Drilling Path Optimization by Optimal Foraging Algorithm

Wei-Bo Zhang, Guang-Yu Zhu

***Abstract*—Drilling path optimization is a crucial issue in current manufacturing systems. In this paper, an optimization algorithm named the Optimal Foraging Algorithm (OFA) is presented based on Optimal Foraging Theory to solve the sequencing problem when many holes must be drilled. The objective of this study is to identify the optimal sequence of drilling operations using OFA to minimize the total processing cost. Since the drilling sequence optimization problem is considered a discrete optimization problem, five operators that can address the integer-encoding vector are built for OFA; thus, a discrete version of OFA is presented. The performance of OFA is evaluated against four other baseline algorithms in solving five real-world problems. The results, including the optimal solutions, the Kruskal-Wallis test, CPU time and the evolution curves, demonstrate that OFA improves the solution by minimizing non-productive time. OFA is verified as a feasible method for solving drilling path optimization problems.**

***Index Terms*—Optimal Foraging Algorithm (OFA), drilling path optimization, hole-making problem, discrete optimization problem, Optimal Foraging Theory**

I. Introduction

AFTER a metal cellphone shell was used by Apple Inc. in iPhone 5, many models of cellphones were launched with a metal shell by the numerous cellphone manufacturers. There are several holes, sometimes a dozen holes, or even dozens of holes that must be made in the metal casting for a cellphone. The cellphone shells are processed in a CNC (Computer Numerically Controlled) drilling-milling center, because the number of holes is large and the machining precision of holes is high. In addition to cellphone shells, there are other special types of products that involve a large number of holes. Examples of such products include boiler plates, drum and trammel screens, connection flanges in steel structures, food-processing separators, and certain portions of printed circuit boards (PCB) [1]. These products can be processed by CNC machines because the holes drilling process using CNC machines has been completely automated. CNC drilling machines can be classified as CNC printed circuit board (PCB) drill, CNC vertical drill, CNC deep-hole drill, drilling center and other large CNC drilling machines [2]. Most commercial CAD software packages include modules for automated generation of CNC code. In CNC code, the common method of tool path selection for drilling is based on the set of ordinary paths such as zig, zig-zag, radial, and spiral tool paths, thus the tool path planning generated from the commercial CAD software is often not fully optimized in terms of the tool travel distance, and ultimately, the total machining time [3].

Due to the point-to-point tool movement in a drilling path optimization (DPO) problem (or hole-making problem), a considerable portion of the processing time is spent on moving the table from one location to another. The fraction of machining time spent on drilling holes can vary from an average of 40% for a typical product to more than 90% for products that involve dense matrices of holes such as the special classes of products introduced in the above paragraph [3]. Therefore, optimization of holes making operations can significantly reduce machining time and directly improve productivity of manufacturing systems.

Because of the significant amount of time required to move the drill bit from one point to another, holes drilling path optimization problems attract the interest of the academicians, researchers, and engineers. The calculation of the minimum tool path length between holes is similar to a highly well-known problem from the operational research field named the traveling salesman problem (TSP). The problem of finding the best path for the points to be drilled can be modeled as TSP in order to reduce the production cost. In this case, the holes to be drilled are the cities, and the cost of travel is the time it takes to move the drill head from one hole to the next. The path of production through a manufacturing facility has often been identified as a TSP [4,5].

Certain meta-heuristic algorithms more suitable for optimizing the drilling process have been found in the literature. Kolahan and Liang [6] employed a tabu-search approach to minimize the total processing cost for drilling operations. Four issues, that is, tool travel time, tool switching time, tool selection and machining speed specification, were considered, and the tabu-search algorithm was used to identify the solution. Adaptive particle swarm optimization (adaptive PSO) was used by Onwubolu and Clerc to solve the problem of path optimization in automated drilling operations [7]. In their study, the tool path of a CNC drilling machine was modeled as a travelling salesman problem (TSP). The authors of reference [8] noted that PSO might convergences prematurely. They proposed a global convergence particle swarm optimization

This work was supported by Collaborative Innovation Center of High-End Equipment Manufacturing in Fujian.

Wei-Bo Zhang is with the School of Mechanical Engineering and Automation, Fuzhou University, Fuzhou, Fujian Province, P.R. China (e-mail: wbzhang@ fzu.edu.cn).

Guang-Yu Zhu is with the School of Mechanical Engineering and Automation, Fuzhou University, Fuzhou, Fujian Province, P.R. China (e-mail: zhugy@ fzu.edu.cn).

algorithm to obtain the global optimization solution. A typical 14-holes work-piece shown in Fig. 1 was proposed in reference [8]. This typical work-piece has been used as a benchmark model for comparison against various heuristic algorithms [9, 10]. These researchers extended their work to different case studies in reference [11]. Dalavia *et al.* attempted to use PSO to reduce overall processing cost of hole-making operations by determining the optimal sequence for hole-making operations [12]. Abbas, *et al.* developed an approach based on ant colony optimization (ACO) to optimize the path planning of a CNC drilling tool between holes for a special class of products that involve a large number of holes arranged in a rectangular matrix [3]. Medina-Rodríguez *et al.* employed a parallel ACO algorithm to identify the best sequence of hole-making to create G code programing for the shortest cutting path [13]. Liu *et al.* presented a new method to optimize the process planning of hole-making with several operations by using different tool sets. The optimization objectives were to minimize the total time of airtime and tool switching time with the aid of ACO algorithm [14]. A firefly algorithm has been used to search for the optimized path in the PCB hole drilling process by Ismail *et al.* [15]. Lim et al. report a combinatorial cuckoo search algorithm for solving a drilling path optimization problem [4]. Tamjidy *et al.* used an evolutionary optimization algorithm based on geographic distribution of biological organisms to resolve the hole-making process problem. The aim of their study was to minimize non-productive time by employing a biogeography based optimization algorithm [16]. Srivastava used one of the newly developed heuristic techniques i.e., intelligent water drop (IWD) algorithm to optimize the PCB drill routing process [9]. Kanagaraj *et al.* presented a charged system search (CSS) algorithm to solve for robotic drilling path optimization involved in printed circuit board (PCB) manufacturing industries [17]. Daadoo used the Shortest Path Search Algorithm to find the optimal route in the PCB hole drilling process [10]. Dalavi *et al.* attempted to use a hybrid method, particle swarm optimization algorithm and shuffled frog leaping algorithm, to optimize the sequence of hole-making operations in a plastic injection mold [18,19]. The path optimization problem is also optimized by using bacterial foraging optimization algorithm [20, 21].

OFA, inspired by the animal Behavioral Ecology Theory—Optimal Foraging Theory, is an efficient stochastic search algorithm [22]. OFA has been used to solve global optimization problems following animal foraging behavior. It is a new population-based evolutionary algorithm. OFA has been tested on benchmark functions that present difficulties common to many global optimization problems. Experimental results indicate the performance of OFA in solving continuous function is better than other tested algorithms.

In this paper, OFA is employed to solve the drilling path optimization problem. OFA is extended to solve discrete optimization problems and provide an approach for solving the drilling path optimization problem.

The main contributions of this study are:

· The drilling path optimization (DPO) problem is solved by OFA. The research shows OFA can be used to solve practical engineering problems. The application field of OFA is extended.

· A discrete version of OFA is presented. OFA can solve continuous function, and can also be used in solving discrete optimization problems.

II. Optimal Foraging Algorithm

In this paper, the optimization problem is defined as follows: $\boldsymbol{X} = [x_1, \cdots, x_i, \cdots x_d]^T$ is required to satisfy $F(\boldsymbol{X}^*) = \min_{\boldsymbol{X} \in R} f(\boldsymbol{X})$, $R = \{\boldsymbol{X} | x_i^L \le x_i \le x_i^U\}$, $i = 1,2,\cdots,d$, where $f(\boldsymbol{X})$ is the objective function; $\boldsymbol{X}$, a d-dimensional state vector, is one solution of the objective function; x_i is the ith component of $\boldsymbol{X}$; $F(\boldsymbol{X})$ is the objective function value and $F(\boldsymbol{X}^*)$ is the optimal objective function value; $\boldsymbol{X}^*$ is the optimal vector, namely, the optimal solution; R is the constraint space constructed by constraints; x_i^L and x_i^U denote the lower and the upper bounds of the ith state vector; and the maximizing optimization problem $f(\boldsymbol{X})$ is equivalent to the minimizing optimization problem $-f(\boldsymbol{X})$.

The process of solving the optimization problem is similar to the process of animal foraging. The neighborhood of the global optimal solution, namely, the local optimal solution of function $f(\boldsymbol{X})$ in the constraint space can be considered the various patches in the animal foraging habitat. The optimization process can be viewed as an animal, following the optimal foraging theory, foraging in different patches to find the optimal patch where the net rate of energy gain can be maximized. After the optimal patch is found, the animal will search the optimal position within the patch according to the model of the best prey. The optimal solution is achieved when the final optimal position is found.

A. Where would the Algorithm Search for the Optimal Solution?

In OFA, the individual is regarded as a foraging animal whose position in a patch denotes a solution of the objective function. This solution is assumed to be the current optimal solution or the optimal foraging position of the individual obtained through foraging. First, the individual must determine where to continue the search. According to the Optimal Foraging Theory, animals always forage in the area of greatest food abundance and always allocate most foraging time to the patch of greatest food[23, 24], since the current forging position and its neighborhood are assumed to be the patch with abundant food, the algorithm will search the optimal solution near the current position. The animal foraging position is expressed by a two-dimensional variable of longitude and latitude and can move along the two directions. The position in OFA is expressed by a d-dimensional variable; the method used to search each vector of the better position near the current position is as follows:

$$x_i^{t+1} = x_i^t - k \times r_{1i} \times \Delta x_i^t + k \times r_{2i} \times \Delta x_i^t \qquad (1)$$

where x_i^{t+1}, is the ith vector of a position obtained after t+1 times foraging of an individual animal, and $\boldsymbol{X}^{t+1} = [x_1^{t+1}, \cdots, x_i^{t+1}, \cdots x_d^{t+1}]^T$ is a new position (obtained solution); x_i^t is the ith vector of a position obtained after t times foraging, and $\boldsymbol{X}^t = [x_1^t, \cdots, x_i^t, \cdots x_d^t]^T$ is the current position; k is the scale factor; $r_{1i}, r_{2i} \in [0,1]$ are uniform random numbers; Δx_i^t is the ith vector increment of t times foraging and $\Delta \boldsymbol{X}^t = [\Delta x_1^t, \cdots, \Delta x_i^t, \cdots, \Delta x_d^t]$ denotes the position increment of t times foraging.

B. *When will the Algorithm Leave the Local Optimal Solution and Continue Searching?*

Studies indicate that animals do not allocate all available time to the area of the greatest food abundance during foraging. According to the optimal foraging theory, animals always allocate the greatest amount of time to the patch of greatest food abundance and progressively less time to progressively worse areas [23]. The marginal value theorem of optimal foraging theory states that an animal should leave a patch when its rate of food intake in the patch drops to the average rate of the habitat [25, 26]. Research on gregarious species, such as birds (*Cranes*, *Gnamptogenys moelleri*), show that leaving rich patches earlier than predicted could benefit the birds in the long term, since it would allow them to visit more patches per day than a simple rate-maximizing rule [27], thus the probability of finding another rich patch will be high [27, 28]. In establishing OFA, we need to find the answers for: when is the best time to leave and how does the individual leave the current local solution? We adopt "recruitment" behavior for the answer to this question. Recruitment is often used by some insects during foraging to achieve the best searching efficiency [29, 30]. During foraging cooperation among animals is often needed. When some animals find prey, the animal will soon recruit other animals; this recruitment behavior ensures animals always tend to the patch of the greatest prey. Therefore, we introduce the concepts of "Group" and "Recruitment" in OFA. "Group" means a fauna composed of N (N=2,3,…) individuals who dispersed in different patches with different quality, whose essence is a set of solutions in the constraint space. The meaning of "Recruitment" is explained as follows: after searching, the objective function value of each individual's position of a group is calculated, and individuals are ordered based on the values from the best to the worse. Next, the individual with a better value recruits an individual with a poorer value; thus the individuals will tend to move from the patches with low quality to the patches with higher quality.

For OFA with the "Group" concept, $\Delta \boldsymbol{X}^t$ is defined as the recruitment increment of tth foraging. The recruitment increment is as follows: during the tth foraging, after being sorted according to the function value to form a sequence, the individuals scattered in various patches are regarded as a recruited individual in turn. From this sequence an individual with a better value compared with another individual (the recruited individual) is selected randomly as the recruit individual. The recruited individual moves one position increment towards the recruit individual; thus the global information of the better individual is shared by other individuals. The best individual (located in the first place of the sequence) will move one position increment towards the worst one, which obviously is unnatural, but it will enlarge the searching space of the group for OFA and will avoid being trapped in a local optimal solution. Each vector increment Δx_{ji}^t of the recruitment increment $\Delta \boldsymbol{X}_j^t$, $\Delta \boldsymbol{X}_j^t = [\Delta x_{j1}^t, \cdots, \Delta x_{ji}^t, \cdots, \Delta x_{jd}^t]$, for the jth individual to move from the current position $\boldsymbol{X}_j^t$ to the new position $\boldsymbol{X}_j^{t+1}$, must be calculated separately. The mathematical description of the ith vector increment Δx_{ji}^t is given:

$$\begin{cases} \Delta x_{ji}^t = x_{bi}^t - x_{ji}^t & \begin{array}{c}(F_b^t < F_j^t\,;\; F_j^t \neq \min(F_1^t, \cdots, F_N^t)\,; \\ b = 1, \cdots, N;\; j = 1, \cdots, N;\;\; i = 1, \cdots, d)\end{array} \\ \Delta x_{ji}^t = x_{Ni}^t - x_{ji}^t & \begin{array}{c}(\,F_j^t = \min(F_1^t, \cdots, F_N^t)\,; \\ F_N^t = \max(F_1^t, \cdots, F_N^t)\,;\;\; i = 1, \cdots, d)\end{array} \end{cases} \tag{2}$$

where Δx_{ji}^t denotes the recruitment increment of the jth individual's ith vector during tth search; x_{bi}^t, x_{ji}^t, x_{Ni}^t denote, respectively, during t times foraging, the ith vector of the position obtained by the recruit individual b, the ith vector of the position obtained by the recruited individual j and the ith vector of the worst position of the group. F_b^t, F_j^t, F_N^t denote the corresponding objective function values of each position.

$$\begin{cases} x_{ji}^{t+1} = x_{ji}^t - k \times r_{1ji} \times (x_{bi}^t - x_{ji}^t) + k \times r_{2ji} \times (x_{bi}^t - x_{ji}^t) \\ \qquad (F_b^t < F_j^t\,;\; b = 1, \cdots, N;\; j = 1, \cdots, N; \\ \qquad F_j^t \neq \min(F_1^t, \cdots, F_N^t)\,;\; i = 1, \cdots, d) \\ x_{ji}^{t+1} = x_{ji}^t - k \times r_{1ji} \times (x_{Ni}^t - x_{ji}^t) + k \times r_{2ji} \times (x_{Ni}^t - x_{ji}^t) \\ \qquad (\,F_N^t = \max(F_1^t, \cdots, F_N^t)\,; \\ \qquad F_j^t = \min(F_1^t, \cdots, F_N^t)\,;\;\; i = 1, \cdots, d;) \end{cases} \tag{3}$$

In OFA, each vector x_{ji}^{t+1} of the new position $\boldsymbol{X}_j^{t+1}$ of any individual j can be obtained with formula (3), which is used to determine when and how the individual will leave the current position. When better positions have been found by some individuals, the current individual will search much better positions near these better positions by moving towards one of these individuals. If the current individual is the best individual, then it will search a better position near the current position by moving towards the worst individual.

Each individual of OFA can forage near his own current position by using formula (3). Each individual, except the best individual, tends to move to the position of a better individual.

C. *How to Verify Whether the Obtained Solution is Better or Not?*

The foraging behavior of the group is completed after all the individuals in the group updated their positions according to formula (3), then the group's foraging number t is plus 1. Next, the following question must be answered: "Whether the obtained new position is better or worse and can this new position be used in the following search?" The precise model of prey choices developed by Krebs et al. [31] is adopted in OFA to answer this question. The mathematical expression of this model is given in formula (4). In this model, it is assumed that only two types of prey exist during animal foraging. They are profitable prey type 1 and unprofitable prey type 2. According to the Optimal Foraging Theory, animals always choose prey with a better energy gain. If formula (4) is satisfied, unprofitable prey type 2 will be ignored by the animal:

$$\frac{\lambda_1 E_1}{1+\lambda_1 h_1} > \frac{E_2}{h_2} \tag{4}$$

where E_1 denotes the net energy gain of profitable prey type 1,

E_2, the net energy gain of unprofitable prey type 2, h_1, the handling time for the profitable prey type 1, h_2, the handling time for the unprofitable prey type 2, λ_1, the rate of encounter of the profitable prey type 1.

In OFA, when judging whether a better position is obtained with the precise model of prey choices, the position is considered to be the prey in the model; the corresponding function value is considered to be the energy of the prey. Whether the position obtained after the t+1 times search by individual j is better than that obtained after t times search is determined as follows: we hypothesize that a better position can be obtained with an increase of the searching times, which means profitable prey is obtained. Thus, the position obtained after t+1 times search is seen as the profitable prey. The corresponding function value is seen as the energy of the profitable prey, namely, F_j^{t+1} is E_1. The position obtained after t times search is seen as unprofitable prey. The corresponding function value is considered to be the energy of the unprofitable prey type, namely, F_j^t is E_2. Formula (4) is deduced from the total gained energy; the time cost for the energy gain is made up of time spent searching for prey. The time cost for searching for the optimal position is similar to the time spent searching for prey, which can be expressed by the group's foraging number t, for the essence of the group's foraging number is also a time measure for searching for better positions. For maximizing the optimization problem, in OFA, formula (4) can be described as:

$$\frac{\lambda_j^{t+1} F_j^{t+1}}{1+\lambda_j^{t+1}(t+1)} > \frac{F_j^t}{t} \qquad (5)$$

For the minimizing optimization problem, the maximizing F is equivalent to the minimizing $-F$; formula (5) is redefined as:

$$\frac{\lambda_j^{t+1} F_j^{t+1}}{1+\lambda_j^{t+1}(t+1)} < \frac{F_j^t}{t} \qquad (6)$$

If formula (6) is satisfied, the position obtained after t+1 times search can be used by the individual for the next search. Otherwise, the position obtained after t+1 times search will be ignored, and the position obtained after t times search is kept for the next search. $\lambda_j^{t+1} \in [0,1]$ is a random number.

Uniting formula (3) and formula (6), the optimal foraging algorithm is defined.

III. Drilling Path Optimization Problem

When drilling a group of holes in a work-piece with a CNC machine, the worktable moves in x-axes and y-axes. The worktable reaches an exact drilling position, the drilling bit moves in the negative z direction to carry out the drilling operation on the work-piece. All holes will be numbered with an integer, indicated in Fig. 1, and the drilling sequence can be described as $\boldsymbol{X} = [x_1, \dots, x_i, \dots, x_n]^T$, for example, $\boldsymbol{X} = [5,3, \dots, 6, \dots, 8]^T$is a drilling sequence, The component "5" is the number of a hole. Measuring the distance between two sequential holes is based on how the worktable moves. The optimum drilling sequence can minimize the total worktable movements, thus shorten the no-cutting time and lengthen working life of the worktable driving system. The total operation time of the process varies when different drilling paths are selected, thus the operation time is selected as the objective function of OFA for drilling path optimization problems. The objective function is given as:

$$f(\boldsymbol{X}) = t_{move} + n \times t_{drill} \qquad (7)$$

where, n is the number of holes to be machined; t_{drill} is the time used for the drilling operation, the parameter is decided by the relative parameters in the numerical control (NC) program and has nothing to do with the selected drilling path; and t_{move} is the time used when the worktable is moving, this parameter is affected by the selected drilling path. Therefore, the formula (7) can be simplified as formula (8).

$$f(\boldsymbol{X}) = t_{move} \qquad (8)$$

The worktable moves in x and y directions driven by the motor in x-axes and y-axes. Suppose there are two holes i and j of the work-piece, the coordinates are (p_{xi},p_{yi}), (p_{xj},p_{yj}), then the rectilinear distances between two holes is:

$$d_{ij}^R = |p_{xi} - p_{xj}| + |p_{yi} - p_{yj}| \qquad (9)$$

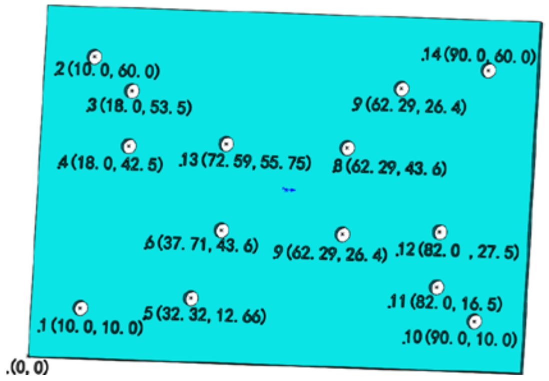

Fig. 1 14-holes work-piece

During the machining operation, the total distance moved by the worktable is the sum of the absolute values of the distances that the worktable moved in the x and y directions. Suppose the worktable moves with a uniform velocity, the time used by the worktable while moving in the x and y directions is given as:

$$t_{move} = \sum \frac{|p_{xi} - p_{xj}|}{v_x} + \sum \frac{|p_{yi} - p_{yj}|}{v_y} \qquad (10)$$

where v_x, v_y are the linear velocities of the worktable in the x-axes and y-axes. In common situations, set $v_x = v_y = v$ in the NC program to simplify formula (10), v has no influence on the non-productive time which is caused by the drilling sequence. Substituting simplified formula (10) into formula (8) to get the objective function $f(\boldsymbol{X})$ of DPO.

$$f(\boldsymbol{X}) = \sum |p_{xi} - p_{xj}| + \sum |p_{yi} - p_{yj}| \qquad (11)$$

IV. OFA for Solving the Drilling Path Optimization Problem

OFA has been proven to be a better algorithm in solving a continuous function. The coding scheme for OFA is a real-coding scheme. However, the DPO problem is a discrete optimization problem and the drilling sequence $\boldsymbol{X} = [x_1, \dots, x_i, \dots, x_n]^T$ is an integer-coding vector. To make OFA fit discrete optimization problems, a discrete version of OFA is presented. Because the state vector (or drilling sequence) $\boldsymbol{X}$ is represented by integer code, the operators add (+), subtract (-), multiply (×) in formula (3) cannot be performed by the method of numerical calculation. Thus, formula (2) and (3) are rewritten as formula (12) and (13). The operators in formula (13) are redefined by improving the method given in reference [32].

$$\begin{cases} \Delta \boldsymbol{X}_j^t = \boldsymbol{X}_b^t \ominus \boldsymbol{X}_j^t & \begin{array}{c}(F_b^t < F_j^t\,;\ F_j^t \neq \min(F_1^t, \cdots, F_N^t)\,; \\ b = 1, \cdots, N;\ j = 1, \cdots, N)\end{array} \\ \Delta \boldsymbol{X}_j^t = \boldsymbol{X}_N^t \ominus \boldsymbol{X}_j^t & \begin{array}{c}(\,F_j^t = \min(F_1^t, \cdots, F_N^t)\,; \\ F_N^t = \max(F_1^t, \cdots, F_N^t)\,)\end{array} \end{cases} \tag{12}$$

$$\begin{cases} \boldsymbol{X}_j^{t+1} = \boldsymbol{X}_j^t - k \otimes r_{1j} \times (\boldsymbol{X}_b^t \ominus \boldsymbol{X}_j^t\,) + k \otimes r_{2j} \times (\boldsymbol{X}_b^t \ominus \boldsymbol{X}_j^t\,) \\ \qquad (F_b^t < F_j^t\,;\ b = 1, \cdots, N;\ j = 1, \cdots, N; \\ \qquad F_j^t \neq \min(F_1^t, \cdots, F_N^t)) \\ \boldsymbol{X}_j^{t+1} = \boldsymbol{X}_j^t - k \otimes r_{1j} \times (\boldsymbol{X}_N^t \ominus \boldsymbol{X}_j^t) + k \otimes r_{2j} \times (\boldsymbol{X}_N^t \ominus \boldsymbol{X}_j^t) \\ \qquad (\,F_N^t = \max(F_1^t, \cdots, F_N^t)\,; \\ \qquad F_j^t = \min(F_1^t, \cdots, F_N^t)) \end{cases} \tag{13}$$

A. *The Operators of the Discrete OFA*

1) *Operator "Θ"*

At time t, there are two random vectors $\boldsymbol{X}_1^t$ and $\boldsymbol{X}_2^t$. During the "Θ" process, the values of the same bit of the two vector's integer code are compared. If the values are different, then the value of the vector before Θ is kept for the same bit in the code of position difference; if the values are the same, then the same bit of the position difference code will set to null.

For example: $\boldsymbol{X}_1^t$ can be assumed as (h_1, h_3, h_4, h_2) and $\boldsymbol{X}_2^t$ as (h_1, h_2, h_3, h_4), then the value of $\Delta \boldsymbol{X}_2^t = \boldsymbol{X}_1^t \ominus \boldsymbol{X}_2^t$ can exist. Since the first bits of $\boldsymbol{X}_1^t$ and $\boldsymbol{X}_2^t$ are the same, the first bit of $\Delta \boldsymbol{X}_2^t$ is then set to null; the second bit of $\boldsymbol{X}_1^t$ is h_3, and the second bit of $\boldsymbol{X}_2^t$ is h_2; they are different, so the second bit of $\Delta \boldsymbol{X}_2^t$ is set to h_3. All bits are set this way, finally we get $\Delta \boldsymbol{X}_2^t = (*, h_3, h_4, h_2)$, where symbol * means null.

2) *Operator "⊗"*

Because $\Delta \boldsymbol{X}_j^t$ is an integer-coding vector, r_{1j} or r_{2j} cannot be a random number between 0 and 1. Thus, in this paper, r_{1j} or r_{2j} is a random binary series, and the series length N is similar to $\Delta \boldsymbol{X}_j^t$. The scale factor k belongs to [0,1]. The operator "⊗" means the values in each bit of r_{1j} or r_{2j} are changed based on the probability. There are N random numbers β_j $(\beta_j \in [0,1])$ $(j = 1, \cdots, N)$. For each bit of r_{1j} or r_{2j}, if $k < \beta_j$, the value in this bit will be changed, such as "1" is changed to "0" or "0" is changed to "1"; if $k > \beta_j$, the value in this bit will not be changed.

For example: $r_{1j} = (0,1,0,1)$. For the first bit, k is less than β_1, "0" will be changed to "1". For the second bit, k is more than β_2, "1" will not be changed. When all bits are set in this way, we obtain $k \otimes r_{1j} = (1,1,0,1)$.

3) *Operator "×"*

A binary series multiply (×) of the recruitment increment $\Delta \boldsymbol{X}_j^t$ is realized as follows. If the bit value of the binary series is 1, then the bit value of $\Delta \boldsymbol{X}_j^t$ is kept in the same bit of the product; if the bit value is 0, then the same bit in the product is set to null. If the bit value of $\Delta \boldsymbol{X}_j^t$ is null, the same bit in the product is set to null.

For example: the binary series is $k \otimes r_{1j} = (1,1,0,1)$, then $k \otimes r_{1j} \times \Delta \boldsymbol{X}_2^t = (1,1,0,1) \times (*, h_3, h_4, h_2) = (*, h_3, *, h_2)$.

4) *Operator "+"*

During the "+" process, the bit value of the current vector (the current position of an individual animal) $\boldsymbol{X}_j^t$ is checked from the first bit. If the value does not appear in the recruitment increment $\Delta \boldsymbol{X}_j^t$, then the bit value is kept in the vector. If the value appears in the $\Delta \boldsymbol{X}_j^t$, then the bit number is recorded and the value of the same bit in $\boldsymbol{X}_j^t$ is exchanged with the current bit value. The current bit value is checked again until it cannot be found in $\Delta \boldsymbol{X}_j^t$, then the other bit values are checked in turn with the same method.

For example, $\boldsymbol{X}_2^t$ can be assumed as (h_1, h_2, h_3, h_4), $\Delta \boldsymbol{X}_2^t$ as $(*, h_3, *, h_2)$, then

$$\begin{aligned} \boldsymbol{X}_2^{t+1} = \boldsymbol{X}_2^t + \Delta \boldsymbol{X}_2^t &= (\boxed{h_1}, h_2, h_3, h_4) + (*, h_3, *, h_2) \\ &= (h_1, \boxed{\underline{h_2}}, h_3, \underline{h_4}) + (*, h_3, *, \underline{h_2}) \\ &= (h_1, \underline{h_4}, h_3, \underline{h_2}) + (*, h_3, *, h_2) \\ &= (h_1, \boxed{h_4}, h_3, h_2) + (*, h_3, *, h_2) \\ &= (h_1, \underline{h_4}, \boxed{h_3}, b_2) + (*, \underline{h_3}, *, h_2) \\ &= (h_1, \underline{h_3}, \boxed{h_4}, h_2) + (*, h_3, *, h_2) \\ &= (h_1, h_3, h_4, \boxed{h_2}) + (*, h_3, *, h_2) = (h_1, h_3, h_4, h_2) \end{aligned}$$

5) *Operator "–"*

During the "–" process, the bit value is checked from the last bit to the first bit for the current vector $\boldsymbol{X}_j^t$, and the first bit to the last bit for $\Delta \boldsymbol{X}_j^t$. For instance, the last bit value of $\boldsymbol{X}_j^t$ is compared with the first bit value of $\Delta \boldsymbol{X}_j^t$, and the second bit value from the last of $\boldsymbol{X}_j^t$ is compared with the second bit value of $\Delta \boldsymbol{X}_j^t$. If the value is the same, the bit value is kept in the vector. If the bit value of $\Delta \boldsymbol{X}_j^t$ is null (*), the bit value in $\boldsymbol{X}_j^t$ is kept. If the two bit values are different, the bit value of $\Delta \boldsymbol{X}_j^t$ is found in $\boldsymbol{X}_j^t$, then the two bit values are exchanged in $\boldsymbol{X}_j^t$.

For example, assume $\boldsymbol{X}_2^t = (h_1, h_2, h_3, h_4)$, $\Delta \boldsymbol{X}_2^t = (*, h_3, *, h_2)$, then

$$\begin{aligned} \boldsymbol{X}_2^{t+1} = \boldsymbol{X}_2^t - \Delta \boldsymbol{X}_2^t &= (h_1, h_2, h_3, \boxed{h_4}) - (\boxed{*}, h_3, *, h_2) \\ &= (h_1, h_2, \boxed{h_3}, h_4) - (*, \boxed{h_3}, *, h_2) \\ &= (h_1, \boxed{h_2}, h_3, h_4) - (*, h_3, \boxed{*}, h_2) \\ &= (\boxed{h_1}, \underline{h_2}, h_3, h_4) - (*, h_3, *, \boxed{h_2}) = (h_2, h_1, h_3, h_4) \end{aligned}$$

B. *The Application Steps of OFA*

When OFA is used in DPO problems, a foraging position represents a drilling sequence. For DPO problems, a foraging position or a drilling sequence is a solution. OFA is implemented in DPO problems as follows: first the initial foraging position of each individual is generated randomly in the constraint space, then the objective function value of each foraging position is calculated by formula (11) and then ordered. Next, the new foraging position of each individual is obtained with formula (13), and the objective function value represented by the new position is checked with formula (6) to determine if the corresponding solution of the objective function value is better. If the solution is better, the new position is preserved for the next foraging, otherwise the new position is ignored and the former position is used for the next foraging. Repeat the above process, searching positions repeatedly with OFA. The best position obtained by each time of foraging is recorded during the search; thus when the algorithm is terminated, the recorded position is the final optimal solution. The flow chart of OFA is given in Fig. 2.

```
1: Initialization. Generate uniformly distributed random
   group $P^1$ composed $N$ individuals
2: Compute the objective function value $F_j^1$ of each
   individual $X_j^1$ in the original group $P^1$;
3: Order $F_j^1$ from best to worse to obtain $F_j^1$ ($j$=1,2,..., $N$),
   and the corresponding sequence of $X_j^1$ ;
4: $t$ = 1;
5:do;
6: Produce new foraging position $X_j^{t+1}$ by using formula
   (13) and Compute $F_j^{t+1}$ by formula (11);
7: By using formula (6), the $X_j^{t+1}$ is determined whether it
   can be used for the next foraging;
8: Order $F_j^{t+1}$ from best to worse and the corresponding
   sequence of $X_j^{t+1}$;
9: Memorize the best foraging position achieved so far;
10: $t$ = $t$ + 1;
11: until $t$ <= $Max\_t$;
```
Fig. 2 The flow chart of Optimal Foraging Algorithm for DPO problem (*Max_t* is the maximum group's foraging number.)

V. Simulation results

To verify the effectiveness of OFA, OFA is compared with four other baseline algorithms to solve four sizes of instances. The four baseline algorithms are genetic algorithms (GAs) [33, 34], differential evolution (DE) [35], adaptive particle swarm optimization algorithm (PSO) [33, 36-38] and ant colony optimization algorithm (ACO) [39, 40].

The performances of these five algorithms are compared by solving two DPO problems and three TSP problems. The data of two DPO include a 9-hole work-piece problem and a 14-hole work-piece problem, and the data of three TSP benchmark problems include a 20-node TSP problem, a 30-node TSP problem named Oliver30 problem [41] and a 51-node TSP problem named Eil51 problem [42].

A. Experimental Setup

The parameter settings for these five algorithms are as follows: the population size is 20 individuals and the iteration number (Max_t) is 20000. Each problem was repeated 30 times with different random seeds. The function value for the optimal solution $\boldsymbol{X}^*$ is $F(\boldsymbol{X}^*)$. The average function value of these thirty $F(\boldsymbol{X}^*)$ is calculated.

The parameters of OFA are the scaling factor $k = t/max\_t$.

For GAs, the linear crossover operator and the non-uniform mutation operator are adopted. The selection operator is normgeomselect. The parameters are the crossover probability $p_c = 0.85$, the probability of mutation $p_m = 0.3$, the parameter b used by the non-uniform mutation is 5.

The DE's parameters are the scaling factor $F = 1.0$ and the crossover rate $C_r = 0.5$.

The PSO's parameters are the acceleration constant $c_1 = c_2 = 2.05$, the adaptive inertia weight $\omega = 0.9 - 0.5 \times (t/Max\_t)$ [43].

The ACO's parameters are the phromone exponential weight $\alpha = 1$, the heuristic exponential weight $\beta = 1$, the evaporation rate $\rho = 0.05$, $Q = 1$ [40].

B. Experimental Results Analysis

The results obtained with these five algorithms are given in Table 1. The results are the average function value of the optimal solutions obtained in 30 times running. The results indicate that OFA can identify the best solution for the 9-hole work-piece, the second best solution with 14-hole work-piece problem and 20-node TSP. DE achieves the best solution with the 9-hole work-piece and 14-hole work-piece problem. ACO achieves the best solution with the 20-node TSP and Oliver30 problem, the second best solution with Eil51 problem. GAs achieves the best solution with the Eil51 problem, the second best solution with 9-hole work-piece and Oliver30 problem. The data in table 1 are analyzed with Kruskal-Wallis test, and the results are given in Table 2. From the results, we can see that p-value (p=0.632) is more than 0.05. From the statistical analysis, there is no significant difference in these five algorithms for solving the five real-word problems.

Table 1 Comparison of computational accuracy.
(The best solution is emphasized in boldface. The second best solution is underlined.)

	The average function value of optimal solutions / Std dev				
	GAs	*DE*	*PSO*	*ACO*	*OFA*
9-hole work-piece problem	<u>285.77</u> /4.23	**285** /0	401.28 /30.5	320.29 /5.8e-14	**285** /0
14-hole work-piece problem	306.1 /29.5	**280.69** /0.5	533.29 /33.93	284.22 /0.58	<u>283.19</u> /4.19
20-node TSP	28.79 /1.83	27.52 /1.12	50.09 /2.66	**24.77** /0.17	<u>27.34</u> /1.94
Oliver30 problem	<u>589.59</u> /38.04	671.14 /41.94	1.1e03 /49.73	**427.64** /1.84	660.11 /57.08
Eil51 problem	**417.65** /30.56	482.75 /24.97	754.67 /36.54	<u>443.25</u> /3.11	472.43 /42.9

Table 2 Kruskal-Wallis ANOVA table

Source	SS	df	MS	Chi-sq	Prob>Chi-sq
Groups	139.1	4	34.78	2.57	0.632
Error	1160.4	20	58.02		
Total	1299.5	24			

The CPU times as the time complexity of each algorithm are given in Table 3. The CPU time of OFA is the third short in five algorithms. Combined with the result of Table 1, OFA get a balance in term of solution quality and time complexity.

Table 3 Comparison of CPU time.

	GAs	*DE*	*PSO*	*ACO*	*OFA*
9-hole work-piece problem	16	5.6	6.9	65.5	10.9
14-hole work-piece problem	16.7	5.9	7.2	93.7	11.7
20-node TSP	43.6	17.3	17.9	149.9	23.4
Oliver30 problem	58.7	23.7	24.8	223.1	31.3
Eil51 problem	58.6	24.2	25.3	389.5	33.2

Fig. 3 - Fig. 7 illustrate the distribution state of the thirty solutions for each algorithm. The solutions of five algorithms are ordered in ascending. The first one is regarded as the

optimal solution. The solutions from the second one to the fourth are regarded as near-optimal solutions. From these figure, it is shown that OFA and GAs can obtain optimal solution or near-optimal solution in four problems except Oliver30. OFA can obtain optimal solution in three problems, including 9-hole work-piece, 14-hole work-piece and Eil51. GAs can also obtain optimal solution in three problems, including 9-hole work-piece, 14-hole work-piece and 20-node TSP. But, the percentage of OFA's optimal solution and near-optimal solution is greater than or equal to GAs'. ACO can obtain optimal solution or near-optimal solution in four problems except Eil51, and can obtain near-optimal solution in two problems. Thus, OFA has more chance to obtain optimal solution or near-optimal solutions.

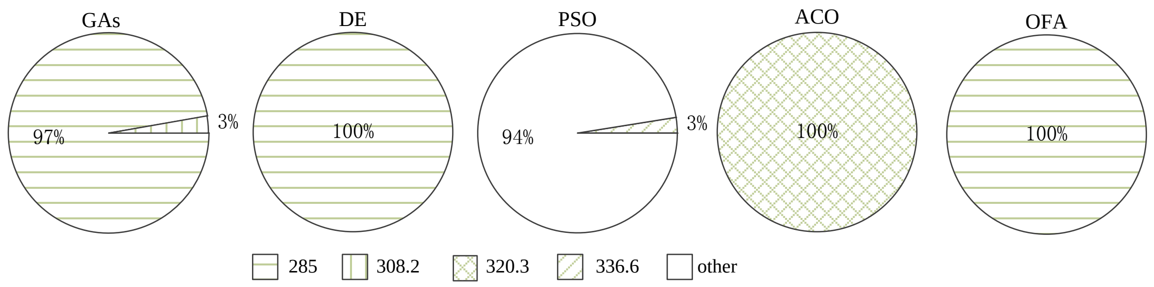

Fig. 3 Distribution drawing of the optimal solution or near-optimal solution of the 9-hole work-piece

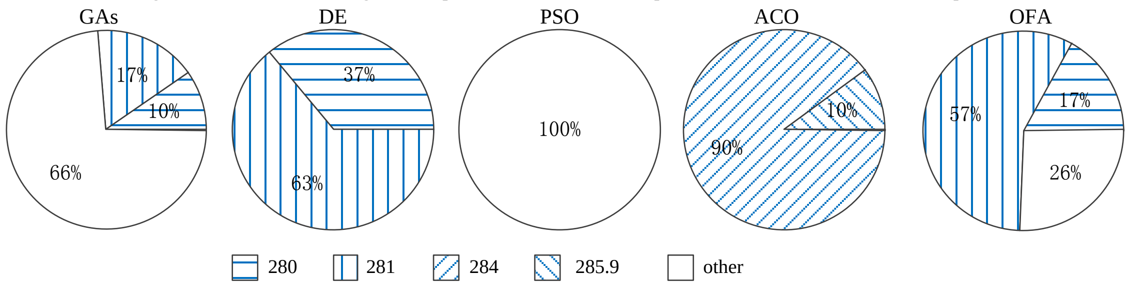

Fig. 4 Distribution drawing of the optimal solution or near-optimal solution of the 14-hole work-piece

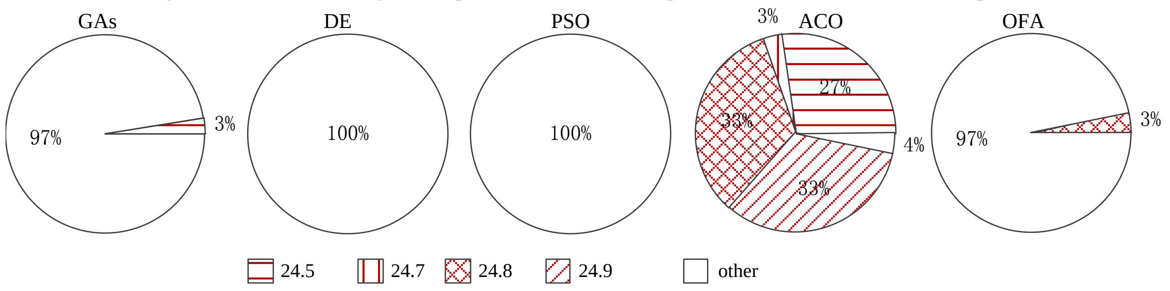

Fig. 5 Distribution drawing of the optimal solution or near-optimal solution of the 20-node TSP

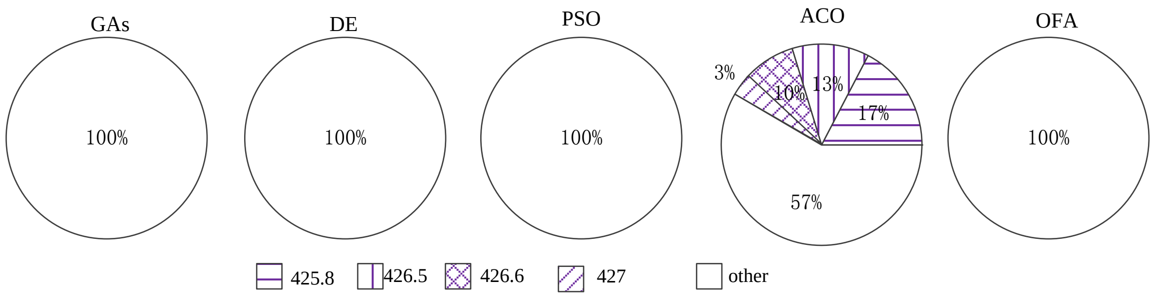

Fig. 6 Distribution drawing of the optimal solution or near-optimal solution of the Oliver30 problem

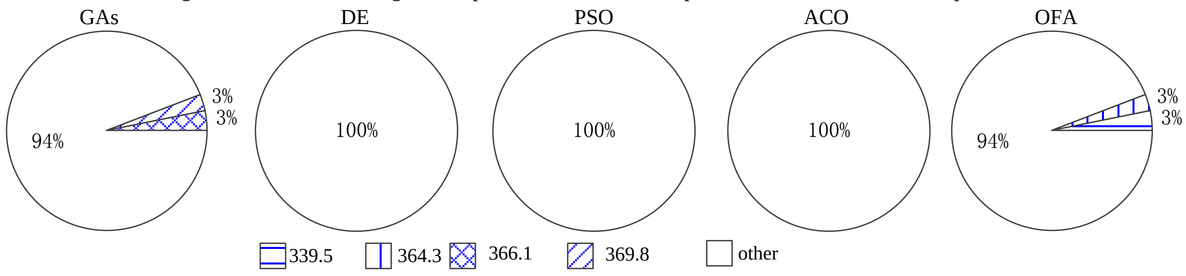

Fig. 7 Distribution drawing of the optimal solution or near-optimal solution of the Eil51 problem

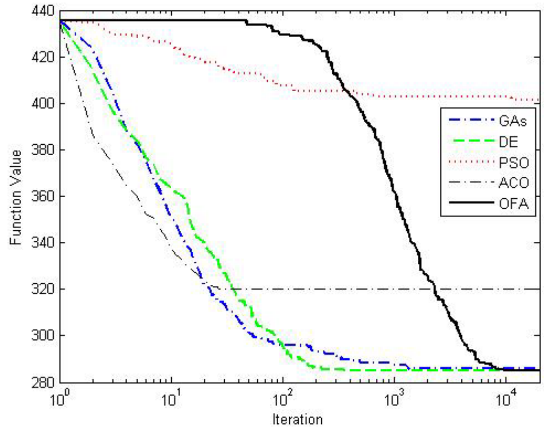

Fig. 8 the 9-hole work-piece problem

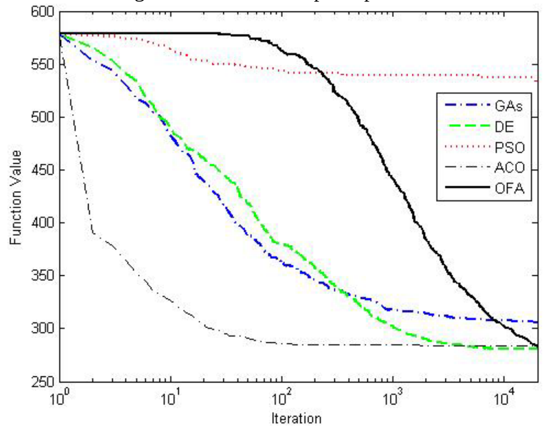

Fig. 9 the 14-hole work-piece problem

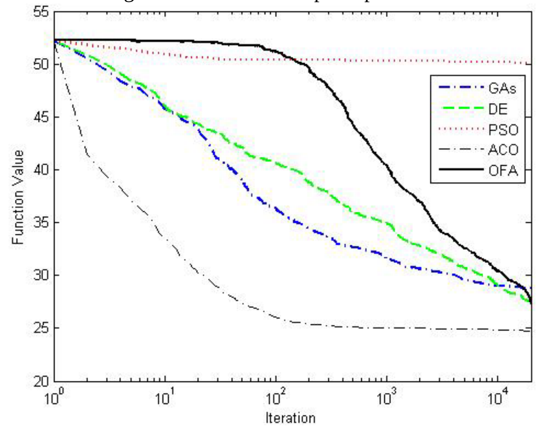

Fig. 10 20-node TSP

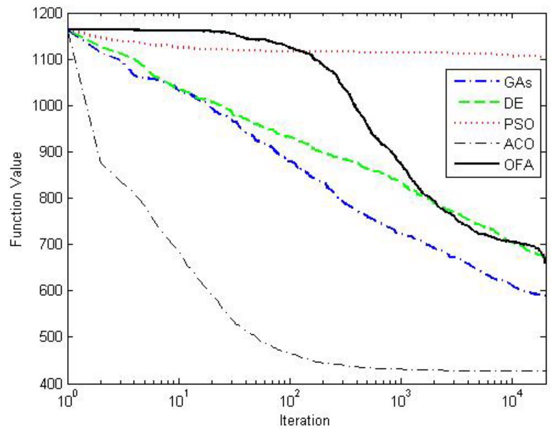

Fig. 11 Oliver30 problem

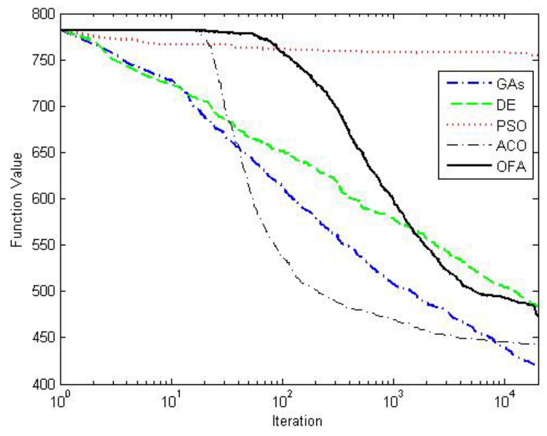

Fig. 12 Eil51 problem

Fig. 8 - Fig. 12 describe the evolution curves of these five problems. The convergence speed of OFA is slow in the initial stage, but then becomes rapid.

Experiments with these five real-world problem are carried out to compare the performance of OFA with GAs, DE, PSO and ACO. From the optimization results obtained in experiments, OFA is as good as or sometimes even better than DE and ACO in terms of the ability to converge to the optimal or the near-optimal solutions. The OFA is definitely an effective optimization tool for a drilling path optimization problem.

VI. Conclusion

A drilling path optimization problem involves a large number of possible sequences to complete drilling operations on a work-piece depending on the location of holes. To achieve efficient drilling operations, proper determination of a drilling sequence, which minimizes the total non-productive time is essential. This paper introduces use of OFA to optimize drilling path optimization problems. The contribution is five operators that can address integer-encoding vectors are built for OFA; thus, a discrete version of OFA is presented. The application area of OFA is extended and OFA can be applied to resolve discrete optimization problems. Four real-world path optimization problems are resolved by OFA and four other test algorithms used for verification. The results demonstrate that OFA is a feasible method for drilling path optimization. OFA is suitable for continuous optimization problems and also discrete optimization problems.

In the future research, different models should be tried in OFA, such as the prey choice model and the patch choice model.

References

[1]. A. T. Abbas, H. Karim, and F. A. Mohamed, “CNC machining path planning optimization for circular hole patterns via a hybrid ant colony optimization approach,” *Mech. Eng. Res.*, vol. 4, no.2, pp.16-29, 2014.
[2] M. Daadoo, Y. A. Daraghmi, “Design and Implementation of Low Cost Computer Numerical Control-Printed Circuit Boards Drilling Machine,” *Int. J. Eng. and Innov. Tech.*, vol.5, no.10, pp.63-67, 2016.
[3] A.T. Abbas, M. F. Aly M F, et. al, “Optimum drilling path planning for a rectangular matrix of holes using ant colony optimization,” *Int. J. Prod. Res.*, vol.49, no.19, pp.5877-5891, 2011.

[4] W. C. E. Lim, G. Kanagaraj, S. G. Ponnambalam, "PCB Drill Path Optimization by Combinatorial Cuckoo Search Algorithm," *Sci. World J.*, Volume 2014, no.2 264518, 2014
[5] J. Li, M. C. Zhou, Q. Sun, et al. "Colored traveling salesman problem". *IEEE T.Cy.*, vol. 45, no.11, pp.2390-2401, 2015.
[6] F. Kolahan, and M. Liang. "Optimization of Hole-making Operations: A Tabu-search Approach." *Int. J. of Mach. Tools and Manuf.* , vol.40, no.12, pp.1735–1753, 2000.
[7] G. C. Onwubolu, and M. Clerc, "Optimal Path for Automated Drilling Operations by a New Heuristic Approach Using Particle Swarm Optimization." *Int. J. Prod. Res.*, vol.42, no.3, pp.473–491, 2004.
[8]G. Y. Zhu, "Drilling path optimization based on swarm intelligent algorithm," *In IEEE Int. Conf. on Robot. Biomimetic*, December. 2006, pp. 193-196.
[9] P. R. Srivastava, "A cooperative approach to optimize the Printed Circuit Boards drill routing process using Intelligent Water Drops," *Comput. Electri. Eng.*, vol.43, pp.270–277, 2015
[10] M. Daadoo, "Path Optimization For Computer Numerical Control-Printed Circuit Boards In Holes Drilling Process - Case Study," *Int. J. Eng. Tech.*, vol.6, no.10, pp.365-377, 2016.
[11] G. Y. Zhu, W. B. Zhang, "Drilling path optimization by the particle swarm optimization algorithm with global convergence characteristics". *Int. J. Prod. Res.*, vol.46, no.8, pp. 2299-2311, 2008.
[12]A.M. Dalavia, P.J. Pawarb, T.P. Singha, "Optimization of hole-making operations for injection mould using particle swarm optimization algorithm". *Int. J. Ind. Eng. Comput.*, vol.6, pp.433–444, 2015.
[13] N. Medina-Rodríguez, O. Montiel-Ross, et. al. "Tool Path Optimization for Computer Numerical Control Machines Based on Parallel Aco." *Eng. Lett.*, vol.20, pp.101–108, 2012.
[14] X. Liu, Y. Hong, N. Zhonghua, et. al. "Process Planning Optimization of Hole-making Operations Using Ant Colony Algorithm." *Int. J. Adv. Manuf. Technol.* Vol.69, nos.1–4, pp753–769, 2013.
[15] M. M. Ismail, M.A. Othman, H. A. Sulaiman, et al. "Firefly Algorithm for Path Optimization in PCB Holes Drilling Process," *Int. Conf. in Green and Ubiquitous Tech.*, Jul. 2012, pp.110-113,
[16] M. Tamjidy, S. Paslar, B.T.H.T. Baharudin, et al. "Biogeography based optimization (BBO) algorithm to minimise non-productive time during hole-making process," *Int. J. Prod. Res.*, vol.53, no.6, pp.1880–1894, 2015
[17] G. Kanagaraj, S.G. Ponnambalam, C.K. Loo, "Charged system search (CSS) algorithm for robotic drill path optimization," in *Proc. Int. Conf. Adv. Mechatronic Sys.*, Aug. 2015, pp. 125-130
[18] A.M. Dalavi, P.J. Pawar, T.P. Singh. "Optimal sequence of hole-making operations using particle swarm optimization and modified shuffled frog leaping algorithm." *Eng. Rev.*, vol.36, no.2, pp.187-196, 2016.
[19] A.M. Dalavi, P.J. Pawar, T.P. Singh. "Tool path planning of hole-making operations in ejector plate of injection mould using modified shuffled frog leaping algorithm." *J. Comput. Des. Eng.*, vol.3, no.3, pp.266-273, 2016.
[20] W. Liu, B. Niu, H. Chen, and Y. Zhu, "Robot path planning using bacterial foraging algorithm," *J. Comput. Theor. Nanos.*, 2013, 10(12): 2890–2896.
[21] C. Liu, J. Wang, Y. T. Leung. Worker assignment and production planning with learning and forgetting in manufacturing cells by hybrid bacteria foraging algorithm. *Comput. Ind. Eng.*, 2016, 96:162-179.
[22]G.Y. Zhu, W.B. Zhang. "Optimal foraging algorithm for global optimization." *Appl. Soft Comput.*, vo. 51, pp.294-313, 2017.
[23]G. H. Pyke, H.R. Pulliam, E.L. Charnov, "Optimal Foraging: A Selective Review of Theory and Tests", *Q. Rev. Biol.*, vol.52, no.2,pp.137-154, 1977.
[24]J. R. Krebs, Alejandro Kacelnik, P. Taylor, "Test of Optimal Sampling by foraging great tits", *Nature*, vol.275, pp.27-31, 1978.
[25] R. J. Cowie, "Optimal foraging in great tits (Parus major)", *Nature*, vol. 268, pp.137-139, 1977.
[26] E. L. Charnov, "Optimal foraging, the Marginal Value Theorem", *Theor. Pop. Biol.*, vol. 9, no.2, 1976.
[27]J.C. Alonso, J.A. Alonso, L.M. Bautista, R. Muńoz-Pulido, "Patch use in cranes: a field test of optimal foraging predictions", *Anim. Behav.*, vol.49, no.5, pp.1367-1379, 1995.
[28] M. P. Hassell, T. R. E. Southwood, "Foraging Strategies of Insects". *Ann. Rev. Ecol. Syst.* , vol. 9, pp.75-78, 1978.
[29]R. Cogni and P. S. Oliveira, "Recruitment Behavior During Foraging in the Neotropical Ant Gnamptogenys moelleri (Formicidae: Ponerinae): Does the Type of Food Matter?", *J. Insect Behav.*, vol.17, no.4, pp.443- 458, 2004.
[30]J.J. Howard, L.M. Henneman, G. Cronin, J.A. Fox and G. Hormiga, "Conditioning of scouts and recruits during foraging by a leaf-cutting ant, Atta colombica", *Anim. Behav.*, vol. 52, no.2, pp.299–306, 1996.
[31]J. R. Krebs, J. T. Erichsen, Michael I. Webber, "Optimal Prey Selection in the great tit(parus major)", *Anim. Behav.*, vol. 25, no.1, pp.30-38, 1977.
[32] G.Y. Zhu, W.B. Zhang. "An improved Shuffled Frog-leaping Algorithm to optimize component pick-and-place sequencing optimization problem." *Expert. Syst. Appl.*, vol.41, no.15, pp.6818-6829, 2014.
[33]V. Roberge, M. Tarbouchi, G. Labonté, "Comparison of parallel genetic algorithm and particle swarm optimization for real-time UAV path planning". *IEEE T. Ind. Inform.*, vol.9, no.1, pp.132-141, 2013.
[34] W.Q. Lin, G.Y. Zhu, "A Genetic Optimization Approach to Optimize the Multi-head Surface Mount Placement Machine", *Lect. Notes Artif. Int.* 5315, Berlin: Springer Press, Part II, 1003–1012. 2008.
[35] S. M. Elsayed, R. A. Sarker, D. L. Essam, "An improved self-adaptive differential evolution algorithm for optimization problems". *IEEE T. Ind. Inform.* vol.9, no.1, pp.89-99, 2013.
[36] J. J. Kim, & J. J. Lee, "Trajectory optimization with particle swarm optimization for manipulator motion planning". *IEEE T. Ind. Inform.*, vol.11, no.3, pp.620-631, 2015.
[37] H. Duan, P. Li, & Y. Yu, "A predator-prey particle swarm optimization approach to multiple UCAV air combat modeled by dynamic game theory." *IEEE/CAA J. of Automatica Sinica*, vol. *2, no.*1, pp.11-18, 2015
[38] W. Dong & M. C. Zhou. "A supervised learning and control method to improve particle swarm optimization algorithms." *IEEE T Syst. Man Cy. A.* vol.47, no.7, pp.1135-1148, 2017
[39] M. Dorigo, M. Birattari, and T. Stutzle. "Ant colony optimization." *IEEE comput. intell. M.* vol.1, no.4, pp.28-39, 2006.
[40] Yarpiz, Ant Colony Optimization (ACO), [Online]. Available: https://cn.mathworks.com/matlabcentral/fileexchange/52859-ant-colony-optimization--aco-?s_tid=srchtitle
[41] W.B. Zhang, G.Y. Zhu. "Comparison and application of four versions of particle swarm optimization algorithms in the sequence optimization". *Expert Syst. Appl*. vol.38, no.7, pp.8858-8864, 2011.
[42] TSPLIB library, http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95 /tsp . (Accesssed 10 October 2017)
[43]M. Taherkhani, R. Safabakhsh. "A novel stability-based adaptive inertia weight for particle swarm optimization." *Appl.Soft Comput.*, vol.38, pp. 281-295, 2016

Wei-Bo Zhang received the Ph.D. degree in mechanical engineering from Jilin University, Changchun, P.R. China, in 2003.

Her present research interests include intelligent design, structure optimization, and automobile parametric design.

From October 2003 to September 2005, she was a Post-doctor at the Tsinghua University, Beijing, P R China. Now, she is an Professor in the School of Mechanical Engineer & Automation at Fuzhou University.

Guang-Yu Zhu received the Ph.D. degree in mechanical engineering from the Beijing Institute of Technology, Beijing, P.R. China in 2005.

His present research interests include optimization design, swarm intelligence, advanced manufacturing technology.

He is a Professor in the School of Mechanical Engineer & Automation at Fuzhou University. He was the runner-up in the National Science Progress Award of P.R. China in 1999.