Improved cuckoo search algorithm and its application to permutation flow shop scheduling problem

Lieping Zhang, Yanlin Yu, Yingxiong Luo and Shenglan Zhang

Abstract

Aiming at the problem that the standard cuckoo search algorithm relies on Levy flights, which leads to the step-size randomness of the search process, a self-adaptive step cuckoo search algorithm based on dynamic balance factor is proposed in our paper. First, two parameters are introduced in our paper, which were iteration number ratio parameter and adaptability ratio parameter. Then, a dynamic balance factor parameter is introduced to adjust the weight number of iteration number ratio parameter and adaptability ratio parameter. Finally, parameter skewness value calculation method and self-adaptive step strategy were proposed combined with the dynamic balance factor. Six typical test functions are used to test the performance of the proposed algorithm, the standard cuckoo search algorithm and the self-adaptive step cuckoo search algorithm which relies only on the iterative times. The test results show that the proposed algorithm had good convergence speed and accuracy. Meanwhile, taking the permutation flow shop scheduling problem as an example, eight operators of Car benchmark class are used as the test data to compare the performance of three algorithms, the effectiveness and superiority of the algorithm in solving the permutation flow shop scheduling problem are verified.

Keywords

Lévy flight, cuckoo search algorithm, dynamic balance factor, self-adaptive step, permutation flow shop scheduling problem, car benchmark instance class

Received 5 September 2019; revised 11 March 2020; accepted 23 June 2020

Introduction

The permutation flow shop scheduling problem (PFSP) is a production scheduling problem that is based on constraint of the flow shop scheduling problem. Meanwhile, it further increases the constraint that all workpieces have the same processing order on any machine. The scheduling optimization can effectively improve the production efficiency of enterprises. Although the process constraint of PFPS is relatively simple, it has been proved in the field of flexible manufacturing industry that PFSP with more than three machines participating in the production process is a typical NP-complete problem, and there is no global optimization algorithm with polynomial computation complexity. The methods of solving PFSP mainly include precise algorithm, heuristic method and intelligent algorithm. The precise algorithm has large computational complexity and is only suitable for solving small-scale problems. The heuristic method can construct the solution in a short time according to the characteristics of the problem, but it is difficult to guarantee the quality of the solution. With the development of computational intelligence, intelligent algorithms for solving PFSP have been widely studied, such as genetic algorithm, particle swarm optimization, differential evolution, cuckoo search, and various hybrid algorithms.

The cuckoo search (CS) algorithm is a new heuristic algorithm proposed by professor Xin-She Yang and S.
Deb of Cambridge University in 2009. It simulates and solves complex optimization problems by simulating the brooding behavior of cuckoos and the Lévy flight patterns of animals such as albatrosses and fruit flies. Due to its simple model, few parameters and high search efficiency, the CS algorithm has been widely applied to various development fields such as artificial intelligence, industrial scheduling, commercial optimization, neural networks and other fields. In general, the performance of the algorithm is not only affected by evolutionary operators. The selection of parameters also affects the quality and speed of the result. In the standard CS algorithm, the Lévy flight is used to generate random step-size, but the step-size is varying. If the step-size is set too small, the algorithm will be biased towards local search, which will accelerate the convergence of the algorithm, but it is easy to produce the premature phenomenon, making the algorithm fall into the local optimal solution. If the step-size is set too large, the algorithm will be biased towards the global search. This will help the algorithm to jump out of the local optimal solution, but it will affect the convergence speed of the algorithm. Therefore, its value needs to be reasonably set according to the feasible region of the problem. In order to improve the reasonableness of the step-size setting of the CS algorithm, many scholars at home and abroad have carried out relevant researches to improve the step-size setting of the standard CS algorithm. For example, the modified cuckoo search algorithm proposed by H. Jiang et al. adapted the step-size control factor by the current fitness and the initial worst fitness, which enhances the local search ability of the algorithm. R.Y. Li et al. introduced the dynamic inertia weights and memory strategies in the preferred random walks by adaptively changing the search step-size in the process of optimization and realized the overall dynamic adjustment of the algorithm. W.Y. Qian et al. proposed an adaptive parameter control strategy to dynamically adjust the step-size factor in the CS algorithm to enhance the search performance of the algorithm. E. Valian et al. proposed an improved cuckoo search (ICS) algorithm by dynamically adjusting the discovery probability and step-size factor in the standard CS algorithm, which effectively improves the accuracy of the algorithm. E. Valian et al. proposed a self-adaptive step cuckoo search algorithm, which improves the search accuracy of the algorithm. L.J. Wang et al. added a compression factor that obeys [0,1] uniform distribution in the updated equation of Lévy flight, which is better than the standard CS algorithm. Z. Jian et al. proposed a self-adaptive step-size and some neighbor-study strategies to enhance search performance, and an improved lambda iteration strategy was used to generate new solutions. J.T. Cheng et al. introduced the memory mechanism into the CS algorithm to dynamically select the appropriate step-size, which differs from many CS variants by incorporating some existing algorithms into CS framework. A reinforced cuckoo search algorithm for multimodal optimization was proposed by T. Kalaipriyan et al., in which a self-adaptive step-size has been introduced to achieve multimodality in the proposed algorithm. Yung, C.S. presented an improved cuckoo search algorithm, considering the effects of coevolution on the displacement dynamics of animal communities, and applied it to the development of maximally distributed physical arrangements. From the several improved algorithms in the above papers, we can see that in the process of searching for the optimal solution, the step-size factor of them gradually decreases with the algorithm iteration process to enhance the local search ability of the algorithm so as to accelerate the convergence speed in the later period. However, these strategies that solely rely on the number of iterations or the fitness or the bird’s nest distance ratio to improve the step-size factor do not balance the convergence speed and the search accuracy during the operation of the algorithm. In some cases, with the increase of the number of iterations, the algorithm falls into some problems such as local optimization, low accuracy, precocity and so on. Moreover, when dealing with complex multi-peak functions, it will lead to large time overhead and low efficiency, and may even make the algorithm converge less accurately than the standard CS algorithm.

Based on the theoretical significance and application value of solving PFSP and the deficiency of the standard CS algorithm, a self-adaptive step length cuckoo search algorithm based on dynamic balance factor is proposed in this paper, which is named as dynamic cuckoo search (DCS) algorithm. Two parameters, iteration number ratio parameter and adaptability ratio parameter, are introduced in the proposed method. A dynamic balance factor $\theta$ is introduced and used to adjust the weight number of iteration number ratio and adaptability ratio. At the same time, the proposed algorithm realizes the dynamic self-adaptive step of the algorithm through the dynamic parameter skewness value $\beta$ in the standard cuckoo search algorithm. The self-adaptive step allows the CS algorithm to be closer to natural evolution and more flexible. The validity of the proposed algorithm is verified by six typical test functions, which are Ackley, Griewank, Rastrigrin, Rosenbrock, Schwefel, and Sphere function. In the second part of the paper, the proposed DCS algorithm is applied to solve the optimization design problem of PFSP, and the feasibility and superiority of the proposed algorithm for solving PFSP are verified by eight operators Car1-Car8 of Car benchmark class.
Standard cuckoo search algorithm

Inspired by cuckoo breeding behavior, the standard CS algorithm is a global intelligent optimization algorithm which imitates cuckoo’s searching nest and spawning process in nature by introducing the Lévy flight mechanism of birds. The standard CS algorithm is inspired by the behavior of cuckoo brooding. It introduces the Lévy flight mechanism of birds to simulate the process of cuckoo search for nests and spawning in nature. It belongs to the global intelligent optimization algorithm. Professor Yang and Deb proposed the following three ideal states set for the algorithm:

1. The cuckoo lays only one egg during each generation and randomly selects a bird nest to hatch.
2. Before finding a better bird nest, take the current best bird nest as the generation’s host for the next reproduction.
3. The total number of bird nests available for each generation is constant, and the probability that a cuckoo egg is found by the host is \( P_a \). If the egg is perceived by the host, the host will either abandon the egg or abandon the entire nest and find a suitable location to build a new bird nest.

Where \( P_a \) represents the probability that the host egg is not its own child, most researchers set \( P_a \) as 0.25.\(^{22}\)

The standard CS algorithm realizes the simulation process of cuckoo search for the nest by producing candidate population, selecting optimal choice and randomly migrating. On the basis of the above three ideal states, the cuckoo searches route and position according to equation (1) and then produce candidate populations:

\[
X_{i+1}^t = X_i^t + \alpha \oplus L(\beta)
\]  

Among them: \( X_{i+1}^t \) and \( X_i^t \) respectively refer to the position vectors of the \( i^{th} \) generation (i = 1, 2, ..., \( n \) is the number of bird nests) bird nest location in the \( t + 1 \) and \( t \) generations (\( X_i^t = X_{i,1}^t, X_{i,2}^t, \ldots, X_{i,d}^t \); \( d \) is the dimension of each bird nest; \( j \) represents any \( j^{th} \) dimension, \( j = 1, 2, \ldots, d \); \( t \) is the algorithm algebra, \( t = 1, 2, \ldots, t_c \); \( t_c \) is the algebra when the algorithm converges); \( \alpha \) is the step-size factor, it is used to control the range of random search. It is a constant greater than zero. It takes different values for different situations; \( \oplus \) refers to point-to-point multiplication; \( L(\beta) \) is random optimization route for Lévy flight; \( \alpha \oplus L(\beta) \) is the step-size of Lévy flight, it represents the distance that cuckoo need to search from the \( i^{th} \) generation (feasible solutions) to the \( i + 1^{th} \) generation of bird nests (feasible solutions) in a random distribution manner of Lévy flight.

The relationship between Lévy flight random optimization route and the time \( t \) obeys the Lévy probability distribution shown in equation (2), among them: \( \beta \) is the exponential coefficient, \( \mu \) is a random number of the normal distribution. From equation (2), we can see that the optimization path of the CS algorithm consists of frequent short jumps and occasional long jumps. This optimization method can make the algorithm have a larger search space, and it is easier to jump out of the local optimum.

\[
\text{Levy}(\beta) \sim \mu = t^\beta, (1 < \beta \leq 3)
\]

To implement the CS algorithm, the method in reference\(^9\) was used to simulate a flight jump path, which is the step-size, as shown in equation (3):

\[
\text{Levy}(\beta) = \mu/|\nu|^\frac{\beta}{\gamma}
\]

Among them: \( \mu, \nu \) are random numbers of the normal distribution, \( \beta \) is the parameter skewness, and the general value is \( \beta = 1.5.\)

In the actual optimization problem, the location of the bird nest \( X_i^t \) represents the effective value space of all variables (\( d \) dimension), and the fitness values of the nest represent the objective function value corresponding to the variable taking different values. In the evolution process, after updating the location of the bird nest by equation (1), use the random number \( r \in [0, 1] \) to compare with \( P_a \). If \( r > P_a \), then randomly change \( X_{i+1}^t \), otherwise, do not change. A group of bird nest locations with better test values are finally reserved, still denoted as \( X_{i+1}^t \).

As far as the step-size control of the CS algorithm is concerned, the Lévy flight random walk strategy is adopted by the CS algorithm to generate a new step-size, but this randomness has no regularity and lacks self-adaptation to the algorithm process. In the standard CS algorithm, the step-size factor \( \alpha \) is fixed during the movement of each individual, generally taking \( \alpha = 1.24 \). However, the step-size in the CS algorithm directly affects the speed and accuracy of the optimization. The smaller the step-size is, the easier it is to search locally, but at the same time, the slower the convergence speed is, the easier it is to fall into a local optimum. The larger the step-size is, the algorithm will have certain global search ability, and the faster the convergence speed is, but it is also easy to skip the optimal value. The phenomenon that the result oscillates back and forth near the optimal solution may occur, and even the phenomenon of population loss may occur. The randomness of the Lévy flight does
not guarantee that the search process converges quickly and steadily towards the optimal solution. The basic solution is to integrate the advantages of global fast search and local careful search to achieve a balance between efficient global search and accurate local search and improve algorithm performance.

**Self-adaptive step cuckoo search algorithm based on dynamic balance factor**

**Algorithm improvement ideas**

Through the analysis of the standard CS algorithm and related research results, we can know that the step-size factor $\alpha$ decreases linearly with the change of the number of iterations, which can accelerate the convergence of the algorithm and enhance the local exploration performance. The step-size factor decreases with the change of fitness, which can improve the convergence accuracy of the algorithm and enhance the quality of the solution. However, in the standard CS algorithm, although the step-size continuously and randomly changes, it does not correlate with the progress of the algorithm, resulting in slow convergence speed in the later period of the algorithm. At the same time, from the step-size calculation equation (3), it can be seen that in the standard CS algorithm, the value of the parameter skewness $b$ plays a decisive role in the generation of the step-size.

Based on the above-mentioned two points, by introducing the number of iterations ratio and the fitness ratio, this paper realizes the dynamic adaptation to the parameter skewness $b$ so as to realize the dynamic adaptation to the step-size of the algorithm. At the same time, by adding a dynamic balance factor $\vartheta$ to balance the weights of the number of iterations ratio and the fitness ratio in the algorithm, the purpose of balancing the convergence speed and convergence accuracy was achieved. The specific method is shown in equation (4):

$$b_t = b_{\min} + \vartheta(1 - \frac{t}{m}) + (1 - \vartheta)\frac{f_{\min}}{f_{\max}}, \beta \in [1, 2]$$

Among them, $b_{\min}$ is the minimum parameter skewness value; $\vartheta$ is the introduced dynamic balance factor, and the range value is $\vartheta \in [0, 1]$; $t$ refers to the current number of iterations; $m$ refers to the maximum number of iterations; $f_{\min}$ refers to the current optimal fitness, and $f_{\max}$ is the initial fitness. The proportion of $t/m$ and $f_{\min}/f_{\max}$ in the calculation is adjusted by adjusting the value of dynamic balance factor $\vartheta$. In the algorithm, as the number of iterations $t$ continues to increase, the step-size will continue to decrease. As the algorithm approaches the optimal solution, the decrease of fitness $f_{\min}$ will result in a smaller step-size. The introduction of $t/m$ will reduce the step-size in the process of the algorithm, so as to accelerate the convergence of the algorithm. The introduction of $f_{\min}/f_{\max}$ solves the problem that the algorithm converges too quickly and falls into premature, and it also avoids the problem of low accuracy caused by fast convergence of the algorithm. Through the adjustment of these two parameters by dynamic balance factor $\vartheta$, it avoids the problem that algorithm blindly and quickly converges with the increase of the number of iterations, and it also solves the problem that the algorithm is premature and falls into a local optimum. At the same time, it also improves the convergence accuracy of the algorithm.

**Improved algorithm implementation process**

According to the proposed improvement idea, the algorithm implementation process can be described as shown in Figure 1.

**Improved algorithm simulation and result analysis**

**Simulation environment and parameters setting.** The simulation test environment of this paper is as follows: Windows 10 operating system, Inter Core CPU T6400 2.10 GHz, 2.00 GB memory, MATLAB 2014b simulation software. Use the six common test functions shown in Table 1 for testing. The tested algorithm is the standard CS algorithm, the ICS algorithm proposed in reference14 and the DCS algorithm proposed in this
### Table 1. Six common test functions.

| Function name | Expression |
|---------------|------------|
| Ackley        | \(f_i(x) = -c_1 \exp \left( -0.2 \sqrt{\sum_{j=1}^{n} x_j^2} \right) - \exp \left[ -\frac{\sum_{j=1}^{n} \cos(2\pi x_j)}{4000} \right] + 20 + e\) |
| Griewank      | \(f_i(x) = \frac{1}{4000} \sum_{j=0}^{n-1} x_j^2 - \sum_{j=0}^{n-1} \cos \left( \frac{x_j}{\sqrt{1 + i}} \right) + 1\) |
| Rastrigrin    | \(f_i(x) = \sum_{i=1}^{n} \left[ x_i^2 - 10\cos(2\pi x_i) + 10 \right]\) |
| Rosenbrock    | \(f_i(x) = \sum_{i=1}^{n} \left[ 100(x_i - x_{i+1}^2)^2 + (x_i - 1)^2 \right]\) |
| Schwefel      | \(f_i(x) = \sum_{i=1}^{n} x_i^2\) |
| Sphere        | \(f_i(x) = \sum_{i=1}^{n} x_i^2\) |

The validity of the proposed algorithm is verified by comparing the test data. The initial ranges of the six test functions in Table 1 are as follows:

1. The value range of Ackley is \(-8 \leq x \leq 8\), and the theoretical optimal value is 0. This function has more local minimums, and it is difficult for the algorithm to discern the optimal direction of the search. It is difficult to find the global optimal solution.

2. The value range of Griewank is \(-600 \leq x \leq 600\), and the theoretical optimal value is 0. This function has many local minimums. It is a classical nonlinear multimode function and the search area is relatively large.

3. The value range of Rastrigrin is \(-100 \leq x \leq 100\), and the theoretical optimal value is 0. This function is a multi-peak function with different peak shapes, making it difficult to find a global optimal solution.

4. The value range of Rosenbrock is \(-10 \leq x \leq 10\), and the theoretical optimum is 0. The function is unimodal and the global optimal solution appears in a gradually descending and long parabolic-shaped ravine. Because the information provided by the function to the optimization algorithm is very limited, it is difficult for the algorithm to discern the search direction, and finding the optimal solution also becomes difficult.

5. The value range of Schwefel is \(-100 \leq x \leq 100\), and the theoretical optimum is 0. It is a unimodal function.

6. The value range of Sphere is \(-5.12 \leq x \leq 5.12\), and the theoretical optimum is 0. It is a unimodal function.

#### Experiment simulation and result analysis.

In the algorithm simulation experiment, the initial discovery probability of the standard CS algorithm and the ICS algorithm is set to \(P_a = 0.25\), the step-size is selected to \(a = 1\), and the dynamic balance factor of the DCS algorithm is \(\vartheta = 0.8\). The minimum, maximum, average, variance and the number of iterations of the optimization results are used as the performance indicators to evaluate the algorithms, and the optimization performances of the CS, ICS and DCS algorithms are compared. The simulation results are shown in Tables 2 and 3, where \(N\) in the table represents the dimension, and the maximum number of iterations of the algorithm is \(T_{\text{max}} = 1000\). The performance index in the table is the average value of the algorithm running 30 times alone. When \(N = 50\), the three algorithms cannot converge successfully at \(T_{\text{max}} = 1000\), so the search performance is further compared when the maximum number of iterations is \(T_{\text{max}} = 2000\) and \(T_{\text{max}} = 3000\), as shown in Tables 4 and 5.

According to the related optimization test data in Tables 2 to 5, it can be seen that:

1. When the dimension \(N = 10\), the ICS and DCS algorithms have a significant improvement over the CS algorithm in both the number of iterations and the convergence accuracy. In terms of the number of iterations, most of the time the ICS algorithm is less than the DCS algorithm, but the overall solution quality of the ICS algorithm is lower than that of the DCS algorithm. It shows that the DCS algorithm can avoid the phenomenon of back and forth fluctuation near the optimal solution at the later period, and it is better than CS algorithm and ICS algorithm in convergence accuracy.
2. When the dimension $N = 30$, the DCS and ICS algorithms are obviously better than the CS algorithm, which greatly improves the performance of the algorithm. From the number of iterations $T_{DCS} < T_{ICS} < T_{CS}$, it can be seen that the number of iterations of the DCS algorithm is the least, that is, it is superior to the ICS algorithm in the convergence speed and has a significant improvement over the performance of the CS algorithm.

3. When the dimension $N = 50$ and $T_{max} = 2000$, except the ICS algorithm can successfully converge sometimes, the CS and DCS algorithms cannot achieve the established convergence accuracy within a specified maximum number of iterations. However, the final minimum, maximum, and average values of the DCS algorithm are superior to both the CS and ICS algorithms. This is because the ICS algorithm adjusts and controls step-size only based on the number of iterations, which makes the algorithm easy to premature in the search optimization process.

4. When the dimension $N = 50$ and $T_{max} = 3000$, due to the increase in the number of iterations, all three algorithms converge successfully, and the search

### Table 2. Test data of six test functions based on CS, ICS and DCS algorithms (where $N = 10$).

| Test function | Algorithm | Minimum  | Maximum  | Average  | Variance  | Number of iterations |
|---------------|-----------|----------|----------|----------|-----------|----------------------|
| Ackley        | CS        | 3.1724e-08 | 1.1169e-05 | 1.1978e-06 | 2.4007e-06 | 653                  |
|               | ICS       | 9.1651e-10 | 5.4678e-06 | 1.5947e-07 | 6.5468e-07 | 281                  |
|               | DCS       | 2.3161e-11 | 3.1495e-08 | 2.3762e-09 | 5.1563e-09 | 212                  |
| Griewank      | CS        | 1.2824e-02 | 1.0010e-01 | 3.4795e-02 | 1.6791e-02 | 324                  |
|               | ICS       | 7.1589e-04 | 1.0724e-01 | 1.8279e-03 | 3.1791e-03 | 276                  |
|               | DCS       | 2.4935e-00 | 1.1197e+01 | 5.8173e+00 | 2.2131e+00 | 682                  |
| Rastrigrin    | CS        | 1.4728e-00 | 5.0812e-00 | 3.1597e-00 | 9.1871e-01 | 306                  |
|               | ICS       | 0.1429e+00 | 3.0793e+00 | 2.5239e+00 | 3.4742e+00 | 287                  |
|               | DCS       | 7.5742e-01 | 7.4146e-01 | 5.1543e-01 | 6.2970e-01 | 725                  |
| Rosenbrock    | CS        | 2.3671e-00 | 1.1031e+00 | 3.1597e+00 | 9.1871e-01 | 306                  |
|               | ICS       | 8.5837e-03 | 7.1482e-00 | 3.2597e+00 | 7.7491e-01 | 289                  |
|               | DCS       | 2.7542e-01 | 7.4146e-01 | 5.1543e+00 | 6.2970e+00 | 725                  |
| Schwefel      | CS        | 3.9764e-10 | 2.7147e+00 | 9.1871e-01 | 3.2777e+00 | 397                  |
|               | ICS       | 7.3825e-11 | 3.7319e-05 | 1.7932e-06 | 8.2936e-06 | 242                  |
|               | DCS       | 8.8532e-04 | 2.4962e-02 | 2.7147e+00 | 6.2970e+00 | 287                  |
| Sphere        | CS        | 4.9826e-18 | 1.5338e-01 | 5.1543e-01 | 8.4421e-01 | 1587                 |
|               | ICS       | 2.3482e-20 | 5.9057e-17 | 3.7763e-01 | 3.2777e+00 | 368                  |
|               | DCS       | 5.8163e-23 | 1.7691e-18 | 8.2784e+00 | 3.1791e-03 | 273                  |

### Table 3. Test data of six test functions based on CS, ICS and DCS algorithms (where $N = 30$).

| Test function | Algorithm | Minimum  | Maximum  | Average  | Variance  | Number of iterations |
|---------------|-----------|----------|----------|----------|-----------|----------------------|
| Ackley        | CS        | 5.7283e-07 | 4.3404e+00 | 2.0926e+00 | 8.4421e-01 | 1587                 |
|               | ICS       | 7.0179e-09 | 2.0393e+00 | 3.0880e-01 | 5.9632e-01 | 963                  |
|               | DCS       | 3.5793e-10 | 3.7319e-05 | 1.7932e-06 | 8.2936e-06 | 242                  |
| Griewank      | CS        | 1.6098e-14 | 7.5394e-02 | 1.0846e-02 | 2.0119e-02 | 1621                 |
|               | ICS       | 4.5697e-16 | 4.9058e-05 | 3.1173e-05 | 1.1340e-06 | 854                  |
|               | DCS       | 7.3825e-11 | 7.1482e-00 | 3.2597e+00 | 7.7491e-01 | 289                  |
| Rastrigrin    | CS        | 2.0423e+01 | 5.9923e+01 | 3.6513e+01 | 9.2702e+00 | 1379                 |
|               | ICS       | 1.1939e+01 | 2.9747e+01 | 2.2296e+01 | 4.1242e+00 | 785                  |
|               | DCS       | 8.4867e+00 | 2.3715e+01 | 9.6713e+00 | 3.5572e+00 | 697                  |
| Rosenbrock    | CS        | 9.8653e+00 | 1.5338e-15 | 2.5086e-16 | 3.5572e-16 | 696                  |
|               | ICS       | 8.5837e-03 | 7.1482e-00 | 3.2597e+00 | 7.7491e-01 | 289                  |
|               | DCS       | 2.3482e-20 | 5.9057e-17 | 4.2599e-01 | 2.6241e-01 | 368                  |
| Schwefel      | CS        | 4.9826e-18 | 1.5338e-01 | 5.1543e-01 | 8.4421e-01 | 1587                 |
|               | ICS       | 2.3482e-20 | 5.9057e-17 | 4.2599e-01 | 2.6241e-01 | 368                  |
|               | DCS       | 5.8163e-23 | 1.7691e-18 | 8.2784e+00 | 3.1791e-03 | 273                  |
| Sphere        | CS        | 1.2421e-15 | 7.3248e-13 | 6.6274e-14 | 1.3432e-13 | 1875                 |
|               | ICS       | 2.9807e-18 | 4.1015e-14 | 7.5438e-16 | 1.1279e-16 | 1107                 |
|               | DCS       | 1.9759e-19 | 5.4671e-16 | 9.4827e-17 | 5.2478e-17 | 961                  |
accuracy of CS algorithm and ICS algorithm has improved. In addition, it can be found that the proposed DCS algorithm has less number of iterations than the CS algorithm and the ICS algorithm, and fully embodies the advantage of the DCS algorithm adjusted by the dynamic balance factor \( \varphi \) in optimizing the performance. It shows that the DCS algorithm can effectively avoid the problem that stochastic fluctuations and slow convergence rate caused by random step-size factor of the CS algorithm, and at the same time it can avoid prematurity caused by the ICS algorithm’s pursuit of convergence speed.

**Application of improved algorithm in permutation flow shop scheduling problem**

**Scheduling problem description**

The basic flow of PFSP can be described as follows:\(^{25}\): \( n \) components are processed on \( m \) machines, each
component requires \( k \) processing steps, and the time that each component \( i \) spends on each device \( j \) when it completes the processing is fixed, assuming the time is \( t_{jk}(i = 1, 2, \ldots, m) \). The problem to be solved by the scheduling is to give an optimal processing sequence for the processing flow of the components, so that the entire processing flow takes the shortest time. In the PFSP, there is not only one component that needs to be processed, but also many components, and the processing sequence of each component on any equipment is unique. All components should be processed in the order of equipment, and the processing time spent on different equipment is also different. From this, it can be seen that taking different component processing sequences on the equipment for the same production process will lead to different results, and the differences between these results are also very large.

The general PFSP assumes the following specific conditions: 1) \( n \) components are processed on \( m \) machines by flow line production. 2) All components are in the same sequence during processing. 3) For each equipment, the order of the processed components is also the same. 4) The readiness time and processing time for all components to be consumed on all equipment are known in advance. 5) All equipment can only process one component individually during processing. The purpose of the PFSP solution is to obtain the optimal scheduling plan for a certain production index under the above five specific conditions, that is, to minimize the preparation time and processing time consumed by the entire processing flow.

If the measurement standard of the scheduling is the maximum completion time, this PFSP problem can be described by the three-element method, that is \( F_m|prmu|C_{\text{max}} \). Among them, \( F_m \) refers to the flow shop scheduling model with \( m \) machines, \( prmu \) means that the processing sequence of each component on any equipment is the same, \( C_{\text{max}} \) refers to the maximum completion time of the entire production process. Suppose \( t_{jk,i} \) is the processing time that component \( j_k \) spends in equipment \( i \) (the readiness time of each component is included in the processing time), \( C_{jk,i} \) is the completion time that component \( j_k \) spends on equipment \( i \), \( \pi \in \{\Omega|j_1, j_1, \ldots, j_m\} \) is the component processing sequence, \( \Omega \) represents the total set of ordered sequences for all elements. For the component processing process, the following constraints are set: the process of component processing cannot be interrupted or abandoned; each equipment is unique and each component is unique. All components should be processed in the order of equipment \( 1 \sim m \). The completion time of each component on each equipment can be calculated using equations (5) to (9).

That is:

\[
C_{jk,i} = \sum_{r=1}^{i} t_{jk,i}, \quad i = 1, 2, \ldots, m \quad (5)
\]

\[
C_{jk,1} = \sum_{k=1}^{k} t_{jk,1}, \quad k = 1, 2, \ldots, n \quad (6)
\]

\[
C_{jk,i} = \max \{C(j_k,i-1), C(j_{k-1},i)\} + t_{jk,i} \quad (k = 2, 3, \ldots, n; i = 2, 3, \ldots, m) \quad (7)
\]

\[
C_{\text{max}}(\pi) = C(j_n,m) \quad (8)
\]

\[
\pi^* = \arg\{C_{\text{max}}(\pi) = C(j_n,m)\} \rightarrow \min, \quad \forall \pi \in \Omega \quad (9)
\]

Among them, equations (5) to (7) are recursive equations for calculating the completion time, equation (8) is the maximum completion time for the batch of workpieces, and equation (9) is the optimal scheduling scheme for minimizing the maximum makespan, \( n \) and \( m \) represent the number of components and the number of equipment respectively. The scheduling process is shown in Figure 2.

**Algorithm coding**

The traditional CS algorithm is generally applied to the continuous optimization problem. Due to the discrete characteristics of PFSP, traditional CS algorithm cannot be used directly for the solution of PFSP. In order to solve the PFSP using the proposed CS algorithm to obtain the optimal scheduling allocation scheme for the production process. It is necessary to construct a reasonable coding method to represent the scheduling problem, that is, the sequential composition of shop scheduling. According to the characteristics of PFSP, this paper selects the random key coding method with the ascending sorting principle. The continuous position vector \( X_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,n}] \) of

![Figure 2. Scheduling Gantt chart of permutation flow shop](image)
the cuckoo in the process of finding the bird nest corresponds to the processing procedure \( \pi = (j_1, j_2, \ldots, j_n) \) of the components on all the equipment in the production process. A certain dimension vector at each bird nest position represents a certain procedure in the processing of components. The position of the bird nest formed by the \( n \)-dimensional vector represents the processing sequence of a group of components in all equipment, so that a bird nest position directly represents a set of scheduling methods. According to this coding method, the corresponding scheduling solution can be calculated by calculating the fitness of the bird nest. At the same time, this coding method based on the multi-dimensional vector to represent the process is compatible with the CS algorithm structure. Therefore, the adoption of this coding method does not require modification of the evolution operator of the CS algorithm.

Scheduling algorithm implementation process

According to the previously established PFSP mathematical model and the proposed coding method, the PFSP implementation process based on the DCS algorithm is as follows:

Step 1: Set parameters and population initialization. Set the population size \( \text{popnum} \), step-size factor \( \alpha \), discovery probability \( P_a \), maximum number of iterations \( N_{\text{max}} \), dynamic balance factor \( \vartheta \), and then initialize the population based on the properties of the problem.

Step 2: The cuckoo randomly selects the bird nest location \( X_i (i = 1, 2, \ldots, m) \) (where \( m \) is the number of nests), then converts into the component processing sequence according to the given coding method, calculates the fitness of each nest (scheduling scheme), and finds the current best nest location \( X^* \).

Step 3: According to the current number of iterations and fitness, use equation (4) to calculate \( \beta \) and equation (3) to calculate the step-size.

Step 4: Generate candidate populations \( X_i \) according to step-size. Progenies are generated according to equation (1). If the progenies are beyond the constraint of the problem, appropriate methods must be used to deal with the infeasible solution.

Step 5: Perform a preferred selection operator. For each individual, compare the fitness of the candidate solution with the current solution and save the high fitness solution to the next iteration.

Step 6: Perform a random migration operator. From the updated candidate population in step 4, select individuals with poor fitness to be mutated according to the probability \( P_a \).

Step 7: Perform the preferred selection operator again. For individual who performed a mutation, compare the fitness before and after mutation. If the fitness after the mutation is higher, the mutated individual is included in the population, otherwise the individual before the mutation is retained, and skip to step 3.

Step 8: Skip to step 9 when the maximum number of searches is reached, otherwise skip to step 3 for the next round of searches.

Step 9: Outputs the current best solution and the scheduling scheme it represents.

Scheduling simulation experiment and result analysis

The PFSP has a wealth of benchmark data, this article uses eight operators of the Car benchmark instance class: \( \text{Car}_1, \text{Car}_2, \ldots, \text{Car}_8 \) as experimental data for testing. The evaluation indicators commonly used in Car benchmark instance class include BRE (best relative error), ARE (average relative error), WRE (worst relative error), SR (success rate) and RE (relative error), which are expressed as the following equations (10) to (13):

\[
\text{BRE} = \left[ \min (C_{\text{max}} - C^*) \right] / C^* \times 100\% \quad (10)
\]

\[
\text{ARE} = \left[ \text{avg}(C_{\text{max}} - C^*) \right] / C^* \times 100\% \quad (11)
\]

\[
\text{WRE} = \left[ \max (C_{\text{max}} - C^*) \right] / C^* \times 100\% \quad (12)
\]

\[
\text{RE} = \left( C_{\text{max}} - C^* \right) / C^* \times 100\% \quad (13)
\]

Among them, \( C_{\text{max}} \) refers to the minimum maximum completion time of the scheme found by the algorithm, and \( C^* \) refers to the lower bound value of the problem.

The parameters of the three comparison algorithms are set as follows: The initial discovery probability of the standard CS algorithm and the ICS algorithm is \( P_a = 0.25 \), that is, the probability that the bird’s nest is abandoned is 0.75, and the step-size factor is \( \alpha = 1 \). The initial discovery probability of the DCS is \( P_a = 0.25 \) and the dynamic balance factor is \( \vartheta = 0.8 \). The population number of each algorithm is set as \( \text{popnum} = 15 \), the maximum number of iterations is \( T_{\text{max}} = 1000 \), \( P \) is the type in the Car class, \( C^* \) is the minimum completion time of the corresponding problem, \( n \) is the number of elements involved in the process, \( m \) is the number of equipment, and each algorithm is repeated 30 times for optimization. The experimental simulation test results are shown in Table 6.

According to the experimental results in Table 6, in the Car benchmark instance class of the three algorithms, the DCS algorithm can find the best solution.
every time and the search process is faster than the CS and ICS algorithms, reflecting that the DCS algorithm has good global search performance. According to the two data of ARE and BRE, the quality of the scheduling solution obtained by the DCS algorithm is also higher than the other two algorithms. At the same time, the initial ARE and BRE values of the DCS algorithm are smaller than the CS and ICS algorithms, indicating that the initial robustness of the DCS algorithm is also stronger than the other two algorithms.

Conclusion

In the traditional CS algorithm, the step-size factor of individual movement is usually fixed, which leads to cannot be satisfied with the need of convergence speed and solution precision. In order to solve this problem, a weight-based dynamic adaptive step-size CS algorithm is proposed and applied to PFSP. The advantages of the proposed algorithm are verified by simulation experiment. The following conclusions can be drawn:

1. The existing improved CS algorithm based on step-size factor usually adopts iteration number ratio or fitness ratio to adjust the step factor separately. In this paper, we introduce the ratio of number ratio and the fitness ratio to achieve the adaptive adjustment of step-size factor, and adjust the weight between them to achieve convergence speed and convergence accuracy balance. The simulation results show that the proposed DCS algorithm has faster convergence speed and higher convergence precision, and has some theoretical value.

2. The proposed DCS algorithm is applied to solve the PFSP by coding by random key encoding of ascending order and using the example class of Car benchmark as the experimental data. The experimental results show that the proposed DCS algorithm has the advantages of higher solution quality, stronger robustness and less precocity compared with the traditional CS algorithm and ICS algorithm. It can be used to solve the PFSP, and it has certain engineering value.

3. In the algorithm simulation tests, the simulation parameters of the three CS algorithms, such as the initial discovery probability, the dynamic balance factor and the parameter skewness value, are selected to test with empirical values. How to optimize these parameters by experiment method or intelligent algorithm optimization method to further
improve the performance and expand the application range of the DCS algorithm is our future research.

Authors' contributions
All authors have contributed equally to this work in terms of data sampling, writing, and reading. All authors read and approved the final manuscript.

Declaration of Conflicting Interests
The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding
The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This research was funded by the National Natural Science Foundation of China (No. 61741303), the Guangxi key laboratory of spatial information and geomatics (Guilin University of Technology) (No.15-140-07-23, No.16-380-25-23).

References
1. Pan QK, Tasgetire MF, Suganthan PN, et al. A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. Inf Sci 2011; 181: 2455–2468.
2. Yenisey MM and Yagmahan B. Multi-objective permutation flow shop scheduling problem: literature review, classification and current trends. Omega 2014; 45: 119–135.
3. Naderi B and Ruiz R. A scatter search algorithm for the distributed permutation flowshop scheduling problem. Eur J Oper Res 2014; 239: 323–334.
4. Rossi FL, Nagano MS and Sagawa JK. An effective constructive heuristic for permutation flow shop scheduling problem with total flow time criterion. Int J Adv Manuf Technol 2017; 90: 93–107.
5. Bewoor LA, Prakash VC and Sapkal SU. Production scheduling optimization in foundry using hybrid particle swarm optimization algorithm. Procedia Manuf 2018; 22: 57–64.
6. Santucci V, Baioletti M and Milani A. Algebraic differential evolution algorithm for the permutation flowshop scheduling problem with total flowtime criterion. IEEE Trans Evol Computat 2016; 20: 682–694.
7. Xu YL and Ye CM. Cuckoo search algorithm for solving permutation flow-shop scheduling problems. Math Theory Appl 2015; 35: 89–96.
8. Zhang Y, Yu YL, Zhang SL, et al. Ant colony optimization for cuckoo search algorithm for permutation flow shop scheduling problem. Syst Sci Control Eng 2019; 7: 20–27.
9. Yang XS and Deb S. Cuckoo search via Lévy flights. In: Proceedings of world congress on nature & biologically inspired Computing, 2009, pp.210–214. Coimbatore: IEEE.
10. Shehab M, Khader AT and Al-Betar MA. A survey on applications and variants of the cuckoo search algorithm. Appl Soft Comput 2017; 61: 1041–1059.
11. Jiang H, Ruan Q, Nian ML, et al. Modified cuckoo search algorithm and its application to optimization in multiple-effect evaporation. Comput Appl Chem 2014; 31: 1363–1368.
12. Li RY and Dai RW. Adaptive step-size cuckoo search algorithm. Comput Sci 2017; 44: 235–240.
13. Qian WY, Hou HC and Jiang SY. New self-adaptive cuckoo search algorithm. Comput Sci 2014; 41: 279–282.
14. Valian E, Mohanna S and Tavakoli S. Improved cuckoo search algorithm for global optimization. Int J Commun Inf Tech 2011; 1: 31–44.
15. Valian E, Tavakoli S, Mohanna S, et al. Improved cuckoo search for reliability optimization problems. Comput Ind Eng 2013; 64: 459–468.
16. Wang LJ, Yin YL and Zhong YW. Cuckoo search with varied scaling factor. Front Comput Sci 2015; 9: 623–635.
17. Zhao J, Liu SX, Zhou MC, et al. Modified cuckoo search algorithm to solve economic power dispatch optimization problems. IEEE/CAA J Autom Sin 2018; 5: 794–806.
18. Cheng JT, Wang L and Xiong Y. Cuckoo search algorithm with memory and the vibrant fault diagnosis for hydroelectric. Eng Comput 2019; 35: 687–702.
19. Thirugnanasambandam K, Prakash S, Subramanian V, et al. Reinforced cuckoo search algorithm-based multimodal optimization. Appl Intell 2019; 49: 2059–2083.
20. Yung CS. A cuckoo search algorithm: effects of co-evolution and application in the development of distributed layouts. J Algorithms Comput Technol 2019; 13: 1–19.
21. Yang XS and Deb S. Cuckoo search: recent advances and application. Neural Comput Appl 2014; 24: 169–174.
22. Naik MK, Maheshwari RN, Wunnava A, et al. A new adaptive cuckoo search algorithm. In: IEEE international conference on recent trends in information systems, 2015, pp.1–5. Kolkata: IEEE.
23. Vazquez RA. Training spiking neural models using cuckoo search algorithm. In: Proceedings of the IEEE congress on evolutionary computation, 2011, pp.679–686. New Orleans, LA: IEEE.
24. Goel S, Sharma A and Bedi P. Cuckoo search clustering algorithm: a novel strategy of biomimicry. In: Proceedings of word congress on information and communication technologies, 2012, pp.916–921. Mumbai: IEEE.
25. Fernandez-Viagas V and Framinan JM. A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem. Int J Prod Res 2015; 53: 1111–1123.
26. Liu YF. Several intelligent algorithms for permutation flowshop scheduling problems. Xi’an: Xidian University, 2012.
27. Tasgetiren MF, Liang YC, Sevkli M, et al. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *Eur J Oper Res* 2007; 177: 1930–1947.

28. Atef J and Aref M. A new variant of cuckoo search algorithm with self adaptive parameters to solve complex RFID network planning problem. *Wirel Netw* 2019; 25: 1585–1604.

29. Liu CP and Ye CM. Solving permutation flow shop scheduling problem by firefly algorithm. *Ind Eng Manage* 2012; 17: 56–59.