Hindawi
Computational Intelligence and Neuroscience
Volume 2023, Article ID 2040866, 20 pages
https://doi.org/10.1155/2023/2040866

Research Article

Cuckoo Algorithm Based on Global Feedback

Xingyu Liu ,[1] Tao Wu ,[1] Wuxing Lai ,[2] Hu Yuan,[1] Qilong Kou,[3] and Jingping Yu[3]

[1]*School of Software, Huazhong University of Science and Technology, Wuhan 430074, China*
[2]*School of Mechanical Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074, China*
[3]*Tencent Technology (Shenzhen) Co. Ltd., Shenzhen 518000, China*

Correspondence should be addressed to Tao Wu; wutao1972@hust.edu.cn and Wuxing Lai; wxlai@hust.edu.cn

Received 22 August 2022; Revised 14 November 2022; Accepted 16 December 2022; Published 7 January 2023

Academic Editor: Mohamed Abdelaziz

This article proposes a cuckoo algorithm (GFCS) based on the global feedback strategy and innovatively introduces a “re-fly” mechanism. In GFCS, the process of the algorithm is adjusted and controlled by a dynamic global variable, and the dynamic global parameter also serves as an indicator of whether the algorithm has fallen into a local optimum. According to the change of the global optimum value of the algorithm in each round, the dynamic global variable value is adjusted to optimize the algorithm. In addition, we set new formulas for the other main parameters, which are also adjusted by the dynamic global variable as the algorithm progresses. When the algorithm converges prematurely and falls into a local optimum, the current optimum is retained, and the algorithm is initialized and re-executed to find a better value. We define the previous process as “re-fly.” To verify the effectiveness of GFCS, we conducted extensive experiments on the CEC2013 test suite. The experimental results show that the GFCS algorithm has better performance compared to other algorithms when considering the quality of the obtained solution.

1. Introduction

Swarm intelligence algorithm is an algorithm designed to simulate the behavior of natural biological groups, which have been extensively applied for solving complex and highly nonlinear optimization problems. As an emerging optimization algorithm, swarm intelligence algorithm has become one of the focuses of increasingly researchers. Researchers have proposed a variety of algorithms, such as ant colony algorithm (ACO) [1], differential evolution (DE) [2], particle swarm optimization algorithm [3] (PSO), artificial bee colony algorithm [4] (ABC), firefly algorithm [5] (FA), and cuckoo search algorithm [6] (CS). At present, these algorithms have been applied to a variety of engineering optimization problems and have a potential research value. Hence, it is still a promising domain to develop more effective swarm intelligence algorithms.

The CS algorithm, inspired by the parasitic brooding behavior of cuckoos, was proposed by Yang and Deb et al. in 2009. This parasitic behavior has become a breeding strategy for cuckoos, and in most cases, they lay their eggs in the nests of other bird species. Therefore, the host bird may discover that the egg is not its own, at which point it either throws away the foreign egg or abandons the nest and builds a new one. In addition, the CS algorithm employs methods such as greedy selection, random walk, and Lévy flight [7] to solve the global optimal solution. Compared with the uniform distribution and Gaussian distribution algorithm, the long-hop mode algorithm provided by the Lévy flight can search the solution domain better. The combination of Lévy flight advantage and local search ability makes the CS algorithm one of the most effective optimization algorithms. Compared with other swarm intelligence algorithms, CS has the advantages of fewer parameters, simple operation, and strong optimization ability, and it is more effective in solving optimization problems. However, on the contrary, there are also the defects of unbalanced exploration ability and mining ability, and it is easy to fall into the local optimal solution.

Since similar search strategies, Lévy flight and random walk strategies, are adopted in most CSs, the search behaviors of cuckoos are similar, which can easily lead the algorithm to fall into a local optimum and enter premature convergence. Sometimes, the algorithm converges to a local optimum at a very early stage, but the whole algorithm ends without obtaining a better fitness value. Under these circumstances, it not only is difficult to obtain a better value but also wastes subsequent computing resources.

Based on this situation, a new type of cuckoo algorithm (GFCS) is proposed. GFCS dynamically adjusts the parameters of the algorithm according to whether each round of the algorithm iteration produces a better fitness value. In the case where the fitness value remains unchanged for a long time, the current optimal value is retained; then, the algorithm will be reset, resulting in better algorithm performance for the same computational generation. Briefly, the core idea of this work is as follows:

The GFCS algorithm is a CS algorithm which employs random walk and Lévy flight to search for the global optimum. We have proposed three innovations based on the original CS algorithm:

(i) We introduce the concept of global feedback to adjust the dynamic global variables by the current round of iterations and determine whether the algorithm falls into a local optimum.

(ii) The fixed parameter pattern of the original CS algorithm is optimized. We set the parameter formulas that vary with the number of iteration rounds and is controlled by the dynamic global variables.

(iii) We introduced the "re-fly" mechanism. When the algorithm falls into the local optimum, the algorithm can save the current global optimum value and the algorithm will be initialized and re-executed to find a better value.

The article is organized as follows. Section 2 reviews the original CS and its technical details. In Section 3, the literature on CS and its application to optimization problems are presented. Section 4 elaborates on the proposed algorithm. A comparative analysis of numerical experiments between GFCS and CS, multiple CS variants, and several other state-of-the-art algorithms is presented in Section 5. Finally, in Section 6, we summarize the proposed algorithm.

2. Basic Cuckoo Search Algorithm

The cuckoo search algorithm (CS) is a swarm intelligence algorithm inspired by the natural behavior of some cuckoo species laying their eggs in other birds' nests. Different from other algorithms, the search process of CS is divided into two stages: global search and local search, corresponding to exploration and exploitation, respectively. The global stage is carried out by the Lévy flight, as the Lévy distribution has infinite mean and variance, which helps to explore the solution space efficiently. The local phase is executed by using the biased random walk.

In the CS algorithm, the number of hosts available is constant. In each iteration, each cuckoo lays only one egg and then randomly places the egg into the host's nest. Each egg is used as a solution to the problem. The host bird has a probability of Pa ($Pa \in [0,1]$) to find the cuckoo laying eggs in its nest. When this happens, the laid eggs are thrown away or the host bird simply abandons the nest to make a new one.

It is assumed that N is the number of cuckoos and D represents the dimension of problem, the position of i th cuckoo is denoted as Xi, and t represents the current iteration. Then, the new position can be generated by the following equations:

$$x_i^{t+1} = x_i^t + \alpha \oplus \text{Lévy}(\text{s}, \lambda), \tag{1}$$

where α is the step size, which should be related to the scale of the problem, and the product $\oplus$ represents the multiplication of the corresponding position of the vector. Therefore, the formula for the Lévy flight is as follows:

$$\text{Lévy}(\text{s}, \lambda) = \frac{\lambda\Gamma(\lambda) \cdot \sin(\pi\lambda/2)}{\pi} \frac{1}{\text{s}^{1+\lambda}}, \tag{2}$$

where

$$\alpha = \alpha_0\left(x_i^t - x_{\text{best}}^t\right), \tag{3}$$

where x_{best}^t represents the optimal solution of the tth generation, $\text{Lévy}(\text{s}, \lambda)$ represents the feature scale, λ represents the power coefficient $(1 < \lambda < 3)$, and Γ represents the gamma function. In addition, α_0 represents the scaling factor, which controls the size of the step.

In formula (1) and formula (2), s represents the step size of the Lévy flight, and it was designed by Mantegna's algorithm [8] as follows:

$$s = \frac{\mu}{|\nu|^{1/\lambda}}, \tag{4}$$

where μ and ν are random numbers drawn from a normal distribution:

$$\mu \sim N\left(0, {\sigma_\mu}^2\right), \nu \sim N\left(0, {\sigma_\nu}^2\right), \tag{5}$$

where the value of σ_ν is usually set to 1, and the formula for σ_μ is shown as follows:

$$\sigma_\mu = \left\{\frac{\Gamma(1+\lambda) \cdot \sin(\pi\lambda/2)}{\Gamma[(1+\lambda)/2] \cdot \lambda \cdot 2^{(\lambda-1)/2}}\right\}^{1/\lambda}. \tag{6}$$

Then, the formula for the local random walk can be expressed as

$$x_i^{t+1} = \begin{cases} x_i^t + r \cdot \left(x_j^t - x_k^t\right), & \text{rand} > P_a, \\ x_i^t, & \text{otherwise.} \end{cases} \tag{7}$$

where r is a scaling factor uniformly distributed in the range [0, 1] and x_j^t and x_k^t represent two different solutions randomly selected in the population.

Based on the previous introduction, the original CS algorithm framework is shown in Algorithm 1.

3. Related Works

The main advantages of the CS algorithm are few parameters, simple operation, easy implementation, optimal random search path, and strong search ability. At present, scholars at home and abroad have also proposed many improvement strategies for the cuckoo algorithm. The main

```
Input: population size $N$, the maximum number of iterations MaxIt, problem dimension $D$, and the probability of the bird's nest being found $Pa$.
(1)  Randomly initialize the solution set with a population size of $N : x_i^t (i = 1, 2, \cdots, N)$;
(2)  Calculate the fitness of all solutions $f_i^t = f(x_i^t)$; Get the best solution $x_{\text{best}}^t$ and its fitness value $f_{\text{best}}$;
(3)  for $t = 1$: MaxIt do
(4)      for $i = 1 : \text{N}$ do
(5)   Generate a new solution $x_{\text{new}}$ by the Lévy flight mode in equation (1);
(6)   Calculate its fitness $f_{\text{new}} = f(x_{\text{new}})$;
(7)   if $f_{\text{new}} < f_i^t$ then
(8)    $x_i^{t+1} = x_{\text{new}}$;
(9)    $f_i^{t+1} = f_{\text{new}}$;
(10)  end if
(11)  end for
(12) Throw away a small fraction (controlled by $Pa$) of the worst solutions and use equation (7) to generate a new solution;
(13) Update the global optimal solution;
(14) end for
(15) Output: Final optimized solution
```

Algorithm 1: Cuckoo search (CS) original algorithm.

research directions in previous years include improving the step size of the Lévy flight and the random walk algorithms, or adjusting the parameter *Pa* by introducing a new *Pa* formula, or setting a new step size adjustment formula to adjust the performance of the algorithm.

Valian et al. proposed an improved cuckoo algorithm [9] for reliability optimization problems. The optimization of the step size of the Lévy flight was introduced into the algorithm, and the probability of cuckoo eggs being found *Pa* was adjusted. With the change of the number of iterations, the step size alpha and *Pa* were gradually reduced according to a certain formula. Naik et al. proposed a new cuckoo algorithm [10] that abandoned the Lévy flight by using a step size based on the number of iterations, the contemporary optimal nest, the contemporary worst nest, and the average nest fitness value. Ong proposed an adaptive cuckoo algorithm [11], which compared the current fitness value with the average fitness value and used different step size algorithms according to the comparison results to ensure that the algorithm had a faster convergence speed in the early stage and a large convergence accuracy in the later stage. Wang et al. proposed a cuckoo algorithm with different scale factors [12]. During the iteration process, random numbers were introduced to make fluctuations at each step, which improved the performance of the algorithm but reduced the stability of the algorithm. Li and Yin proposed a modified cuckoo search algorithm with a self-adaptive parameter method [13]. A linear change of parameters was achieved by introducing the ratio of the current algebra to the total algebra, and according to the success rate of the evolution of the previous generation, different schemes of *Pa* were selected. Huang et al. proposed a cuckoo search (CS) algorithm using an elite opposition-based strategy [14], in which the proposed algorithm generated the opposite solutions of elite individuals in the population by an opposition-based strategy. The algorithm was guided to explore the optimal solution by simultaneously evaluating the current population and the opposite population. Based on the elite opposition-based strategy mentioned previously, Baset et al. proposed a new cuckoo search algorithm [15] for solving integer programming problems, which had faster convergence and higher computational accuracy and was more effective.

Some scholars adjusted or replaced the Lévy flight with some new models, such as introducing other algorithms or introducing Gaussian functions to speed up the optimization. Kamoona et al. proposed an enhanced cuckoo algorithm [16], which replaced the Lévy flight with the Gaussian virus diffusion idea, and innovatively introduced the search formula of the artificial bee colony algorithm. Zheng and Zhou proposed a new cuckoo algorithm based on Gaussian distribution optimization [17]. The algorithm replaced the Lévy flight with the Gaussian distribution to a certain extent, and the algorithm had relatively good performance in local optimization performance. He et al. proposed a Spark-based Gaussian Bare-bones cuckoo Search with dynamic parameter selection [18]. For *Pa* values, a pool of candidate *Pa* values in the range [0.01, 0.5] was introduced, and the value for the step size was generated by a Gaussian distribution.

Inspired by the organizational evolutionary algorithm for numerical optimization, Zheng and Zhou designed a novel algorithm, the cooperative co-evolutionary cuckoo search algorithm (CCCS) [19], which combined dynamic populations and evolutionary operators for solving both unconstrained, constrained optimization and engineering problems. Cheng and Wang proposed a neighborhood-attracting cuckoo algorithm [20], which introduced the concept of neighborhood in the cuckoo algorithm, making the fitness value of the bird's nest closer to the best bird's nest in the area. At the same time, the difference between the fitness value of the individual and the best individual was analysed, and the step size of this iteration was determined by the difference.

Some researchers tried to optimize the cuckoo algorithm by mixing multiple algorithms, randomly or based on certain feedback to select the most suitable algorithm for the current generation, which diversified the iteration selection of the algorithm. The disadvantage was that while promoting the algorithm's search ability, it also increased the instability of the algorithm. Rakhshani and Rahati proposed a new cuckoo algorithm based on Snap-Drift [21]. The algorithm divided the entire optimization process into two modes: snap and drift, selecting the best search mode according to the optimization effect of the current algebra. Peng et al. proposed a multistrategy serial cuckoo algorithm [22], which divided the execution process of the overall algorithm into three stages, including jump learning, Gaussian walking learning, and begging behavior. Different strategies were formulated at each stage and achieved better results. In addition, he also proposed another similar algorithm, the multistrategy reconciliatory cuckoo search algorithm [23], which updated individuals based on a harmonic strategy. The adaptive step size guided the cuckoo to seek optimization in a better direction, and three improved update methods were explored analytically from three perspectives: their own neighborhood, the current optimal individual, and the random position. Gao et al. proposed a multistrategy adaptive cuckoo algorithm [24], and the algorithm was designed with five different search ideas. According to the performance of each iteration, a certain selection ratio was set for these five strategies, making the algorithm tend to be diversified.

In addition, the cuckoo algorithm also has some other improvements. Zhang et al. proposed a dynamic adaptive cuckoo search algorithm [25], which introduced feedback into the algorithm framework and established a closed-loop control system for CS algorithm parameters. The improvement rate was maintained at 20 percent, and Pa and α were dynamically adjusted. Walton et al. proposed an adjusted cuckoo algorithm [26] to classify bird nests. The excellent parts were set as the top nests; then, new nests were constructed by the relationship between the top nests. Excellent nests were used to construct new nests, thus selecting the best ones and improving the local exploration ability of the algorithm, while the poor nests used a larger step size to improve the global ability of the algorithm.

At the same time, the cuckoo algorithm and its improved algorithm also have a very broad application, such as PID controller design [27], grayscale image enhancement [28], cryptanalysis of Vigenere ciphers [29], data privacy protection [30], image segmentation [31], spam detection [32], the design of multidocument summarization extractor [33], and robot path planning [34, 35].

4. Cuckoo Algorithm Based on Global Feedback

The cuckoo algorithm based on global feedback is described in the following sections.

4.1. Motivation. In Lévy flights, it is difficult to balance the large-scale global exploration in the early stages and the local fine-grained search in the later stages with a fixed step size and probability of nest discovery. To improve the dynamic search characteristics of the CS algorithm, an adaptive adjustment scheme of step size α and the probability of nest discovery Pa are proposed to coordinate the overall search performance of the algorithm according to the evolution of the globally optimal individuals. Due to the guidance of the optimal solution in the population, the CS algorithm is easy to converge prematurely when solving some complex optimization problems and falls into the local optimal value prematurely. Even at the end of the whole algorithm, the algorithm fails to jump out of the local optimum. Given this situation, we introduce additional dynamic parameters to adjust the step size and Pa according to the change of the best fitness value of the current generation. If it has been determined that the algorithm is stuck in a local optimum after many rounds, then we consider resetting the algorithm.

4.2. Step Size and **Pa** *Adjustment.* The main parameters in the CS algorithm are step size α and the probability of the bird's nest being found Pa. α mainly controls the step size of the algorithm, and the value of Pa mainly controls the diversity of the algorithm for each round of exploration. If a smaller value of Pa is used, the global bird's nest will tend to be concentrated, and the local search ability of the algorithm will be strengthened, but the diversity of the algorithm gets worse. If the value of Pa is small but the value of α is large, the performance of the algorithm will be poor, resulting in a large increase in the number of iterations. If the value of Pa is large but the value of α is small, the convergence rate is fast, but the optimal solution may not be found. In the original CS algorithm, both Pa and α use fixed values, which cannot be changed during the algorithm iteration. It is difficult for the algorithm to guarantee the speed of global exploration in the early stage, and the lack of overall exploration capability may contribute to the inability of the algorithm to explore the global optimum. And in the later stage, due to the slightly larger step size, it is difficult for the algorithm to perform a very fine search locally. To improve the performance of the algorithm, an adaptive step size formula is introduced as follows:

$$\alpha(t) = m * \alpha_{\max} \exp(c), \tag{8}$$

where

$$c = \ln\left(\frac{\alpha_{\min}}{\alpha_{\max}}\right) * \frac{t}{\text{MaxIt}}, \tag{9}$$

and the formula for Pa is as follows:

$$P_a(t) = \frac{m * 2Pa_{\max}}{(1 + \exp(T * t/\text{MaxIt}))}, \tag{10}$$

where t represents the current number of iterations, MaxIt represents the total number of iterations, $\alpha_{\min}$ and $\alpha_{\max}$ represent the preset minimum step size and the preset maximum step size, respectively, $Pa_{\max}$ represents the preset maximum value of Pa, T represents a constant between 1 and 10, and m represents a dynamically adjusted global parameter, which will be introduced in the following sections.

Equations (8) and (9) refer to parts of [9], on which we change the values of the upper and lower bounds of their definitions and use dynamic global variables to adjust their values again. So, we choose $\alpha_{\max} = 0.5$ and$\alpha_{\min} = 0.005$.

For equation (10), inspired by the sigmoid function [36] in deep learning, we hope the value of parameter Pa to smoothly over from a larger value in the early stage to a smaller value in the later stage like the inverted sigmoid function during the whole algorithm, so equation (10) is designed. In order to prevent the curve from changing too drastically or slowly, an adjustable variable T is added on top of it to control the whole change process.

To ensure that $P_a(t)$ can obtain a large value when t is small at the beginning of the algorithm, we choose $Pa_{\max} = 0.5$. To ensure that the curve of $P_a(t)$ changes quickly but not too steeply, we did experiments on the value of T. The experimental results show that the overall effect is best when T is taken as 6, so we choose $T = 6$.

4.3. Global Feedback and Re-Fly.

4.3. Global Feedback and Re-Fly. The previous formulas of Pa and α ensure that Pa and α can be quickly changed from larger values in the previous stage to relatively small values as the algorithm progresses. This process is essentially irreversible, and the algorithm is likely to fall into a local optimum, thus affecting the subsequent global exploration. For this, we introduce a dynamic adjustment parameter m and set the initial value of m to 1. As the algorithm runs, the formula for m is as follows:

$$m = \begin{cases} 1, & f_{\text{best}}^{\text{new}} < f_{\text{best}}, \\ m + 0.001, & \text{otherwise.} \end{cases} \tag{11}$$

If the optimal fitness value does not change after this round of iteration, we will increase the value of m. Due to the influence of m, the step size α and Pa of the algorithm are enlarged, and the global exploration ability of the algorithm is improved. If the optimal fitness value of the algorithm changes after a certain round of iteration, we consider that the algorithm has found a better solution and reset m to 1 at this point. If the algorithm falls into a local optimal value (nonglobal optimal value), as m continues to increase, it still does not obtain a better optimal fitness value. When $m > 2$, we will make a judgment at this time.

If $t < \text{MaxIt}/2$, we will keep the current best fitness value, reset all nests, set m to 1, and re-execute the algorithm. This stage is called "re-fly." Otherwise, we will continue to execute the algorithm and expand m, making the step size gradually expand until a better fitness value appears.

In the traditional CS algorithm and some improved versions of the CS algorithm, the algorithm may converge prematurely at the beginning of the algorithm and fall into a local optimum, and at the end of the entire algorithm, the algorithm does not obtain a better value. Therefore, the "re-fly" method is introduced in GFCS. If the best fitness value does not change after 1000 iterations and the current number of rounds does not reach half of the total number of rounds, the subsequent computing power is reserved to obtain better results.

In terms of the previous descriptions, the implementation of GFCS is shown in Algorithm 2.

To show the algorithm process more visually, the flowchart of the algorithm is shown in Figure 1.

4.4. Algorithm Complexity Analysis. To demonstrate that the GFCS algorithm does not increase the time and space complexity of the CS algorithm, we analyse the algorithmic complexity of GFCS and CS. For the CS algorithm, assuming that the dimension of the problem is D, the time to evaluate the D-dimensional function is positively related to D, the total number of iterations is G, and the population size is n. Therefore, the time complexity of the CS algorithm is approximately: $O(\text{GDn})$.

GFCS is improved only in the Lévy flight phase and is consistent with the CS algorithm in the local walk phase, so we only need to analyse the differences in the previous phase. GFCS requires additional calculations of variables α and Pa in each iteration, determines and updates the value of the parameter m. The time consumption during the calculation is only related to the parameters G and not to D and n. In addition, "re-fly" is performed at most once in the algorithm, and the time consumption is only related to n and D and not to G. Therefore, the total time complexity of GFCS is still $O(\text{GDn})$.

In terms of space complexity, GFCS does not use extra storage space to store data, so it does not increase the space complexity of the algorithm. In summary, the complexity of GFCS is in the same order of magnitude as that of the original CS. In the subsequent experiments, we will further compare the total time consumption of GFCS with the original CS on the test set functions.

5. Experimental Study

The experimental study is described in the following sections.

5.1. Experimental Environment and Benchmark Functions. To verify the performance of GFCS, experiments are carried out on the test set of CEC2013 [37], which is widely used internationally. CEC2013 contains 28 test functions, among which $f1$–$f5$ are unimodal functions, $f6$–$f20$ are multimodal functions, and $f21$–$f28$ are combination functions. The solution constraints of the functions are in the range of $[-100, 100]$. All experiments were performed on the Windows 10 platform, and all algorithms were implemented in MATLAB R2021a.

In our experiments, f_{opt} represents the standard optimal value of the objective function and $f_{\min}$ represents the actual optimal value of the objective function obtained by our algorithm. We record the following equation as the criterion for the algorithm detection:

$$\text{res} = f_{\min} - f_{\text{opt}}. \tag{12}$$

Therefore, the closer res is to 0, the better it is. Furthermore, to reduce the statistical error, the average error of all these independently running functions was chosen as the

```
     Input: population size $N$, maximum number of iterations MaxIt, problem dimension $D$, mode switching parameter $Pa$.
(1)  count = 0;
(2)  Randomly initialize the solution set with a population size of $N : x_i^t (i = 1, 2, \cdots, N)$;
(3)  Calculate the fitness of all solutions $f_i^t = f(x_i^t)$; Get the best solution $x_{\text{best}}^t$ and its fitness value $f_{\text{best}}$; $m = 1$;
(4)  for $t = 1$: MaxIt do
(5)    Calculate $\alpha$ and $Pa$ using equations (8) and (10) with $t$, respectively;
(6)    for $i = 1 : N$ do
(7)    Generate a new solution $x_{\text{new}}$ by the Lévy flight mode in equation (1);
(8)      if $f_{\text{new}} < f_i^t$ then
(9)        $x_i^{t+1} = x_{\text{new}}$; $f_i^{t+1} = f_{\text{new}}$;
(10)     end if
(11)   end for
(12)   Throw away a small fraction (controlled by $Pa$) of the worst solutions and use equation (7) to generate a new solution;
(13)   Update the global optimal solution $f_{\text{best}}^{\text{new}}$;
(14)   if $f_{\text{best}}^{\text{new}} < f_{\text{best}}$ $m = 1$;
(15)   else $m = m + 0.001$;
(16)   end if
(17)   if $m > 2$ && $t <$ MaxIt/2 && count = 0
(18)     Repeat 2 and 3, count = 1 (will only be executed once at most)
(19)   end if
(20)   $t = t + 1$;
(21) end for
(22) Output: Final optimized solution
```

Algorithm 2: Cuckoo search based on the global feedback.

performance metric. To ensure the fairness of the experiment, the population size n is set to 25, with the dimension $D = 30$ and the maximum number of iterations MaxIt = 20000. Each test function was run 30 times in the same environment, and its mean and standard error values were recorded.

5.2. Comparison with CS and Other Variants. To explore the accuracy and convergence of GFCS, a comparative analysis was carried out with the original CS and its seven variants. The seven CS variants were CS [6], ACS [10], NACS [20], GCS [17], ICS [9], ACSA [11], VCS [12], and MSRCS [23]. Table 1 lists the core ideas and specific parameters of each algorithm.

In this section, we compare GFCS with the original CS and 7 improved CS variants. Tables 2 and 3 show the 30-dimensional test results of GFCS and other 8 CS algorithms in the CEC2013 test set. In the tables, bold letters indicate the best results, "Mean" and "Std" represent the mean and standard error values, respectively. In addition, the average ranking results of the Friedman test are added at the bottom of the table, where "+" indicates that the results are better than the algorithm, "−" indicates that are worse, and "≈" indicates that the results are not much different.

The data in Tables 2 and 3 show that GFCS works best on $f2, f4, f6, f7, f9, f12, f13, f15, f18, f19, f20, f23, f24, f25, f26$, and $f27$, which include unimodal functions, multimodal functions, and combined functions. In unimodal functions, the effect of GFCS is clearly better than other algorithms on $f2$ and $f4$, and the performance of each algorithm tends to be consistent on $f1$, $f3$, and $f5$. The results of GFCS on multimodal functions $f6, f7, f9, f12, f13, f15, f18, f19$, and $f20$ are all better than other algorithms, and the results of $f17$ are better than other algorithms except VCS. And in the combined functions $f23, f24, f25, f26$, and $f27$, the results of GFCS are better, as it obtains better optimal values. Through the results of the Friedman test at the bottom of Tables 2 and 3, it can be found that GFCS beats other algorithms on at least 18 functions compared to other algorithms. And the average ranking of GFCS for the 28 tested functions in the two tables are 1.43 and 1.46, respectively, which ranks first compared to the other 8 CS algorithms. The analysis shows that GFCS has good stability and convergence for different types of problems.

In addition, to visually display the ranking of the results in Tables 2 and 3, the ranking of the average error value (minimization problem) of each function is summarized, and stacked histograms based on the ranking statistics are drawn. Figures 2 and 3 shows that each ranking is represented by a color block. The better the algorithm performs, the lighter the color of the corresponding block is. Figure 2 shows the ranking of GCS, ACSA, NACS, VCS, and GFCS, and Figure 3 shows the ranking of MSRCS, ACS, ICS, CS, and GFCS.

As can be seen from the figures, the white block that marks the first has the largest proportion in GFCS, and GFCS does not obtain the red square that marks the fifth, indicating that GFCS has the best performance compared to other algorithms. In addition, on some functions, such as $f11$ and $f16$, GFCS does not achieve the first place, but still locates in the second or third position, showing that GFCS is still competitive. Overall, for the CEC2013 test set with $D = 30$, GFCS has a considerable advantage over other algorithms.

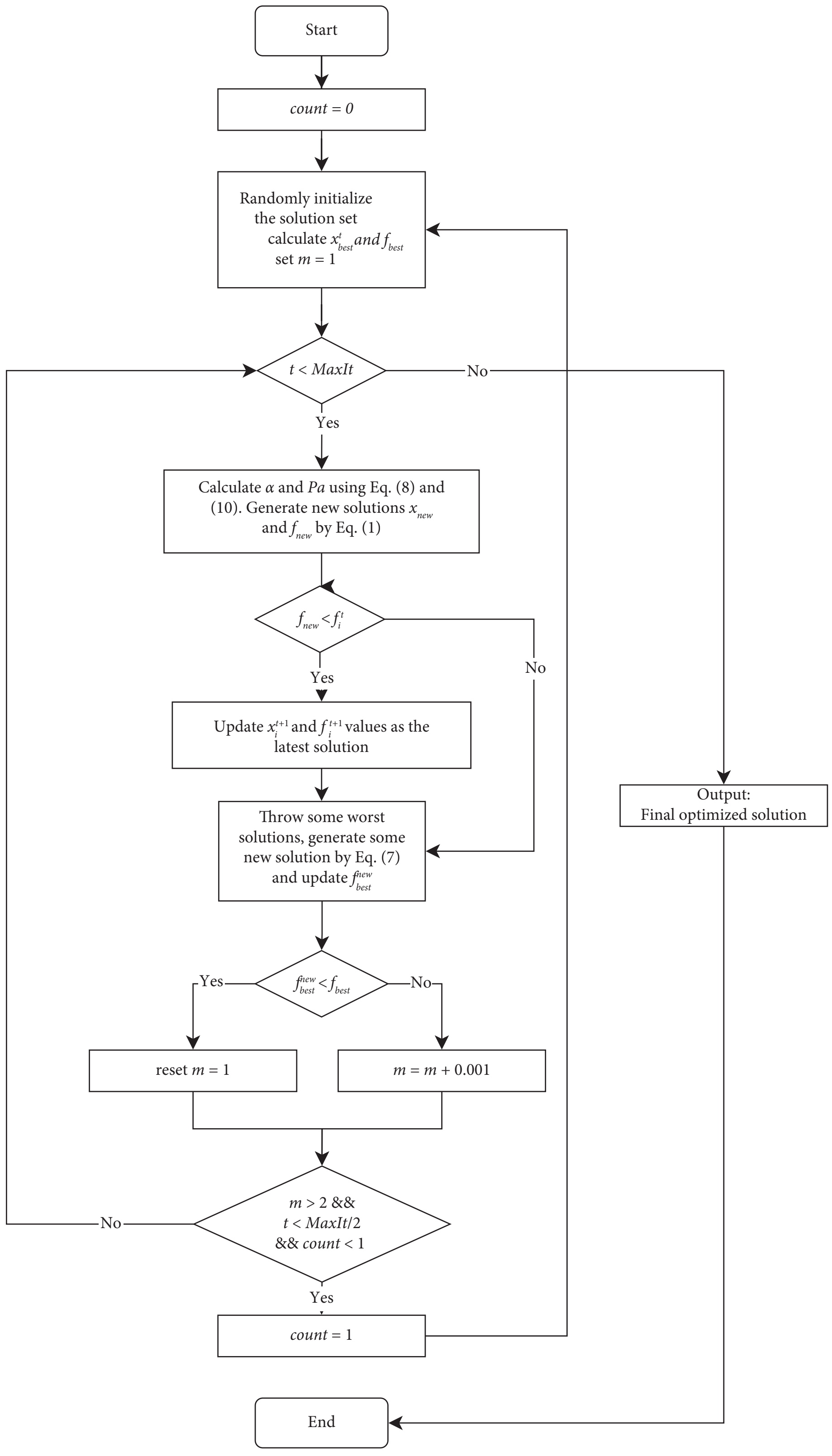
Start
$count = 0$
Randomly initialize the solution set calculate x^t_{best} and f_{best} set $m = 1$
$t < MaxIt$
No
Yes
Calculate α and Pa using Eq. (8) and (10). Generate new solutions x_{new} and f_{new} by Eq. (1)
$f_{new} < f^t_i$
Yes
No
Update x^{t+1}_i and f^{t+1}_i values as the latest solution
Throw some worst solutions, generate some new solution by Eq. (7) and update f^{new}_{best}
$f^{new}_{best} < f_{best}$
Yes
No
reset $m = 1$
$m = m + 0.001$
$m > 2$ && $t < MaxIt/2$ && $count < 1$
No
Yes
$count = 1$
Output: Final optimized solution
End

Figure 1: The flowchart of the GFCS algorithm.

To further illustrate the convergence capability and optimization accuracy of GFCS, the original CS and four representative CS variants are selected in conjunction with the comparison results above. In addition, we select representative functions from the unimodal, multimodal, and combinatorial functions of the CEC2013 test set. They are $f2$,

Table 1: Core ideas and parameter settings for CSs.

Algorithms	Core ideas	Parameter settings
CS	Basic CS	$\alpha = 0.01$ and $P_a = 0.25$
ACS	The Lévy flight is abandoned and a new step control formula and iteration formula are introduced	$P_a = 0.25$
NACS	Neighborhood attraction is introduced to improve the performance of CS	$Pa = 0.25, m = 3,$ and $Ps = 0.8$
ICS	Control the step size and Pa with the number of iterations to improve the performance of CS	$\alpha_{\max} = 0.5$ and $\alpha_{\min} = 0.01$ $P_{a\max} = 0.5$ and $P_{a\min} = 0.005$
ACSA	Compare the current fitness value with the average fitness value to decide on the two different step size methods	$\alpha_U = 0.8$ and $\alpha_L = 0.2$
VCS	In VCS, a varied scale factor is added to the step adjustment of the Lévy flight	$\beta = 1.5$ and $P_a = 0.25$
GCS	Using Gaussian distribution instead of the Lévy flight to improve the algorithm performance	$\mu = 0.0001, \sigma_0 = 0.5, and\ P_a = 1.5$
MSRCS	Based on reconciliatory strategy to update solutions, MSRCS tries to strike a balance between exploration and exploitation	$p_a = 0.25,\ p_1 = 0.8, and\ p_2 = 0.5$
GFCS	Dynamic variables and "re-fly" mechanism are added to the operation of the algorithm, and feedback is obtained on a per-round iteration to improve algorithm performance	$\alpha_{\max} = 0.5, \alpha_{\min} = 0.005$ $Pa_{\max} = 0.5,$ and $T = 6$

Table 2: The test result of GCS, ASCA, NACS, VCS, and GFCS on CEC2013 ($D = 30$).

Function	Mean/std	GCS	ASCS	NACS	VCS	GFCS
***F*1**	Mean	**0.00*E* + 00**	**0.00*E* + 00**	**0.00*E* + 00**	**0.00*E* + 00**	**0.00*E* + 00**
	Std	**0.00*E* + 00**	**0.00*E* + 00**	**0.00*E* + 00**	**0.00*E* + 00**	**0.00*E* + 00**
***F*2**	Mean	9.47*E* + 05	2.22*E* + 06	1.41*E* + 05	1.44*E* + 06	**1.96*E* + 03**
	Std	3.22*E* + 05	9.51*E* + 05	6.01*E* + 04	5.45*E* + 05	**2.48*E* + 03**
***F*3**	Mean	**1.00*E* + 10**	**1.00*E* + 10**	**1.00*E* + 10**	**1.00*E* + 10**	**1.00*E* + 10**
	Std	**0.00*E* + 00**	**0.00*E* + 00**	**0.00*E* + 00**	**0.00*E* + 00**	**0.00*E* + 00**
***F*4**	Mean	4.15*E* + 03	8.89*E* + 02	1.12*E* + 00	7.49*E* + 02	**3.97*E* − 01**
	Std	1.46*E* + 03	2.88*E* + 02	**4.04*E* − 01**	2.51*E* + 02	4.98*E* − 01
***F*5**	Mean	**0.00*E* + 00**	**0.00*E* + 00**	**0.00*E* + 00**	**0.00*E* + 00**	**0.00*E* + 00**
	Std	**0.00*E* + 00**	**0.00*E* + 00**	**0.00*E* + 00**	**0.00*E* + 00**	**0.00*E* + 00**
***F*6**	Mean	1.52*E* + 00	3.74*E* + 00	1.33*E* + 01	2.33*E* + 00	**1.36*E* − 13**
	Std	2.89*E* + 00	8.59*E* + 00	1.51*E* + 01	6.58*E* + 00	**4.37*E* − 13**
***F*7**	Mean	6.80*E* + 01	5.22*E* + 01	1.39*E* + 02	5.77*E* + 01	**3.53*E* + 01**
	Std	3.03*E* + 01	2.60*E* + 01	3.50*E* + 01	1.39*E* + 01	**7.09*E* + 00**
***F*8**	Mean	**2.09*E* + 01**	**2.09*E* + 01**	**2.09*E* + 01**	**2.09*E* + 01**	**2.09*E* + 01**
	Std	**2.40*E* − 02**	5.16*E* − 02	3.27*E* − 02	3.19*E* − 02	5.64*E* − 02
***F*9**	Mean	2.91*E* + 01	2.86*E* + 01	3.14*E* + 01	2.73*E* + 01	**2.16*E* + 01**
	Std	**9.52*E* − 01**	2.60*E* + 00	2.54*E* + 00	1.82*E* + 00	2.68*E* + 00
***F*10**	Mean	1.74*E* − 02	**1.47*E* − 02**	1.44*E* − 01	2.02*E* − 02	1.63*E* − 02
	Std	2.01*E* − 02	**1.48*E* − 02**	8.60*E* − 02	1.87*E* − 02	1.49*E* − 02
***F*11**	Mean	1.91*E* + 01	**4.03*E* + 00**	6.91*E* + 00	8.51*E* + 00	1.21*E* + 01
	Std	1.17*E* + 01	**2.08*E* + 00**	3.97*E* + 00	3.97*E* + 00	3.20*E* + 00
***F*12**	Mean	9.73*E* + 01	9.08*E* + 01	3.07*E* + 02	7.66*E* + 01	**5.38*E* + 01**
	Std	1.76*E* + 01	1.26*E* + 01	9.22*E* + 01	1.17*E* + 01	**1.12*E* + 01**
***F*13**	Mean	1.48*E* + 02	1.37*E* + 02	3.33*E* + 02	1.19*E* + 02	**1.00*E* + 02**
	Std	**1.28*E* + 01**	2.09*E* + 01	5.26*E* + 01	1.99*E* + 01	2.62*E* + 01
***F*14**	Mean	1.18*E* + 03	9.85*E* + 02	**7.07*E* + 02**	8.30*E* + 02	1.02*E* + 03
	Std	3.68*E* + 02	3.49*E* + 02	2.73*E* + 02	4.00*E* + 02	**2.68*E* + 02**
***F*15**	Mean	4.59*E* + 03	4.41*E* + 03	3.65*E* + 03	4.27*E* + 03	**3.46*E* + 03**
	Std	5.07*E* + 02	**3.29*E* + 02**	3.98*E* + 02	3.47*E* + 02	7.47*E* + 02
***F*16**	Mean	1.60*E* + 00	1.46*E* + 00	**5.04*E* − 01**	1.42*E* + 00	1.38*E* + 00
	Std	1.20*E* − 02	1.78*E* − 01	**9.83*E* − 02**	2.37*E* − 01	3.41*E* − 01

Table 2: Continued.

Function	Mean/std	GCS	ASCS	NACS	VCS	GFCS
***F*17**	Mean	7.53E+01	5.35E+01	1.49E+02	**4.70E+01**	5.29E+01
	Std	9.63E+00	9.61E+00	5.27E+01	**7.15E+00**	7.81E+00
***F*18**	Mean	1.59E+02	1.45E+02	3.63E+02	1.32E+02	**8.25E+01**
	Std	**8.17E+00**	1.59E+01	9.91E+01	9.88E+00	1.89E+01
***F*19**	Mean	8.20E+00	5.95E+00	1.75E+01	4.34E+00	**2.29E+00**
	Std	1.44E+00	1.85E+00	5.57E+00	1.45E+00	**4.77E−01**
***F*20**	Mean	1.21E+01	1.19E+01	1.33E+01	1.19E+01	**1.11E+01**
	Std	**2.94E−01**	3.75E−01	8.24E−01	3.65E−01	4.84E−01
***F*21**	Mean	2.48E+02	2.56E+02	3.25E+02	2.63E+02	**2.47E+02**
	Std	**4.62E+01**	6.35E+01	1.03E+02	5.64E+01	5.16E+01
***F*22**	Mean	1.94E+03	1.39E+03	6.07E+02	9.87E+02	1.04E+03
	Std	5.21E+02	5.55E+02	**2.66E+02**	5.76E+02	3.16E+02
***F*23**	Mean	5.04E+03	5.08E+03	4.75E+03	4.77E+03	**3.68E+03**
	Std	5.96E+02	3.70E+02	4.20E+02	**3.00E+02**	5.32E+02
***F*24**	Mean	2.75E+02	2.75E+02	2.91E+02	2.73E+02	**2.59E+02**
	Std	7.36E+00	**6.57E+00**	8.92E+00	1.05E+01	7.22E+00
***F*25**	Mean	2.90E+02	2.84E+02	3.06E+02	2.85E+02	**2.75E+02**
	Std	**3.32E+00**	9.33E+00	9.78E+00	6.18E+00	6.45E+00
***F*26**	Mean	**2.00E+02**	**2.00E+02**	2.54E+02	**2.00E+02**	**2.00E+02**
	Std	1.74E−02	3.31E−02	8.41E+01	2.32E−02	**1.57E−04**
***F*27**	Mean	1.11E+03	1.03E+03	1.17E+03	1.02E+03	**8.57E+02**
	Std	**2.58E+01**	1.51E+02	6.59E+01	3.50E+01	1.42E+02
***F*28**	Mean	**3.00E+02**	**3.00E+02**	1.90E+03	**3.00E+02**	**3.00E+02**
	Std	**0.00E+00**	**0.00E+00**	1.33E+03	**0.00E+00**	**0.00E+00**
Average ranking		2.71	2.79	3.46	2.25	1.43
+/ − / ≈		21/1/6	19/4/5	20/4/4	18/4/6	

In the table, bold letters indicate the best results, "Mean" and "Std" represent the mean and standard error values, respectively.

Table 3: The test result of CEC2013 of MSRCS, ACS, ICS, CS, and GFCS on CEC2013 ($D = 30$).

Function	Mean/std	MSRCS	ACS	ICS	CS	GFCS
***F*1**	Mean	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
	Std	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
***F*2**	Mean	3.78E+04	1.06E+06	6.65E+04	8.91E+05	**1.96E+03**
	Std	1.59E+04	3.74E+05	4.40E+04	3.76E+05	**2.48E+03**
***F*3**	Mean	**1.00E+10**	**1.00E+10**	**1.00E+10**	**1.00E+10**	**1.00E+10**
	Std	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
***F*4**	Mean	6.87E−01	1.07E+03	6.14E+00	2.10E+03	**3.97E−01**
	Std	5.74E−01	3.42E+02	5.00E+00	8.76E+02	**4.98E−01**
***F*5**	Mean	**0.00E+00**	**0.00E+00**	1.14E−14	**0.00E+00**	**0.00E+00**
	Std	**0.00E+00**	**0.00E+00**	3.50E−14	**0.00E+00**	**0.00E+00**
***F*6**	Mean	6.70E+00	7.04E+00	2.08E−03	1.12E+00	**1.36E−13**
	Std	1.17E+01	1.05E+01	6.29E−03	2.63E+00	**4.37E−13**
***F*7**	Mean	3.62E+01	1.03E+02	6.03E+01	1.06E+02	**3.53E+01**
	Std	1.43E+01	2.15E+01	2.85E+01	2.27E+01	**7.09E+00**
***F*8**	Mean	**2.08E+01**	2.09E+01	2.09E+01	2.09E+01	2.09E+01
	Std	9.27E−02	**4.26E−02**	4.27E−02	5.48E−02	5.64E−02
***F*9**	Mean	2.42E+01	2.70E+01	2.61E+01	2.89E+01	**2.16E+01**
	Std	3.73E+00	2.51E+00	4.07E+00	**1.45E+00**	2.68E+00
***F*10**	Mean	1.03E−01	1.24E−02	**3.82E−03**	1.39E−02	1.63E−02
	Std	5.94E−02	8.24E−03	**4.91E−03**	1.14E−02	1.49E−02
***F*11**	Mean	1.95E+01	8.06E+00	**3.23E+00**	4.53E+00	1.21E+01
	Std	7.68E+00	4.55E+00	**1.51E+00**	3.62E+00	3.20E+00
***F*12**	Mean	6.16E+01	1.31E+02	1.18E+02	1.62E+02	**5.38E+01**
	Std	1.57E+01	2.51E+01	2.90E+01	3.10E+01	**1.12E+01**

Table 3: Continued.

Function	Mean/std	MSRCS	ACS	ICS	CS	GFCS
***F*13**	Mean	$1.13E+02$	$1.58E+02$	$1.44E+02$	$1.80E+02$	$\mathbf{1.00E+02}$
	Std	$2.39E+01$	$3.03E+01$	$2.51E+01$	$\mathbf{2.30E+01}$	$2.62E+01$
***F*14**	Mean	$1.70E+03$	$1.28E+03$	$8.96E+02$	$7.41E+02$	$1.02E+03$
	Std	$4.70E+02$	$3.23E+02$	$3.04E+02$	$\mathbf{1.74E+02}$	$2.68E+02$
***F*15**	Mean	$3.89E+03$	$4.12E+03$	$3.96E+03$	$4.17E+03$	$\mathbf{3.46E+03}$
	Std	$5.76E+02$	$3.75E+02$	$5.93E+02$	$\mathbf{2.70E+02}$	$7.47E+02$
***F*16**	Mean	$\mathbf{6.84E-01}$	$9.30E-01$	$1.52E+00$	$1.42E+00$	$1.38E+00$
	Std	$4.53E-01$	$\mathbf{1.31E-01}$	$2.74E-01$	$2.40E-01$	$3.41E-01$
***F*17**	Mean	$6.51E+01$	$9.12E+01$	$5.85E+01$	$6.36E+01$	$\mathbf{5.29E+01}$
	Std	$1.20E+01$	$1.35E+01$	$\mathbf{7.11E+00}$	$9.61E+00$	$7.81E+00$
***F*18**	Mean	$8.70E+01$	$2.22E+02$	$1.69E+02$	$1.95E+02$	$\mathbf{8.25E+01}$
	Std	$\mathbf{1.62E+01}$	$2.78E+01$	$2.63E+01$	$2.94E+01$	$1.89E+01$
***F*19**	Mean	$3.45E+00$	$8.99E+00$	$5.71E+00$	$6.94E+00$	$\mathbf{2.29E+00}$
	Std	$1.16E+00$	$1.44E+00$	$2.08E+00$	$1.75E+00$	$\mathbf{4.77E-01}$
***F*20**	Mean	$1.12E+01$	$1.21E+01$	$1.20E+01$	$1.24E+01$	$\mathbf{1.11E+01}$
	Std	$5.63E-01$	$\mathbf{3.46E-01}$	$4.08E-01$	$3.88E-01$	$4.84E-01$
***F*21**	Mean	$3.01E+02$	$2.61E+02$	$\mathbf{2.05E+02}$	$2.44E+02$	$2.47E+02$
	Std	$9.54E+01$	$4.66E+01$	$\mathbf{3.94E+01}$	$4.78E+01$	$5.16E+01$
***F*22**	Mean	$1.63E+03$	$1.95E+03$	$1.16E+03$	$1.17E+03$	$\mathbf{1.04E+03}$
	Std	$8.40E+02$	$5.17E+02$	$4.25E+02$	$\mathbf{3.07E+02}$	$3.16E+02$
***F*23**	Mean	$4.04E+03$	$4.81E+03$	$5.07E+03$	$5.11E+03$	$\mathbf{3.68E+03}$
	Std	$4.98E+02$	$4.42E+02$	$5.54E+02$	$\mathbf{4.41E+02}$	$5.32E+02$
***F*24**	Mean	$2.65E+02$	$2.76E+02$	$2.67E+02$	$2.81E+02$	$\mathbf{2.59E+02}$
	Std	$8.67E+00$	$\mathbf{6.06E+00}$	$1.39E+01$	$1.07E+01$	$7.22E+00$
***F*25**	Mean	$2.75E+02$	$2.91E+02$	$2.83E+02$	$3.00E+02$	$\mathbf{2.73E+02}$
	Std	$1.07E+01$	$6.77E+00$	$1.08E+01$	$\mathbf{4.68E+00}$	$6.45E+00$
***F*26**	Mean	$2.16E+02$	$\mathbf{2.00E+02}$	$\mathbf{2.00E+02}$	$\mathbf{2.00E+02}$	$\mathbf{2.00E+02}$
	Std	$4.84E+01$	$2.41E-02$	$3.14E-03$	$1.43E-02$	$\mathbf{1.57E-04}$
***F*27**	Mean	$8.79E+02$	$1.05E+03$	$9.47E+02$	$1.03E+03$	$\mathbf{8.57E+02}$
	Std	$1.21E+02$	$\mathbf{5.27E+01}$	$2.04E+02$	$2.20E+02$	$1.42E+02$
***F*28**	Mean	$3.00E+02$	$\mathbf{3.00E+02}$	$\mathbf{3.00E+02}$	$\mathbf{3.00E+02}$	$\mathbf{3.00E+02}$
	Std	$1.04E-13$	$\mathbf{0.00E+00}$	$\mathbf{0.00E+00}$	$\mathbf{0.00E+00}$	$\mathbf{0.00E+00}$
Average ranking		2.18	3.43	2.50	3.36	1.46
$+/-/\approx$		22/2/4	19/3/6	19/4/5	18/4/6	

In the table, bold letters indicate the best results, "Mean" and "Std" represent the mean and standard error values, respectively.

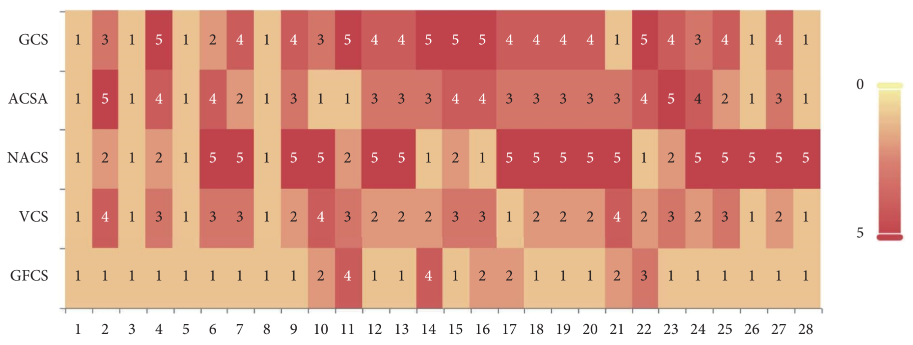

Figure 2: System-stacked histogram of ranking for GCS, ACSA, NACS, VCS, and GFCS on CEC2013 ($D = 30$).

$f4, f6, f7, f9, f12, f13, f15, f18, f19, f23$, and $f24$, where $f2$ and $f4$ are unimodal functions, $f6$, $f7$, $f9$, $f12$, $f13$, $f15$, $f18$, and $f19$ are multimodal functions, and $f23$ and $f24$ are combined functions. Based on the above algorithms and test functions, their convergence curves are plotted in Figure 4, where the horizontal axis indicates the number of iterations and the vertical axis indicates the error values.

From Figure 4, it can be observed that GFCS converges significantly faster than other competitors on $f2$, $f4$, $f6$, $f12$, $f19$, and $f24$. For the remaining functions, although the

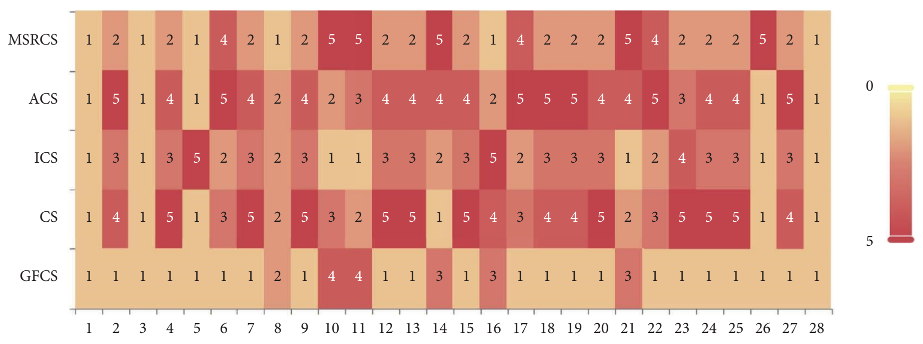

FIGURE 3: System-stacked histogram of ranking for MSRCS, ACS, ICS, CS, and GFCS on CEC2013 ($D = 30$).

convergence speed of GFCS is not the fastest compared with other algorithms, the best optimization results can still be achieved as the iteration progresses.

In addition, we can find that, for functions $f2$, $f4$, $f6$, $f13$, $f19$, and $f24$, GFCS can obtain a smoother descending curve during the convergence process because GFCS can quickly adjust the parameters Pa and step size α, to obtain faster convergence speed and more precise search results.

For other multimodal functions or mixed functions, these functions tend to make the algorithm fall into a local optimum. After the algorithm falls into the local optimum in a very early period, its optimum value is usually difficult to change greatly. For these functions, the advantages of GFCS are even more obvious. As $f9$, $f12$, $f15$, $f18$, and $f23$, Figure 4 shows that the optimization curve of GFCS falls rapidly again after a period of stagnation, jumping out of the current local optimum. At this time, the global feedback part and "re-fly" of GFCS play a role, making the algorithm jump out of the local optimum and find a better solution. Since the algorithm balances exploration and exploitation according to the optimization of each round of iterations, the convergence curve is not a continuous decline but presents a state of gradual optimization in stages to approach the global optimal solution.

According to the previous analysis, GFCS has a good convergence speed and optimization ability on various types of test functions and is able to jump out of local extremes. Therefore, we can conclude that GFCS achieves better performance than other algorithms when dealing with 30-dimensional problems.

5.3. Effect of Dimension Growth. As can be seen in the above sections, GFCS outperforms others in handling the 30-dimensional functions on the CEC2013 benchmark features. However, for a good algorithm, it should also be able to generate high-quality solutions to high-dimensional problems. To study the impact of dimensional growth on GFCS performance, we investigate the scalability of the algorithm on 28 test functions of CEC2013 with problem dimension size scaled from 30 to 50 In this section, we choose MSRCS, VCS, ACS, ECS, ICS, and GFCS, which performed better in the previous sections for comparison experiments, and the experimental results are shown in Table 4 and Figure 5.

From Table 4, GFCS wins on $f2$, $f7$, $f9$, $f12$, $f13$, $f19$, $f21$, and $f27$. Although it does not finish first on many other functions, it still achieves a relatively high ranking. Likewise, MSRCS is the champion on $f4$, $f8$, $f15$, $f16$, $f18$, $f20$, $f23$, $f24$, and $f25$. ICS is the $f10$ and $f11$ champion. VCS obtains the best results on $f14$, $f17$, and $f22$, CS obtains the best solution on $f5$, and ACS obtains the best solution on $f6$. Furthermore, all algorithms achieve the same result on $f3$. As can be seen at the bottom of the table, compared with other algorithms, GFCS outperforms at least 17 functions and has an average rank of 1.96. In addition, Figure 5 shows that GFCS still has the lightest overall color block and high rankings on most functions. More specifically, GFCS achieves the first- or second-best results on most functions. It still has a clear advantage over other algorithms. Based on all previous experimental analyses, we can conclude that although the advantage of GFCS mildly decreases when the dimensionality of the problem increases from 30 to 50 dimensions, GFCS is still the best algorithm to handle these benchmark functions combining the results of the previous experiments.

To visually compare the optimization of GFCS with other algorithms in the case of $D = 50$, we draw the optimization images of some functions in Figure 6. Figure 6 shows that, on the six functions $f2$, $f7$, $f12$, $f13$, $f19$, and $f21$, GFCS is significantly faster than the other competitors and is able to achieve better optimal values. On $f4$, $f9$, and $f27$, although GFCS is not significantly faster than the other competitors, it is relatively fast and can eventually achieve the best fitness value. In conclusion, the proposed GFCS has a better performance compared to the other competitors.

5.4. Comparison with Other Evolutionary Algorithms. To further confirm the superiority of GFCS, we select some other evolutionary algorithms for comparison. The differential evolution algorithm [2] (DE) and firefly algorithm [5] (FA) are widely studied and used swarm intelligence optimization algorithms. To further verify the performance of GFCS, DE, FA, and some variants of DE, ABC, and BSO are selected for comparison, the variants being NABC [38], ABCX [39], CUDE [40], and MSBSO [41].

In view of the fairness of the experiments, the population size = 25, the problem dimension = 30, and the number of evaluations = $1E6$ are set for these algorithms, and each test function independently runs 30 times. For some other

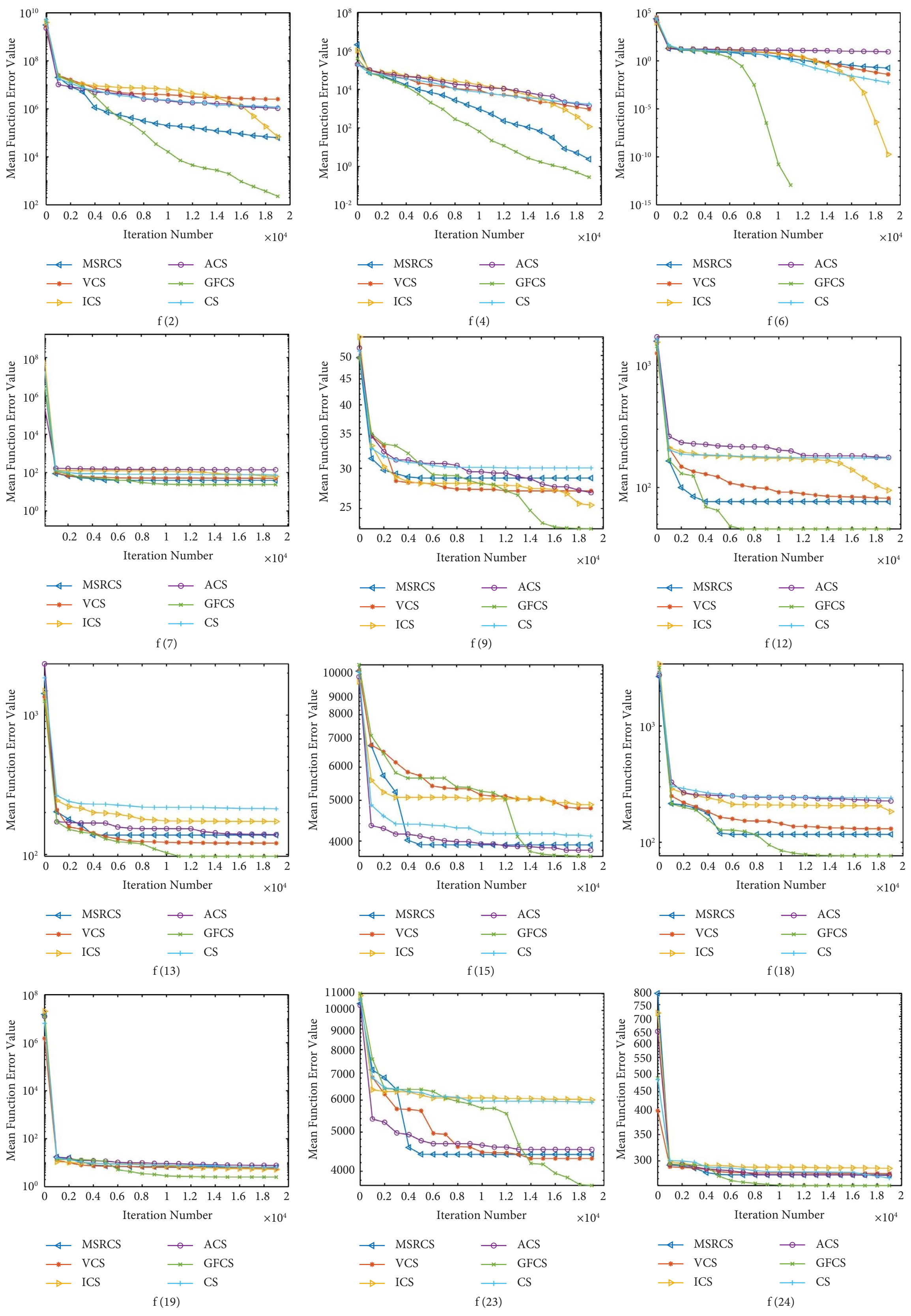

Figure 4: Convergence curves of MSRCS, VCS, ICS, ACS, CS, and GFCS on part functions of CEC2013 ($D = 30$).

Table 4: The test result of CEC2013 of MSRCS, VCA, CS, ACS, ICS, and GFCS on CEC2013 ($D = 50$).

Function	Mean/std	MSRCS	VCS	CS	ACS	ICS	GFCS
F1	Mean	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
	Std	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F2	Mean	1.99E+05	6.35E+06	3.58E+06	4.01E+06	8.87E+05	**1.37E+05**
	Std	8.87E+04	1.74E+06	9.78E+05	8.16E+05	2.02E+05	**2.48E+03**
F3	Mean	**1.00E+10**	**1.00E+10**	**1.00E+10**	**1.00E+10**	**1.00E+10**	**1.00E+10**
	Std	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F4	Mean	**6.40E+01**	5.29E+03	7.79E+03	5.34E+03	6.01E+02	1.07E+02
	Std	**4.70E+01**	1.16E+03	1.50E+03	2.22E+03	2.04E+02	4.91E+01
F5	Mean	1.14E−13	1.14E−13	5.68E−14	7.96E−14	**1.02E−13**	1.06E−13
	Std	**0.00E+00**	**0.00E+00**	5.99E−14	5.49E−14	3.60E−14	2.94E−14
F6	Mean	4.46E+01	4.40E+01	4.44E+01	**4.09E+01**	4.40E+01	4.38E+01
	Std	2.39E+00	**1.80E+00**	1.70E+01	1.02E+01	**1.80E+00**	1.47E+00
F7	Mean	7.62E+01	1.05E+02	1.23E+02	1.14E+02	9.54E+01	**7.53E+01**
	Std	1.75E+01	1.59E+01	1.63E+01	1.72E+01	2.74E+01	**1.26E+01**
F8	Mean	**2.10E+01**	2.11E+01	2.11E+01	2.11E+01	2.11E+01	2.11E+01
	Std	6.27E−02	2.95E−02	**2.22E−02**	2.84E−02	4.96E−02	2.36E−02
F9	Mean	5.12E+01	5.68E+01	5.81E+01	5.49E+01	5.64E+01	**4.94E+01**
	Std	6.36E+00	**1.47E+00**	2.10E+00	3.22E+00	3.49E+00	3.98E+00
F10	Mean	1.10E−01	4.56E−2	5.75E−02	5.30E−02	**2.69E−02**	6.03E−02
	Std	6.13E−02	2.65E−02	3.73E−02	**1.98E−02**	3.24E−02	3.12E−02
F11	Mean	6.83E+01	4.29E+01	3.22E+01	5.43E+01	**3.13E+01**	5.01E+01
	Std	1.47E+01	1.25E+01	7.68E+00	1.72E+01	**7.54E+00**	9.25E+00
F12	Mean	1.80E+02	2.03E+02	3.99E+02	3.70E+02	3.55E+02	**1.37E+02**
	Std	3.09E+01	2.79E+01	6.07E+01	6.24E+01	7.74E+01	**2.30E+01**
F13	Mean	2.65E+02	2.94E+02	4.73E+02	4.08E+02	4.18E+02	**2.50E+02**
	Std	3.92E+01	3.97E+01	8.72E+01	**3.91E+01**	4.27E+01	4.04E+01
F14	Mean	3.49E+03	**2.06E+03**	2.32E+03	4.48E+03	2.39E+03	2.49E+03
	Std	5.35E+02	7.55E+02	4.70E+02	6.53E+02	5.51E+02	**4.00E+02**
F15	Mean	**7.91E+03**	9.48E+03	8.51E+03	8.48E+03	9.19E+03	8.19E+03
	Std	7.64E+02	5.39E+02	4.79E+02	**3.52E+02**	4.16E+02	1.32E+03
F16	Mean	**9.56E−01**	2.24E+00	2.36E+00	1.70E+00	2.51E+00	2.25E+00
	Std	4.04E−01	2.04E−01	2.14E−01	**1.75E−01**	2.12E−01	4.81E−01
F17	Mean	1.55E+02	**1.23E+02**	1.96E+02	3.02E+02	1.50E+02	1.27E+02
	Std	3.73E+01	**1.20E+01**	3.01E+01	3.98E+01	2.81E+01	1.87E+01
F18	Mean	**1.85E+02**	3.20E+02	5.04E+02	6.05E+02	4.21E+02	2.99E+02
	Std	**2.49E+01**	3.16E+01	8.35E+01	6.32E+01	8.50E+01	8.86E+01
F19	Mean	8.36E+00	1.17E+01	2.82E+01	3.01E+01	1.52E+01	**5.67E+00**
	Std	**1.03E+00**	4.43E+00	7.78E+00	3.14E+00	3.25E+00	1.39E+00
F20	Mean	**2.06E+01**	2.18E+01	2.24E+01	2.19E+01	2.20E+01	2.09E+01
	Std	8.43E−01	5.08E−01	7.66E−01	**4.01E−01**	7.31E−01	5.90E−01
F21	Mean	9.16E+02	6.02E+02	6.04E+02	5.52E+02	4.48E+02	**3.46E+02**
	Std	**2.89E+02**	4.36E+02	4.38E+02	4.38E+02	4.07E+02	3.09E+02
F22	Mean	3.90E+03	**1.98E+03**	3.76E+03	5.27E+03	3.11E+03	2.65E+03
	Std	9.09E+02	**8.13E+02**	8.16E+02	1.06E+03	8.90E+02	1.12E+03
F23	Mean	**7.91E+03**	1.06E+04	1.04E+04	1.00E+04	1.10E+04	9.12E+03
	Std	1.06E+03	6.87E+02	**5.56E+02**	8.50E+02	8.63E+02	1.54E+03
F24	Mean	**3.08E+02**	3.52E+02	3.68E+02	3.49E+02	3.39E+02	3.21E+02
	Std	1.76E+01	**9.07E+00**	1.37E+01	1.68E+01	2.62E+01	1.51E+01
F25	Mean	**3.36E+02**	3.76E+02	3.98E+02	3.82E+02	3.80E+02	3.65E+02
	Std	1.22E+01	3.41E+00	**6.96E+00**	1.40E+01	1.28E+01	1.17E+01
F26	Mean	2.48E+02	2.25E+02	**2.00E+02**	2.01E+02	**2.00E+02**	**2.00E+02**
	Std	1.02E+02	7.71E+01	1.09E−01	1.76E−01	2.49E−02	**3.17E−03**
F27	Mean	1.55E+03	1.82E+03	1.92E+03	1.79E+03	1.78E+03	**1.54E+03**
	Std	1.59E+02	**2.96E+01**	4.95E+01	8.82E+01	2.19E+02	9.50E+01

Table 4: Continued.

Function	Mean/std	MSRCS	VCS	CS	ACS	ICS	GFCS
F28	Mean	7.21E + 02	**4.00E + 02**	7.30E + 02	1.04E + 03	**4.00E + 02**	**4.00E + 02**
	Std	1.02E + 03	**0.00E + 00**	1.04E + 03	1.35E + 03	1.33E − 13	**0.00E + 00**
Average ranking		2.79	3.29	4.25	3.93	3.11	1.96
+/ − / ≈		17/2/9	18/3/7	20/3/5	21/3/4	18/6/4	

In the table, bold letters indicate the best results, "Mean" and "Std" represent the mean and standard error values, respectively.

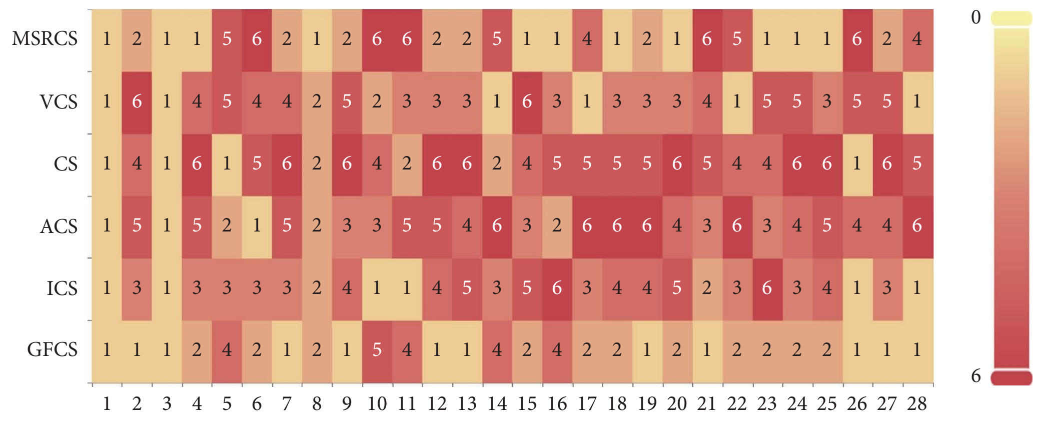

Figure 5: System-stacked histogram of ranking for MSRCS, VCS, CS, ACS, ICS, and GFCS on CEC2013 ($D = 50$).

parameters of the NABC, ABCX, CUDE, and MSBSO, we follow the settings in the literature, and the parameters of GFCS are consistent with the previous tests. The experimental results are shown in Table 5, and the best result is shown in **bold**.

According to Table 5, GFCS finds the best values on $f1$, $f2, f5, f6, f10, f12$, and $f26$. Likewise, NABC behaves well on $f11$, $f15$, $f16$, $f19$, $f25$, and $f26$ and FA provides the best solutions on $f8$ and $f9$, while does not yield optimal results for any function. In addition, ABCX performs best on $f14$, $f17, f21, f22, f23, f27$, and $f28$, CUDE finds the best values on $f3$ and $f4$, and MSBSO provides the best solutions on $f1$, $f5$, $f7$, $f12$, $f13$, $f18$, $f20$, and $f24$. In terms of average ranking results, GFCS generates an average rank value of 3.00, and ranks first, followed by the MSBSO algorithm with a ranking of 3.03, CUDE with a ranking of 3.43, NABC with a ranking of 3.71, and ABCX with a ranking of 4.25, respectively. The results of the average rank obtained by the Friedman test show that GFCS is still competitive with other swarm intelligence algorithms. In addition, it can be seen from the bottom of Table 5 that, compared with these algorithms, GFCS surpasses the other algorithms in most functions, which show that GFCS still has a relatively large advantage compared with other evolutionary algorithms.

Figure 7 shows that we plotted a superimposed histogram based on the ranking statistics to better visualize the ranking of the results in Table 5. Figure 7 shows that GFCS has lighter overall color block rankings and ranks in the top three on 19 functions. Except for the eighth ranking of GFCS on function 3, GFCS does not achieve any other sixth or seventh ranking. By comparing the ranking with other algorithms, we can see that GFCS is still competitive.

To further verify the performance of GFCS, we select some other classical evolutionary algorithms for comparison. The genetic algorithm [42], GA, particle swarm optimization [3] (PSO), ant colony algorithm [43] (ACO), artificial bee colony algorithm [4] (ABC), and brain storm optimization algorithm [44] (BSO) are selected for comparison.

In view of the fairness of the experiments, the population size $N = 25$, the problem dimension $D = 30$, and the number of evaluations = $1E6$ are set for these algorithms, and each test function independently runs 30 times. For GA, the probability of crossover is set to 1, and the probability of mutation is set to 0.01. For PSO, we set the personal learning coefficient = 1.5 and the global learning coefficient = 2.0. For ACO, the evaporation rate of pheromone is set to 0.1. Moreover, for ABC, the parameter limit is set to $(0.6 * \mathrm{N}) * D$. For BSO, the number of clusters is set to five. For other parameters of the GA, PSO, ACO, ABC, and BSO, we followed the settings in the literature, and the parameters of GFCS are consistent with the previous tests. The experimental results are shown in Table 5, and the best result is shown in bold.

According to Table 6, GFCS finds the best values on all functions except $f3$, $f8$, $f10$, $f15$, and $f16$. In terms of the average ranking results, GFCS produces an average ranking value of 1.39, which has a greater advantage over other SI algorithms. In addition, it can be seen from the bottom of Table 6 that GFCS beats these classical algorithms on most functions, which shows that the GFCS algorithm has a greater advantage over them.

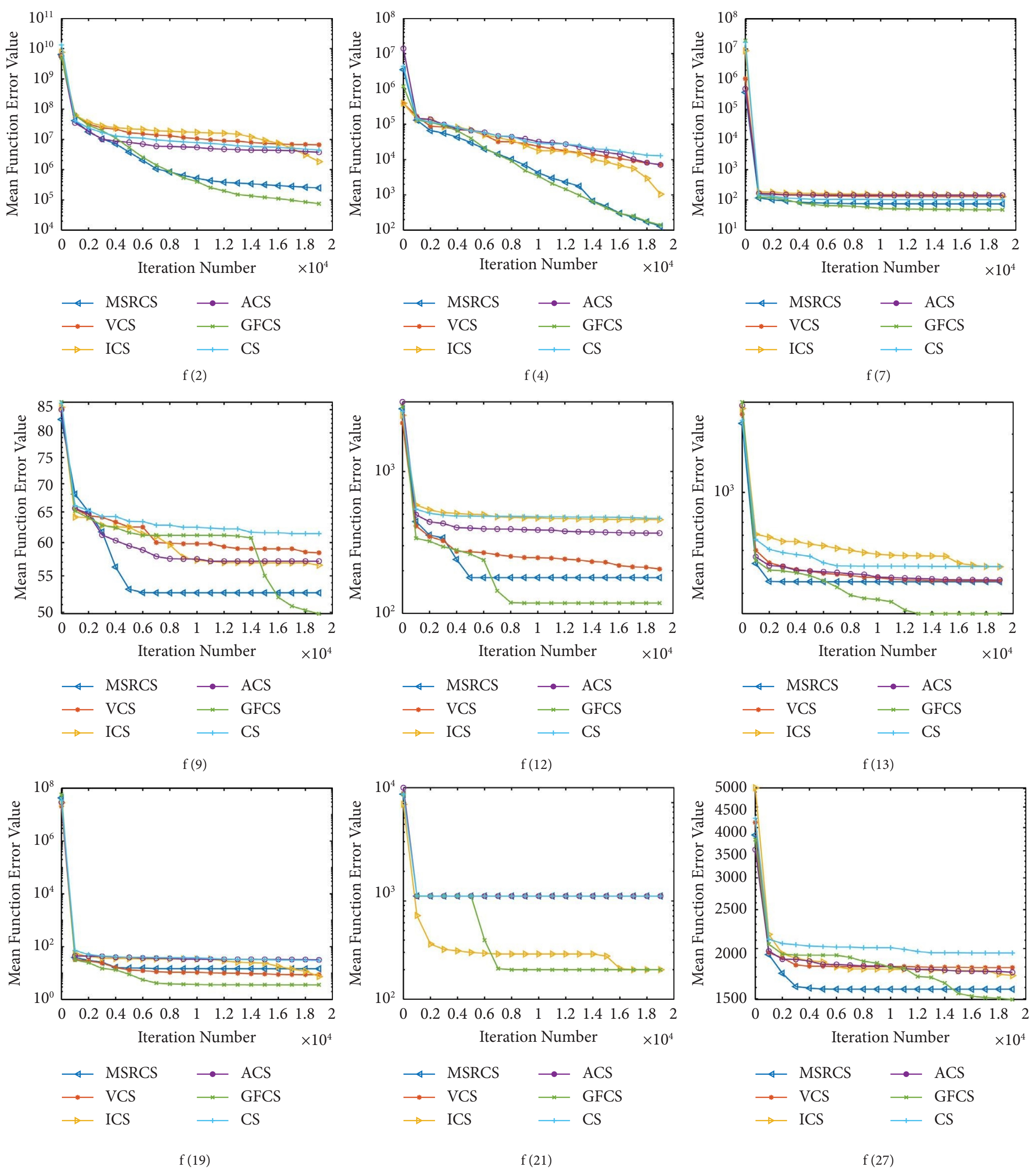

Figure 6: Convergence curves of MSRCS, VCS, ICS, ACS, CS, and GFCS on part functions of CEC2013 ($D = 50$).

We draw a stacked histogram based on the ranking statistics to better visualize the ranking of the results in Table 6, as detailed in Figure 8. Figure 8 shows that GFCS has lighter overall color blocks and ranks first on 23 functions. Moreover, GFCS does not achieve any other fifth or sixth ranking except the sixth ranking on function 3. By comparing the rankings with these SI algorithms, we can conclude that GFCS has a clear superiority in searching for the global optimum.

5.5. Comparison of Calculation Time. To demonstrate the effectiveness of GFCS in terms of running time, we calculate the time of running the 28 test functions of the CEC2013 test set with GFCS and the original CS algorithm in different dimensions. Among them, the dimensions are set to 30 and 50. For the other parameters, the upper limit of the number of iterations is set to 20,000, and the population size is set to 25. To exclude measurement chance, each function on the CEC2013 test set runs 10 times independently, and the total run time is calculated. The experimental results are shown in Table 7.

Table 7 shows that there is no significant difference in runtime between CS and GFCS for either dimension $D = 30$ or $D = 50$. This again validates the complexity analysis of CS and GFCS in Section 4.4, where there is no significant difference between GFCS and CS in terms of time complexity.

Table 5: The test result of CEC2013 of DE, FA, NABC, ABCX, CUDE, MSBSO, and GFCS on CEC2013 ($D = 30$).

Function	Mean/std	DE	FA	NABC	ABCX	CUDE	MSBSO	GFCS
***F*1**	Mean	1.06E − 13	6.82E − 13	4.09E − 13	2.37E − 13	1.82E − 13	**0.00E + 00**	**0.00E + 00**
	Std	1.17E − 13	1.61E − 13	9.59E − 14	7.15E − 14	1.44E − 13	**0.00E + 00**	**0.00E + 00**
***F*2**	Mean	4.10E + 07	4.55E + 04	2.00E + 06	1.76E + 07	8.16E + 03	5.74E + 04	**1.96E + 03**
	Std	9.32E + 06	3.21E + 04	5.24E + 05	3.01E + 06	4.97E + 03	1.67E + 04	**2.48E + 03**
***F*3**	Mean	3.38E + 07	1.82E + 08	2.74E + 07	8.99E + 08	**3.69E + 05**	1.04E + 06	1.00E + 10
	Std	3.93E + 07	3.07E + 08	1.76E + 07	9.89E + 07	6.62E + 05	1.18E + 06	**0.00E + 00**
***F*4**	Mean	6.85E + 03	3.34E + 02	5.19E + 04	6.12E + 04	**6.43E − 10**	9.19E − 03	3.97E − 01
	Std	1.14E + 03	3.73E + 02	5.34E + 03	6.03E + 03	**1.64E − 09**	1.21E − 02	4.98E − 01
***F*5**	Mean	1.14E − 13	2.27E − 12	6.03E − 13	2.72E − 13	2.73E − 13	**0.00E + 00**	**0.00E + 00**
	Std	**0.00E + 00**	1.73E − 12	9.36E − 14	6.72E − 14	5.87E − 14	**0.00E + 00**	**0.00E + 00**
***F*6**	Mean	1.43E + 01	1.38E + 01	1.46E + 01	1.49E + 01	3.32E + 00	1.09E + 01	**1.36E − 13**
	Std	4.53E + 00	1.10E − 11	7.04E − 01	4.69E + 00	6.15E + 00	1.08E + 01	**4.37E − 13**
***F*7**	Mean	3.89E + 01	7.13E + 01	6.01E + 01	8.43E + 01	3.97E + 01	**1.07E + 01**	3.53E + 01
	Std	1.16E + 01	1.30E + 01	7.26E + 00	9.02E + 00	1.09E + 01	**7.00E + 00**	7.09E + 00
***F*8**	Mean	2.09E + 01	**2.08E + 01**	2.09E + 01	2.09E + 01	2.09E + 01	2.09E + 01	2.09E + 01
	Std	**2.87E − 02**	5.41E − 02	3.16E − 02	5.28E − 02	4.48E − 02	5.44E − 02	5.64E − 02
***F*9**	Mean	3.16E + 01	**2.07E + 01**	2.59E + 01	2.71E + 01	2.23E + 01	2.45E + 01	2.16E + 01
	Std	**1.34E + 00**	2.49E + 00	3.79E + 00	1.99E + 00	3.99E + 00	5.94E+00	2.68E + 00
***F*10**	Mean	5.07E − 01	1.85E − 01	7.00E − 02	1.41E + 00	1.21E − 01	4.66E − 02	**1.63E − 02**
	Std	6.21E − 01	8.20E − 02	7.74E − 02	2.20E − 01	5.24E − 02	2.68E − 02	**1.49E − 02**
***F*11**	Mean	3.32E − 01	7.32E + 01	**5.68E − 14**	6.19E − 14	9.05E + 00	2.31E + 01	1.21E + 01
	Std	6.14E − 01	2.26E + 01	**0.00E + 00**	1.65E − 14	7.66E + 00	4.49E + 00	3.20E + 00
***F*12**	Mean	1.50E + 02	8.58E + 01	1.42E + 02	2.29E + 02	6.83E + 01	**3.84E + 01**	5.38E + 01
	Std	1.13E + 01	2.24E + 01	2.64E + 01	1.59E + 01	1.62E + 01	**9.20E + 00**	1.12E + 01
***F*13**	Mean	1.58E + 02	1.54E + 02	2.01E + 02	2.00E + 02	1.12E + 02	**9.31E + 01**	1.00E + 02
	Std	1.51E + 01	4.33E + 01	3.00E + 01	**1.04E + 01**	2.43E + 01	1.69E + 01	2.62E + 01
***F*14**	Mean	1.62E + 02	2.20E + 03	**3.27E − 01**	4.03E − 02	1.69E + 01	4.37E + 02	1.02E + 03
	Std	9.94E + 01	6.46E + 02	**3.32E − 01**	4.23E − 02	3.78E + 01	2.15E + 02	2.68E + 02
***F*15**	Mean	6.89E + 03	3.62E + 03	**2.99E + 03**	4.21E + 03	3.84E + 03	3.67E + 03	3.46E + 03
	Std	3.50E + 02	6.47E + 02	8.31E + 02	**2.80E + 02**	5.55E + 02	6.38E + 02	7.47E + 02
***F*16**	Mean	2.31E + 00	1.89E + 00	**6.69E − 01**	1.62E + 00	9.01E − 01	2.09E + 00	1.38E + 00
	Std	3.01E − 01	**2.28E − 01**	2.35E − 01	2.77E − 01	5.00E − 01	3.82E − 01	3.41E − 01
***F*17**	Mean	3.07E + 01	8.09E + 01	3.04E + 01	**3.03E + 01**	3.18E + 01	5.69E + 01	5.29E + 01
	Std	3.11E − 01	1.13E + 01	**5.09E − 03**	5.88E − 03	2.66E + 00	8.30E + 00	7.81E + 00
***F*18**	Mean	2.05E + 02	8.38E + 01	1.97E + 02	1.52E + 02	8.59E + 01	**6.29E + 01**	8.25E + 01
	Std	**8.95E + 00**	5.06E + 01	3.36E + 01	1.56E + 01	2.95E + 01	9.01E + 00	1.89E + 01
***F*19**	Mean	4.27E + 00	3.40E + 00	**5.14E − 02**	3.28E − 01	3.33E + 00	3.11E + 00	2.29E + 00
	Std	3.72E − 01	5.52E − 01	**4.54E − 02**	2.23E − 01	2.19E + 00	9.72E − 01	4.77E − 01
***F*20**	Mean	1.24E + 01	1.50E + 01	1.47E+01	1.37E + 01	1.01E + 01	**9.71E + 00**	1.11E + 01
	Std	2.59E − 01	**0.00E + 00**	7.24E − 01	3.98E − 01	8.52E − 01	7.59E − 01	4.84E − 01
***F*21**	Mean	2.96E + 02	3.57E + 02	3.76E+02	**2.03E + 02**	3.47E + 02	3.04E + 02	2.47E + 02
	Std	**2.79E + 01**	7.86E + 01	9.17E + 01	4.42E + 01	8.82E + 01	5.81E + 01	5.16E + 01
***F*22**	Mean	1.50E + 02	2.23E + 03	8.46E + 01	**5.67E + 01**	1.23E + 02	4.39E + 02	1.04E + 03
	Std	3.41E + 01	8.12E + 02	4.13E + 01	4.83E + 01	**1.13E + 01**	1.45E + 02	3.16E + 02
***F*23**	Mean	7.09E + 03	3.89E + 03	4.39E + 03	**5.01E + 01**	3.87E + 03	4.16E + 03	3.68E + 03
	Std	3.84E + 02	8.23E + 02	7.12E + 02	**3.58E + 02**	6.47E + 02	6.81E + 02	5.32E + 02
***F*24**	Mean	2.76E + 02	2.55E + 02	2.31E + 02	2.63E + 02	2.31E + 02	**2.19E + 02**	2.59E + 02
	Std	4.51E + 00	1.00E + 01	**2.66E + 00**	7.45E + 00	3.95E + 00	7.63E + 00	7.22E + 00
***F*25**	Mean	2.83E + 02	2.70E + 02	**2.59E + 02**	2.92E + 02	2.72E + 02	2.62E + 02	2.75E + 02
	Std	4.58E + 00	7.87E + 00	4.13E + 01	**4.23E + 00**	2.04E + 01	8.13E + 00	6.45E + 00
***F*26**	Mean	2.16E + 02	**2.00E + 02**	**2.00E + 02**	2.01E + 02	2.12E + 02	2.11E + 02	**2.00E + 02**
	Std	4.49E + 01	4.20E − 03	1.66E − 02	1.54E − 01	3.79E + 01	3.56E + 01	**1.57E − 04**
***F*27**	Mean	1.08E + 03	7.65E + 02	5.49E + 02	**4.02E + 02**	6.41E + 02	6.07E + 02	8.57E + 02
	Std	3.50E + 01	5.78E + 01	1.93E + 02	1.44E + 00	**1.18E + 02**	1.75E + 02	1.42E + 02
***F*28**	Mean	3.00E + 02	5.21E + 02	3.00E + 02	**2.83E + 02**	4.12E + 02	3.00E + 02	3.00E + 02
	Std	1.52E − 13	4.93E + 02	2.48E − 13	5.69E + 01	3.55E + 02	2.14E − 13	**0.00E + 00**
Average ranking		5.07	4.79	3.71	4.25	3.43	3.03	3.00
+/ − / ≈		21/2/5	20/2/6	14/3/11	18/1/9	16/1/11	12/4/12	

In the table, bold letters indicate the best results, "Mean" and "Std" represent the mean and standard error values, respectively.

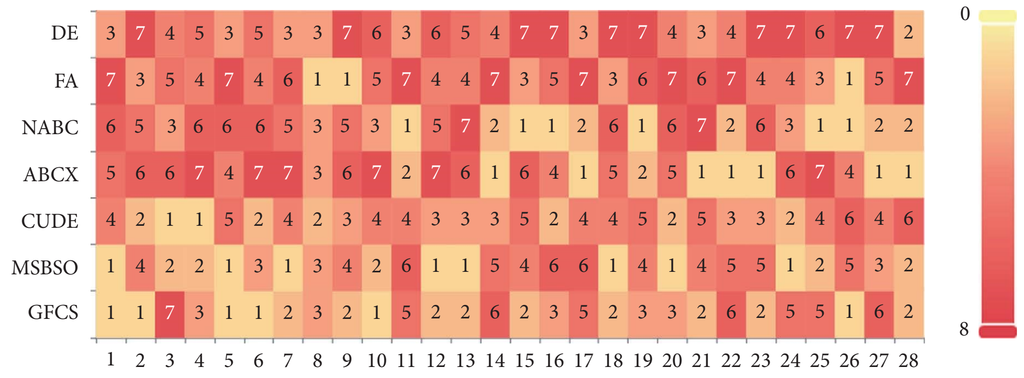

Figure 7: System-stacked histogram of ranking for DE, FA, NABC, ABCX, CUDE, MSBSO, and GFCS on CEC2013 ($D = 30$).

Table 6: The test result of CEC2013 of GA, ACO, ABC, BSO, and GFCS on CEC2013 ($D = 30$).

Function	Mean/ std	GA	PSO	ACO	ABC	BSO	GFCS
F1	Mean	1.55*E* − 07	4.00*E* − 05	1.40*E* − 02	5.23*E* − 13	1.32*E* − 02	**0.00*E* + 00**
	Std	1.14*E* − 07	6.05*E* − 05	1.77*E* − 03	1.53*E* − 13	1.23*E* + 00	**0.00*E* + 00**
F2	Mean	4.08*E* + 06	1.64*E* + 06	1.61*E* + 06	1.12*E* + 08	8.92*E* + 05	**1.96*E* + 03**
	Std	3.18*E* + 06	9.48*E* + 05	7.01*E* + 05	1.43*E* + 07	3.66*E* + 05	**2.48*E* + 03**
F3	Mean	8.21*E* + 09	4.04*E* + 08	3.68*E* + 08	**9.40*E* + 01**	1.29*E* + 08	1.00*E* + 10
	Std	4.11*E* + 09	5.77*E* + 08	4.76*E* + 08	**6.61*E* + 01**	2.38*E* + 08	0.00*E* + 00
F4	Mean	8.10*E* + 04	4.38*E* + 00	1.07*E* + 03	6.52*E* + 04	1.10*E* + 03	**3.97*E* − 01**
	Std	2.52*E* + 04	2.61*E* + 00	3.42*E* + 02	1.31*E* + 04	9.76*E* + 02	**4.98*E* − 01**
F5	Mean	3.20*E* + 02	2.00*E* − 05	5.71*E* + 04	1.09*E* − 03	5.33*E* − 02	**0.00*E* + 00**
	Std	2.25*E* + 02	6.12*E* − 05	1.43*E* + 04	2.96*E* − 04	1.38*E* − 02	**0.00*E* + 00**
F6	Mean	8.36*E* + 01	5.72*E* + 01	5.05*E* + 00	1.39*E* + 01	3.12*E* + 01	**1.36*E* − 13**
	Std	2.04*E* + 01	2.73*E* + 01	2.66*E* + 00	6.96*E* − 02	2.63*E* + 01	**4.37*E* − 13**
F7	Mean	3.28*E* + 04	1.01*E* + 02	1.15*E* + 02	7.09*E* + 01	1.06*E* + 02	**3.53*E* + 01**
	Std	5.29*E* + 04	1.52*E* + 01	1.80*E* + 01	**3.05*E* + 00**	5.27*E* + 01	7.09*E* + 00
F8	Mean	2.10*E* + 01	**2.09*E* + 01**	**2.09*E* + 01**	**2.09*E* + 01**	**2.09*E* + 01**	**2.09*E* + 01**
	Std	6.37*E* − 02	5.51*E* − 02	6.08*E* − 02	**3.08*E* − 02**	1.48*E* − 01	5.64*E* − 02
F9	Mean	3.63*E* + 01	2.76*E* + 01	2.31*E* + 01	3.85*E* + 01	3.89*E* + 01	**2.16*E* + 01**
	Std	4.05*E* + 00	2.31*E* + 00	1.07*E* + 00	**9.03*E* − 01**	2.45*E* + 00	2.68*E* + 00
F10	Mean	5.93*E* + 00	2.63*E* + 00	2.42*E* − 01	**1.50*E* − 03**	1.03*E* + 00	1.63*E* − 02
	Std	5.65*E* + 00	2.61*E* + 00	3.00*E* − 02	**3.11*E* − 03**	1.14*E* − 01	1.49*E* − 02
F11	Mean	3.90*E* + 02	1.18*E* + 02	2.88*E* + 02	1.87*E* + 02	5.53*E* + 02	**1.21*E* + 01**
	Std	1.07*E* + 02	3.02*E* + 01	8.09*E* + 01	1.13*E* + 01	1.62*E* + 02	**3.20*E* + 00**
F12	Mean	3.07*E* + 02	1.71*E* + 02	2.51*E* + 02	1.98*E* + 02	6.62*E* + 02	**5.38*E* + 01**
	Std	4.81*E* + 01	5.48*E* + 01	6.52*E* + 01	1.47*E* + 01	9.10*E* + 01	**1.12*E* + 01**
F13	Mean	4.02*E* + 02	2.62*E* + 02	3.96*E* + 02	2.01*E* + 02	6.80*E* + 02	**1.00*E* + 02**
	Std	3.67*E* + 01	6.05*E* + 01	8.17*E* + 01	**1.32*E* + 01**	8.30*E* + 01	2.62*E* + 01
F14	Mean	3.98*E* + 03	3.32*E* + 03	2.76*E* + 03	6.50*E* + 03	4.41*E* + 03	**1.02*E* + 03**
	Std	3.46*E* + 02	7.04*E* + 02	5.81*E* + 02	2.86*E* + 02	6.74*E* + 02	**2.68*E* + 02**
F15	Mean	4.49*E* + 03	4.39*E* + 03	**2.94*E* + 03**	7.25*E* + 03	4.17*E* + 03	3.46*E* + 03
	Std	6.58*E* + 02	8.03*E* + 02	3.62*E* + 02	**3.03*E* + 02**	7.30*E* + 02	7.47*E* + 02
F16	Mean	3.11*E* + 00	1.16*E* + 00	**1.84*E* − 01**	2.18*E* + 00	3.42*E* − 01	1.38*E* + 00
	Std	9.06*E* − 01	5.13*E* − 01	**4.21*E* − 02**	2.95*E* − 01	1.40*E* − 01	3.41*E* − 01
F17	Mean	4.70*E* + 02	1.29*E* + 02	4.98*E* + 02	2.32*E* + 02	5.36*E* + 02	**5.29*E* + 01**
	Std	1.11*E* + 02	3.94*E* + 01	9.58*E* + 01	1.46*E* + 01	9.61*E* + 00	**7.81*E* + 00**
F18	Mean	4.14*E* + 02	1.37*E* + 02	4.13*E* + 02	2.43*E* + 02	4.95*E* + 02	**8.25*E* + 01**
	Std	1.20*E* + 02	4.63*E* + 01	4.94*E* + 01	**1.60*E* + 01**	7.94*E* + 01	1.89*E* + 01
F19	Mean	3.65*E* + 02	4.97*E* + 00	1.91*E* + 01	1.88*E* + 01	1.14*E* + 01	**2.29*E* + 00**
	Std	6.07*E* + 01	2.09*E* + 00	6.13*E* + 00	6.31*E* − 01	2.75*E* + 00	**4.77*E* − 01**
F20	Mean	1.46*E* + 01	1.44*E* + 01	1.45*E* + 01	1.28*E* + 01	1.44*E* + 01	**1.11*E* + 01**
	Std	2.22*E* − 01	5.77*E* − 01	5.33*E* − 03	**1.46*E* − 01**	3.88*E* − 02	4.84*E* − 01
F21	Mean	2.80*E* + 02	3.42*E* + 02	3.02*E* + 02	2.80*E* + 02	3.74*E* + 02	**2.47*E* + 02**
	Std	4.47*E* + 01	1.23*E* + 02	**7.98*E* − 02**	4.22*E* + 01	1.08*E* + 02	5.16*E* + 01
F22	Mean	4.84*E* + 03	4.10*E* + 03	3.84*E* + 03	7.08*E* + 03	5.17*E* + 03	**1.04*E* + 03**
	Std	1.13*E* + 03	7.08*E* + 02	4.11*E* + 02	**2.48*E* + 02**	1.07*E* + 03	3.16*E* + 02

Table 6: Continued.

Function	Mean/ std	GA	PSO	ACO	ABC	BSO	GFCS
***F*23**	Mean	5.62*E* + 03	5.25*E* + 03	3.74*E* + 03	7.36*E* + 03	5.11*E* + 03	**3.68*E* + 03**
	Std	9.48*E* + 02	1.01*E* + 03	6.66*E* + 02	**3.11*E* + 02**	7.41*E* + 02	5.32*E* + 02
***F*24**	Mean	3.47*E* + 02	2.75*E* + 02	2.82*E* + 02	2.93*E* + 02	3.61*E* + 02	**2.59*E* + 02**
	Std	1.29*E* + 01	1.22*E* + 01	**3.07*E* + 00**	3.24*E* + 00	2.17*E* + 02	7.22*E* + 00
***F*25**	Mean	3.78*E* + 02	3.00*E* + 02	3.05*E* + 02	3.02*E* + 02	3.54*E* + 02	**2.73*E* + 02**
	Std	1.43*E* + 01	1.30*E* + 01	5.82*E* + 00	**1.92*E* + 00**	2.21*E* + 01	6.45*E* + 00
***F*26**	Mean	3.92*E* + 02	3.29*E* + 02	**2.00*E* + 02**	2.21*E* + 02	2.20*E* + 02	**2.00*E* + 02**
	Std	7.58*E* + 00	6.79*E* + 01	7.68*E* − 03	3.45*E* + 00	5.13*E* + 01	**1.57*E* − 04**
***F*27**	Mean	1.32*E* + 03	1.03*E* + 03	8.93*E* + 02	1.26*E* + 03	1.32*E* + 03	**8.57*E* + 02**
	Std	6.71*E* + 01	8.74*E* + 01	6.50*E* + 01	**2.06*E* + 01**	8.20*E* + 01	1.42*E* + 02
***F*28**	Mean	3.76*E* + 03	4.04*E* + 02	9.15*E* + 02	**3.00*E* + 02**	4.70*E* + 03	**3.00*E* + 02**
	Std	5.71*E* + 02	3.94*E* + 02	6.98*E* + 02	7.90*E* − 06	6.40*E* + 02	**0.00*E* + 00**
Average ranking		5.14	3.14	3.25	3.46	4.14	1.39
+/ − / ≈		27/0/1	25/1/2	23/1/4	25/1/2	25/1/2	

In the tables, bold letters indicate the best results, "Mean" and "Std" represent the mean and standard error values, respectively.

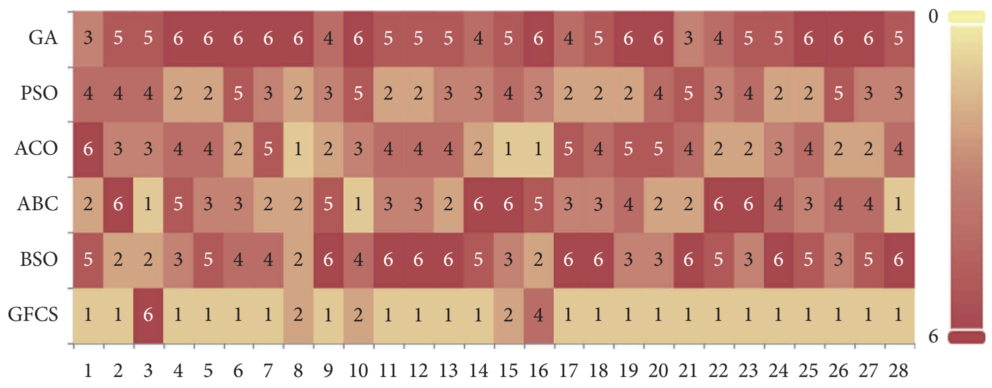

Figure 8: System-stacked histogram of ranking for GA, PSO, ACO, ABC, BSO, and GFCS on CEC2013 ($D = 30$).

Table 7: The runtime of CS and GFCS on CEC2013 (unit: seconds).

Dimension/algorithm	CS	GFCS
$D = 30$	5.483*E* + 03	5.443*E* + 03
$D = 50$	8.645*E* + 03	8.652*E* + 03

6. Conclusion

This article proposes a global feedback-based cuckoo search algorithm (GFCS). In GFCS, we introduce the concept of global feedback and the "re-fly" mechanism. In addition, we set new parameter formulas that change with the number of iteration rounds and are controlled by the dynamic global variables. To evaluate the performance of the GFCS algorithm, GFCS is compared with the other eight variants of the CS algorithm and several classical evolutionary algorithms and their variants. Based on the experimental results, the following conclusions can be drawn:

(i) GFCS algorithm adopts a global feedback strategy and the "re-fly" mechanism in the optimization search process. According to the evolution of the current generation, GFCS adjusts the parameters of the algorithm globally during the evolution process, effectively accelerates the algorithm's convergence speed, and enriches the population and the diversity of learning.

(ii) Compared with CS, some CS variants, and several SI algorithms in the experiment, the GFCS algorithm has faster convergence speed and better convergence accuracy.

(iii) As we compare in Sections 4.4 and 5.5, the time and space complexity of GFCS is comparable to that of the traditional CS algorithm, which means that the GFCS algorithm does not improve the complexity of the algorithm.

In the future, we intend to extend our current work in the following directions. Firstly, for the switching parameters, we will try to adjust the adaptive adjustment mechanism or introduce a multistrategy mechanism to further improve the search ability. Secondly, we will consider the application of GFCS to some other scientific problems, such as applying the algorithm to the field of machine learning or deep learning. Thirdly, we will discuss the application of the algorithm to some practical problems.

Data Availability

The datasets generated and/or analysed in this study can be obtained from the corresponding authors upon reasonable request.

Conflicts of Interest

The authors declare that they have no known conflicts of financial interest or personal relationships that could have influenced the work reported in this paper.

Acknowledgments

This work was supported by Tencent Technology (Shenzhen) Co., Ltd. and Natural Science Foundation of China under Grants 51575204 and 52175536.

References

[1] M. Dorigo, M. Birattari, and T. Stutzle, “Ant colony optimization,” *IEEE Computational Intelligence Magazine*, vol. 1, no. 4, pp. 28–39, 2006.

[2] R. Storn and K. Price, “Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces,” *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[3] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of the IEEE Icnn95-international Conference on Neural Networks*, Perth, WA, Australia, December, 1995.

[4] D. Karaboga and B. Basturk, “A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm,” *Journal of Global Optimization*, vol. 39, no. 3, pp. 459–471, Apr. 2007.

[5] X. S. Yang, “Firefly algorithm, stochastic test functions and design optimisation,” *International Journal of Bio-Inspired Computation*, vol. 2, no. 2, p. 78, 2010.

[6] X.-S. Yang and S. Deb, “Cuckoo search: recent advances and applications,” *Neural Computing & Applications*, vol. 24, no. 1, pp. 169–174, Jan. 2014.

[7] C. T. Brown, L. S. Liebovitch, and R. Glendon, “Lévy flights in dobe Ju/’hoansi foraging patterns,” *Human Ecology*, vol. 35, no. 1, pp. 129–138, Feb. 2007.

[8] R. N. Mantegna, “Fast, accurate algorithm for numerical simulation of levy stable stochastic processes,” *Physical Review A*, vol. 49, no. 5, pp. 4677–4683, 1994.

[9] E. Valian, S. Tavakoli, S. Mohanna, and A. Haghi, “Improved cuckoo search for reliability optimization problems,” *Computers & Industrial Engineering*, vol. 64, no. 1, pp. 459–468, Jan. 2013.

[10] M. Naik, M. R. Nath, and S. Sahany, “A new adaptive cuckoo search algorithm,” in *Proceedings of the IEEE International Conference on Recent Trends in Information Systems*, p. 5, Kolkata, India, September, 2015.

[11] P. Ong, “Adaptive cuckoo search algorithm for unconstrained optimization,” *The Scientific World Journal*, vol. 2014, Article ID 943403, 8 pages, 2014.

[12] L. Wang, Y. Yin, and Y. Zhong, “Cuckoo search with varied scaling factor,” *Frontiers of Computer Science*, vol. 9, no. 4, pp. 623–635, 2015.

[13] X. Li and M. Yin, “Modified cuckoo search algorithm with self adaptive parameter method,” *Information Sciences*, vol. 298, pp. 80–97, 2015.

[14] K. Huang, Y. Zhou, X. Wu, and Q. Luo, “A cuckoo search algorithm with elite opposition-based strategy,” *Journal of Intelligent Systems*, vol. 25, no. 4, pp. 567–593, 2016.

[15] M. A. Baset, Y. Zhou, and M. Ismail, “An improved cuckoo search algorithm for integer programming problems,” *International Journal of Computing Science and Mathematics*, vol. 9, no. 1, pp. 66–81, 2018.

[16] A. M. Kamoona, J. C. Patra, and A. Stojcevski, “An enhanced cuckoo search algorithm for solving optimization problems,” in *Proceedings of the 2018 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–6, Rio de Janeiro, Brazil, July. 2018.

[17] H. Zheng and Y. Zhou, “A novel cuckoo search optimization algorithm base on gauss distribution,” *Journal of Computational Information Systems*, vol. 8, no. 10, pp. 4193–4200, 2012.

[18] Z. He, H. Peng, and C. Deng, “A spark-based Gaussian barebones cuckoo search with dynamic parameter selection,” in *Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1220–1227, Wellington, New Zealand, June. 2019.

[19] H. Zheng and Y. Zhou, “A cooperative coevolutionary cuckoo search algorithm for optimization problem,” *Journal of Applied Mathematics*, vol. 2013, Article ID 912056, 9 pages, 2013.

[20] J. Cheng and L. Wang, “Cuckoo search algorithm with neighborhood attraction for numerical optimization,” *IEEE Access*, vol. 7, pp. 122261–122274, 2019.

[21] H. Rakhshani and A. Rahati, “Snap-drift cuckoo search: a novel cuckoo search optimization algorithm,” *Applied Soft Computing*, vol. 52, pp. 771–794, Mar. 2017.

[22] H. Peng, Z. Zeng, C. Deng, and Z. Wu, “Multi-strategy serial cuckoo search algorithm for global optimization,” *Knowledge-Based Systems*, vol. 214, Article ID 106729, 2021.

[23] H. Peng, Y. Li, and C. Deng, “Multi-strategy reconciliatory cuckoo search algorithm,” *Computer Engineering*, vol. 4, pp. 1–17, 2022.

[24] S. Gao, Y. Gao, Y. Zhang, and T. Li, “Adaptive cuckoo algorithm with multiple search strategies,” *Applied Soft Computing*, vol. 106, Article ID 107181, 2021.

[25] Y. Zhang, L. Wang, and Q. Wu, “Dynamic adaptation cuckoo search algorithm,” *Control and Decision*, vol. 29, no. 4, pp. 617–622, 2014.

[26] S. Walton, O. Hassan, K. Morgan, and M. Brown, “Modified cuckoo search: a new gradient free optimisation algorithm,” *Chaos, Solitons & Fractals*, vol. 44, no. 9, pp. 710–718, Sep. 2011.

[27] Q. Jin, L. Qi, B. Jiang, and Q. Wang, “Novel improved cuckoo search for PID controller design,” *Transactions of the Institute of Measurement and Control*, vol. 37, no. 6, pp. 721–731, Jul. 2015.

[28] S. Arora and P. Kaur, “Grayscale image enhancement using improved cuckoo search algorithm,” *Progress in Intelligent Computing Techniques: Theory, Practice, and Applications*, vol. 518, pp. 141–148, 2018.

[29] A. K. Bhateja, A. Bhateja, S. Chaudhury, and P. Saxena, “Cryptanalysis of Vigenere cipher using cuckoo search,” *Applied Soft Computing*, vol. 26, pp. 315–324, Jan. 2015.

[30] G. K. Shailaja and C. G. Rao, "Opposition intensity-based cuckoo search algorithm for data privacy preservation," *Journal of Intelligent Systems*, vol. 29, no. 1, pp. 1441–1452, 2019.

[31] J. Fan, W. Xu, Y. Huang, and R. Dinesh Jackson Samuel, "Application of chaos cuckoo search algorithm in computer vision technology," *Soft Computing*, vol. 25, no. 18, pp. 12373–12387, 2021.

[32] A. C. Pandey and D. S. Rajpoot, "Spam review detection using spiral cuckoo search clustering method," *Evolutionary Intelligence*, vol. 12, no. 2, pp. 147–164, 2019.

[33] R. Rautray, R. C. Balabantaray, R. Dash, and R. Dash, "CSMDSE-cuckoo search based multi document summary extractor: cuckoo search based summary extractor," *International Journal of Cognitive Informatics and Natural Intelligence*, vol. 13, no. 4, pp. 56–70, 2019.

[34] K. Sharma, S. Singh, and R. Doriya, "Optimized cuckoo search algorithm using tournament selection function for robot path planning," *International Journal of Advanced Robotic Systems*, vol. 18, no. 3, Article ID 172988142199613, 2021.

[35] D. Troxler, T. Hanne, and R. Dornberger, "A multi-threaded cuckoo search algorithm for the capacitated vehicle routing problem," in *Proceedings of the 2020 4th International Conference on Intelligent Systems*, pp. 105–110, Metaheuristics & Swarm Intelligence, Thimphu Bhutan, March. 2020.

[36] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," in *From Natural to Artificial Neural Computation*, J. Mira and F. Sandoval, Eds., vol. 930, pp. 195–201, Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.

[37] J. J. Liang, B. Y. Qu, and P. N. Suganthan, *Problem Definitions and Evaluation Criteria for the CEC 2013 Special Session on Real-Parameter Optimization*, pp. 281–295, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Nanyang Technological University, Singapore, Technical Report, 2013.

[38] H. Peng, C. Deng, and Z. Wu, "Best neighbor-guided artificial bee colony algorithm for continuous optimization problems," *Soft Computing*, vol. 23, no. 18, pp. 8723–8740, 2019.

[39] H. Hakli and M. S. Kiran, "An improved artificial bee colony algorithm for balancing local and global search behaviors in continuous optimization," *International Journal of Machine Learning and Cybernetics*, vol. 11, no. 9, pp. 2051–2076, 2020.

[40] H. Peng, Z. Wu, and C. Deng, "Enhancing differential evolution with commensal learning and uniform local search," *Chinese Journal of Electronics*, vol. 26, no. 4, pp. 725–733, 2017.

[41] J. Liu, H. Peng, Z. Wu, J. Chen, and C. Deng, "Multi-strategy brain storm optimization algorithm with dynamic parameters adjustment," *Applied Intelligence*, vol. 50, no. 4, pp. 1289–1315, 2020.

[42] J. H. Holland, "Genetic algorithms and the optimal allocation of trials," *SIAM Journal on Computing*, vol. 2, no. 2, pp. 88–105, 1973.

[43] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant System: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics—Part B*, vol. 26, no. 1, pp. 29–41, 1996.

[44] Y. Shi, "Brain storm optimization algorithm," in *International Conference in Swarm Intelligence*, pp. 303–309, Springer, Heidelberg, Germany, 2011.