IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 9 November 2022, accepted 24 November 2022, date of publication 28 November 2022,
date of current version 2 December 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3225448

RESEARCH ARTICLE

Meta-Heuristic Algorithms for the Generalized Extensible Bin Packing Problem With Overload Cost

RAN DING, BIN DENG, AND WEIDONG LI
School of Mathematics and Statistics, Yunnan University, Kunming 650504, China
Corresponding author: Bin Deng (dengbin96@126.com)

This work was supported by the Postgraduate Research and Innovation Foundation of Yunnan University under Grant 2021Y325.

ABSTRACT In this paper, we consider a generalized extensible bin packing problem with overload cost, first proposed by Denton et al. in 2010, in which the total size of items packed into a bin is allowed to exceed its capacity, and the cost incurred each bin is equal to the fixed cost plus the overload cost, the objective is to minimize the total cost of all bins. According to the characteristics of the problem, we first propose an improved ant colony optimization algorithm (IACO), which enhances the positive feedback effect of ACO by improving the update method of pheromone and the adaptive adjustment parameters. We also introduce a variable neighborhood search method in ACO to improve the convergence of the algorithm and get rid of the phenomenon of local extrema. Then, we present a discrete particle swarm optimization algorithm (DPSO) to solve the problem. In order to ensure the uniform distribution and high quality of the initial particle swarm, we use some heuristic methods in the initialization process of the swarm, so that the initial particle can cover the entire search space with a large probability, which effectively improves the performance of DPSO algorithm. Finally, we compare and analyze the performance of these proposed algorithms through two sets of computational experimental frameworks. Compared with some algorithms in the literature, computational results signify that the improved ACO algorithm and MDPSO algorithm are more competitive than some other metaheuristic algorithms.

INDEX TERMS Extensible bin packing, regular working time, scheduling, ant colony optimization algorithm, discrete particle swarm optimization.

I. INTRODUCTION

A. RESEARCH MOTIVATION

In a multiprocessor system, tasks in an order need to be scheduled to the processors in the shortest possible time. Each processor has a given capacity and is charged a fixed power-on/machine-wear cost. However, an overload cost will be charged if the total size of tasks scheduled to it exceeds the processor's service capacity. We aim to minimize the total cost of processing all tasks. Based on the application background in production and life, we consider the load balancing problem on identical parallel machines in this paper, called the *generalized extensible bin packing problem with overload cost* (GEBPOC). This problem was first proposed by Denton et al. [1], [2], who were inspired by a healthcare delivery application in outpatient surgery centers that dynamically assign patients to operating rooms, with the goal of minimizing the total cost of opening an operating room and using overtime to complete a day of surgery when both the number of scheduled patients and the duration of the procedure are unknown.

The associate editor coordinating the review of this manuscript and approving it for publication was Christian Pilato.

B. MODEL DESCRIPTION

$$\min \quad totalcost = mt + c \cdot \sum_{i=1}^{m} \max\{\sum_{j=1}^{n} p_j x_{ij} - t, 0\} \quad (1)$$

$$s.t. \sum_{i=1}^{m} x_{ij} = 1, \ \text{for } j = 1, 2, \cdots, n. \quad (2)$$

$$x_{ij} = \begin{cases} 1, & \text{if job } J_j \text{ is assigned to machine } M_i \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

C. LITERATURE REVIEW

It can be known from literature [3] that GEBPOC is also NP-hard in the strong sense, so it is unlikely to obtain the optimal schedule through polynomial time-bounded algorithms. Over the years there has been a great deal of research to develop efficient approaches for the problem in terms of performance guarantee. In GEBPOC, if $c = 1$, our problem is equivalent to extensible bin packing with unequal bin sizes (EBP-UBS), where an inexhaustible set of unequal bins are given. Unlike the traditional bin packing, a bin's capacity in EBP-UBS is allowed to be exceeded if needed. In addition, in the study of EBP-UBS, many researchers regard the capacity of each bin and the fixed cost as unitary, or the unit cost of the part beyond the bin is also 1. [4] considered this type of problem for the first time, and they studied both offline and online versions under the assumption that the largest item size does not exceed the smallest bin size. They shown that the worst-case ratio of the longest processing time first algorithm (LPT) is $4 - 2\sqrt{2}$ in the offline case, and that for the online version, the list scheduling algorithm (LS) has an upper bound of $\frac{5}{4}$, and this bound is tight. They also proved that any online algorithm has an approximation ratio is not less than $\frac{7}{6}$. [5] studied the online versions of this problem for $m = 2, 3, 4$, and proved that the designed online algorithm obtains tight bounds of $\frac{7}{6}$, $\frac{11}{9}$, $\frac{19}{16}$ for these three cases, and when $m = 2$, this algorithms are the best possible. They also given an improved algorithm for $m = 3$. The vector scheduling problem in asymmetric settings was studied by [6], who gave a polynomial time approximation scheme (PTAS) for the EBP-UBS using dynamic programming techniques, where the state space is a vector that depends on the dimension of $\frac{1}{\varepsilon}$.

When $c = t = 1$ in GEBPOC, the extensible bin packing problem (EBP) is formed. In [3], Dell'Olmo et al. proved that the worst-case ratio of the LPT algorithm is $\frac{13}{12}$. When the number of bins m is fixed, according to the discussion in [7], there is a fully polynomial time approximation scheme (FPTAS) for this problem. If m is not fixed, EBP can be solved by using the (efficient) PTAS idea for the identical machine scheduling problem in [8]. The online version of EBP is proved by [9] using the LS algorithm that both its upper and lower bounds are equal to $\frac{5}{4}$, and a heuristic method that depends on the parameter $x(0 < x < 1)$ is designed to improve this bound, the new algorithm H_x assigned tasks to machines when a machine's load is less than or equal to H_x, and its worst-case ratio is equal to 1.228. For the lower bound of the online problem, they also found an instance in the case of $m = 2$ to proved that the competitive ratio of any online algorithm will not be less than $\frac{7}{6}$. A fully polynomial time asymptotic approximation scheme (FPTAAS) for this problem was developed by [10]. On the other hand, if $t = 1$ in GEBPOC, [11] investigated the bin packing problem with overload cost (BPOC), where the number of identical bin is infinite, and they presented the lower and upper bounds of any deterministic online algorithm for BPOC according to the value of c. [12] considered the more general case of EBP, *i.e.,* the generalized extensible bin packing problem (GEBP), where $c = \sigma_i$ and the capacity of all machines is allowed to be different. They developed an EPTAS based on using the shifting technique followed by a solution of a polynomial number of n-fold programming instances. When the number of machines of each type is not part of the input but part of the solution, they presented an asymptotic fully polynomial time approximation scheme (AFPTAS) for a related bin packing type variant of the problem (denoted by GEBP-BPV) similar to the variant of EBP.

In the context of surgical scheduling, inspired by the practical application of dynamically assigning patients to operating rooms in outpatient procedure centers, [1] considered the more general case of the EBP problem, where the decision maker needs to choose the number of bins of size S to be opened. The fixed cost of each opened bin is c^f, and the overtime cost per unit time is c^v. The goal is to minimize the total cost of opening an operating room and working overtime to complete a day of surgery. Afterwards, [2] investigated some faster approximation algorithms for solving the online version of the problem (DEBP), and they showed that every $(1 + \rho)$-approximation algorithm for the EBP problem produces a $(1 + \rho \cdot \frac{Sc^v}{c^f})$-approximation algorithm in this more general setting. They also considered a two-stage randomized version of the problem, in which emergency patients need to be assigned to the operating room along with preassigned elective patients. This is also the first attempt to account for random cases in the EBP problem. Based on this stochastic setting, [13] studied the stochastic extensible bin packing problem (SEBP).

There is a quite some literature that focuses on the case of where the fixed cost of each bin is equal to 0, called the late work minimization scheduling problem, which is equivalent to the early work maximization problem when considering optimal solutions. The early work denotes a part of a job executed before a due date, while the late work represents a part of a job executed after a due date. Late work minimization problem was first proposed by [14]. For the offline late work minimization problem, [15] proved that is binary NP-hard on two identical machines, and they designed an optimal online algorithm with competitive ratio of $\sqrt{5} - 1$. For the offline early work maximization problem, [16] proposed a polynomial time approximation scheme on two identical machines. [17] proved that the LPT algorithm has a worst case ratio is $\frac{10}{9}$ on two identical machines, and proposed a branch-and-bound algorithm for the general case with arbitrary due date. When there are m machines, [18] proposed a pseudo-polynomial time dynamic programming algorithm and a fully polynomial time approximation scheme. Recently, [19] proposed a more efficient new dynamic programming algorithm and a FPTAS for the problem in [18]. Reference [20] presented an efficient polynomial time approximation scheme for the problem in [18]. Reference [21] considered four semi-online scheduling problems with a common due date to maximize the total early work. Next, [22] studied several online and semi-online early

work maximization problems on two hierarchical machines. Reference [23] also considered three semi-online early work maximization problems on two hierarchical machines with partial information of processing time.

The above algorithms will expose the characteristics of poor solution accuracy for large scale problem instances. The rise of some meta-heuristic algorithms has provided new research and development directions for solving instances of combinatorial optimization problems such as large scale identical machine scheduling. In the pursuit of obtaining high-quality solutions within an effective time frame, meta-heuristics play an indispensable role. It defines a series of processing frameworks based on natural laws, biology and other phenomena, which can be used to solve any optimization problem. In recent years, it has aroused great research interest of scholars. They have successively applied simulated annealing (SA) [24], particle swarm algorithm (PSO) [25], ant colony optimization algorithm (ACO) [26], and genetic algorithm (GA) [27] to solve the identical machine scheduling problem, and achieved certain results. When solving a particular problem through a meta-heuristic algorithm, it is necessary to define the expression of the problem. Then the feasible solution set is iterated based on the initial solution, the bad solutions are eliminated and the high-quality solutions are retained. Once the algorithm satisfies the termination condition, the best solution currently found is output. Note that the solution obtained by these algorithms is not necessarily optimal, but must be a feasible solution of great quality and performance.

An ant colony algorithm for the no-wait flow shop scheduling problem with the goal of minimizing makespan is considered by [28]. Some literatures (*e.g.,* [29], [30], [31], [32], [33], and [34]) studied the application of ACO algorithm to single-machine scheduling problem with tardiness penalty. Later, [35] considered the scheduling problem of minimizing maximum tardiness with m identical machines, and they applied an ant colony algorithm with four different specific heuristics in the construction of solution. As we all know, there are many parameters in the ACO algorithm, and different parameter settings will lead to different results. Reference [36] considered the setting of ACO's parameters as a combinatorial optimization problem, and they solved the problem by the PSO algorithm and proposed an adaptive parameter setting strategy. In addition, [37] proposed three population solving algorithms for the problem of fair resource allocation: discrete artificial bee colony algorithm, discrete artificial fish swarm algorithm and discrete hybrid frog leaping algorithm, and verified the effectiveness of these algorithms through computational experiments. Chen et al. [38] used a DPSO algorithm to consider the minimizing total late work scheduling problem in a flow shop system with different due dates and learning effects. Reference [39] considered the two criteria of fairness and cost respectively for the crew scheduling problem, and proposed an improved honey badger optimization algorithm to solve this problem through genetic algorithm and Levy flight. Compared with other algorithms, the algorithm has good performance.

Reference [40] proposed a DPSO algorithm for minimize makespan criterion, and they also investigated the effectiveness of hybrid DPSO by this algorithm with an efficient local search heuristic. Reference [41] considers a no-wait flow shop scheduling problem with the goal of minimizing makespan and total flow time. They proposed a hybrid DPSO algorithm related to variable neighborhood search, and explored the selection method of control parameters and the effect of embedding variable neighborhood search on the optimization performance of the algorithm. Reference [42] used particle swarm optimization for parameter optimization related to improving the ability of soil surface process models to simulate soil moisture. They also used a particle swarm optimization algorithm in [43] to calibrate parameters related to turbulence in the surface layer in the source region of the Yellow River. Reference [44] takes the total weighted earliness and tardiness penalty as the optimization goal under the condition of common due date on single machine environment, and proposes a DSPO algorithm, and improves the local search ability of the algorithm and jumps out of the local extreme value ability by embedding variable neighborhood search. Reference [45] proposed a discrete DPSO with new information sharing mechanism for minimizing the makespan problem. Reference [46] proposed a discrete DPSO based on genetic algorithm crossover operator and mutation operator, and compared the effectiveness of several crossover and mutation operators. Taking into account the objectives of makespan and total flow time. Reference [47] considered the effect of artificial intelligence particle swarm optimization method on the calibration of freeze-thaw related parameters in the improvement of climate-vegetation model freeze-thaw process. Reference [48] proposed a combined particle swarm optimization (CPSO) and employs a simulated annealing algorithm to enhance the ability of CPSO to get rid of local extrema value. For the job shop scheduling problem, [49] presented a hybrid DPSO with the tabu search algorithm for the makespan minimization problem. Reference [50] presented the DPSO for solving the problem of minimizing the total weighted earliness and tardiness time, and confirmed that this algorithm outperforms the research results of related literature.

Compared with other heuristic algorithms, the ACO algorithm has strong robustness in solving performance, that is, the basic ACO algorithm model can be applied to solve the problem with a slight modification according to the specific characteristics of a optimization problem. Moreover, the DPSO algorithm has the advantages of relatively fast approaching the optimal solution, effectively optimizing the parameters of the system, and strong robustness. Therefore, in this research, we solve GEBPOC based on ACO and DPSO algorithms, and use the LPT algorithm as a base comparator algorithm to evaluate our results. To the best of our knowledge, this is the first attempt to solve this problem using a population algorithm.

D. MAIN RESULTS

The main contributions of this paper are mainly summarized as follows.

1) According to the characteristics of the problem, the corresponding ant colony model is defined, and the positive feedback effect of ant colony optimization is enhanced by improving the state transition rules and dynamic adaptive parameters. In order to avoid the premature or stagnant phenomenon of the ant colony algorithm in the search, the variable neighborhood search method is also introduced to improve the ant colony algorithm, and further improve the global search ability and convergence speed of the algorithm.
2) The DPSO algorithm is a new computing technology based on swarm intelligence theory to solve many discrete optimization problems. In order to ensure the uniform distribution and high-quality characteristics of the initial population, some heuristic methods are adopted in the initialization process of the particle swarm, so that the initial particle can cover the entire search space with a high probability, and avoid inefficiency caused by blind search.
3) A large number of computational experiments are designed, and the results of the proposed meta-heuristic algorithm and LPT algorithm in terms of performance ratio, running time, function value, Friedman rank and convergence are compared, which shows that these meta-heuristic algorithms have good stability and strong optimization ability when solving large-scale instances.

The reminder of the paper is organized as follows: In the next section, we briefly introduce the concepts and ideas of ant colony optimization and particle swarm optimization. Section 3 introduces the improved ant colony algorithm for GEBPOC. Section 4 presents a discrete particle swarm optimization algorithm for GEBPOC. Section 5 compares and analyzes the results and effectiveness of the proposed algorithm through computational experiments. A summary of this paper and an outlook for future research are presented in Section 6.

II. BASIC ACO AND PSO

A. BASIC ACO

Ant colony optimization algorithm is a population-based evolutionary algorithm proposed by Dorigo et al. [26] by simulating the trail-finding method of ants' foraging behavior in nature. As shown in Fig. 1, the behavior of real ants generating near-optimal trails can be explained by four steps ($a \longrightarrow b \longrightarrow c \longrightarrow d$). During the movement of the ant colony, it can leave pheromone substances on the trail for information transmission, and the ants can perceive this substance and guide their movement direction. When there are many ants foraging, each ant will randomly choose a trail at the beginning and release pheromone in the trail. Ants with a short trail will reach the destination earlier than ants with a long trail, and the frequency of round trips will also be faster, the pheromone left on this trail will be correspondingly more concentrated. But pheromones also evaporate over time. When the next generation of ants forage, they will choose the trail of pheromone concentration, and the more ants who choose this trail, will release more pheromone. Therefore, the behavior of the ant colony composed of a large number of ants will show a positive information feedback phenomenon: the more ants walking on a certain trail, the greater the probability of the latecomers to choose this trail. ACO has the characteristics of distributed computing, positive feedback of information and heuristic search, and is essentially a heuristic global optimization algorithm in evolutionary algorithms.

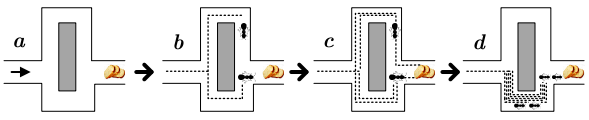


FIGURE 1. Dynamic behavior of basic ant colony optimization, and the gray areas represent the number of pheromones on each trail.

ACO was originally proposed for the traveling salesman problem (TSP) problem. The following introduces the basic ant colony system model by taking the traveling salesman problem (TSP) as an example. Let m and n be the number of ants in the ant colony and the number of cities in the TSP problem, respectively, d_{ij} represents the distance between city i and city j, the number of ants in city j at time z is represented by $b_j(z)$. $\tau_{ij}(z)$ represents the residual pheromone concentration on the trail from city i to city j at time z. In ACO, the walking trail of ants represents a feasible solution of the optimization problem, and all trails of the whole ant colony constitute a solution space of the problem. The general procedure of the ACO algorithm is as follows:

1) Set the number of ant populations according to the specific problem, and assume that the pheromone concentration on each trail is equal at the initial moment, *i.e.,* $\tau_{ij}(0) = R$ (R is a constant), and then search in parallel. After each ant completes a trip, it will release pheromones on the trail, and the pheromone is proportional to the quality of problem solution.
2) Construct the trail. This step includes the selection of the initial city and the determination of the next arrival city. Each ant randomly selects a city as its starting point, and maintains a trail tabu table to store the cities the ants pass through in sequence. During the movement of ant k ($k = 1, \cdots, m$), the trail selection adopts a random local search strategy, and the direction of transfer is determined by the pheromone concentration on each trail. The probability of ant k transferring from city i to city j at time z is represented by $p_{ij}^k(z)$, *i.e.,*

$$p_{ij}^k(z) = \begin{cases} \frac{[\tau_{ij}(z)]^\alpha [\eta_{ij}z]^\beta}{\sum_{s \in allowed_k} [\tau_{is}(z)]^\alpha [\eta_{is}z]^\beta}, & j \in allowed_k, \\ 0, & \text{otherwise.} \end{cases} \quad (4)$$

where α is a importance factor of the residual pheromone on the trail (i, j), β is the importance of

the information transferred from city i to city j, and η_{ij} is the prior knowledge, generally taking $\eta_{ij} = \frac{1}{d_{ij}}$, which means that the closer cities are more likely to be selected. Unlike the ant colony in actual life, the artificial ant colony system has a memory function. The set of cities that ant k has walked through is recorded by $tabu_k$, which is called *tabu table*, and it will make dynamic adjustments with the evolution process. And $allowed_k$ represents the city set that ant k is allowed to transfer in the next step, that is, the complement of $tabu_k$.

3) After n moments, all ants have completed the traversal of n cities, then set $tabu_k$ to empty, calculate the length of the trail traveled by each ant, and write the shortest trail. All ants start the next round of search and traversal from the previous starting point.
4) In order to avoid too much pheromone causing the residual pheromone to drown the heuristic information, one need to update the residual pheromone on the trail after each ant walks a step or traverses n cities. Note that ants of the same generation are not affected by the pheromones left by previous ants. The process of pheromone renewal includes both the evaporation of previous pheromone and the increase of pheromone on the trail traversed. The pheromone update formula is as follows:

$$\tau_{ij}(z+1) = (1-\rho)\cdot\tau_{ij}(z) + \sum_{k=1}^{m}\Delta_{ij}^{k}(z), \tag{5}$$

$$\Delta_{ij}^{k}(z) = \begin{cases} \frac{Q}{L_k}, & \text{if ant } k \text{ moves from city } i \text{ to city } j, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

where $0 \le \rho \le 1$ is the pheromone evaporation coefficient, $\Delta_{ij}^{k}(z)$ represents the pheromone increment left by the kth ant on the trail (i, j) in the current iteration, Q is a constant, and L_k denotes the distance traveled by the kth ant in the current iteration trail length.

5) When the algorithm reaches a predetermined maximum number of iterations or a stagnant state occurs, the algorithm terminates and outputs the shortest trail found so far.

B. BASIC PSO

The particle swarm optimization algorithm was proposed by Kennedy and Eberhard [25] and is one of the latest meta-heuristic algorithm based on population intelligence for optimizing continuous nonlinear functions. Its biological inspiration is based on the metaphor of social interaction and communication in a flock of birds or school of fishes. It is simple and easy to implement, requires few parameters to be adjusted, and has the characteristics of strong global convergence ability and robustness. PSO algorithm has been widely used in function optimization, neural network training, fuzzy system control and other fields.In the PSO algorithm, individuals are regarded as particles with positions and velocities, where the particle's position represents a feasible candidate solution to the problem. Starting from the initial population, the particles fly continuously in the search space, each particle searches for the optimal solution in the search space individually, and records it as the current individual extreme value, and then shares it with other particles in the entire particle swarm. Finally, the optimal individual extreme value found is used as the current global optimal solution of the entire particle swarm. All particles in the particle swarm adjust their speed and position according to the current individual extremum found by themselves and the current global optimal solution shared by the entire particle swarm to gradually approach the optimal solution.

Let $v_k = (v_{k1}, v_{k2}, \cdots, v_{kn})$ and $x_k = (x_{k1}, x_{k2,\cdots,x_{kn}})$ be the velocity and position of particle k, respectively, and $pB_k = (pb_{k1}, pb_{k2}, \cdots, pb_{kn})$ and $gB = (gb_1, gb_2, \cdots, gb_n)$ denote the optimal position of the individual and the population, respectively. Then in the $(z+1)$th round of iteration, the update rule for the velocity and position of particle k is as follows:

$$v_k^{z+1} = wv_k^k + c_1r_1(pB_k^z - x_k^z) + c_2r_2(gB^z - x_k^z), \tag{7}$$

$$x_k^{z+1} = x_k^z + v_k^{z+1}. \tag{8}$$

where w is the inertia factor weight, which reflects the influence of the particle's original speed on the next speed. The constants c_1 and c_2 are called cognitive coefficients and social coefficients, respectively, which reflect the extent to which particles are affected by their own optimal solution and the optimal solution of population. Both r_1 and c_2 are random numbers in the interval [0, 1], pB_k^z represents the personal best solution found by particle k in first z rounds of iterations, and gB^z represents the global best solution obtained by all particles in first z rounds of iterations. The procedure of the basic PSO algorithm is shown in Fig. 2.

In fact, particle swarm optimization (PSO) is a new evolutionary algorithm (EA). Similar to the genetic algorithm, PSO also starts from a random solution, finds the optimal solution through iteration, and evaluates the quality of the solution through fitness, but it is simpler than the genetic algorithm rule. PSO does not have the ''crossover'' and ''mutation'' operations of the genetic algorithm. It can find the global optimal solution by following the currently searched local optimal solution. As we all know, the genetic algorithm first needs to encode the problem, and after finding the optimal solution, it needs to decode the problem. In addition, many parameters are needed in the implementation of the three operators of selection, crossover and mutation, such as crossover rate and mutation rate, and the choice of these parameters is mostly based on experience, which will seriously affect the quality of the solution.

Compared with genetic algorithm, PSO has the advantages of simple and easy to implement process, less parameters to be adjusted, stronger convergence ability and robustness.

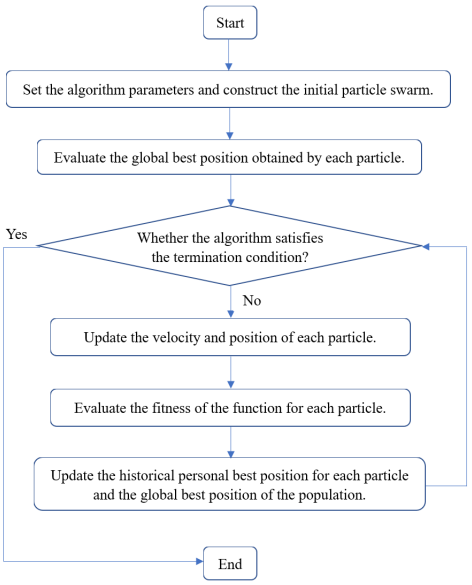


FIGURE 2. Program diagram of basic particle swarm optimization algorithm.

Since GEBPOC is strongly NP-hard, the optimal solution can be found in a reasonable time for small-scale problem instances, while the computational time of the exact algorithm explodes when the problem is large. In this paper, we will design a DPSO algorithm based on the discrete characteristics of the problem to find near-optimal or optimal solutions with acceptable time and memory requirements.

III. IMPROVED ANT COLONY OPTIMIZATION ALGORITHM FOR GEBPOC

Identical machine scheduling problem and bin packing problem are typical combinatorial optimization problems. They are obviously different from TSP problems. Therefore, some adjustments and modifications must be made to the representation and update method of pheromone in the basic ACO algorithm according to the characteristics of the problem, in order to make it suitable for solving specific problems. In this section, we define the ant colony model of GEBPOC according to the characteristics of this problem, and enhance the positive feedback effect of ACO by improving the state transition rules and dynamic adaptive parameters. In order to avoid the premature or stagnant phenomenon of the ant colony algorithm in the search, we also introduced the variable neighborhood search method to improve the ACO to further enhance the global search ability and convergence speed of the algorithm.

A. CONSTRUCTION OF ANT COLONY MODEL

Suppose there are d ants, m machines and n jobs in the system. Each step an ant takes to select a target machine for a job, the ant completes a tour after n times, that is, the schedule process of n jobs is completed, thereby obtaining a scheduling scheme for this problem, which is represented by an n-dimensional vector $(d_1, d_2, \cdots, d_n)$, where d_j $(1 \leq d_j \leq m)$ denotes that job J_j is executed by machine M_{d_j} For example, when $m = 2$ and $n = 4$, the vector $(1, 2, 2, 1)$ corresponds to a feasible schedule, which means that both jobs J_1 and J_4 are assigned to machine M_1, while both J_2 and J_3 are scheduled on M_2.

For GEBPOC, if the load of each machine is greater than or less than the regular working time t, then such schedule is the optimal solution of this problem, and the corresponding objective value is the lower bound LB of this problem, *i.e.,*

$$totalcost \geq \max\{mt, mt + c \cdot \sum_{i=1}^{m} \max\{t_i - t, 0\}\} = LB. \tag{9}$$

In scheduling problems, the quality of the solution is generally evaluated by fitness. According to the characteristics of the problem, we need to consider not only the overtime of each machine, but also the longest machine load in the solution space, so we can define the fitness function as

$$F(sol) = \max\{t_i(sol)\} + \sum_{i=1}^{m} |t_i - t|\}. \tag{10}$$

The ACO algorithm finally outputs the optimal schedule corresponding to the solution with the minimal fitness value searched by the ant colony after many iterations.

B. PHEROMONE EXPRESSION FOR GEBPOC

In GEBPOC, how to reasonably express and store pheromone is the key to the realization of ant colony algorithm. Since the performance of each machine in the problem is identical, jobs are processed regardless of which machine executes it, they are only affected by those jobs that are assigned to the same machine. In this paper, we refer to this property as a job's matching degree (denoted as τ_{ij}) and store it as the pheromone concentration to guide the ants in choosing a appropriate machine for jobs.

If the job set that has been processed on the machine M_i is S_i, then for the next new job J_j, whether it can also be arranged to be processed on M_i needs to be selected with reference to the pheromone left by the ants and the state transition probability. We denote the average degree of matching between job J_j and the assigned job subset S_i on machine M_i at time t (*amount of information*) as $\sigma_{ij}(z)$, *i.e.,*

$$\sigma_{ij}(z) = \begin{cases} \sum_{j:J_j \in S_i} \frac{\tau_{ij}(z)}{|S_i|}, & S_i \neq \emptyset, \\ \frac{1}{m}, & otherwise. \end{cases} \tag{11}$$

where $|S_i|$ refers to the number of jobs that have been processed on machine M_i.

C. STATE TRANSITION STRATEGY

In order to better select a suitable machine, ant k needs to select a machine M_i for job J_j to process it according to the random probability rule of (4):

$$s = \begin{cases} \arg\max_{s \in allowed_k} \tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}, & q \leq q_0, \\ p_{ij}(z), & otherwise. \end{cases} \tag{12}$$

$$p_{ij}(z) = \begin{cases} \frac{[\sigma_{ij}(z)]^{\alpha}[t/(t-(t_i+p_j))]^{\beta}}{\sum_{r=1}^{m}[\sigma_{ri}(z)]^{\alpha}[[t/(t-(t_r+p_j))]^{\beta}}, & t_i + p_j \le t, \\ 0, & \text{otherwise.} \end{cases} \quad (13)$$

where $q_0 = \frac{\log N_c}{\log N_{max}} \in [0, 1]$ is the adaptive threshold of the ACO algorithm (N_c is the current number of iterations of the algorithm, N_{max} is the preset maximal number of iterations), q is a uniformly distributed random number in interval [0, 1]. The state transition probability of the ant at time z is represented by $p_{ij}(z)$, which reflects the probability that the ant assigns the job J_j to M_i at time z according to the residual pheromone and heuristic information during the search process. It can be concluded from the above rules that when $q \le q_0$, the ant can choose the next point according to the previous knowledge, otherwise it will choose the next point according to the random probability. The prior knowledge takes into account the load of the current machine. When the load of the machine M_i is small, the probability of the machine being selected is relatively large, and vice versa. Once the machine load exceeds the regular working time t, the probability of M_i being selected is almost zero. (Note that the value of q_0 is very small at the beginning of the iteration, which implies that the ants will randomly search to ensure that the search space is large enough. As the value of q_0 increases gradually, the ants will conduct deterministic search with a large probability, and then select the trail traveled by the elite ants, so that the ants gradually approach the optimal solution area.)

D. VARIABLE NEIGHBORHOOD SEARCH

In the ant colony algorithm, the search process often falls into a local optimum phenomenon. For this reason, we introduce a variable neighborhood search algorithm into ACO to search the optimal solution generated by the ant colony in each iteration by multiple neighborhood structures. Thus, the search efficiency of ACO is improved. For GEBPOC, the selection of the neighborhood structure needs to consider the permutation of jobs on different machines and the machine with the largest load. In this paper, we present the following three neighborhood structures (*e.g.* $m = 2, n = 5, t = 7$).

- **Move.** In the solution π, the job J_3 processed on M_2 is moved to M_1 for execution, and a new solution π_1 is generated (cf. Fig. 3(a)).
- **Symmetric swap. (in short, *Swap*)** In solution π, a new solution π_2 is generated by exchanging the job J_2 on M_1 with a job J_4 on M_2 (cf. Fig. 3(b)).
- **Asymmetric swap. (in short, *Aswap*)** In the solution π, the two jobs J_3 and J_4 on M_2 are exchanged with J_2 on M_1, and a new solution π_3 is generated (cf. Fig. 3(c)).

After many experiments, it has been shown that the algorithm can show better performance if the algorithm is performed in the sequence of *Move*, *Swap*, and *Aswap*, so we will introduce these three neighborhood structures in this sequence. According to the above three neighborhood structures, the following variable neighborhood search algorithm is designed by us:

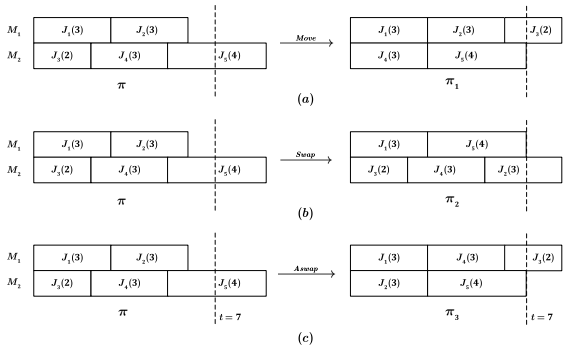


FIGURE 3. Three Neighborhood Structures of VNS.

Algorithm 1 VNS

Initialization, Determine the neighborhood structure $\mathcal{N}_k$ ($k = 1, 2, \cdots, k_{max}$), where k_{max} represents the number of neighborhoods. Let $iter_{max}$ be the loop length and $i = 1$, and select the best solution in this iteration as the initial solution $\pi = \pi_0$.
while $i < iter_{max}$ **do**
  A new solution π is obtained by randomly exchanging jobs on any two machines in π_0;
  for $k = 1$ to k_{max} **do**
    Find the best solution π' in the neighborhood $\mathcal{N}_k(\pi)$;
    if $totalcost(\pi') < totalcost(\pi)$ **then**
      $\pi = \pi'$ and $k = 1$
    else
      $k = k + 1$;
    end if
  end for
  if $totalcost(\pi) < totalcost(\pi_0)$ **then**
    $\pi_0 = \pi$
  else
    $i = i + 1$;
  end if
end while
Output the best found solution π_0.

E. IMPROVED PHEROMONE UPDATE RULES

The pheromone needs to be updated immediately after the ant completes a tour, and the update rule is carried out according to the following equations,

$$\tau_{ij}(z+1) = (1-\rho) \cdot \tau_{ij}(z) + \Delta_{ij}(z), \quad (14)$$

$$\Delta_{ij}(z) = \begin{cases} \frac{R}{L^*}, & \text{if } J_i \text{ and } J_j \text{ are assigned to a same machine,} \\ 0, & \text{otherwise} \end{cases} \quad (15)$$

where L^* represents the total overtime in the best found schedule so far.

In this paper, we adopt the global pheromone update rule. After the ant completes a traversal, the pheromone is updated according to Eqs. (14) and (15), so the amount of information between the jobs assigned to the same machine will increase. Based on the above analysis, we design the following *improved ant olony optimization algorithm* (IACO) for solving the problem GEBPOC:

Algorithm 2 IACO

1: **Initialization,** Set the initial pheromone and $t = 0$. Let the current number of loops and the maximum number of iterations be denoted by N_c and N_{max}, respectively. Let the initial amount of information $\sigma_{ij}(0) = R$. m ants randomly select a job J_j and assign it to any machine randomly, then $tabu_k = \{J_j\}$ and $allowed_k = \mathcal{J} \setminus tabu_k$, and the optimal solution is denoted by π^*;
2: Calculate the lower bound LB of problem;
3: **while** $totalcost(\pi^*) \neq LB$ and $N_c < N_{max}$ **do**
4: **for** $k = 1$ to m **do**
5: Ant k selects the target machine for each job according to the random probability selection principle of (12) until the set $allowed_k$ is empty;
6: The schedule obtained by ant k is evaluated using the fitness evaluation function ((10));
7: Let $\pi_0 = \pi^*$ and call the **Algorithm 1**;
8: **end for**
9: The pheromone update is performed by (14);
10: Update the current best solution found by ant colony;
11: **end while**
12: Output the historical best found solution π_0.

IV. IMPROVED DISCRETE PARTICLE SWARM OPTIMIZATION ALGORITHM FOR GEBPOC

In Section 2.2, we have introduced the idea of the basic particle swarm optimization algorithm (PSO), which presents a feasible solution to a specific optimization problem through the position structure of particles, and then uses the iterative process of particle velocity and position changes to continuously evolve, so as to gradually approach the best position. However, the performance of PSO for some discrete variables is not very satisfactory, because the original PSO algorithm can only optimize problems in which the elements of the solution are continuous real numbers [40]. This section describes how the discrete particle swarm optimization (DPSO) algorithm can solve the problem GEBPOC. In order to obtain better quality initial particles, we employ some heuristics in the population initialization process, so that the particles can cover the entire search space with a large probability, and denote this modified algorithm as MDPSO.

It is acknowledged that the PSO algorithm provides a general framework for solving optimization problems. However, for a specific problem, the key lies in the representation of solution, the definition method of operators, the construction method of initial solution and the setting of termination conditions. Some related technologies of the MDPSO algorithm proposed for GEBPOC are introduced as follows:

1) **Representation of the solution.**
In order to establish a direct relationship between the solution space of GEBPOC and particles, the solutions corresponding to the assignment of jobs to machine are represented by an n-dimensional array $(d_1, d_2, \cdots, d_n)$ similar to that in Section 3.1. For example, when $m = 2$ and $n = 5$, the 4-dimensional array (1, 2, 2, 1, 1) corresponds to a feasible schedule, which implies that jobs J_1, J_4 and J_5 are all processed on machine M_1, while jobs J_2 and J_3 are both executed by M_2.
2) **Definition of operators in particle update.**
Based on the idea of the basic PSO algorithm in Section 2.2, we give the following update method for the new velocity and position of particle k in the $(z+1)$th iteration, combining the characteristics of the specific problem. In effect, we redefine what each operator expresses, *i.e.,*

$$V_k^{z+1} = V_k^z \oplus (R_1 \otimes (pB_k^z \ominus X_k^z)) \oplus (R_2 \otimes (gB^z \ominus X_k^z)), \quad (16)$$

$$X_k^{z+1} = X_k^z \oplus V_k^{z+1}, \quad (17)$$

where R_1 and R_2 are n-dimensional random variables consisting of 0 and 1. The addition $\oplus$, subtraction $\ominus$, and multiplication $\otimes$ operations between tuples need to be performed during the algorithm iteration. Therefore, we have to redefine these three operators according to the nature of the problem. The technique of redefining operators in this paper refers to the ideas in literature [40].

a) *Redefinition of the subtract operator* $\ominus$.
The subtract operator $\ominus$ is mainly used to express the difference between the current position (X_k^z) and the best position (pB_k^z or gB^z) of particle k. The difference between two tuples is calculated by comparing whether the element at each position in X_k^z is the same as the element at the corresponding position in pB_k^z or gB^z. If so, the element at the corresponding position in the final tuple is assigned zero; otherwise, the value at the corresponding position in the tuple pB_k^z (or gB^z) is reserved as the calculation result. Note that the jobs corresponding to the positions with the same elements in X_k^z and pB_k^z (or gB^z) need to be arranged according to the longest processing time (LPT) strategy and assigned to machine in turn when the machine is idle. The redefinition process of the subtract operator $\ominus$ is shown in Fig. 4 (still take $m = 2,\ n = 5$ as an example).
b) *Redefinition of the multiply operator* $\otimes$.
The multiply operator $\otimes$ is mainly used for the operation between the random variables R_1, R_2 and the result of the subtract operation.

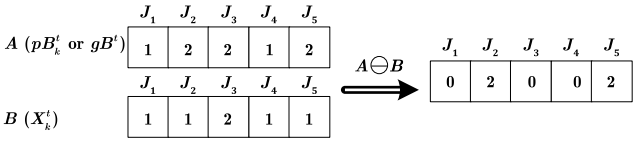


FIGURE 4. Redefinition of subtract operator ⊖ (where jobs J_2 and J_5 are both assigned to machine M_2, while jobs J_1, J_3 and J_4 are assigned to machines based on LPT rule).

It is also a process of data selection, which can improve the search ability of DPSO. This operator first defines R_1 and R_2 as two n-dimensional arrays, each of which has a randomly generated value of 0 or 1. $A \otimes B$ is a simple multiply arithmetic operation between the elements of two n-dimensional tuples. If the element at the jth position in tuple A is 1, then the element at the jth position of the operation result is equal to the element at the jth position in tuple B; otherwise, if the element at the jth position in A is 0, then the element at the jth position of the operation result is also 0, where $j = 1, 2, \cdots, n$. The process of redefining the multiply operator ⊗ is shown in Fig. 5.

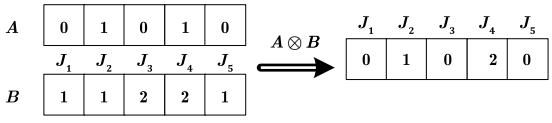


FIGURE 5. Redefinition of multiply operator ⊗ (where jobs J_2 and J_4 are assigned to M_1 and M_2, respectively, while jobs J_1, J_3 and J_5 are assigned to machines based on LPT rule).

c) *Redefinition of the add operator* ⊕.
The add operator ⊕ is the final operation to obtain the velocity and position of a particle in a new iteration, and this operator must guarantee that the obtained result is a reasonable solution. In this paper, we regard ⊕ as the crossover operator in genetic algorithm. In fact, we randomly select two cut points from the particle chain, and then exchange the chain between these two cut points, which tends to produce two new particle chains in the end. During the operation, we generally randomly select a particle chain as the result of the add operation. The process of redefining the add operator ⊕ is shown in Fig. 6.

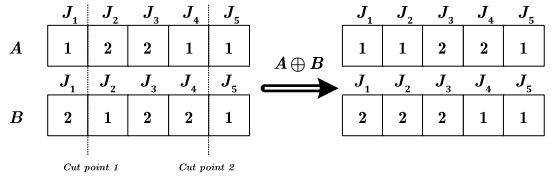


FIGURE 6. Redefinition of multiply operator ⊕ (one of the two new particle chains generated will be randomly selected as the result of the $A \oplus B$ operation).

3) **Initialize the population.**
The DPSO algorithm in this paper defines the population size as 20, that is, there are 20 particles in the system, and they are all n-dimensional arrays generated in a random way. In order to enable particles to cover the entire search space with a large probability, prevent the algorithm from falling into local optimum, together with avoid reducing the search efficiency due to blind search, we adopt some simple heuristic algorithms (such as longest processing time first (LPT), shortest processing time first (SPT)) in Algorithm 3 to obtain a higher quality initial population.
4) **The termination condition of algorithm.**
The DPSO algorithm is essentially a process of gradually replacing and seeking the optimum. A particle k combines the best solution pB_k^z searched by itself and the global best solution gB^z searched by the population to continuously update its speed and position, making it gradually approach the position of the optimal solution. When the algorithm triggers the termination condition, it will stop running and output the schedule corresponding to the current global best solution gB^z found by the population.
In this paper, the MDPSO algorithm stops as soon as one of the following two situations occurs: the currently found global best solution gB^z has not changed after 200 iterations or the lower bound in (9) is just obtained.

Combined with the introduction of the above related technologies, we formally present the MDPSO algorithm for GEBPOC.

V. EXPERIMENTAL RESULTS AND DISCUSSION

A. EXPERIMENTAL SETUP

In this section, we compare the performance of some algorithms proposed in the previous two sections through data experiments.In addition to this, we also compare the results of IACO and MDPSO with those of several other well-known meta-heuristics: variable neighborhood search algorithm (VNS) [51], simulated annealing algorithm (SA) [24], ant colony optimization algorithm (ACO) [26] and particle swarm algorithm (PSO) [25]. These algorithms have been proven to have excellent results when solving engineering optimization problems. We evaluate the ability of these algorithms to solve the GEBPOC problem by setting small-scale and large-scale instances, respectively. All the instances in this section are from literatures [52] and [24]. Since the calculation of the objective function value involves the selection of regular working time t, we set $t = \lceil h \cdot P \rceil$ to verify the influence of different values of t on algorithms, where $h \in \{0.2,\ 0.4,\ 0.6,\ 0.8\}$ and $P = \sum_{j:J_j \in \mathcal{J}} p_j$. For convenience, in the whole experiment, we assume the unit time cost of overload $c = 2$. Therefore, the following two experimental frameworks are finally generated (cf. Table 1), where m, n and p represent the number of machines, the number of jobs and the processing time of a job, respectively. $U(a, b)$ means

TABLE 1. Parameter settings for computational experiments.

	m	n	p	h
$E1$	3, 4, 5	$2m$, $3m$, $5m$	$U(1,20)$, $U(20,50)$	0.2, 0.4, 0.6, 0.8
$E2$	2, 3, 4, 6, 8, 10	10, 30, 50, 100	$U(100,800)$	$\frac{1}{m}$

Algorithm 3 MDPSO

Initialization, Set $z = 0$ and there are 20 particles in the population;
for $k = 1$ to N **do**
  particle X_k^z at random;
  Let $pB_k^z = X_k^z$;
end for
$gB^z = \{X_l^z | l = \arg\min_k \{totalcost(X_k^z)\}\}$;
while the maximal number of iterations $z < 200$ and $totalcost \neq LB$ **do**
  for $k = 1$ to N **do**
    The velocity of particle k is updated by (16);
    The position of particle k is updated by (17);
    if $totalcost(X_k^{z+1}) < totalcost(pB_k^z)$ **then**
      Let $pB_k^{z+1} = X_k^{z+1}$
    else
      Let $pB_k^{z+1} = pB_k^z$;
    end if
  end for
  if $totalcost(gB^z) > \min_k \{totalcost(pB_k^{z+1})\}$ **then**
    Let $gB^{z+1} = \{pB_l^{z+1} | l = \arg\min_k \{totalcost(pB_k^{z+1})\}\}$
  else
    $gB^{z+1} = pB_k^{z+1}$;
  end if
  Set $z = z + 1$;
end while
Output the historical best found solution gB^z.

that the processing time of each job is distributed randomly and uniformly in the interval $[a, b]$.

These two sets of experimental frameworks yield $3 \times 3 \times 2 \times 4 + 6 \times 4 = 96$ different (m, n, p, h) combination settings in total. To increase the reliability of the algorithm, 30 sets of test data were randomly generated for each (m, n, p, h) combination, and the results were averaged. In this way, there are a total of 2880 computing instances. In this paper, all algorithm experiments are coded in Python 3.8, and tested on a laptop with Ryzen Core R7-4800H 2.90 GHz CPU and 16 GB RAM.

We test the performance of each of these seven algorithms (LPT, VNS, SA, ACO, IACO, DPSO, and MDPSO) using the data generated by the above methods. The parameters of algorithm ACO are set as follows: the number of ants $d = 20$, $\alpha = 1$, $\beta = 5$, and $N_{max} = 200$. The population size in DPSO is equal to the number of ants in ACO, and the maximal number of iterations is also 200, which of course is a parameter involved in several other meta-heuristics. To avoid chance of results, for each combination of (m, n, p, h), 30 instances were generated.

B. ANALYSIS OF RESULTS: RATIO AND TIME

In this section, we test the performance ratio and required execution time of these meta-heuristics. The average results obtained from experiments $E1$ and $E2$ are shown in Tables 2-5. In each table, the columns "Ratio" calculate the ratio between the criterion values given by proposed algorithms (the optimal or near optimal) and LB, and they are presented with 4-digit precision, while the columns "Avg.time" indicate the time consumption of the corresponding approach, where LPT in milliseconds, other algorithms in seconds.

The results from Table 2 show that the ratio of the intelligent optimization algorithm proposed in this paper is closer to 1.0000 than LPT algorithm when solving the same problem instance for the case of $m = 3$, due to the small scale of the problem, while the LPT algorithm run much faster than other algorithms in general. For a specific n, the output value will be closer to the optimal value when a certain algorithm is used to solve the problem as the value of h increases. In particular, when $h = 0.8$, the ratio corresponding to almost all instances is 1.0000, which means that when the regular working time is large, the jobs can be processed without overtime. However, the average running time is not much different. For both ACO and IACO algorithms, IACO is stronger than basic ACO in terms of ratio, because the search ability of the global optimal solution of the algorithm is significantly improved after the variable neighborhood search algorithm is introduced in IACO, but the running time will be slightly increased, the average running time of IACO is about 7 times that of ACO. Compared with the DPSO algorithm, the ratio of MDPSO is closer to 1.0000, due to some heuristic improvement strategies adopted in the population initialization process, but the average running time of MDPSO is 2 times slower than that of DPSO. In most problems (except a few simple problems), the average solutions obtained by the IACO algorithm and the MDPSO algorithm are better than LPT, VNS, SA, ACO and DPSO. In addition, the running time of the above seven algorithms becomes longer as the number of jobs increases.

Tables 3 and 4 report the comparison results obtained from the case of $m = 4$ and $m = 5$, respectively. The variation in ratio and average running time of these two cases is similar to the case of $m = 3$. As the number of machines increases, the running time of each algorithm also increases, but the increase in LPT is larger than that of the other meta-heuristics. Compared with the results in Table 2, the ratios corresponding to Tables 3 and 4 to when $h = 0.8$ are rarely

TABLE 2. Results for experiment $E1$: $m = 3$.

m	n	p	h	LPT		VNS		SA		ACO		IACO		DPSO		MDPSO	
				Ratio	Avg.time(ms)	Ratio	Avg.time(s)	Ratio	Avg.time(s)	Ratio	Avg.time(s)	Ratio	Avg.time(s)	Ratio	Avg.time(s)	Ratio	Avg.time(s)
3	6	$U(1,20)$	0.2	1.0524	2.33	1.0521	0.04	1.0523	0.05	1.0527	0.05	1.0523	0.08	1.0524	0.04	1.0524	0.06
			0.4	1.0427	2.31	1.0476	0.05	1.0474	0.05	1.0533	0.06	1.0527	0.06	1.0527	0.04	1.0527	0.08
			0.6	1.0000	2.12	1.0025	0.05	1.0112	0.04	1.0000	0.05	1.0000	0.09	1.0002	0.05	1.0000	0.08
			0.8	1.0000	2.16	1.0003	0.04	1.0007	0.04	1.0000	0.04	1.0000	0.12	1.0000	0.05	1.0000	0.07
	9		0.2	1.0117	2.78	1.0364	0.06	1.0102	0.07	1.0117	0.07	1.0112	0.18	1.0109	0.08	1.0027	0.09
			0.4	1.0356	3.34	1.0072	0.07	1.0327	0.06	1.0356	0.07	1.0343	0.11	1.0348	0.07	1.0235	1.14
			0.6	1.0007	2.66	1.0019	0.05	1.0005	0.07	1.0004	0.08	1.0000	0.15	1.0011	0.08	1.0000	1.12
			0.8	1.0000	2.32	1.0008	0.07	1.0000	0.06	1.0000	0.05	1.0000	0.10	1.0006	0.07	1.0000	1.12
	15		0.2	1.0278	5.24	1.0283	0.07	1.0264	0.08	1.0264	0.08	1.0196	1.23	1.0278	0.09	1.0164	1.16
			0.4	1.0265	6.17	1.0174	0.07	1.0138	0.07	1.0176	0.08	1.0171	1.28	1.0176	0.08	1.0163	1.28
			0.6	1.0013	5.23	1.0042	0.08	1.0017	0.06	1.0002	0.07	1.0001	1.35	1.0002	0.11	1.0000	1.37
			0.8	1.0000	5.11	1.0013	0.06	1.0002	0.08	1.0018	0.08	1.0006	1.36	1.0018	0.09	1.0002	1.32
	6	$U(20,50)$	0.2	1.0328	2.73	1.0328	0.06	1.0312	0.04	1.0328	0.04	1.0316	0.08	1.0328	0.04	1.0312	0.08
			0.4	1.0363	2.95	1.0376	0.05	1.0354	0.06	1.0371	0.05	1.0323	0.09	1.0364	0.04	1.0328	0.06
			0.6	1.0000	2.67	1.0054	0.05	1.0173	0.05	1.0000	0.05	1.0000	0.06	1.0002	0.06	1.0000	0.07
			0.8	1.0000	2.54	1.0007	0.04	1.0008	0.05	1.0000	0.05	1.0000	0.07	1.0008	0.05	1.0001	0.05
	9		0.2	1.0214	5.82	1.0332	0.07	1.0224	0.09	1.0214	0.07	1.0208	0.13	1.0211	0.07	1.0200	1.16
			0.4	1.0207	6.17	1.0215	0.08	1.0201	0.08	1.0202	0.07	1.0200	0.17	1.0197	0.05	1.0154	1.29
			0.6	1.0037	6.21	1.0103	0.07	1.0018	0.11	1.0022	0.08	1.0016	0.12	1.0009	0.08	1.0000	1.24
			0.8	1.0001	6.05	1.0024	0.08	1.0001	0.08	1.0000	0.06	1.0000	0.11	1.0006	0.09	1.0000	1.23
	15		0.2	1.0183	7.52	1.0208	0.09	1.0173	0.12	1.0185	0.09	1.0181	1.36	1.0167	1.13	1.0078	1.45
			0.4	1.0205	7.88	1.0149	0.08	1.0168	0.10	1.0187	0.09	1.0176	1.39	1.0312	1.38	1.0165	1.47
			0.6	1.0018	8.10	1.0062	0.10	1.0012	0.09	1.0013	0.10	1.0010	1.34	1.0018	1.32	1.0005	1.45
			0.8	1.0002	7.73	1.0014	0.07	1.0003	0.08	1.0000	0.09	1.0000	1.33	1.0000	1.28	1.0000	1.48
	Average			1.0148	4.51	1.0161	0.06	1.0151	0.07	1.0147	0.07	1.0138	0.52	1.0151	0.27	1.0120	0.83

TABLE 3. Results for experiment $E1$: $m = 4$.

m	n	p	h	LPT		VNS		SA		ACO		IACO		DPSO		MDPSO	
				Mean	Avg.time(ms)	Mean	Avg.time(s)	Mean	Avg.time(s)	Mean	Avg.time(s)	Mean	Avg.time(s)	Mean	Avg.time(s)	Mean	Avg.time(s)
4	8	$U(1,20)$	0.2	1.0436	3.24	1.0436	0.16	1.0447	0.14	1.0436	0.14	1.0423	0.36	1.0435	0.16	1.0441	0.47
			0.4	1.0553	3.20	1.0547	0.16	1.0525	0.15	1.0534	0.14	1.0525	0.47	1.0527	0.16	1.0465	0.42
			0.6	1.0057	3.31	1.0043	0.12	1.0240	0.19	1.0046	0.18	1.0041	0.33	1.0123	0.18	1.0116	0.53
			0.8	1.0000	3.25	1.0000	0.15	1.0021	0.22	1.0000	0.15	1.0001	0.34	1.0014	0.19	1.0008	0.51
	12		0.2	1.0479	5.07	1.0475	0.48	1.0384	0.56	1.0472	0.47	1.0448	0.88	1.0362	0.42	1.0109	0.96
			0.4	1.0638	4.89	1.0612	0.40	1.0352	0.42	1.0603	0.51	1.0343	0.93	1.0348	0.69	1.0235	1.14
			0.6	1.0019	4.51	1.0023	0.45	1.0136	0.58	1.0011	0.44	1.0000	0.75	1.0107	0.48	1.0082	1.05
			0.8	1.0000	4.77	1.0002	0.50	1.0027	0.50	1.0000	0.41	1.0000	0.77	1.0011	0.53	1.0000	1.12
	20		0.2	1.0764	5.89	1.0712	1.32	1.0563	1.42	1.0673	1.24	1.0526	1.84	1.0547	1.37	1.0164	1.65
			0.4	1.0539	5.81	1.0483	1.28	1.0422	1.45	1.0329	1.36	1.0138	1.62	1.0436	1.46	1.0163	1.86
			0.6	1.0113	5.73	1.0107	1.35	1.0074	1.37	1.0062	1.33	1.0047	1.74	1.0065	1.42	1.0057	1.83
			0.8	1.0008	5.70	1.0003	1.30	1.0012	1.51	1.0003	1.28	1.0003	1.75	1.0002	1.48	1.0001	1.92
	8	$U(20,50)$	0.2	1.0497	3.32	1.0482	0.35	1.0463	0.28	1.0492	0.27	1.0327	0.58	1.0474	0.23	1.0312	0.49
			0.4	1.0582	3.27	1.0527	0.21	1.0465	0.22	1.0567	0.19	1.0349	0.64	1.0462	0.27	1.0343	0.53
			0.6	1.0086	3.36	1.0069	0.20	1.0121	0.32	1.0073	0.24	1.0023	0.62	1.0058	0.31	1.0024	0.62
			0.8	1.0000	3.18	1.0001	0.24	1.0014	0.31	1.0000	0.28	1.0000	0.62	1.0001	0.34	1.0000	0.59
	12		0.2	1.0481	5.63	1.0472	0.83	1.0352	1.69	1.0471	0.73	1.0402	1.39	1.0298	1.54	1.0200	1.36
			0.4	1.0627	5.48	1.0569	0.74	1.0248	1.47	1.0538	0.70	1.0473	1.34	1.0265	1.31	1.0192	1.35
			0.6	1.0037	5.66	1.0082	0.74	1.0069	1.26	1.0035	0.68	1.0009	1.28	1.0054	1.07	1.0023	1.19
			0.8	1.0001	5.41	1.0007	0.63	1.0007	1.32	1.0001	0.73	1.0000	1.31	1.0003	1.33	1.0000	1.07
	20		0.2	1.0794	6.02	1.0375	1.72	1.0512	2.74	1.0767	1.56	1.0702	1.99	1.0478	2.67	1.0432	2.42
			0.4	1.0463	6.14	1.0109	1.65	1.0307	2.66	1.0275	1.63	1.0346	2.15	1.0274	2.96	1.0108	1.97
			0.6	1.0118	6.01	1.0012	1.56	1.0126	2.84	1.0084	1.48	1.0082	2.18	1.0071	2.88	1.0022	2.03
			0.8	1.0006	5.76	1.0002	1.59	1.0035	2.83	1.0005	1.62	1.0002	2.12	1.0002	2.93	1.0001	2.12
	Average			1.0304	4.78	1.0256	0.76	1.0247	1.10	1.0270	0.74	1.0217	1.17	1.0226	1.10	1.0146	1.22

equal to 1.0000, which is also due to the increase in the number of machines. In Table 3, the average running time of IACO is about 2 times that of ACO, while the average running time of MDPSO and DPSO differs by only 0.02s. In Table 4, the average running time of IACO is only 0.27s away from ACO, and MDPSO runs faster than DPSO in average time. Although VNS, SA, ACO, and DPSO have smaller average execution time than IACO and MDPSO, the former are more prone to falling into local minima. Therefore, in terms of running time, as the number of machines increases, the effects of IACO algorithm and MDPSO algorithm begin to emerge.

The second set of experiments $E2$ were tested on some large scale problems, and a longer job processing time was set. The experimental results are shown in Table. 5. This set of experiments omits the setting of different regular working time, since experiment $E1$ has already verified its effect on algorithms. The experimental results show that at the level of the number of jobs, the time consumed by the LPT algorithm is basically not affected by the number of machines, which is due to the time complexity $O(n \log n)$ of LPT. As the problem size increases, the average performance of the LPT algorithm becomes weaker. The difference in running time between LPT algorithm and other meta-heuristics grows with the number of jobs. This implies that the bigger instances, the higher improvement in time consumption can be observed, which is also a big advantage of these proposed meta-heuristic algorithms from the practical point of view. Secondly, compared to the LPT, VNS, SA, ACO and DPSO, the improved IACO and MDPSO beats them not only from the point of view of time efficiency, but also from the point of view of the ratio and the ability to solve problem instances in a reasonable time, which means that the decrease in computational complexity for the improved IACO and MDPSO algorithms in comparison to the previous algorithms allowed for predicting the decrease of running time, but the

TABLE 4. Results for experiment $E1$: $m = 5$.

m	n	p	h	LPT		VNS		SA		ACO		IACO		DPSO		MDPSO	
				Mean	Avg.time(ms)	Mean	Avg.time(s)	Mean	Avg.time(s)	Mean	Avg.time(s)	Mean	Avg.time(s)	Mean	Avg.time(s)	Mean	Avg.time(s)
5	10	$U(1,20)$	0.2	1.1627	6.53	1.1524	1.32	1.1473	1.38	1.1472	0.94	1.1317	1.18	1.0927	1.36	1.0852	1.33
			0.4	1.1482	6.72	1.1328	1.17	1.1025	1.34	1.1254	1.32	1.1054	1.23	1.0942	1.30	1.0663	1.37
			0.6	1.0730	6.33	1.0704	1.08	1.0674	1.34	1.0656	1.14	1.0473	1.14	1.0536	1.33	1.0172	1.35
			0.8	1.0026	6.38	1.0019	1.24	1.0128	1.32	1.0007	1.16	1.0007	1.18	1.0035	1.28	1.0004	1.38
	15		0.2	1.0836	8.77	1.0712	1.42	1.0742	1.56	1.0723	1.50	1.0636	1.68	1.0632	1.47	1.0372	1.28
			0.4	1.0504	8.23	1.0485	1.56	1.0452	1.64	1.0483	1.62	1.0275	1.74	1.0477	1.65	1.0154	1.20
			0.6	1.0162	9.06	1.0143	1.53	1.0163	1.66	1.0056	1.58	1.0108	1.46	1.0062	1.62	1.0039	1.36
			0.8	1.0013	8.82	1.0016	1.61	1.0014	1.63	1.0006	1.53	1.0000	1.61	1.0014	1.68	1.0006	1.52
	20		0.2	1.0746	23.21	1.0625	1.68	1.0489	1.87	1.0538	1.72	1.0362	1.86	1.0464	1.94	1.0336	1.72
			0.4	1.0547	23.09	1.0403	1.72	1.0356	2.44	1.0376	1.64	1.0282	1.69	1.0304	1.95	1.0250	1.76
			0.6	1.0301	25.43	1.0214	1.64	1.0175	2.07	1.0174	1.64	1.0043	1.80	1.0161	1.90	1.0082	1.42
			0.8	1.0019	24.77	1.0016	1.73	1.0021	2.16	1.0012	1.62	1.0009	1.83	1.0013	1.91	1.0002	1.77
	10	$U(20,50)$	0.2	1.0638	7.62	1.0607	0.76	1.0562	1.42	1.0566	0.83	1.0433	0.93	1.0580	1.37	1.0442	1.41
			0.4	1.0672	7.29	1.0537	0.89	1.0470	1.44	1.0482	0.96	1.0278	1.33	1.0457	1.48	1.0149	1.38
			0.6	1.0252	6.58	1.0149	0.97	1.0128	1.53	1.0107	0.91	1.0057	1.27	1.0072	1.43	1.0028	1.30
			0.8	1.0107	7.04	1.0038	0.86	1.0023	1.48	1.0039	0.93	1.0016	1.30	1.0012	1.50	1.0011	1.32
	15		0.2	1.0481	10.24	1.0468	1.26	1.0336	1.46	1.0462	1.28	1.0442	1.81	1.0302	1.36	1.0256	1.39
			0.4	1.0433	13.21	1.0382	1.34	1.0385	1.39	1.0367	1.31	1.0176	1.50	1.0387	1.37	1.0104	1.46
			0.6	1.0108	12.80	1.0102	1.30	1.0154	1.40	1.0083	1.32	1.0036	1.92	1.0165	1.30	1.0072	1.45
			0.8	1.0054	13.62	1.0013	1.32	1.0013	1.44	1.0009	1.32	1.0000	1.86	1.0003	1.35	1.0000	1.48
	20		0.2	1.0636	32.70	1.0572	1.71	1.0586	2.06	1.0551	1.74	1.0462	1.83	1.0547	1.83	1.0517	1.23
			0.4	1.0480	34.56	1.0324	1.69	1.0418	2.15	1.0205	1.66	1.0147	2.43	1.0341	1.70	1.0323	1.36
			0.6	1.0132	31.62	1.0107	1.82	1.0084	1.94	1.0076	1.70	1.0018	2.65	1.0044	1.76	1.0083	1.29
			0.8	1.0021	34.05	1.0026	1.76	1.0014	1.88	1.0023	1.74	1.0011	2.47	1.0006	1.82	1.0005	1.24
	Average			1.0459	15.36	1.0396	1.39	1.0370	1.67	1.0364	1.38	1.0277	1.65	1.0312	1.57	1.0205	1.41

TABLE 5. Results for experiment $E2$.

m	n	p	h	LPT		VNS		SA		ACO		IACO		DPSO		MDPSO	
				Mean	Avg.time(ms)	Mean	Avg.time(s)	Mean	Avg.time(s)	Mean	Avg.time(s)	Mean	Avg.time(s)	Mean	Avg.time(s)	Mean	Avg.time(s)
2	10	$U(100,800)$	$\frac{1}{m}$	1.0014	2.46	1.0014	0.07	1.0014	0.19	1.0014	0.06	1.0006	0.14	1.0014	0.09	1.0014	0.13
3				1.0057	2.67	1.0036	0.05	1.0048	0.24	1.0027	0.07	1.0018	0.19	1.0032	0.12	1.0019	0.19
4				1.0093	2.93	1.0102	0.06	1.0083	0.37	1.0068	0.07	1.0020	0.26	1.0063	0.16	1.0018	0.15
6				1.0147	2.78	1.0127	0.08	1.0116	0.35	1.0093	0.09	1.0057	0.28	1.0094	0.27	1.0032	0.24
8				1.0462	2.93	1.0432	0.18	1.0278	0.42	1.0382	0.16	1.0084	0.29	1.0274	0.31	1.0103	0.34
10				1.0538	3.43	1.0475	0.25	1.0364	0.48	1.0408	0.28	1.0136	0.35	1.0358	0.34	1.0116	0.20
2	30			1.0017	8.69	1.0016	0.44	1.0033	0.94	1.0013	0.56	1.0002	0.73	1.0015	0.54	1.0000	0.64
3				1.0076	9.42	1.0065	0.89	1.0067	1.53	1.0054	0.94	1.0015	1.49	1.0051	0.83	1.0008	0.80
4				1.0184	9.88	1.0149	1.23	1.0216	2.18	1.0139	1.17	1.0039	1.52	1.0147	1.26	1.0048	0.79
6				1.0394	9.65	1.0327	1.28	1.0238	2.34	1.0256	1.45	1.0084	1.58	1.0242	1.47	1.0115	1.16
8				1.0477	9.70	1.0454	1.33	1.0402	2.46	1.0396	1.68	1.0175	1.73	1.0361	1.73	1.0139	1.50
10				1.0551	9.59	1.0402	1.46	1.0442	2.37	1.0428	1.83	1.0241	1.96	1.0411	1.92	1.0205	1.21
2	50			1.0023	37.23	1.0033	0.83	1.0027	2.41	1.0023	0.78	1.0008	0.85	1.0020	0.71	1.0012	0.96
3				1.0107	36.43	1.0069	0.97	1.0085	2.79	1.0074	1.39	1.0019	1.43	1.0072	1.28	1.0014	1.23
4				1.0236	38.40	1.0184	0.94	1.0176	3.06	1.0178	1.64	1.0032	1.73	1.0149	1.59	1.0029	1.07
6				1.0340	36.70	1.0245	2.06	1.0278	2.96	1.0252	2.18	1.0094	2.39	1.0226	1.93	1.0136	1.82
8				1.0485	37.51	1.0417	2.43	1.0354	4.17	1.0374	2.53	1.0127	2.84	1.0315	2.67	1.0124	2.04
10				1.0615	35.94	1.0528	3.96	1.0438	6.64	1.0484	3.86	1.0183	3.62	1.0403	4.05	1.0165	2.50
2	100			1.0007	162.08	1.0000	0.78	1.0032	3.75	1.0000	0.96	1.0000	1.28	1.0000	1.36	1.0000	1.48
3				1.0035	166.71	1.0034	3.86	1.0046	7.46	1.0025	3.77	1.0005	3.52	1.0021	3.48	1.0000	1.79
4				1.0064	172.35	1.0041	7.97	1.0023	15.02	1.0038	7.84	1.0018	6.49	1.0024	7.54	1.0014	3.24
6				1.0730	176.16	1.0536	15.62	1.0226	28.34	1.0483	14.08	1.0174	12.56	1.0172	16.28	1.0162	6.27
8				1.0108	185.47	1.0078	31.97	1.0075	29.69	1.0103	28.76	1.0062	21.63	1.0069	24.33	1.0053	17.30
10				1.0083	194.85	1.0054	48.63	1.0068	41.54	1.0056	43.25	1.0017	32.04	1.0042	35.49	1.0018	31.04
	Average			1.0243	56.42	1.0201	5.31	1.0172	6.74	1.0182	4.98	1.0067	4.20	1.0149	4.57	1.0064	3.25

computational experiments showed how significant this decrease is in practice. More precisely, we observed that both theoretical and practical improvements are significant for big instances and both of these improvements are relatively minor when scale of instances is small, that is, IACO and MDPSO algorithms have better stability when solving large scale instances.

C. ANALYSIS OF RESULTS: FUNCTION VALUE AND FRIEDMAN RANK

To further illustrate the significant advantages of the proposed algorithm, we also select 8 instances from the two sets of experimental frameworks in Table 1, *i.e.,* (3, 15, $U(1, 20)$, 0.2), (3, 15, $U(20, 50)$, 0.4), (4, 20, $U(1, 20)$, 0.2), (4, 20, $U(20, 50)$, 0.4), (5, 20, $U(1, 20)$, 0.2), (5, 20, $U(20, 50)$, 0.4), (10, 50, $U(100, 800)$, 0.1) and (10, 100, $U(100, 800)$, 0.1), we separately recorded the best function value, worst function value, mean and standard deviation of each algorithm in 30 runs, since evolutionary algorithms have a certain degree of randomness in each run. The experimental results are shown in Table 6. Four performance indicators (include "Best", "Worst", "Mean", and "Standard deviation (SD)") are used to validate the effectiveness of the proposed IACO and MDPSO in conjunction with other state-of-the-art optimization algorithms. Moreover, a non-parametric statistical test called the Friedman ranking test is applied for a fair performance comparison with other existing optimization methods. The Feldman ranking test is a non-parametric multiple hypothesis test of repeated measures ANOVA that gives the ranking of different algorithms on each dataset and finally calculates the mean of the ranking of each algorithm on all datasets, if all algorithms have no performance difference, then the average ranking of their performance should be equal.

It can be clearly seen from the information in Table 6 that our proposed IACO and MDPSO algorithms are better than LPT, VNS, SA, ACO and DPSO in terms of best fitness,

TABLE 6. Best, worst, mean, standard deviation and Friedman ranking results obtained for 30 independent runs of GEBPOC.

m	n	p	h	Value	LPT	VNS	SA	ACO	IACO	DPSO	MDPSO
				Best	96	94	94	89	87	93	87
				Worst	172	168	162	165	149	157	142
3	15	$U(1,20)$	0.2	Mean	132.2	131.9	127.2	120.7	118.4	126.2	114.7
				SD	21.85	21.61	19.31	23.78	17.90	21.24	14.73
				Rank	6	5	3	7	2	4	1
				Best	748	726	728	728	624	684	624
				Worst	1122	1074	1154	1036	997	1028	962
3	15	$U(20,50)$	0.4	Mean	944.3	909.4	930.7	879.2	798.5	879.6	776.7
				SD	102.38	95.26	96.60	95.29	94.28	95.21	92.50
				Rank	7	4	6	5	2	3	1
				Best	214	207	207	196	172	185	172
				Worst	372	258	365	347	316	324	293
4	20	$U(1,20)$	0.2	Mean	283.5	281.6	296.9	280.5	238.4	251.5	220.5
				SD	42.15	42.89	40.47	41.67	38.46	39.51	37.19
				Rank	6	7	4	5	2	3	1
				Best	1079	1054	1062	1054	996	1032	992
				Worst	1584	1593	1547	1518	1438	1495	1427
4	20	$U(20,50)$	0.4	Mean	1351.1	1137.5	1304.83	1241.2	1234.3	1289.0	1204.7
				SD	132.67	121.25	130.48	129.32	126.62	127.58	125.68
				Rank	7	1	6	5	3	4	2
				Best	268	262	257	260	235	258	235
				Worst	456	451	464	463	432	446	436
5	20	$U(1,20)$	0.2	Mean	355.8	352.5	348.6	377.5	346.1	344.2	346.4
				SD	59.98	60.06	59.21	57.60	56.22	55.76	55.78
				Rank	6	7	5	4	3	1	2
				Best	1328	1276	1262	1278	1240	1264	1240
				Worst	1764	1760	1783	1742	1674	1751	1653
5	20	$U(20,50)$	0.4	Mean	1542.6	1475.2	1498.6	1486.2	1443.3	1518.5	1454.7
				SD	123.10	122.67	126.41	123.07	117.65	119.68	110.82
				Rank	6	4	7	5	2	3	1
				Best	24518	24134	23870	24157	23420	23682	23420
				Worst	46328	44265	45382	44816	44265	45328	44056
10	50	$U(100,800)$	0.1	Mean	35433.6	35081.7	36475.9	35101.8	33769.5	35488.3	32954.8
				SD	6542.80	6329.57	6430.65	6285.43	6146.79	5982.37	5664.81
				Rank	7	5	6	4	3	2	1
				Best	44732	43957	44235	43783	43165	44062	43147
				Worst	78429	76934	73054	61796	61796	58837	55936
10	100	$U(100,800)$	0.1	Mean	59018.6	59486.5	58439.5	52645.8	52237.7	50765.8	51253.4
				SD	8951.58	8465.43	8578.49	5655.87	5485.52	5533.59	3268.76
				Rank	7	5	6	4	2	3	1
		Mean rank			6.5	4.8	5.4	4.9	2.4	2.9	1.3
		Rank			7	4	6	5	2	3	1

worst fitness and average fitness. In particular, MDPSO is more competitive than IACO in all of these areas. MDPSO also outperforms more than half of the algorithms in terms of standard deviation. Therefore, we can see that a more balanced scheduling scheme can be obtained significantly by the MDPSO algorithm. It is worth mentioning that, for each instance, the standard deviation presented in the results in Table 6 is actually consistent with the Friedman ranking. Although VNS and DPSO ranked first in the 4th and 5th groups of instances, respectively, in the final Friedman ranking, the MDPSO algorithm was firmly in first place, followed by IACO and DPSO, respectively, while the LPT algorithm ranks last. This means that the quality of the solutions obtained by MDPSO and IACO is relatively excellent, and as the scale of the problem instance increases, the solution accuracy of LPT will decrease significantly, and VNS, SA and DPSO will also easily fall into local optimum.

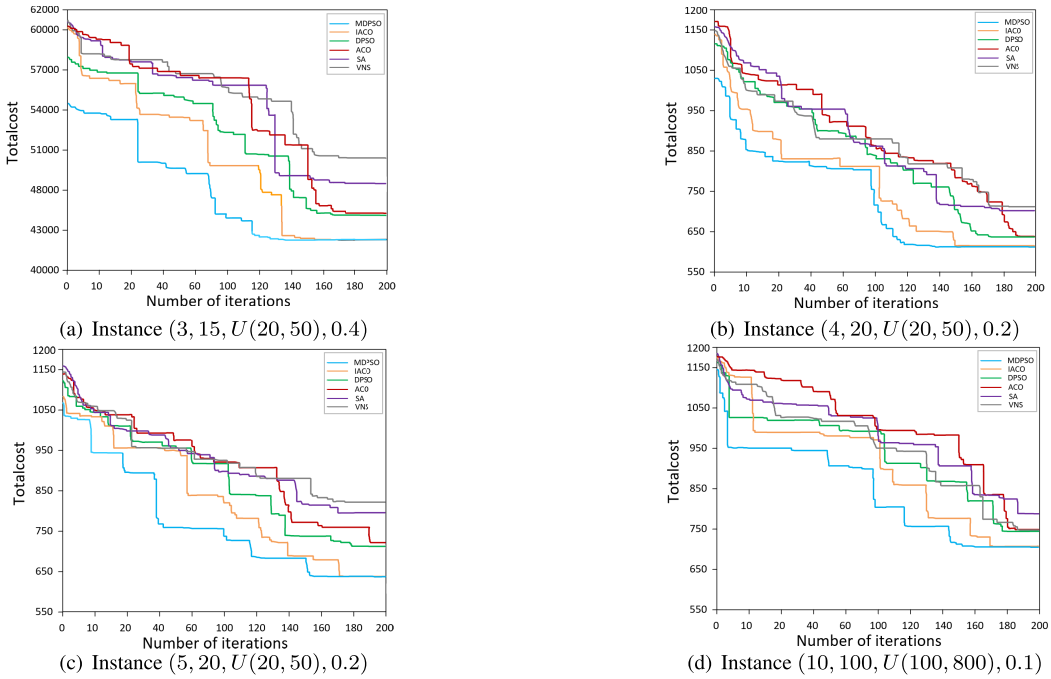
D. ANALYSIS OF RESULTS: SOLUTION COMPARISON AND CONVERGENCE

In order to more comprehensively present the number of good-quality solutions output by these algorithms, we also record the number of corresponding solutions obtained by comparing some previous algorithms with IACO and MDPSO, respectively. We only record the results of 30 independent runs for the 8 instances selected in the previous subsection. The experimental results are shown in Table 7, where we gives an indication on how many times a given algorithm reports a better objective value comparing with respect to the other algorithm. For example, a value num_1/num_2 in column DPSO/MDPSO implies that, out of 30 problems generated by each combination, there are num_1 problems for which DPSO yeilds a better solution than MDPSO, num_2 problems for which MDPSO performs better, and the other $30 - num_1 - num_2$ problems for which DPSO and MDPSO yeild the same total cost.

It can be seen from the comparison results that after 30 experiments on the same set of instances, LPT, VNS, SA, ACO and DPSO produce significantly less good solutions than IACO and MDPSO, among which DPSO has the best performance, with an average proportion of 26.6%, and the average LPT algorithm with the worst performance is only 10.0%. In addition, the larger the instance size is, the smaller the number of times the output solutions by other algorithms

TABLE 7. Quantitative comparison of good solutions.

m	n	p	h	LPT/IACO	VNS/IACO	SA/IACO	ACO/IACO	DPSO/IACO	LPT/MDPSO	VNS/MDPSO	SA/MDPSO	ACO/MDPSO	DPSO/MDPSO	IACO/MDPSO
3	15	$U(1,20)$	0.2	3/18	3/19	2/14	2/14	3/18	4/23	4/22	5/24	5/24	4/23	12/16
3	15	$U(20,50)$	0.4	6/14	5/4	4/6	5/11	3/17	2/7	9/8	12/16	3/5	2/11	14/15
4	20	$U(1,20)$	0.2	3/25	4/23	5/17	8/11	6/19	1/9	1/9	3/16	5/19	11/17	8/9
4	20	$U(20,50)$	0.4	3/22	7/21	5/16	5/8	9/15	2/13	1/5	1/6	8/21	10/15	10/13
5	20	$U(1,20)$	0.2	5/21	4/13	1/8	8/15	5/12	3/17	6/15	4/9	5/24	7/11	14/15
5	20	$U(20,50)$	0.4	2/13	1/9	1/8	9/15	12/17	4/24	3/19	5/18	5/19	6/9	12/13
10	50	$U(100,800)$	0.1	1/28	1/24	2/25	0/21	3/27	1/29	1/26	1/29	3/25	0/11	13/14
10	100	$U(100,800)$	0.1	1/29	1/28	1/29	0/16	0/19	0/29	0/30	0/30	0/30	1/16	1/2
		Average		3/21.3	3.3/17.6	2.6/15.4	4.6/13.9	5.1/18	2.1/18.9	3.1/16.8	3.9/18.5	4.3/20.9	5.1/14.1	10.5/12.1



(a) Instance $(3, 15, U(20, 50), 0.4)$ (b) Instance $(4, 20, U(20, 50), 0.2)$

(c) Instance $(5, 20, U(20, 50), 0.2)$ (d) Instance $(10, 100, U(100, 800), 0.1)$

FIGURE 7. Comparison of convergence of several metaheuristic algorithms.

beat IACO and MDPSO, which highlights the superiority of our improved algorithm in solving GEBPOC. Of course, the experimental results also show that the number of good solutions output by IACO is less than that of MDPSO.

To further observe the convergence of the above several meta-heuristic algorithms, Fig. 7 also shows the change curve of the minimum total cost after 200 iterations when we select instances $(3, 15, U(20, 50), 0.4)$, $(4, 20, U(20, 50), 0.2)$, $(5, 20, U(20, 50), 0.2)$ and $(10, 100, U(100, 800), 0.1)$, respectively. In these four convergence curves, the algorithms IACO and MDPSO have faster convergence rates than other meta-heuristic algorithms. The point to be made here is that the global optimization capability of IACO is stronger than that of VNS, SA, ACO and DPSO. For example, it can be seen from the Fig. 7 (d), because of the basic ACO is not optimized and easily falls into local optimal, the algorithm only converges after the 185^{th} iteration, while the IACO has already converged at the 170^{th} iteration due to the introduction of the VNS method in the algorithm, which impling the ability of IACO to jump out of the local otimum is better than basic ACO algorithm. The basic DPSO algorithm and the improved MDPSO algorithm obtain the optimal solutions at the 179^{th} iteration and 161^{th} iteration, respectively. Therefore, VNS, SA, ACO and DPSO algorithms have significantly weaker convergence than IACO and MDPSO, respectively, because their performance has not been optimized. ACO and DPSO can only obtain sub-optimal solutions to the problem, while the improved algorithm can get close to optimal solutions. Meanwhile, IACO has better convergence than DPSO, but slower than MDPSO.

VI. CONCLUSION

In this paper, we consider the generalized extensible bin packing problem with overload cost using some meta-heuristics based on the existing work of Denton et al. [1], [2], to the best of our knowledge, this is the first attempt to solve this problem with an meta-heuristics algorithm. This model provides a powerful tool for management decision-making in outpatient operating room allocation in a healthcare setting and in servers handling large tasks. According to the characteristics of the problem, we use improved ACO algorithm and discrete particle swarm optimization algorithms to solve this problem. We enhance the positive feedback effect of ant colony optimization by improving the state transition rules

and dynamic adaptive parameters. In order to avoid the premature or stagnant phenomenon of ant colony algorithm in the search, a variable neighborhood search method is also introduced, which further improves the global search ability and convergence speed of the algorithm. In addition, in order to ensure the uniform distribution and high-quality characteristics of the initial particle swarm, some heuristic methods are adopted in the initialization process of the particle swarm, so that the initial particle can cover the entire search space with a large probability. Computational experiments show that for the same problem instance, the proposed IACO and MDPSO algorithms outperform other metaheuristic algorithms in most cases. A number of results are achieved from solving this model and managers can make a reasonable choice according to the actual situation they are dealing with, which is very in line with the needs of economic, social, medical and green manufacturing.

For further studies, since the model studied has many uncertainties in practice, we need to consider more constraints, such as the regular working time of each server may be different, and the cost of overloading each machine may also be different. Of course, in the actual operating room allocation process, since the number of operations to be completed is dynamically known, we must also dynamically make decisions accordingly. We will conduct further research on these cases. Moreover, in the DPSO algorithm, it can be seen from the experimental results that it is not enough to get rid of particles trapped in local minima only by improving the quality of the initial particle population, and the search process of the best solution is often difficult to take into account the balance of ''detection'' and ''development'' at the same time. The new and improved DPSO algorithm will be a very interesting research direction in the future.

REFERENCES

[1] B. T. Denton, A. J. Miller, H. J. Balasubramanian, and T. R. Huschka, ''Optimal allocation of surgery blocks to operating rooms under uncertainty,'' *Oper. Res.*, vol. 58, no. 1, pp. 802–816, 2010.

[2] B. P. Berg and B. T. Denton, ''Fast approximation methods for online scheduling of outpatient procedure centers,'' *Informs J. Comput.*, vol. 29, no. 4, pp. 631–644, Nov. 2017.

[3] P. Dell'Olmo, H. Kellerer, M. G. Speranza, and Z. Tuza, ''A 13/12 approximation algorithm for bin packing with extendable bins,'' *Inf. Process. Lett.*, vol. 65, no. 5, pp. 229–233, Mar. 1998.

[4] P. Dell'Olmo and M. G. Speranza, ''Approximation algorithms for partitioning small items in unequal bins to minimize the total size,'' *Discrete Appl. Math.*, vol. 94, nos. 1–3, pp. 181–191, May 1999.

[5] D. Ye and G. Zhang, ''On-line extensible bin packing with unequal bin sizes,'' *Discrete Math. Theor. Comput. Sci.*, vol. 11, no. 1, pp. 141–152, Jan. 2009.

[6] L. Epstein and T. Tassa, ''Vector assignment schemes for asymmetric settings,'' *Acta Inf.*, vol. 42, nos. 6–7, pp. 501–514, Mar. 2006.

[7] G. J. Woeginger, ''When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)?'' *Informs J. Comput.*, vol. 12, no. 1, pp. 57–74, Feb. 2000.

[8] N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid, ''Approximation schemes for scheduling on parallel machines,'' *J. Scheduling*, vol. 1, no. 1, pp. 55–66, Jun. 1998.

[9] M. G. Speranza and Z. Tuza, ''On-line approximation algorithms for scheduling tasks on identical machines withextendable working time,'' *Ann. Oper. Res.*, vol. 86, pp. 491–506, Jan. 1999.

[10] E. G. Coffman and G. S. Lueker, ''Approximation algorithms for extensible bin packing,'' *J. Scheduling*, vol. 9, no. 1, pp. 63–69, Feb. 2006.

[11] K. Luo and F. C. Spieksma, ''Online bin packing with overload cost,'' in *Algorithms and Discrete Applied Mathematics*, A. Mudgal and C. R. Subramanian, Eds. Cham, Switzerland: Springer, 2021, pp. 3–15.

[12] A. Levin, ''Approximation schemes for the generalized extensible bin packing problem,'' *Algorithmica*, vol. 84, no. 2, pp. 325–343, Feb. 2022.

[13] G. Sagnol and D. S. G. Waldschmidt, ''Stochastic extensible bin packing,'' 2020, *arXiv:2002.00060*.

[14] J. Blazewicz, ''Scheduling preemptible tasks on parallel processors with information loss,'' *Technique et Sci. Informatiques*, vol. 3, no. 6, pp. 415–420, 1984.

[15] X. Chen, M. Sterna, X. Han, and J. Błażewicz, ''Scheduling on parallel identical machines with late work criterion: Offline and online cases,'' *J. Scheduling*, vol. 19, no. 6, pp. 729–736, Dec. 2016.

[16] M. Sterna and K. Czerniachowska, ''Polynomial time approximation scheme for two parallel machines scheduling with a common due date to maximize early work,'' *J. Optim. Theory Appl.*, vol. 174, no. 3, pp. 927–944, Sep. 2017.

[17] X. Chen, W. Wang, P. Xie, X. Zhang, M. Sterna, and J. Błażewicz, ''Exact and heuristic algorithms for scheduling on two identical machines with early work maximization,'' *Comput. Ind. Eng.*, vol. 144, Jun. 2020, Art. no. 106449.

[18] X. Chen, Y. Liang, M. Sterna, W. Wang, and J. Błażewicz, ''Fully polynomial time approximation scheme to maximize early work on parallel machines with common due date,'' *Eur. J. Oper. Res.*, vol. 284, no. 1, pp. 67–74, Jul. 2020.

[19] X. Chen, X. Shen, M. Y. Kovalyov, M. Sterna, and J. Blazewicz, ''Alternative algorithms for identical machines scheduling to maximize total early work with a common due date,'' *Comput. Ind. Eng.*, vol. 171, Sep. 2022, Art. no. 108386.

[20] W.-D. Li, ''Improved approximation schemes for early work scheduling on identical parallel machines with a common due date,'' *J. Oper. Res. Soc. China*, pp. 1–10, Jun. 2022, doi: 10.1007%2Fs40305-022-00402-y.

[21] X. Chen, S. Kovalev, Y. Liu, M. Sterna, I. Chalamon, and J. Błażewicz, ''Semi-online scheduling on two identical machines with a common due date to maximize total early work,'' *Discrete Appl. Math.*, vol. 290, pp. 71–78, Feb. 2021.

[22] M. Xiao, X. Liu, W. Li, X. Chen, M. Sterna, and J. Blazewicz, ''Online and semi-online scheduling on two hierarchical machines with a common due date to maximize the total early work,'' 2022, *arXiv:2209.08704*.

[23] M. Xiao, X. Liu, and W. Li, ''Semi-online early work maximization problem on two hierarchical machines with partial information of processing time,'' in *Proc. Int. Conf. Algorithmic Appl. Manage.* Cham, Switzerland: Springer, 2021, pp. 146–156.

[24] W.-C. Lee, C.-C. Wu, and P. Chen, ''A simulated annealing approach to makespan minimization on identical parallel machines,'' *Int. J. Adv. Manuf. Technol.*, vol. 31, nos. 3–4, pp. 328–334, Nov. 2006.

[25] J. Kennedy and R. Eberhart, ''Particle swarm optimization,'' in *Proc. Int. Conf. Neural Netw. (ICNN)*, vol. 4, Aug. 1995, pp. 1942–1948.

[26] M. Dorigo, ''Optimization, learning and natural algorithms,'' M.S. thesis, Politecnico Di Milan, Milan, Italy, 1992.

[27] D. Whitley, ''A genetic algorithm tutorial,'' *Statist. Comput.*, vol. 4, no. 2, pp. 65–85, 1994.

[28] Q.-K. Pan, B.-H. Zhao, and Y.-G. Qu, ''Ant-colony heuristic algorithm for no-wait flow shop problem with makespan criterion,'' *Comput. Integr. Manuf. Systems-Beijing*, vol. 13, no. 9, pp. 1801–1804 and 1815, 2007.

[29] A. Bauer, B. Bullnheimer, R. F. Hartl, and C. Strauss, ''An ant colony optimization approach for the single machine total tardiness problem,'' in *Proc. Congr. Evol. Comput. (CEC)*, 1999, pp. 1445–1450.

[30] M. D. Besten, T. Stützle, and M. Dorigo, ''Ant colony optimization for the total weighted tardiness problem,'' in *Proc. Int. Conf. Parallel Problem Solving Nature*. Cham, Switzerland: Springer, 2000, pp. 611–620.

[31] D. Merkle and M. Middendorf, ''An ant algorithm with a new pheromone evaluation rule for total tardiness problems,'' in *Proc. Workshops Real-World Appl. Evol. Comput.* Cham, Switzerland: Springer, 2000, pp. 290–299.

[32] D. Merkle and M. Middendorf, ''Ant colony optimization with global pheromone evaluation for scheduling a single machine,'' *Appl. Intell.*, vol. 18, no. 1, pp. 105–111, 2003.

[33] O. Holthaus and C. Rajendran, ''A fast ant-colony algorithm for single-machine scheduling to minimize the sum of weighted tardiness of jobs,'' *J. Oper. Res. Soc.*, vol. 56, no. 8, pp. 947–953, Aug. 2005.

[34] T. C. E. Cheng, A. A. Lazarev, and E. R. Gafarov, "A hybrid algorithm for the single-machine total tardiness problem," *Comput. Oper. Res.*, vol. 36, no. 2, pp. 308–315, Feb. 2009.
[35] C. R. Gatica de Videla, S. C. Esquivel, and G. M. Leguizamón, "An ACO approach for the parallel machines scheduling problem," *Intel. Artif.*, vol. 14, no. 46, pp. 84–95, Mar. 2010.
[36] Z.-F. Hao, R.-C. Cai, and H. Huang, "An adaptive parameter control strategy for ACO," in *Proc. Int. Conf. Mach. Learn. Cybern.*, 2006, pp. 203–206.
[37] X. Liu, X. Zhang, W. Li, and X. Zhang, "Swarm optimization algorithms applied to multi-resource fair allocation in heterogeneous cloud computing systems," *Computing*, vol. 99, no. 12, pp. 1231–1255, Dec. 2017.
[38] X. Chen, V. Chau, P. Xie, M. Sterna, and J. Błażewicz, "Complexity of late work minimization in flow shop systems and a particle swarm optimization algorithm for learning effect," *Comput. Ind. Eng.*, vol. 111, pp. 176–182, Sep. 2017.
[39] B. Deng, "An improved honey badger algorithm by genetic algorithm and levy flight distribution for solving airline crew rostering problem," *IEEE Access*, vol. 10, pp. 108075–108088, 2022.
[40] A. H. Kashan and B. Karimi, "A discrete particle swarm optimization algorithm for scheduling parallel machines," *Comput. Ind. Eng.*, vol. 56, no. 1, pp. 216–223, Feb. 2009.
[41] Q. K. Pan, M. F. Tasgetiren, and Y.-C. Liang, "A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem," *Comput. Oper. Res.*, vol. 35, no. 9, pp. 2807–2839, Sep. 2008.
[42] Q. Yang, H. Zuo, and W. Li, "Land surface model and particle swarm optimization algorithm based on the model-optimization method for improving soil moisture simulation in a semi-arid region," *PLoS ONE*, vol. 11, no. 3, pp. 1–17, 2016.
[43] Q. Yang, J. Wu, Y. Li, W. Li, L. Wang, and Y. Yang, "Using the particle swarm optimization algorithm to calibrate the parameters relating to the turbulent flux in the surface layer in the source region of the Yellow River," *Agricult. Forest Meteorol.*, vol. 232, pp. 606–622, Jan. 2017.
[44] Q.-K. Pan, M. F. Tasgetiren, and Y.-C. Liang, "A discrete particle swarm optimization algorithm for single machine total earliness and tardiness problem with a common due date," in *Proc. IEEE Int. Conf. Evol. Comput.*, Sep. 2006, pp. 3281–3288.
[45] C. Zhou, L. Gao, and H.-B. Gao, "Particle swarm optimization based algorithm for permutation flow shop scheduling," *Acta Electonica Sinica*, vol. 34, no. 11, pp. 2008–2011, 2006.
[46] Z. Lian, X. Gu, and B. Jiao, "A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan," *Appl. Math. Comput.*, vol. 175, no. 1, pp. 773–785, 2006.
[47] Q. Yang, L. Dan, J. Wu, R. Jiang, J. Dan, W. Li, F. Yang, X. Yang, and L. Xia, "The improved freeze–thaw process of a climate-vegetation model: Calibration and validation tests in the source region of the yellow river," *J. Geophys. Res., Atmos.*, vol. 123, no. 23, pp. 13–346, Dec. 2018.
[48] B. Jarboui, N. Damak, P. Siarry, and A. Rebai, "A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems," *Appl. Math. Comput.*, vol. 195, no. 1, pp. 299–308, 2008.
[49] D. Y. Sha and C.-Y. Hsu, "A hybrid particle swarm optimization for job shop scheduling problem," *Comput. Ind. Eng.*, vol. 51, no. 4, pp. 791–808, Dec. 2006.
[50] C.-T. Tseng and C.-J. Liao, "A discrete particle swarm optimization for lot-streaming flowshop scheduling problem," *Eur. J. Oper. Res.*, vol. 191, no. 2, pp. 360–373, Dec. 2008.
[51] N. Mladenović and P. Hansen, "Variable neighborhood search," *Comput. Oper. Res.*, vol. 24, no. 11, pp. 1097–1100, Nov. 1997.
[52] X. Chen, Z. Wang, E. Pesch, M. Sterna, and J. Błażewicz, "Two-machine flow-shop scheduling to minimize total late work: Revisited," *Eng. Optim.*, vol. 51, no. 7, pp. 1268–1278, Jul. 2019.

RAN DING received the B.S. degree from Henan Normal University, Xinxiang, China. He is currently pursuing the M.S. degree in operations research and cybernetics with Yunnan University, Kunming, China. His research interests include discrete optimization, scheduling theory, and intelligent algorithms.

BIN DENG received the B.S. degree in surveying engineering from the Southwest University of Science and Technology, Mianyang, China. He is currently pursuing the Ph.D. degree in operations research and cybernetics with Yunnan University, Kunming, China. His research interests include aviation operations, fair distribution, and intelligent algorithms.

WEIDONG LI received the B.S., M.S., and Ph.D. degrees from Yunnan University, Kunming, China, in 2004, 2007, and 2010, respectively. He is currently working as a Professor and a Doctoral Supervisor with the School of Mathematics and Statistics, Yunnan University. His current research interests include discrete optimization, theoretical computer science, computational economics, and algorithmic game theory and its applications.