evolution metaheuristics for distributed limited-buffer flowshop scheduling with makespan criterion,” Comput. Oper. Res., vol. 108, pp. 33–43, Aug. 2019.

[23] G. Zhang, K. Xing, and F. Cao, “Discrete differential evolution algorithm for distributed flowshop scheduling with makespan criterion,” Eng. Appl. Artif. Intell., vol. 76, pp. 96–107, Nov. 2018.

[24] S.-Y. Wang and L. Wang, “An estimation of distribution algorithm-based memetic algorithm for the distributed assembly permutation flow shop scheduling problem,” IEEE Trans. Syst., Man, Cybern., Syst., vol. 46, no. 1, pp. 139–149, Jan. 2016.

[25] J. Lin and S. Zhang, “An effective hybrid biogeography-based optimization algorithm for the distributed assembly permutation flow shop scheduling problem,” Comput. Ind. Eng., vol. 97, pp. 128–136, Jul. 2016.

[26] J. Lin, Z.-J. Wang, and X. Li, “A backtracking search hyper-heuristic for the distributed assembly flow shop scheduling problem,” Swarm Evol. Comput., vol. 36, pp. 1–16, Sep-Oct. 2017.

[27] H.-Y. Sang, Q.-K. Pan, J.-Q. Li, P. Wang, Y.-Y. Han, K.-Z. Gao, and P. Duan, “Effective invasive weed optimization algorithms for distributed assembly permutation flowshop problem with total flowtime criterion,” Swarm Evol. Comput., vol. 44, pp. 64–73, Feb. 2019.

[28] D. Ferone, S. Hatami, E. M. González-Neira, A. A. Juan, and P. Festa, “A biased-randomized iterated local search for the distributed assembly permutation flowshop scheduling problem,” Int. Trans. Oper. Res., vol. 16, no. 5, pp. 1361–1368, Aug. 2020.

[29] H. Ochi and O. B. Driss, “Scheduling the distributed assembly flowshop problem to minimize the makespan,” Procedia Comput. Sci., vol. 164, pp. 471–477, Jul. 2019.

[30] G. Zhang, K. Xing, and F. Cao, “Scheduling distributed flowshops with flexible assembly and set-up time to minimise makespan,” Int. J. Prod. Syst., vol. 56, no. 9, pp. 3226–3244, May 2018.

[31] G. Zhang and K. Xing, “A biogeography optimization algorithm for scheduling two-stage assembly flowshop in a distributed environment,” Comput. Ind. Eng., vol. 125, pp. 423–433, Nov. 2018.

[32] H. Ishibuchi, T. Yoshida, and T. Murata, “Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling,” IEEE Trans. Evol. Comput., vol. 7, no. 2, pp. 204–223, Apr. 2003.

[33] N. Vin and A. Ghosh, “A memetic differential evolution algorithm for energy-efficient parallel machine scheduling,” Omega, vol. 32, pp. 155–165, Jan. 2015.

[34] B. Liu, L. Wang, and Y.-H. Jin, “An effective PSO-based memetic algorithm for flowshop scheduling,” IEEE Trans. Syst., Man Cybern., B, Cybern., vol. 37, no. 1, pp. 18–27, Feb. 2007.

[35] Q.-K. Pan, L. Wang, H.-Y. Sang, J.-Q. Li, and M. Liu, “A high performing memetic algorithm for the flowshop scheduling problem with blocking,” IEEE Trans. Autom. Sci. Eng., vol. 10, no. 3, pp. 741–756, Jul. 2013.
[36] B. Qian, L. Wang, D.-X. Huang, and X. Wang, “Scheduling multi-objective job shops using a memetic algorithm based on differential evolution,” *Int. J. Adv. Manuf. Technol.*, vol. 35, nos. 9–10, pp. 1014–1027, Jan. 2008.

[37] M. A. González and C. R. Vela, “An efficient memetic algorithm for total weighted tardiness minimization in a single machine with setups,” *Appl. Soft Comput.*, vol. 37, pp. 506–518, Dec. 2015.

[38] W. Shao, D. Pi, and Z. Shao, “Memetic algorithm with node and edge histogram for no-idle flow shop scheduling problem to minimize the makespan criterion,” *Appl. Soft Comput.*, vol. 54, pp. 164–182, May 2017.

[39] J. Deng and L. Wang, “A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem,” *Swarm Evol. Comput.*, vol. 32, pp. 121–131, Feb. 2017.

[40] E. Cuevas, M. Cienfuegos, D. Zaldívar, and M. Pérez-Cisneros, “A swarm optimization algorithm inspired in the behavior of the social-spider,” *Expert Syst. Appl.*, vol. 40, no. 16, pp. 6374–6384, Nov. 2013.

[41] Y. S. Ong and A. J. Keane, “Meta-lamarckian learning in memetic algorithms,” *IEEE Trans. Evol. Comput.*, vol. 8, no. 2, pp. 99–110, Apr. 2004.

[42] Y. Zhou, Y. Zhou, Q. Luo, and M. Abdel-Basset, “A simplex method-based social spider optimization algorithm for clustering analysis,” *Eng. Appl. Artif. Intell.*, vol. 64, pp. 67–82, Sep. 2017.

[43] V. Fernandez-Viagas, P. Perez-Gonzalez, and J. M. Framinan, “The distributed permutation flow shop to minimise the total flowtime,” *Comput. Ind. Eng.*, vol. 118, pp. 464–477, Apr. 2018.

[44] D. C. Montgomery, *Design and Analysis of Experiments*. Hoboken, NJ, USA: Wiley, 2005.

[45] S. García, D. Molina, M. Lozano, and F. Herrera, “A study on the use of non-parametric tests for analyzing the evolutionary algorithms’ behaviour: A case study on the CEC’2005 special session on real parameter optimization,” *J. Heuristics*, vol. 15, no. 6, pp. 617–644, Dec. 2009.

**GUANGHUI ZHANG** received the Ph.D. degree in control science and engineering from Xi’an Jiaotong University, Xi’an, China, in 2018. He is currently an Associate Professor with the School of Information Science and Technology, Hebei Agricultural University, Baoding, China. His research interests include intelligent optimization and scheduling algorithms.

**KEYI XING** received the B.S. degree in mathematics from Northwest University, Xi’an, China, in 1982, the M.S. degree in applied mathematics from Xidian University, Xi’an, in 1985, and the Ph.D. degree in systems engineering from Xi’an Jiaotong University, Xi’an, in 1994. In 1985, he was with Xidian University. Since 2004, he has been with Xi’an Jiaotong University, where he is currently a Professor of systems engineering with the State Key Laboratory for Manufacturing Systems Engineering and with the Systems Engineering Institute. His research interests include control and scheduling of automated manufacturing, discrete-event, and hybrid systems.

**GUANGYUN ZHANG** received the B.S. degree from the China University of Mining and Technology, Xuzhou, China, in 2007, and the Ph.D. degree in electrical engineering from the University of New South Wales, Sydney, Australia, in 2013. He has been with Nanjing Tech University, where he is a Professor. His research interests include imaging spectrometry, control, and scheduling of hybrid systems.

**ZHENXUE HE** received the Ph.D. degree in computer architecture from Beihang University, Beijing, China, in 2018. He is currently an Associate Professor with Hebei Agricultural University. His research interests include low power integrated circuit design and optimization, multiple-valued logic circuits, and intelligent algorithms.