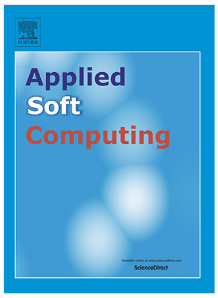
Journal Pre-proof



A Parallel Compact cuckoo search algorithm for three-dimensional path planning

Pei-Cheng Song, Jeng-Shyang Pan, Shu-Chuan Chu

PII: S1568-4946(20)30383-5
DOI: https://doi.org/10.1016/j.asoc.2020.106443
Reference: ASOC 106443

To appear in: *Applied Soft Computing Journal*

Received date : 17 November 2019
Revised date : 15 May 2020
Accepted date : 16 May 2020

Please cite this article as: P.-C. Song, J.-S. Pan and S.-C. Chu, A Parallel Compact cuckoo search algorithm for three-dimensional path planning, *Applied Soft Computing Journal* (2020), doi: https://doi.org/10.1016/j.asoc.2020.106443.

Highlights

- Compact Cuckoo Search Algorithm
- Parallel Cuckoo Search Algorithm
- Algorithm that reduce the computation time and the memory
- Parallel Compact Cuckoo Search Algorithm
- Optimization methods for 3D path planning

A Parallel Compact Cuckoo Search Algorithm for Three-dimensional Path Planning

Pei-Cheng Song[a], Jeng-Shyang Pan[a,b], Shu-Chuan Chu[a,b,*]

[a]*College of Computer Science and Engineering, Shandong University of Science and Technology,Qingdao 266590, China*
[b]*College of Science and Engineering, Flinders University, 1284 South Road, Tonsley SA 5042, Australia*

Abstract

The three-dimensional (3D) path planning of unmanned robots focuses on avoiding collisions with obstacles and finding an optimized path to the target location in a complex three-dimensional environment. An improved cuckoo search algorithm based on compact and parallel techniques for three-dimensional path planning problems is proposed. This paper implements the compact cuckoo search algorithm, and then, a new parallel communication strategy is proposed. The compact scheme can effectively save the memory of the unmanned robot. The parallel scheme can increase the accuracy and achieve faster convergence. The proposed algorithm is tested on several selected functions and three-dimensional path planning. Results compared with other methods show that the proposed algorithm can provide more competitive results and achieve more efficient execution.

Keywords: 3D path planning, cuckoo search algorithm, parallel communication strategy, compact strategy

1. Introduction

The meta-heuristic algorithm can effectively solve various optimization problems and is suitable for dealing with problems that are not solved by specific

*Corresponding author
Email addresses: spacewe@outlook.com (Pei-Cheng Song), jspan@cc.kuas.edu.tw (Jeng-Shyang Pan), scchu0803@gmail.com (Shu-Chuan Chu)

effective methods[1]. They can be used to solve relevant problems in the fields of industry, finance, mathematics, etc. According to the No Free Lunch Theorem (NFL) [2, 3], there is no meta-heuristic algorithm is suitable for dealing with all types of optimization problems. So new meta-heuristic algorithms are constantly being proposed every year, and continuous improvements are made on existing algorithms to deal with increasingly complex problems [4].

Cuckoo Search algorithm (CS) simulates the parasitic and brooding behavior of cuckoos to solve complex optimization problems effectively[5, 6, 7, 8]. A nest of cuckoo represents a possible solution in the algorithm, and the algorithm is combined with Lévy flight to constantly update the position of the nest to find a potentially better solution to be a new cuckoo's nest. The key parameters of the CS algorithm are less than other similar algorithms, and the algorithm is easy to implement. The combination with Lévy flight search mechanism makes the algorithm have better global optimization ability, which has shown high efficiency in engineering optimization problems[9, 10, 11]. The CS algorithm has evolved many variations and improvements from the original algorithm, such as Modified Cuckoo Search algorithm (MCS)[12], Multiobjective Cuckoo Search algorithm (MOCS)[9], Chaotic Cuckoo Search algorithm (CCS)[13], Binary Cuckoo Search algorithm (BCS)[14],etc. Multiple optimization problems can be solved using the CS algorithm, such as the feedforward neural network training[15],the permutation flow shop scheduling problems[16], the traveling salesman problem[17], the data clustering problem[18]. Although the CS algorithm is widely used in many problems, there are few related kinds of research on 3D path planning problems. Compared with other heuristic algorithms, the CS algorithm was proposed later. The CS algorithm still has much room for improvement in some areas, so this paper attempts to use the improved CS algorithm to deal with 3D path planning problems.

The three-dimensional route planning problem is to obtain the optimized shortest path to the destination location through the corresponding algorithm after establishing the environmental model according to specific optimization requirements[19, 20]. It is used to realize the intelligence and adaptability of unmanned mechanical equipment in complex environments. It is often used in underwater unmanned submersibles, unmanned smart cars, or unmanned aerial vehicles[21]. In recent years, there have been several effective 3D path planning strategies, including bio-inspired algorithms, node-based algorithms, sampling-based algorithms, multi-fusion based algorithms, and mathematical model-based algorithms[21, 22]. There are numerous corresponding specific algorithms in each strategy. There are several practical algorithms in bio-inspired algorithms, including genetic algorithm (GA)[23], memetic algorithm (MA)[24], classical particle swarm optimization algorithm (PSO)[25, 26], evolutionary algorithm (EA)[27], classical ant colony optimization algorithm(ACO)[28] and shuffled frog leaping algorithm(SFLA)[29], etc. These algorithms are not only applied in different fields, but can also be applied to 3D path planning problems. Bio-inspired algorithms can converge stably compared to other strategies to find an optimal path that meets the requirements.

There are some studies using CS algorithm for path planning. Reference [7] achieves drone path optimization in two-dimensional space with chaotic cuckoo search algorithm. Reference [30] also utilizes a cuckoo search algorithm to optimize the path of a mobile robot in two dimensions. Reference [31] uses a hybrid differential evolution algorithm and cuckoo algorithm to optimize the trajectory of an uninhabited combat air vehicle (UCAV), it uses a B-Spline curve to smooth the path. The effects of the bat and the cuckoo algorithms on path planning problems are compared in [32]. As a result, the bat algorithm can provide better results. Reference [33] is a mixture of cuckoo and bat algorithms for better results. [34] implements path planning in 3D space by using a hybrid genetic algorithm and cuckoo algorithm. Compared with the ACO algorithm and other hybrid algorithms, the original CS algorithm cannot handle 3D path planning problems well. In addition, there are other researches using other similar heuristics to deal with two-dimensional (2D) or 3D spatial path planning [35, 36, 37, 38, 39].

However, there is no improved method of the CS algorithm used to handle the application problem on devices with limited memory. This paper attempts to improve CS by combining new parallel communication strategy and compact method, which are applied to 3D path planning problems. Compact technology uses a probability model to represent the entire population for operation, which can effectively save memory and is suitable for small robots and other devices with limited memory. So this paper first uses the compact method to improve the CS algorithm to meet the needs of memory-constrained devices. The common parallel communication strategy usually divides the entire population into multiple groups, and each group has multiple individuals [40, 41]. In this paper, a single compact CS algorithm is considered as an individual, and it is updated using the proposed parallel communication strategy. Improved algorithm can adapt to devices with different memory spaces by using the different number of groups.

The compact method [42, 43, 44, 45, 46, 47] uses a macro probabilistic model to represent the original population, which in turn enables the operation of the original population-based algorithm. The compact method uses the distribution characteristics of the original population to generate a probability model[42], and the algorithm replaces the operation of the original population with the operation of the probability model, so that the probability model requires fewer variables than the original population, and can save more memory. Some algorithms that use probabilistic models are also constantly being proposed, such as compact genetic algorithm (cGA)[43], compact differential evolution algorithm (cDE)[44], compact particle swarm optimization algorithm (cPSO)[42], compact bat algorithm (cBA)[45], compact articial bee colony optimization (cABC)[48], etc. However, there is no version of the compact method for the CS algorithm. Parallelism is an important algorithm optimization method[49][50][51][52], which can exchange information between groups, thus achieving faster convergence and better solutions. According to different algorithms and applications, a variety of parallel communication strategies can be proposed to enhance algorithm performance. This paper proposes a parallel compact cuckoo search algorithm (pcCS) and applies it to 3D path planning.

Based on the above, this paper attempts to improve the CS algorithm using the compact method and uses the parallel communication strategy to make further improvements to meet the needs of different memory devices. Finally, this paper attempts to apply the improved algorithm to the 3D path planning problem. The main contributions of the paper are listed as follows:

- Improved and implemented six different compact CS algorithms, and then compared and tested the proposed algorithms.
- A single compact CS algorithm as a particle, using parallel communication strategies to improve and update.
- Aiming at the characteristics of poor localized search ability of the cuckoo search algorithm, a new parallel communication strategy for optimizing local search ability is proposed.
- Hybrid parallel and compact technology improves the original cuckoo algorithm and applies it to 3D path planning problems.
- The improved CS algorithm was tested using some selected benchmark functions and then compared to other compact and general algorithms.
- The application of the improved cuckoo search algorithm in the discrete optimization problem of 3D path planning.

The rest of the paper is organized as follows. The related work introduced the principle of the CS algorithm and the problem of 3D path planning. In Section 3, the steps and methods of algorithm improvement are introduced, and the process of using compact to improve CS algorithm and the application of parallel communication strategies are introduced. The performance of the proposed algorithm is tested in Section 4, and the results of different algorithms are shown and analyzed. Section 5 introduces the application of the proposed algorithm in 3D path planning. Finally, the conclusion is given in Section 6.

2. Related work

A brief introduction to the CS algorithm and a 3D route planning problem is provided in this section.

2.1. Cuckoo search algorithm

The CS algorithm is characterized by its ability to simulate the breeding strategy of cuckoos in nature[5]. The cuckoo will search for the nest location most suitable for its spawning in a random or similar random way. The main method of the CS algorithm is to imitate the parasitic brooding behavior of the cuckoo and use Lévy flight to simulate the random walk of the cuckoo. In nature, the movement patterns of many birds share the characteristics of Lévy flight. The Lévy flight process includes frequent short-distance flights and occasional long-distance flights. The flight steps are in accordance with the Lévy distribution, and occasional long-distance flights may expand the search range and also avoid trapping into local optimal values.

To simplify the implementation of this algorithm , three simple and idealized rules are set for the cuckoo search algorithm[5]. 1) Only one cuckoo egg is produced by each cuckoo at a time, then randomly selects a location for hatching. 2) The nest with the best egg will be preserved and used on to the next iteration. 3) The number of nests that can be used is fixed, and the probability of the eggs in the nest being found is $p_a \in [0, 1]$. When the egg is found, the owner of the nest will construct a new nest elsewhere or throw the cuckoo egg. According to the above rules for CS behavior, when x_i^{t+1} represents the next generation solution, x_i^t represents the current solution, i is a cuckoo in the solution, the update method for the location of the search process is as follows:

$$x_i^{t+1} = x_i^t + \rho L\acute{e}vy(\delta) \tag{1}$$

In Equation 1, ρ indicates the moving step size scaling factor and $\rho > 0$, usually use $\rho = 1$. The setting of ρ is usually related to the size of the problem decision space, and it is necessary to set the appropriate value to make the search more efficient without jumping out of the decision space too quickly [53]. In [5], the above formula is actually a stochastic equation that implements random walks. Usually, a Markov chain is used as the random walk whose next position depends only on the current position (the first term in the above formula) and the transition probability (the second term). The random step size in Lévy flight is generated using the Lévy probability distribution.

$$Lévy(\delta) \sim u = t^{-1-\delta}, \quad (0 < \delta \leq 2) \tag{2}$$

In addition to Equation 1 and 2, the Cuckoo Algorithm also performs a local search operation. CS algorithm uses the switch parameter p_a to achieve the balance between local random search and global search [53]. The operation can be written as

$$x_i^{t+1} = x_i^t + \rho \times s \otimes H(p_a - c) \otimes \left(x_j^t - x_k^t\right) \tag{3}$$

In Equation 3, s indicates the step size, x_k^t and x_j^t are two randomly selected solutions from the whole population. c is a random number generated with a uniform distribution in the range 0 to 1. $H(u)$ indicates a Heaviside function. Following to the above introduction, the pseudo-code of the CS algorithm is shown in Algorithm 1.

After the original CS algorithm was proposed, many different improved versions appeared. In addition to the introduction in Section 1, there are many different improved methods. In [12], the new algorithm improves the parameter ρ of the Equation 1, so that the parameter ρ decreases continuously with the increase of the number of iteration. The second modification is to increase the process of information exchange between individuals, by generating a new solution closer to the optimal solution, speeding up convergence to a minimum. In [15], the algorithm improves the parameters p_a and ρ. The two parameters will change dynamically with the iterative process and gradually become smaller, but the two are reduced in different ways. In addition, there are many researches on improving CS algorithm parameters [13, 54, 55].

Algorithm 1: CS

Fitness function $f(x), x = (x_1, \ldots, x_d)^T$;
Initializing a population of n nests, $x_i (i \leq n)$;
while *(t< Max Generation) or (stop criterion)* **do**
    Update cuckoo x_i by Lévy flights via Eq. 1;
    Get its quality/fitness F_i;
    Choose a nest x_j randomly;
    if $(F_i > F_j)$ **then**
        replace x_j by the new solution;
    Use fraction (p_a) to reset worse nests via Eq. 3;
    Keep the best nest and rank the solutions, find the current best;
Final result output and presentation;

In [56], algorithm not only improves the parameters but also uses the operation method of the DE algorithm, adding the effect of the global optimal solution on other solutions. In [57], the local search formula of the CS algorithm is improved, and the global optimal solution is used to guide the movement of other individuals. In addition, there are other literatures that use global optimal solutions to improve CS [58, 59].

In addition to improving the parameters and adding the effect of global optimization, there are also improved methods, such as mixing with other algorithms [60].

2.2. Three-dimensional path planning

Unmanned devices moving in three-dimensional space need to avoid obstacles in complex environments. Modeling threat sources is an important task before path planning[19]. The method used in this paper is to divide the three-dimensional space into a grid, so that the path from start to endpoint can be planned from the three-dimensional grid[61]. After acquiring the relevant data of the three-dimensional map, then we can create a 3D spatial coordinate sys-

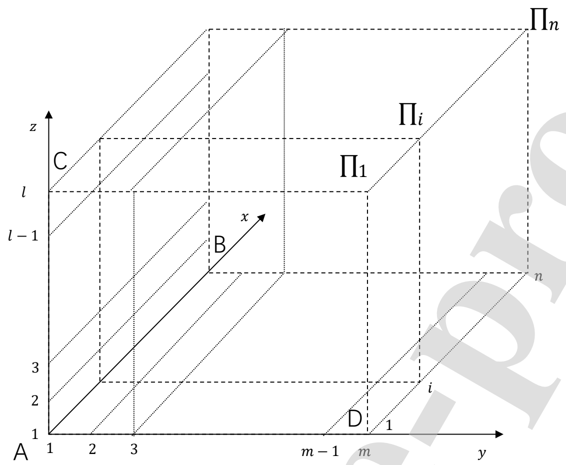


Figure 1: Three-dimensional space division

tem, with the vertices in the lower-left corner as the origin of the coordinate. In this space coordinate, the x-axis as the direction in which the longitude increases, the y-axis as the direction in which the latitude increases, and the z-axis as the direction perpendicular to the sea level. After that, the space represented by the three-dimensional coordinate system is equally divided into three-dimensional meshes. First, the three-dimensional space is equally divided along with the edge AB to obtain n planes $\Pi_i (i = 1, 2, \cdots, n)$. Then, the n planes are equally divided into m along the edge AD, and the n planes are equally divided into l along the edge AC to form each intersection in the grid. The effect on the three-dimensional space division is shown in Figure 2. This sets the entire 3D space as a collection of 3D points.Each point in the collection can be represented in two ways, including the serial number coordinates $p_1(a, b, c)(a = 1, 2, \cdots, n; b = 1, 2, \cdots, m; c = 1, 2, \cdots, l)$ and the actual position coordinates $p_2 (x_i, y_i, z_i)$, where a, b and c represent the sequence numbers along the three sides.

The algorithm plans on these three-dimensional discrete points to find a route that meets the optimization requirements from the start point to the destination point. The common 3D path optimization requires finding the optimized shortest path to the endpoint under the premise of avoiding the obstacle.

In the grid established for the three-dimensional space, Q_{wv} represents the distance between the current point w and the point v on the next adjacent plane.

$$Tp = \sum_{w=1}^{n-1} Q_{wv} = \sum_{w=1}^{n-1} \sqrt{(x_w - x_v)^2 + (y_w - y_v)^2 + (z_w - z_v)^2} \quad (2 \leq v \leq n) \tag{4}$$

The total path Tp from the starting point to the destination point is to add the distances of two adjacent planar path points. In this space, the distance from the start point to the endpoint can be obtained from Equation 4. Similarly, the fitness function optimized by the improved cuckoo algorithm in this paper is also Equation 4. The goal of algorithm optimization is to minimize Tp.

3. Compact CS and parallel strategy

In this section, the process of improving the cs algorithm is divided into two parts. The first part is the introduction of the compact method and the implementation of the compact CS algorithm. The second part is the proposal of the parallel communication strategy and the combination with the compact CS algorithm.

3.1. Compact scheme and CS

The essence of the estimated distribution algorithm (EDA) is a method based on the probability model, which uses a probabilistic model to describe the distribution characteristics of the original population[62]. Firstly, the probabilistic model is constructed using the original population distribution, then the probabilistic model is used to perform the algorithm operation, and the constant search of the solution space is realized by updating the probabilistic model [42, 43, 44, 45, 46, 47].

Perturbation Vector (PV) is often used to describe the macroscopic probability distribution of the entire existing population [42, 44, 45]. PV is constantly changing with the operation of the algorithm, which can be organized as: $PV^t = [\mu^t, \delta^t]$, where μ is used to representing the mean value of the Perturbation Vector ,and δ is the standard deviation of the Perturbation Vector, t is used to represent the number of current iterations. Each pair of mean value and standard deviation in PV corresponds to the corresponding probability density function (PDF), and PDF is truncated at $[-1, 1]$, and PDF normalizes the area of the amplitude to 1[63]. In the compact method, a solution x_i can be generated by using the probability density function formed by the PV. By constructing a Chebyshev polynomial, the PDF can have the ability to correspond to a cumulative distribution function (CDF) with a range of values from 0 to 1 [64, 65]. Since the PDF function is truncated at [-1, 1], the calculation method of CDF is shown as

$$CDF = \int_{-1}^{x} PDF dx = \int_{-1}^{x} \frac{\sqrt{\frac{2}{\pi}} e^{-\frac{(x-\mu)^2}{2\delta^2}}}{\delta \left(erf\left(\frac{\mu+1}{\sqrt{2}\delta}\right) - erf\left(\frac{\mu-1}{\sqrt{2}\delta}\right) \right)} dx \tag{5}$$

In Equation 5, the value range of x is [-1,1], μ and δ are mean and standard deviation in PV. erf is the error function [66]. So the CDF function can be expressed as Equation 6.

$$CDF = \frac{erf\left(\frac{\mu+1}{\sqrt{2}\delta}\right) + erf\left(\frac{x-\mu}{\sqrt{2}\delta}\right)}{erf\left(\frac{\mu+1}{\sqrt{2}\delta}\right) - erf\left(\frac{\mu-1}{\sqrt{2}\delta}\right)} \tag{6}$$

The CDF can be used to describe the value of the actual solution of the population by a similar probability distribution, while PDF is the macroscopic probability distribution constructed by the population distribution. Using the PV vector, the solution x_i can be randomly generated by the inverse CDF. In the process of algorithm execution, it is necessary to continuously update the PV. After generating two solutions, it is usually by comparing the values of the fitness functions of the two solutions to determine which is better, the better solution is as the *winner*, the worse solution is as the *loser*, and then the PV is updated[47]. The formula for updating each standard deviation and the average value in the PV using *winner* and *loser* is as follows:

$$\mu_i^{t+1} = \mu_i^t + \frac{1}{N_p}\left(winner_i - loser_i\right) \tag{7}$$

In Equation 7, μ_i^{t+1} represents the newly generated average, and N_p is the virtual population. The update rule for δ is as follows:

$$\delta i^{t+1} = \sqrt{\left(\delta i^t\right)^2 + \left(\mu_i^t\right)^2 - \left(\mu_i^{t+1}\right)^2 + \frac{1}{N_p}\left(winner_i^2 - loser_i^2\right)} \tag{8}$$

Based on the introduction of the compact scheme above, the compact CS (cCS) algorithm can be implemented. First, use a uniform distribution to generate a random number x in the range 0 to 1. Next, use PV to generate a solution through the inverse function of CDF. The inverse function of CDF is written as

$$y = \sqrt{2}\delta erf^{-1}\left(-erf\left(\frac{\mu+1}{\sqrt{2}\delta}\right) - xerf\left(\frac{\mu-1}{\sqrt{2}\delta}\right) + xerf\left(\frac{\mu+1}{\sqrt{2}\delta}\right)\right) + \mu \tag{9}$$

In Equation 9, erf^{-1} is the inverse function of erf. Since the range of solutions generated using Equation 9 is between -1 and 1. That is, the value of y ranges from -1 to 1, it is necessary to map the generated solutions to the actual decision space. Suppose an n-dimensional problem, lb and ub are the limits of the sampling range in a certain dimension. Then the function that maps the solution y generated by inverse CDF to the actual decision space can be written as

$$y_{ds} = y \times \frac{1}{2}(ub - lb) + \frac{1}{2}(ub + lb) \tag{10}$$

After mapping the generated solution to the actual decision space using Equation 10, levy random walk can be performed using Equation 1. However, in the cCS algorithm, Equation 3 in the original CS algorithm cannot be used directly to implement the local search operation. Because Equation 3 needs to randomly select two solutions from the entire population to update the current solution, but for cCS there is only one solution generated using PV. In addition, the previous literature [67] did not improve the local search formula. So this paper modified this step and proposed six functions to implement Equation 3 in compact scheme. By using different variants of Equation 3, different versions of the cCS algorithm will be obtained: cCS_V1, cCS_V2, cCS_V3, cCS_V4, cCS_V5 and cCS_V6. The specific form of each function is shown in Table 1.

Table 1: Compact CS algorithm versions with different variants of Equation 3

Name	Equation variant
cCS_V1	$x_3 = x_2 + \rho s \otimes H(p_a - c) \otimes (x_{t2} - x_{t1})$
cCS_V2	$x_3 = x_2 + \rho s \otimes H(p_a - c) \otimes (best - x_1)$
cCS_V3	$x_3 = x_2 + \rho s \otimes H(p_a - c) \otimes (\alpha(best - x_1) + \beta(x_{t2} - x_{t1}))$
cCS_V4	$x_3 = H(p_a - c) \otimes best + (1 - H(p_a - c)) \otimes x_2$
cCS_V5	$x_3 = x_2 + H(p_a - c) \otimes u$
cCS_V6	$x_3 = \begin{cases} x_2 + \rho s \otimes H(p_a - c) \otimes (best - x_1), & rand \geq 0.5 \\ H(p_a - c) \otimes best + (1 - H(p_a - c)) \otimes x_2, & else \end{cases}$

For cCS_V1 in Table 1, the form of the equation is the same as Equation 3, but x_{t1} and x_{t2} are generated using PV. This method can better retain the characteristics of the original equation, but the extra x_{t1} and x_{t2} will take up more space. For cCS_V2, x_1 is the initial solution generated using PV, x_2 is the solution obtained by performing Equation 1 on x_1, and x_3 is the solution that needs to be updated on the basis of x_2 using equation. *best* is the optimal solution retained by the algorithm during execution and will be continuously updated.

cCS_V3 combines the characteristics of cCS_V2 and cCS_V1, which do not only retain the influence of the global optimal solution, but also increases the number of times of PV sampling. cCS_V4 uses the switching parameter p_a to combine the value of *best* and x_2. For a certain dimension of the solution, when the random number is less than p_a, the value of the *best* of the current dimension is selected; otherwise, x_2 is selected. cCS_V5 uses p_a to perturb x_2 to achieve a local random search, where u is a random number generated using a standard normal distribution. cCS_V6 combines the characteristics of cCS_V2 and cCS_V4, and controls the switching of different generation methods through a random number *rand*. The random number *rand* ranges from 0 to 1.

After completing the above steps, the compact algorithm needs to compare the generated solutions to determine the *winner* and *loser*, and then use the *winner* and *loser* to update the PV. It should be noted that when updating PV using *winner* and *loser*, it needs to be mapped to the interval of -1 to 1 using the inverse of Equation 10.

The PV vector and the generated individual solution are stored during algorithm execution, instead of storing the location of the entire population solution and the motion vector, which achieves less runtime memory usage and is beneficial for use on resource-constrained devices. Based on the above compact methods, this paper uses the formula of cCS_V2 to introduce, the pseudocode is shown as Algorithm 2.

Algorithm 2: cCS (cCS_V2)

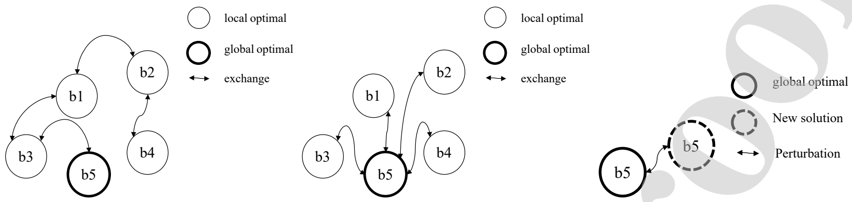
for $i = 1 : d$ **do**
    initialize $\mu[i] = 0$; $\delta[i] = \lambda = 10$;
Initializing $best = up\_bound$;$Fmin = Inf$;
while *(t< Max Generation) or (stop criterion)* **do**
    Get x_1 from PV via Eq. 9 10;
    x_1 Lévy random walk generates x_2 via Eq. 1;
    Update x_2 with fraction (p_a) to get x_3 via cCS_V2's Eq in Table 1;
    $[winner, loser, fit_{winner}]$ =compete(x_3, x_1);
    for $i = 1 : d$ **do**
        Update PV via Eq. 7, Eq. 8;
    if $fit_{winner} < Fmin$ **then**
        $best = winner$;$Fmin = fit_{winner}$;

For other versions of cCS, the corresponding changes need to be made according to the different formulas in Table 1. For cCS_V1 and cCS_V3, Algorithm 2 needs to use PV to generate x_{t2} and x_{t1} before generating x_3. For cCS_V6, the algorithm needs to generate a random number to choose which formula to execute.In Algorithm 2, d represents the dimension of the problem, and *up_bound* represents the upper limit of the decision space, which is a d-dimensional vector. $Fmin$ is used to store the fitness value of the global optimal solution. Inf represents an infinite number. fit_{winner} is used to store the fitness value of the $winner$.

3.2. Parallel communication strategy and cCS

Algorithm can improve the search ability and convergence speed by using a parallel communication strategy and can find better solutions faster[49][41]. For the meta-heuristic algorithm using the parallel communication strategy, the population is divided into multiple subgroups, each subgroup is calculated independently, and communicates after each iteration. The common basic idea is to use better solutions of some groups to replace the worse ones of the other groups. There are many ways to communicate with each subgroup in a parallel communication strategy [49, 41, 51]. In this paper, a parallel communication method is proposed to strengthen the cCS algorithm.

According to the introduction in the previous section, this paper gives the implementation of the cCS algorithm. The compact method can utilise the probability model to represent the entire population and achieve similar performance to the original algorithm with less memory. As introduced in [43, 44, 42, 45, 48], compact technology can achieve similar effects to the original algorithm on memory-constrained devices. Sometimes the memory-constrained device may not be enough to support the entire population of the original algorithm, but the memory is sufficient when using the compact algorithm, and there may be extra space. In order to further enhance the performance of the compact algorithm and make full use of the existing memory space of the device, this paper proposes a new parallel communication strategy to enhance the performance of the cCS algorithm.



(a) Part1:Random exchange (b) Part2:Optimal exchange (c) Part3:perturbation

Figure 2: Three parts of the parallel communication strategy.

This paper assumes that each group includes its own PV and optimal solution *best*, and uses x_1, x_2, and x_3 generated by PV as temporary variables. This paper uses two strategies and randomly selects different strategies to execute when communicating between groups. The composition of the two strategies can be divided into three parts. Three components mimic the principles of common biological heuristics.

This paper uses Figure 2 to introduce the three parts of the parallel communication strategy. Assume that there are five groups of cCS, each of which executes its own algorithmic process. After each iteration is completed, the optimal value and updated PV are obtained. Then assume that the optimal value of each group is used as the basic particle, and update it using the three methods shown in Figure 2.

For part 1 in Figure 2, suppose $b1$ is the best solution *best* obtained by the first group. Next, the algorithm needs to choose one of the remaining 4 optimal solutions, assuming $b4$. If the fitness value of $b4$ is better than $b1$, then $b1$ will be set to be the same as $b4$. If the fitness value of $b4$ is worse than that of $b1$, the algorithm will execute the third part to perturb $b1$. So strategy 1 is a combination of part 1 and part 3.

As shown in part 2 of Figure 2, for a non-global optimal solution, the algorithm will update it to a global optimal solution. However, the global optimal solution $b5$ cannot find a better solution than itself, so the algorithm will execute part 3 and perturb $b5$ to try to find a better solution. So strategy 2 is a combination of part 2 and part 3.

According to the introduction above, it is assumed that the optimal solution of each group is a basic particle, and the three components of the two strategies realize the execution effect of common biological heuristic algorithms. The first part realizes the communication between different particles, the second part realizes the approach to the optimal solution, and the third part realizes the search of the local area. This paper also implements the optimization of the cCS algorithm using a parallel communication strategy. Since the next improved algorithm is based on the cCS algorithm, and the cCS algorithm has six different implementations, for convenience, this paper uses cCS_V2 to represent the cCS algorithm, the pseudocode is shown in Algorithm 3.

According to Algorithm 3, *rand* is a random number in the range of 0 to 1. The initialization of PV, *best*, and $Fmin$ for each group is the same as in Algorithm 2. *GlobalBest* is updated to the optimal solution in all groups in Algorithm 3. The way to perturb and update the optimal solution $G(i).best$ of each group is to add a perturbation term to the solution. The perturbation term can be randomly generated using a standard normal distribution.

4. Experimental analysis of the algorithm

4.1. Benchmark functions and algorithm parameters

This section selects a variety of numerical optimization functions and problems to test the effect of the proposed pcCS algorithm [42, 68, 69]. The selected test functions include both unimodal and multimodal functions, and their dimensions are variable. This paper sets the dimension of each test function to $D = 30$. The expressions and ranges of all benchmark test functions are shown in Tables 2 and 3. The benchmark test functions $f_1 - f_7$ and $f_{13} - f_{18}$ are taken from [69], which is the same as the benchmarks used in [70]. f_{25} is from [71], and f_{10} is from [72]. The remaining other algorithms come from [68]. All the tested algorithms maintain consistent parameter settings.

This paper first analyzes and tests the six versions of the cCS algorithm, and then selects the better version for subsequent comparison and improve-

Algorithm 3: pcCS (use cCS_V2)

```
Set the number of groups g, each group is G(i), (i ≤ g);
Initialize the G(i).PV, G(i).best and G(i).Fmin of each group;
Initialize GlobalBest = G(1).best; GlobalFmin = G(1).Fmin;
while (t< Max Generation ) or (stop criterion) do
    for i = 1 : g do
        Get x_1 ,x_2, x_3 via Eq. 9 10, Eq. 1, and cCS_V2's Eq in Table 1;
        [winner, loser, fit_winner] =compete(x_3, x_1);
        for i = 1 : d do
            Update G(i).PV via Eq. 7, Eq. 8;
        if fit_winner < G(i).Fmin then
            G(i).best = winner;G(i).Fmin = fit_winner;
    Update GlobalFmin and GlobalBest;
    for i = 1 : g do
        if rand < 0.5 then
            Randomly select a group k,k ≠ i; // Strategy 1;
            if G(i).Fmin > G(k).Fmin then
                //Part 1;
                G(i).best = G(k).best; G(i).Fmin = G(k).Fmin;
            else
                Disturb and update G(i).best; //Part 3;
        else
            //Strategy 2;
            if G(i).Fmin > GlobalFmin then
                //Part 2;
                G(i).best = GlobalBest; G(i).Fmin = GlobalFmin;
            else
                Disturb and update G(i).best; //Part 3;
```

Table 2: Unimodal benchmark functions.

Name	Function	D	Range
Sphere	$f_1 = \sum_{i=1}^{D} x_i^2$	30	[-100,100]
Schwefel 2.22	$f_2 = \sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	30	[-10,10]
Schwefel 1.2	$f_3 = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_j \right)^2$	30	[-100,100]
Schwefel 2.21	$f_4 = \max_{1 \le i \le D} \lvert x_i \rvert$	30	[-100,100]
Rosenbrock	$f_5 = \sum_{i=1}^{D-1} \left[100 \left(x_{i+1} - x_i^2 \right)^2 + (x_i - 1)^2 \right]$	30	[-30,30]
Step	$f_6 = \sum_{i=1}^{D} \left(\lfloor \lvert x_i \rvert \rfloor \right)$	30	[-100,100]
Quartic	$f_7 = \sum_{i=1}^{D} i x_i^4 + random[0,1)$	30	[-1.28,1.28]
Zakharov	$f_8 = \sum_{i=1}^{n} x_i^2 + \left(\frac{1}{2} \sum_{i=1}^{n} i x_i \right)^2 + \left(\frac{1}{2} \sum_{i=1}^{n} i x_i \right)^4$	30	[-5,10]
Sum Squares	$f_9 = \sum_{i=1}^{D} i x_i^2$	30	[-10,10]
Ridge	$f_{10}(x) = x_1 + d \left(\sum_{i=2}^{n} x_i^2 \right)^{\alpha}, d = 1, \alpha = 0.5$	30	[-5,5]
Xin-She Yang 3	$f_{11} = [e^{-\sum_{i=1}^{D} (x_i/\beta)^{2m}} - 2e^{-\sum_{i=1}^{D} (x_i)^2} \cdot \prod_{i=1}^{D} \cos^2(x_i)]$	30	[-20,20]
Dixon & Price	$f_{12} = (x_1 - 1)^2 + \sum_{i=2}^{D} i \left(2x_i^2 - x_{i-1} \right)^2$	30	[-10,10]

ment. The cCS algorithm is compared with other common compact algorithms, including cPSO [42], cBA [45], cABC [48], and cDE [44]. This paper then uses a parallel communication strategy to improve the cCS algorithm, and both improved algorithms are compared with the original CS algorithm [5] and the improved adaptive CS algorithm ACS [70].

Next, this paper compares the proposed pcCS algorithm with classic PSO [73], GA algorithms and common algorithms, including the Bat Algorithm (BA) in 2010 [74], the Flower Pollination Algorithm(FPA) in 2012 [75], Sine Cosine Algorithm (SCA) in 2016 [76], Butterfly optimization algorithm (BOA) in 2018 [77]. The parameter settings of related algorithms are shown in Table 4. Np represents the population size. When comparing different algorithms, the total number of function evaluations of all algorithms is the same.

Table 3: Multimodal benchmark functions.

Name	Function	D	Range
Schwefel 2.26	$f_{13}=-\frac{1}{D}\sum_{i=1}^{D}x_i\sin\sqrt{\lvert x_i\rvert}$	30	[-500,500]
Rastrigin	$f_{14}=\sum_{i=1}^{D}\left[x_i^2-10\cos\left(2\pi x_i\right)+10\right)\right]$	30	[-5.12,5.12]
Ackley 1	$f_{15}=-20e^{-0.02\sqrt{D^{-1}\sum_{i=1}^{D}x_i^2}}$ $-e^{D^{-1}\sum_{i=1}^{D}\cos(2\pi x_i)}+20+e$	30	[-35,35]
Griewank	$f_{16}=\sum_{i=1}^{D}\frac{x_i^2}{4000}-\prod\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$	30	[-100,100]
Generalized penalized function 1	$f_{17}=\frac{\pi}{n}\{10\sin^2\pi y_1$ $+\sum_{i=1}^{n}\left(\left(y_i-1\right)^2\left(1+10\sin^2\pi y_i\right)\right)$ $+\left(y_n-1\right)^2\}+\sum_{i=1}^{n}u\left(x_i,10,100,4\right)$	30	[-50,50]
Generalized penalized function 2	$f_{18}=\frac{1}{10}\{\sin^2 3\pi x_1$ $+\sum_{i=1}^{n-1}\left(\left(x_i-1\right)^2\left(1+\sin^2 3\pi x_{i+1}\right)\right)\}$ $+\frac{1}{10}\{\left(x_n-1\right)\left(1+\sin 2\pi x_n\right)^2\}$ $+\sum_{i=1}^{n}u(x_i,5,100,4)$	30	[-50,50]
Periodic	$f_{19}=1+\sin^2\left(x_1\right)$ $+\sin^2\left(x_2\right)-0.1e^{-\left(x_1^2+x_2^2\right)}$	30	[-10,10]
Ackley 4	$f_{20}=\sum_{i=1}^{D}(e^{-0.2}\sqrt{x_i^2+x_{i+1}^2}$ $+3\left(\cos\left(2x_i\right)+\sin\left(2x_{i+1}\right)\right))$	30	[-35,35]
Xin-She Yang 2	$f_{21}=\left(\sum_{i=1}^{D}\lvert x_i\rvert\right)\exp\left[-\sum_{i=1}^{D}\sin\left(x_i^2\right)\right]$	30	[-2π,2π]
Xin-She Yang 4	$f_{22}=\left[\sum_{i=1}^{D}\sin^2\left(x_i\right)-e^{-\sum_{i=1}^{D}x_i^2}\right]$ $\cdot e^{-\sum_{i=1}^{D}\sin^2\sqrt{\lvert x_i\rvert}}$	30	[-10,10]
Styblinski-Tang	$f_{23}=\frac{1}{2}\sum_{i=1}^{D}\left(x_i^4-16x_i^2+5x_i\right)$	30	[-5,5]
Salomon	$f_{24}=1-\cos(2\pi\sqrt{\sum_{i=1}^{D}x_i^2})$ $+0.1\sqrt{\sum_{i=1}^{D}x_i^2}$	30	[-100,100]
Happy Cat	$f_{25}=\left[\left(\lVert\mathbf{x}\rVert^2-n\right)^2\right]^{\alpha}$ $+\frac{1}{n}\left(\frac{1}{2}\lVert\mathbf{x}\rVert^2+\sum_{i=1}^{n}x_i\right)+\frac{1}{2}$	30	[-2,2]
Qing	$f_{26}=\sum_{i=1}^{D}\left(x_i^2-i\right)^2$	30	[-500,500]

Table 4: Parameter settings for each related algorithm

Name	Parameter
cCS	Virtual Np=200, p_a=0.25 (cCS_V3: $\alpha, \beta \in [0, 1]$)
pcCS	Virtual Np=200, p_a=0.25, groups = 3
cPSO	Virtual Np=200, ϕ_1 = -0.2, ϕ_2 = -0.07, ϕ_3 = 3.74, γ_1 = 1, γ_2 = 1
cBA	Virtual Np=200, loudness = 0.5, pulse rate = 0.5, f_{min}=0, f_{max}=2
cABC	Virtual Np=200, limit = 100
cDE	Virtual Np=200, F = 0.5, Cr = 0.5
CS/ACS	Np=20, p_a=0.25
PSO	Np=20, ϕ_1 = 1, ϕ_2 = 1.5, ϕ_3 = 2, γ_1 = 1, γ_2 = 1
GA	Np=20, mutation Rate = 0.1, crossover Rate = 0.4
BA	Np=20, loudness = 0.5, pulse rate = 0.5,f_{min}=0, f_{max}=2
FPA	Np=20, probabibility switch = 0.8
BOA	Np=20, probabibility switch=0.8, modular modality=0.01, power exponent=0.1
ACO	Np=20, pheromone decay rate = 0.9

4.2. Comparison of cCS algorithms

According to Algorithm 2 and Table 1, this paper compares the different versions of the proposed cCS algorithm. Table 5 shows the differences in memory usage and function evaluation times for the six versions of the cCS algorithm. *Dim* represents the dimension of the problem in Table 5. According to Algorithm 2, in the D-dimensional problem, the six versions of the algorithm will retain multiple D-dimensional variables, including x_1, x_2, x_3, $best$, and PV vectors. However, due to the different formulas used in cCS_V1 and cCS_V3, two additional D-dimensional variables x_{t1} and x_{t2} are required. Although different algorithms use different memory spaces, during the algorithm execution, when the total number of iterations is the same, the function evaluation times of different versions of the cCS algorithm are the same.

Table 5: Memory and function evaluation times of different versions of cCS algorithm

Name	Dim	Memory size	Variables	Function calls
cCS_V1	D	8×D	$x_1, x_2, x_3, best, \mu, \delta, x_{t1}, x_{t2}$	2×iteration
cCS_V2	D	6×D	$x_1, x_2, x_3, best, \mu, \delta$	2×iteration
cCS_V3	D	8×D	$x_1, x_2, x_3, best, \mu, \delta, x_{t1}, x_{t2}$	2×iteration
cCS_V4	D	6×D	$x_1, x_2, x_3, best, \mu, \delta$	2×iteration
cCS_V5	D	6×D	$x_1, x_2, x_3, best, \mu, \delta$	2×iteration
cCS_V6	D	6×D	$x_1, x_2, x_3, best, \mu, \delta$	2×iteration

This paper uses the proposed benchmark functions to compare six different versions of the cCS algorithm. Each algorithm runs 51 times on each benchmark function. All algorithms have the same number of function evaluations. The test results of the six algorithms are shown in Table 6. Win indicates how many benchmark functions the algorithm has achieved the best results. *Average value* represents the average of the results obtained by the algorithm on all benchmark functions. *Function calls* is the number of function evaluations of the algorithm during each test. *Average time* is used to represent the average execution time of the algorithm in all benchmark functions, which is used to measure the complexity of the algorithm.

Table 6: Performance comparison results of different versions of cCS

Name	Win	Average value	Function calls	Average time
cCS_V1	0	4.14E+09	10240	22.15121
cCS_V2	12	2.11E+08	10240	9.21137
cCS_V3	0	2.79E+08	10240	15.8338
cCS_V4	12	2.67E+08	10240	9.204615
cCS_V5	0	4.17E+09	10240	9.351995
cCS_V6	2	3.05E+08	10240	9.299054

According to the data in Table 6, cCS_V2 and cCS_V4 have the best results, and both algorithms have achieved the best results in 12 benchmark test functions. cCS_V6 combines the characteristics of cCS_V2 and cCS_V4, but the final result is not as good as cCS_V2 and cCS_V4. For *Average value* in Table 6, the results of cCS_V2 and cCS_V4 are similar. The effect of cCS_V3 is the closest to the previous two, but the algorithm consumes more time. The time complexity of cCS_V5 and cCS_V6 is similar to cCS_V1 and cCS_V2, but the results obtained are not as suitable. Based on the comparison above, the version of the cCS algorithm used in the following improvements and tests is cCS_V2.

Different versions of the cCS algorithm use Equation 1 to generate x_2, but the local search formula is different. As shown in Table 1, cCS_V2 and cCS_V4, which can obtain better results, use the global optimal solution to update x_3. According to the data in Table 6 and related literature, Levy flight helps jump out of the local optimum. But like cCS_V3 and cCS_V1, the two solutions randomly generated using PV have no guiding significance to the current solution. Although the randomness of the exploration is increased, it does not guarantee that it will gradually approach the potentially better area. cCS_V2 and cCS_V4 combine the characteristics of random exploration and convergence to global optimum, so they can achieve better results compared to other versions of cCS.

4.3. Comparison with other compact algorithms

In this section, the proposed cCS algorithm is compared with several other compact algorithms. This paper chooses the cPSO, cBA, cABC and cDE algorithms for comparison. All algorithms were run five times on each benchmark function, while the means, standard deviations, and optimal values are recorded. In the following description and testing process, cCS means cCS_V2. The parameter settings of other compact algorithms for comparison are shown in Table 4. At the same time, the overall performance of each algorithm compared with the cCS algorithm was measured at a significant level $\alpha = 0.05$ under the Wilcoxon's sign rank test.

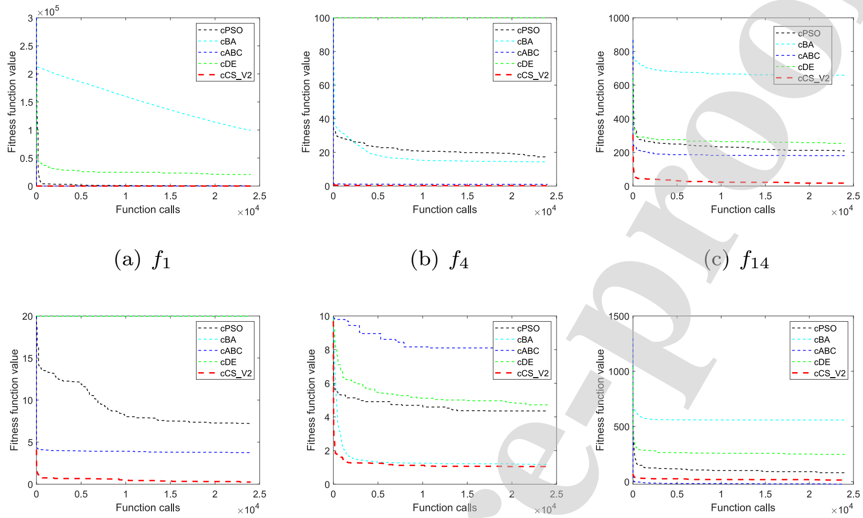
Table 7 shows the comparison test results of cCS algorithm with other compact algorithms. The test results of each algorithm on each function are compared with the results of the cCS algorithm. The symbol ($<$) indicates that the algorithm does not perform as well as the cCS algorithm on the current function. The symbol ($>$) indicates that the cCS algorithm performs poorly. The symbol ($=$) indicates that the two algorithms perform similarly on the current benchmark function. The comparison results on all benchmark functions are summarized in the last row of the table.

According to the data in Table 7, compared with the other four compact algorithms, the proposed cCS algorithm performs better on the selected benchmark test function. Compared with the cPSO algorithm, the proposed cCS algorithm achieves better results in 19 benchmark functions, and the performance on 6 functions is not as good as the cPSO algorithm. Compared with the cCS algorithm, the cBA algorithm achieves better results on 8 functions. For the cABC algorithm, the proposed cCS algorithm performs better on 18 benchmark test functions, and the performance on functions f_{11}, f_{17}, and f_{21} are the same. Compared with the cCS algorithm, the cDE algorithm achieves better results on the 6 benchmark functions, and the effect on other functions is not as good as the cCS algorithm.

In order to further compare the effects of different compact algorithms, this paper uses the convergence curves of the algorithms to evaluate the degree of

Table 7: Comparison of the cCS with the cPSO, cBA, cABC and cDE.Each algorithm is measured under Wilcoxon's signed rank test with the significant level $\alpha = 0.05$.

Function	cPSO		cBA		cABC		cDE		cCS_V2
f_1	5.036E+02	(<)	9.937E+04	(<)	1.002E+01	(<)	2.055E+04	(<)	1.593E-01
f_2	1.233E+01	(<)	2.086E+21	(<)	1.330E+01	(<)	5.310E+01	(<)	8.627E-01
f_3	3.626E+03	(<)	2.013E+07	(<)	1.180E+01	(<)	4.045E+04	(<)	4.604E-01
f_4	1.735E+01	(<)	1.442E+01	(<)	1.004E+00	(<)	1.000E+02	(<)	1.899E-01
f_5	5.560E+04	(<)	1.304E+04	(<)	1.515E+03	(<)	3.766E+07	(<)	3.708E+01
f_6	8.312E+02	(<)	6.976E+04	(<)	1.080E+01	(<)	2.256E+04	(<)	0.000E+00
f_7	2.545E+00	(<)	2.297E-01	(>)	7.854E+01	(<)	1.159E+01	(<)	1.089E+00
f_8	1.443E+02	(<)	6.525E+01	(<)	1.380E+01	(<)	2.353E+02	(<)	3.687E+00
f_9	9.129E+01	(<)	6.195E+00	(<)	1.337E+02	(<)	3.013E+03	(<)	1.137E+00
f_{10}	-5.096E+72	(>)	-1.013E+02	(>)	1.289E+00	(<)	-1.214E+29	(>)	-1.876E-01
f_{11}	0.000E+00	(<)	0.000E+00	(<)	9.950E-01	(=)	0.000E+00	(<)	9.950E-01
f_{12}	5.817E+02	(<)	3.555E+01	(<)	4.489E+02	(<)	2.430E+05	(<)	2.259E+00
f_{13}	-6.696E+84	(>)	-1.255E+04	(>)	-4.627E+02	(>)	-7.812E+89	(>)	-1.091E+01
f_{14}	2.085E+02	(<)	6.590E+02	(<)	1.809E+02	(<)	2.536E+02	(<)	1.734E+01
f_{15}	7.184E+00	(<)	1.996E+01	(<)	3.734E+00	(<)	1.997E+01	(<)	2.553E-01
f_{16}	5.054E+00	(<)	1.329E+03	(<)	3.802E-01	(<)	4.573E+02	(<)	5.244E-03
f_{17}	4.470E+01	(<)	1.984E+02	(<)	4.577E-01	(=)	1.451E+02	(<)	5.400E-01
f_{18}	5.009E+01	(<)	2.133E+03	(<)	2.973E+00	(<)	8.450E+02	(<)	2.749E+00
f_{19}	4.355E+00	(<)	1.176E+00	(<)	7.931E+00	(<)	4.716E+00	(<)	1.050E+00
f_{20}	8.109E+01	(<)	5.587E+02	(<)	-2.264E+01	(>)	2.465E+02	(<)	1.507E+01
f_{21}	3.378E-11	(=)	3.262E-11	(>)	3.378E-11	(=)	3.337E-11	(=)	3.378E-11
f_{22}	9.514E-13	(<)	6.186E-03	(<)	8.154E-09	(<)	1.874E-10	(<)	-4.695E-02
f_{23}	-7.999E+02	(>)	-1.025E+03	(>)	-4.323E+02	(>)	-8.344E+02	(>)	-1.216E+02
f_{24}	4.607E+00	(<)	2.748E+01	(<)	3.934E-01	(<)	1.468E+01	(<)	9.991E-02
f_{25}	1.038E+00	(>)	4.608E-01	(>)	6.300E-01	(>)	3.688E-01	(>)	1.641E+01
f_{26}	3.690E+07	(<)	7.278E+11	(<)	5.855E+03	(>)	8.331E+10	(<)	8.984E+03
</=/>	19/1/6		18/0/8		18/3/5		19/1/6		-



(a) f_1 (b) f_4 (c) f_{14}

(d) f_{15} (e) f_{19} (f) f_{20}

Figure 3: Comparison results of convergence performance of compact algorithms.

convergence.

As shown in Figure 3, all algorithms are performing the same number of function evaluations, and the horizontal axis is the number of function evaluations, that is, *Function calls*, and the vertical axis is the fitness value of different algorithms. Because the convergence effects of different algorithms on the same function may be similar, it is challenging to distinguish the different degrees of convergence of different algorithms, so this paper selects several images with scattered curves for display.

In Figure 3, the proposed cCS algorithm can quickly find the optimal region on the functions f_1, f_4, which is significantly better than the cBA algorithm. In the functions f_{14}, f_{15}, and f_{19}, the cCS algorithm has achieved better convergence results than other algorithms, but the global search capability in the function f_{20} is not as good as the cABC algorithm.

4.4. Comparison with the original CS algorithm

This section compares the performance of the proposed cCS algorithm with the original CS algorithm and ACS algorithm. At the same time, it also uses the pcCS algorithm to compare with the cCS, CS, and ACS algorithms.

Although the number of virtual populations is 200 according to Table 4, in fact, the number of populations used by the cCS algorithm is 1. Compared with the CS and ACS algorithms, the cCS algorithm uses less memory and performs fewer function evaluations in one execution. At the same time, the pcCS algorithm obtained by using the parallel communication strategy to improve the cCS algorithm can be divided into multiple groups, so its memory usage is uncertain. According to Table 4, pcCS uses three groups. In order to better compare with CS algorithm, this paper lists the memory usage of related algorithms and the comparison of function evaluation times, as shown in Table 8.

Table 8: Comparison of memory usage and function evaluation times of cCS, pcCS and CS

Name	Population size	Dim	Memory size	Function calls
cCS_V2	1	D	6×D	2×iteration
pcCS	3	D	12×D	9×iteration
CS	N	D	2×N×D	2×N×iteration

Next, the performance of the related algorithm is tested on the benchmark functions, where N is 20, and the test results are shown in Table 9.

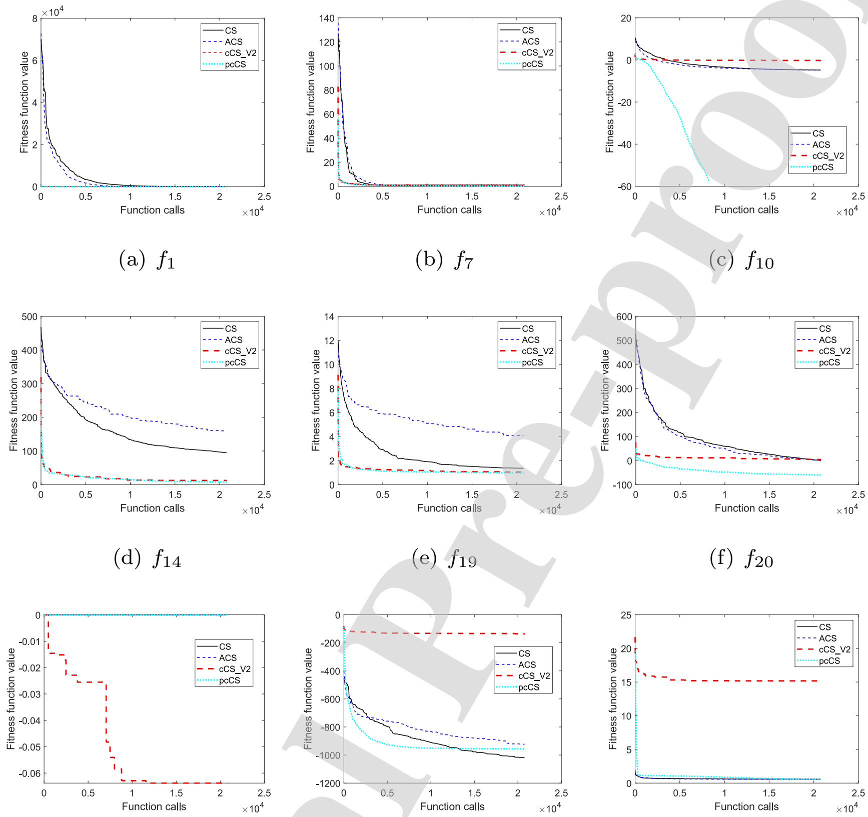
According to the results in Table 9, the cCS algorithm is compared with the CS and ACS algorithms. Compared with the CS algorithm, better results are obtained on 17 benchmark functions. The cCS algorithm achieved better results than the ACS algorithm on 16 benchmark functions, but the performance on the benchmark functions f_7, f_9, f_{10},f_{12}, f_{17}, f_{20}, f_{21}, f_{23}, and f_{25} was not good.

In the implementation of cCS algorithm and CS, ACS algorithm comparison, the paper also compares the performance of pcCS algorithm with the above three algorithms. Compared with the cCS algorithm in Table 9, the pcCS algorithm performs better overall. Compared with CS and ACS algorithms, pcCS algorithm achieves better results while using less memory space. Compared with cCS algorithm, although pcCS algorithm uses more memory space, it still does not get better results than cCS on the benchmark functions f_1, f_2, f_{22}.

Table 9: Comparison of the cCS, pcCS with the CS and ACS. Each algorithm is measured under Wilcoxon's signed rank test with the significant level $\alpha = 0.05$ in comparison with the cCS and pcCS.

Function	CS			ACS			cCS		pcCS
f_1	8.2055E+00	(<)	(<)	7.8677E-01	(<)	(<)	7.0573E-02	(>)	1.7937E-01
f_2	4.2063E+00	(<)	(<)	5.4301E+00	(<)	(<)	8.6082E-01	(>)	1.0642E+00
f_3	1.3098E+03	(<)	(<)	8.2056E+02	(<)	(<)	2.3109E-01	(<)	6.4009E-01
f_4	1.3827E+01	(<)	(<)	4.4075E+00	(<)	(<)	1.1854E-01	(<)	1.3895E-01
f_5	2.0556E+03	(<)	(<)	1.4579E+02	(<)	(<)	4.2498E+01	(<)	4.0630E+01
f_6	3.1200E+01	(<)	(<)	1.6000E+00	(<)	(<)	0.0000E+00	(=)	0.0000E+00
f_7	1.2487E-01	(>)	(>)	1.5841E-01	(>)	(>)	1.0964E+00	(<)	3.1196E-01
f_8	1.3597E+02	(<)	(<)	1.0358E+02	(<)	(<)	4.6997E+00	(<)	2.1906E+00
f_9	9.5658E-01	(>)	(<)	1.2588E+00	(>)	(<)	1.5503E+00	(<)	1.0622E+00
f_{10}	-4.7868E+00	(>)	(<)	-4.7923E+00	(>)	(<)	-2.5261E-01	(<)	6.5535E+04
f_{11}	9.9502E-01	(=)	(<)	9.9502E-01	(=)	(<)	9.9502E-01	(<)	0.0000E+00
f_{12}	-8.0834E+03	(>)	(>)	-7.9968E+03	(>)	(>)	-1.1837E+01	(<)	-3.2014E+03
f_{13}	9.4979E+01	(<)	(<)	1.5894E+02	(<)	(<)	1.2478E+01	(<)	7.9097E+00
f_{14}	5.6008E+00	(<)	(<)	2.5931E+00	(<)	(<)	4.3924E-01	(<)	4.0902E-01
f_{15}	1.0471E+00	(<)	(<)	6.8408E-01	(<)	(<)	3.0600E-03	(<)	6.9230E-03
f_{16}	2.7580E+01	(<)	(<)	1.9034E+01	(<)	(<)	4.3669E-01	(<)	1.4608E-01
f_{17}	1.2834E+01	(<)	(<)	5.2364E-01	(>)	(>)	2.7109E+00	(<)	1.7302E+00
f_{18}	1.2182E+01	(<)	(<)	7.9493E+00	(<)	(<)	2.6056E+00	(<)	2.0716E+00
f_{19}	1.3687E+00	(<)	(<)	4.0709E+00	(<)	(<)	1.0440E+00	(<)	9.7830E-01
f_{20}	-6.3805E-01	(>)	(<)	2.2287E+00	(>)	(<)	4.5301E+00	(<)	-5.9292E+01
f_{21}	1.8788E-11	(>)	(>)	1.5174E-11	(>)	(>)	3.3780E-11	(<)	2.6734E-11
f_{22}	4.6360E-13	(<)	(<)	1.7515E-12	(<)	(<)	-6.3803E-02	(>)	1.4774E-13
f_{23}	-1.0176E+03	(>)	(>)	-9.2287E+02	(>)	(=)	-1.3594E+02	(<)	-9.5554E+02
f_{24}	3.1170E+00	(<)	(<)	1.9027E+00	(<)	(<)	9.9936E-02	(=)	9.9909E-02
f_{25}	5.9314E-01	(>)	(=)	5.5123E-01	(>)	(=)	1.5183E+01	(<)	5.5237E-01
f_{26}	1.0000E+10	(<)	(<)	1.0000E+10	(<)	(<)	8.9129E+03	(<)	2.3289E+03
</=/>	17/1/8			16/1/9			-		
</=/>	20/1/5			20/2/4			21/2/3		-

Next, this paper compares the convergence curves of the two improved algorithms and CS and ACS algorithms, as shown in Figure 4. Compared with the CS and ACS algorithms, the pcCS algorithm has a faster convergence speed, but the performance on some benchmark functions is not as good as the CS and ACS algorithms, such as the benchmark functions f_{22} and f_{23}. In the benchmark functions f_1, f_7, f_{14}, and f_{19}, the convergence effect of the cCS algorithm and the pcCS algorithm is similar. In f_{22}, the pcCS algorithm with a parallel communication strategy is not as effective as the cCS algorithm. In f_{23} and f_{25},



(a) f_1 (b) f_7 (c) f_{10}

(d) f_{14} (e) f_{19} (f) f_{20}

(g) f_{22} (h) f_{23} (i) f_{25}

Figure 4: Comparison results of convergence performance of cCS, pcCS, CS and ACS algorithms.

pcCS algorithm is improved compared with cCS algorithm, but the effect is not as good as CS and ACS algorithms.

4.5. Comparison of pcCS with different groups

The pcCS algorithm proposed in this paper uses the optimal solution obtained by the cCS algorithm as an individual to execute a parallel communication strategy. The previous section compares the comparison results of the pcCS algorithm with related algorithms when the number of groups is 3, so this section compares the performance and memory usage of the pcCS algorithm when the number of groups is different.

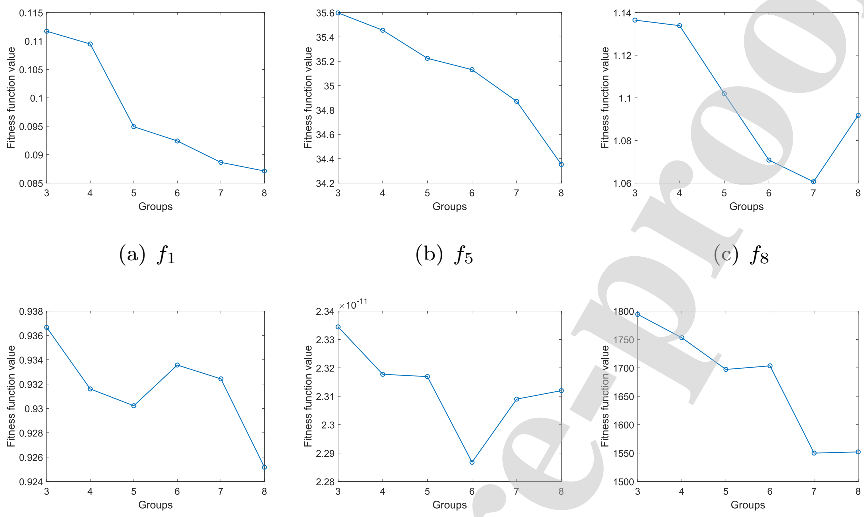
In this paper, the test functions are used to test the pcCS algorithm in different groups. The different algorithms are run 51 times in each function and the average value is retained. Since the number of individuals generated in the algorithm is different for different groups, the number of evaluations in each iteration In Table 10, *Eva* represents the number of function evaluations per iteration, when the number of groups is n, then the cCS algorithm of each group will evaluate the generated x_3 and x_1, and the number of evaluations in this part is $n \times 2$. In addition, n optimal solutions will be evaluated in the parallel communication stage, so the number of algorithm evaluations in each iteration of the algorithm is $n \times 2 + n$.

When the number of groups is different, the number of individuals generated is different, and the memory used is also different. When generating n groups, each group needs to save its own PV, including μ and δ. In addition, it is also necessary to save the optimal solutions of the n groups. The x_1, x_2 and x_3 generated by each group are temporary variables and can be shared, so the memory size is $n \times 2 + 3 + n$. This paper compares the pcCS algorithms with different group size. The total number of function evaluations is the same. The results are shown in Table 10.

Table 10: Comparison of pcCS algorithm with different groups

Groups	Function calls	Memory size	Eva	Average value	Win
3	20480	3×2+3+3	3×2+3	2415.073	3
4	20480	4×2+3+4	4×2+4	2415.641	1
5	20480	5×2+3+5	5×2+5	2414.238	0
6	20480	6×2+3+6	6×2+6	2411.135	4
7	20480	7×2+3+7	7×2+7	2399.846	7
8	20480	8×2+3+8	8×2+8	2409.386	11

The *Average value* in Table 10 represents the average value of the algorithm's calculation results on all functions, and the overall calculation results are better as the number of groups increases. Win means that when set to the



(a) f_1 (b) f_5 (c) f_8

(d) f_{19} (e) f_{21} (f) f_{26}

Figure 5: Comparison results of pcCS with different group sizes in different functions.

current number of groups, the algorithm has obtained the optimal value in how many test functions.

In Table 10, the result when the number of groups is 4 is slightly worse than the result when the number of groups is 3, and the result when the number of groups is 8 is also slightly worse than the result when the number of groups is 7. But in general, under the same function evaluation times, as the number of groups increases, the performance of the algorithm also gradually improves. According to the data in Table 10, the results obtained when the number of groups is different are shown in Figure 5.

As shown in Figure 5, when the group size of pcCS algorithms increases, the results on functions f_1, f_5, and f_{26} become better, but the increase in the number of groups does not guarantee better results on other functions. In function f_8, the best result can be obtained when the number of groups is 7. In function f_{21}, the best result can be obtained when the number of groups is 6, and when the number of groups is 8, the result is not as good as when the number of groups is 7. In f_{19}, although the best result was obtained when the number of groups was 8, the result when the number of groups was 4 was still better than the result when the number of groups was 7.

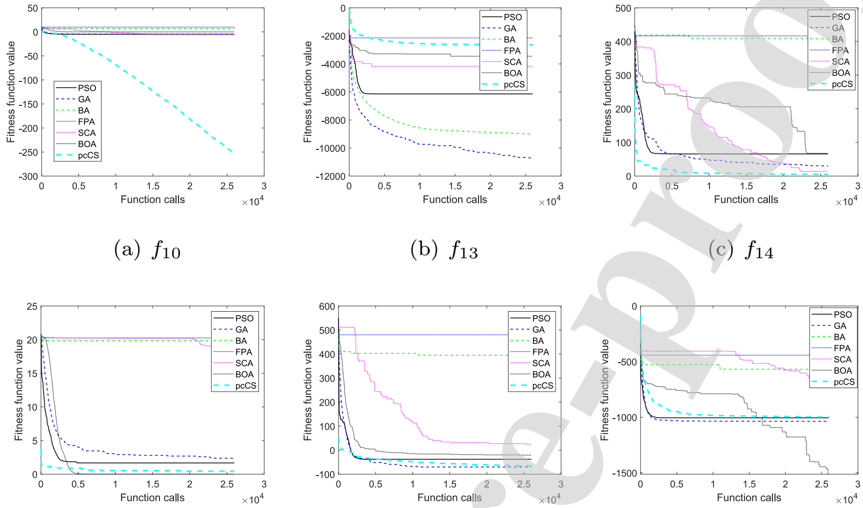
4.6. Comparison with common optimization algorithms

This section compares the effects of the proposed pcCS algorithm with other common algorithms. All settings of the pcCS algorithm are the same as in the previous section. The parameter settings and population size of related algorithms are shown in Table 4. The comparison result with pcCS algorithm is shown in Table 11.

Table 11: Comparison of the pcCS with the PSO, GA, BA, FPA, SCA and BOA. Each algorithm is measured under Wilcoxon's signed rank test with the significant level $\alpha = 0.05$ in comparison with the pcCS.

Function	PSO		GA		BA		FPA		SCA		BOA		pcCS
f_1	1.005E-02	(>)	3.200E+01	(<)	3.458E+04	(<)	4.788E+04	(<)	6.087E-03	(>)	1.211E-12	(>)	1.294E-01
f_2	1.764E-01	(>)	1.297E+00	(<)	9.162E+06	(<)	2.792E+11	(<)	1.293E-05	(>)	4.487E-10	(>)	1.109E+00
f_3	3.039E+02	(<)	2.310E+03	(<)	1.027E+05	(<)	1.691E+05	(<)	2.775E+03	(<)	1.256E-12	(>)	2.774E-01
f_4	1.025E+00	(<)	6.387E+00	(<)	7.458E+01	(<)	8.079E+01	(<)	1.846E+01	(<)	6.534E-10	(>)	1.526E-01
f_5	8.680E+01	(<)	9.463E+02	(<)	1.283E+08	(<)	1.948E+08	(<)	6.483E+01	(<)	2.894E+01	(>)	3.737E+01
f_6	2.333E+00	(<)	5.067E+01	(<)	3.867E+04	(<)	6.726E+04	(<)	0.000E+00	(=)	0.000E+00	(=)	0.000E+00
f_7	2.195E-02	(>)	3.531E-02	(>)	8.684E+01	(<)	1.032E+02	(<)	1.302E-02	(>)	1.827E-03	(>)	2.513E-01
f_8	1.407E+00	(>)	1.622E+01	(<)	4.951E+02	(<)	1.455E+08	(<)	1.106E+01	(<)	1.065E-12	(>)	1.875E+00
f_9	5.920E-05	(>)	3.575E+00	(<)	5.567E+03	(<)	9.747E+03	(<)	2.139E-04	(>)	1.029E-12	(>)	6.883E-01
f_{10}	-4.997E+00	(<)	-4.728E+00	(<)	6.898E+00	(<)	9.840E+00	(<)	-4.849E+00	(<)	4.021E-10	(<)	-2.521E+02
f_{11}	9.952E-01	(<)	9.950E-01	(<)	0.000E+00	(=)	9.968E-01	(<)	9.958E-01	(<)	9.952E-01	(<)	0.000E+00
f_{12}	1.081E+00	(>)	1.982E+01	(<)	4.722E+05	(<)	1.695E+06	(<)	1.165E+00	(>)	9.920E-01	(>)	1.204E+00
f_{13}	-6.140E+03	(>)	-1.071E+04	(>)	-8.998E+03	(>)	-2.143E+03	(<)	-4.166E+03	(>)	-3.451E+03	(>)	-2.628E+03
f_{14}	6.567E+01	(<)	2.992E+01	(<)	4.097E+02	(<)	4.167E+02	(<)	1.374E+01	(<)	6.754E+01	(<)	4.798E+00
f_{15}	1.676E+00	(<)	2.376E+00	(<)	1.986E+01	(<)	2.025E+01	(<)	1.868E+01	(<)	6.702E-10	(>)	4.336E-01
f_{16}	5.112E-02	(<)	1.232E+00	(<)	3.556E+02	(<)	6.318E+02	(<)	4.237E-01	(<)	4.846E-13	(>)	7.721E-03
f_{17}	3.484E-02	(>)	6.057E-02	(>)	3.149E+02	(<)	5.269E+02	(<)	6.669E-01	(<)	4.901E-01	(<)	1.132E-01
f_{18}	2.276E-01	(<)	1.653E+00	(<)	1.312E+03	(<)	2.216E+03	(<)	2.511E+00	(<)	2.996E+00	(<)	9.958E-01
f_{19}	1.000E+00	(<)	1.117E+00	(<)	8.470E+00	(<)	9.546E+00	(<)	4.598E+00	(<)	8.135E+00	(<)	9.685E-01
f_{20}	-3.791E+01	(<)	-7.012E+01	(>)	3.948E+02	(<)	4.799E+02	(<)	2.474E+01	(<)	-2.066E+01	(<)	-6.550E+01
f_{21}	5.081E-12	(>)	9.956E-12	(>)	3.197E-11	(<)	4.464E-07	(<)	4.202E-10	(<)	4.569E-07	(<)	2.457E-11
f_{22}	6.637E-18	(>)	1.650E-13	(<)	9.538E-09	(<)	3.583E-08	(<)	3.904E-10	(<)	1.229E-11	(<)	8.013E-14
f_{23}	-1.005E+03	(>)	-1.037E+03	(>)	-5.687E+02	(<)	-4.424E+02	(<)	-6.471E+02	(<)	-1.512E+03	(>)	-9.989E+02
f_{24}	1.367E+00	(<)	1.567E+00	(<)	2.113E+01	(<)	2.631E+01	(<)	4.688E-01	(<)	3.322E-01	(<)	9.990E-02
f_{25}	7.699E-01	(<)	2.994E-01	(>)	1.044E+00	(<)	1.743E+00	(<)	4.045E-01	(<)	7.080E-01	(<)	3.281E-01
f_{26}	1.056E-01	(>)	1.129E+05	(<)	9.873E+10	(<)	1.930E+11	(<)	5.236E+03	(<)	2.623E+03	(<)	2.424E+03
</=/>	14/0/12		19/0/7		24/1/1		26/0/0		19/1/6		12/1/13		-

According to the data in Table 11, compared with the PSO algorithm, the pcCS algorithm has better results than the PSO algorithm in 14 functions, but



(a) f_{10} (b) f_{13} (c) f_{14}

(d) f_{15} (e) f_{20} (f) f_{23}

Figure 6: Comparison results of convergence performance of common optimization algorithms.

the effect on other benchmark functions is not as good as PSO. Especially in the f_{26} function, the effect of pcCS algorithm is obviously inferior to that of PSO algorithm. Compared with GA algorithm, the results obtained by pcCS algorithm on 7 functions are not good.

The pcCS algorithm obtained better results than the BA algorithm on 24 benchmark functions. The same result is obtained in f_{11}, the result in f_{13} is not as good as the BA algorithm. Compared to the FPA algorithm, pcCS achieves better results in all benchmark functions. In the benchmark function f_6, the algorithms SCA, BOA and pcCS all found the same solution. The pcCS algorithm achieves better results than the SCA algorithm on most benchmark functions, but the results in the functions f_1, f_2, f_7, f_9, f_{12}, and f_{13} are not as good as the SCA algorithm. Compared with the newer BOA algorithm, the pcCS algorithm cannot achieve more competitive results and is better than the BOA algorithm on only 12 benchmark functions. In half of the benchmark functions, the pcCS algorithm does not get better results than the BOA algorithm, and the search ability of the BOA algorithm is stronger in these functions.

Next, the paper evaluates the convergence ability of different algorithms, and the results are shown in Figure 6. The pcCS algorithm can obtain better results in the benchmark functions f_{10}, f_{14}, but it is not the most competitive in other functions. The performance of the pcCS algorithm proposed on the functions f_{13}, f_{15}, f_{20}, and f_{23} is not as good as other algorithms, but the convergence speed on the functions f_{13}, f_{15} is faster. Compared with the BOA algorithm, the pcCS algorithm still has the problem of being easily trapped into a local optimum.

5. Application in 3D path planning

For three-dimensional path planning mentioned in Section 2.2, using the bio-inspired algorithms to plan the three-dimensional path, firstly, the three-dimensional environment needs to mesh. Then, the obstacles are modeled so that the algorithm can perform path planning, and the process is simplified by dividing the plane. In general, the algorithm needs to calculate the optimal path from the starting point to the destination point in the 3D Scenes.

This paper describes the method of modeling three-dimensional obstacles in Section 2.2, which is used to carry out underwater three-dimensional path planning problem for underwater submersibles. This paper constructs a three-dimensional virtual space, in which the deepest point height in the three-dimensional terrain is 0, the length and width are both 21km, and the height is 2km. According to the method of dividing the three-dimensional space according to the Section 2.2, the virtual space is divided into a three-dimensional grid with a length and a width of 21 and a height of 10. The start position and the destination position of the whole path are a point in the first plane Π_1 and a point in the last plane Π_n . For the algorithm, the starting point is fixed to the destination point. The indexes of the introduced 3D path planning model are shown in Table 12.

For the introduction in Section 1, there are several common bio-inspired algorithms for 3D path planning problems. This paper uses the classical GA,

Table 12: The index of 3D path planning scenario.

Content	Detail
Actual simulated distance	21km×21km×2km
Meshing status	21×21×10
Number of waypoints	21
scenario 1 start and end point	(12,5),(8,5)
scenario 2 start and end point	(10,5),(8,5)

PSO and classical ACO to compare with the proposed pcCS algorithms. The CS is also compared to pcCS. Algorithms ACO, CS, PSO and GA all have a population of 20, the pcCS algorithm has a group number of 7, and the remaining parameters are the same as in Table 4 without modification. Since the algorithm ACS uses an adaptive strategy, the change of the solution becomes smaller and smaller with the increase of the number of iterations, which is not enough to change the current path. Therefore, this paper does not use the ACS algorithm to deal with the path planning problem. In addition, the algorithm initialization process of pcCS adds the pre-calculation process of the solutions in all groups to ensure a better initial path. However, in order to deal with the proposed 3D path planning problem, the implementation of the proposed pcCS algorithm has been modified in this paper.

During the execution of the pcCS algorithm, in order to sample in discrete space, the algorithm first needs to use Equation 10 to map the solution generated by PV to the current space. After the mapping is completed, the value is discretized using rounding. In the initial process, the mean μ in the PV is set to the midpoint of each dimension, and the standard deviation δ is still set to 10. According to Algorithm 3, during the parallel communication phase of the algorithm, the optimal solution of the group is perturbed in part3. The perturbation method is to add a perturbation term. In order to be more suitable for the current problem, this paper modified the disturbance operation to the following format:

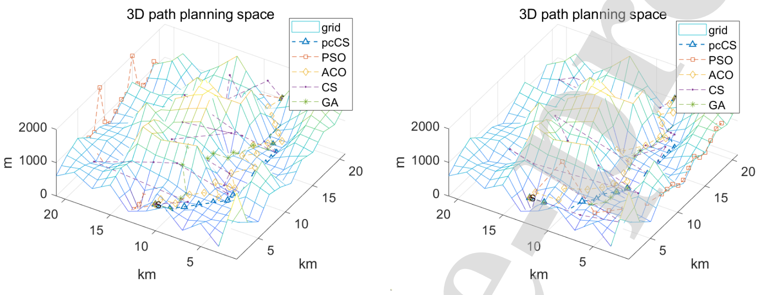
$$x_i = rem(round(x_i + \gamma \times k), M) \quad (11)$$

In Equation 11, k represents a random number in the range 0 to 1 generated using a uniform distribution. M is the maximum value of the current dimension. For the y-axis, M is 21, and for the z-axis, M is 10. x_i represents the value of the i-th dimension. The $round()$ function means rounding the values in parentheses, the $rem()$ function represents the remainder operation. γ represents a self-incrementing integer with a step size of 1, its initial value is 2, and its maximum value is M. When it increases to M, it will be re-initialized to 2. When x_i becomes 0 due to the remainder, it will be reset to the midpoint of the dimension.

For all algorithms, a particle is used to represent a path. Since there are a total of 21 planes, that is, 21 path points, each path point has two dimensions, so each particle is a 42-dimensional vector. At the same time, because the range of the two dimensions of each path point is different, it is necessary to constrain each of its dimensions in the calculation process.

According to the Algorithm 3 and the improved perturbation Equation 11, the pcCS algorithm can continuously try to find a better location in each dimension during the perturbation. Based on the above introduction, this paper uses three-dimensional path planning problems to compare the proposed algorithms, and the results are shown in Table 13. Table 13 shows that the pcCS algorithm can find a shorter path than other algorithms, and the pcCS algorithm execution effect is more stable.

In this paper, the pcCS algorithm and other algorithms are tested in the built 3D environment. The path planning calculation results are shown in Figure 7. Since the path point of the algorithm planning is in a fixed grid, the limitation of the algorithm is to avoid the obstacle at the position of the grid point, but the path between the two security grid points is still likely to touch the obstacle, this is where we need to improve next. This paper tests the execution of the algorithm at different starting points. The average performance trend calculated

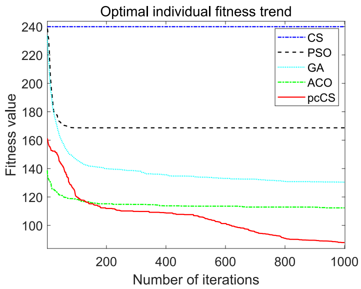


(a) scenario 1 (b) scenario 2

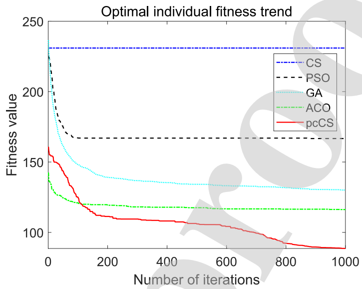
Figure 7: 3D path planning results at different starting points.

Table 13: Comparison of the pcCS with the GA, PSO, ACO and CS.

Approaches	Mean	Best	Std.
CS	2.40E+02	2.20E+02	9.68E+00
PSO	1.69E+02	1.51E+02	1.27E+01
GA	1.30E+02	1.11E+02	1.22E+01
ACO	1.12E+02	1.07E+02	2.93E+00
pcCS	8.79E+01	8.52E+01	2.49E+00



(a) scenario 1



(b) scenario 2

Figure 8: Comparisons of the pcCS with the PSO,GA,ACO and CS for 3D path planning at different starting points.

in the case of two different starting points is not much different, which shows that the correlation algorithm has better stability for different starting points.

Figure 8 shows the comparison between the pcCS algorithm and the ACO algorithm, PSO algorithm, GA algorithm, and CS algorithm. The curve in Figure 8 is the average value taken after the algorithm is executed 10 times. Compared with PSO, GA, and CS algorithms, the execution effect of pcCS algorithm is better. Compared with the classical ACO algorithm, it can avoid falling into the local optimum and get a competitive solution. Compared with the continuous attempts of pcCS algorithm to the surrounding path points, since the CS algorithm does not have a good constraint function and does not focus on enhancing the local search ability, the effect of the CS algorithm is not ideal. The pcCS algorithm proposed in this paper can better deal with the problem of 3D path planning after being improved, but there are still problems such as insufficient accuracy and so on. It needs to be improved to obtain better results.

6. Conclusion

An improved CS algorithm based on a hybrid compact and parallel communication strategy is proposed, and the 3D path planning problem of underwater unmanned submersibles is applied. The compact scheme can be used to replace the operation of the original population with the operation of the probabilistic model, which uses the probability model to represent the distribution characteristics of the original population.The compact scheme can reduce the optimized variables, requiring less running memory. The parallel communication strategy achieves faster convergence and can find better solutions faster. This paper also increases the perturbation of the optimal values of subgroups in parallel communication strategy, which can prevent the proposed algorithm from trapping into local optimal values effectively. The implemented parallel and compact schemes show obvious advantages in the test, which is significantly improved compared to the original algorithm. This variable number of groups can bring better adaptability. As the number of groups increases, the performance is generally improved, but the effect is unstable, and more analysis and improvement are needed. In Section 4 and 5, the proposed algorithm pcCS is evaluated using a set of selected problems and 3D path planning problems. The comparison between the pcCS algorithm proposed in this paper and other algorithms indicates that the proposed improved algorithm has a nice effect on the tested problems.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (61872085), the Natural Science Foundation of Fujian Province (2018J01638) and the Fujian Provincial Department of Science and Technology (2018Y3001).

References

[1] I. BoussaïD, J. Lepagnot, P. Siarry, A survey on optimization metaheuristics, Information Sciences 237 (2013) 82–117.

[2] D. H. Wolpert, W. G. Macready, No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation 1 (1) (1997) 67–82.

[3] T. Joyce, J. M. Herrmann, A review of no free lunch theorems, and their implications for metaheuristic optimisation, in: Nature-inspired algorithms and applied optimization, Springer, 2018, pp. 27–51.

[4] A. Kaveh, A. Dadras, A novel meta-heuristic optimization algorithm: thermal exchange optimization, Advances in Engineering Software 110 (2017) 69–84.

[5] X.-S. Yang, S. Deb, Cuckoo Search via Lévy flights, in: 2009 World Congress on Nature Biologically Inspired Computing (NaBIC), IEEE, 2009, pp. 210–214. doi:10.1109/NABIC.2009.5393690.

[6] I. Fister, X.-S. Yang, D. Fister, Cuckoo search: a brief literature review, in: Cuckoo search and firefly algorithm, Springer, 2014, pp. 49–62.

[7] J.-S. Pan, J.-L. Liu, S.-C. Hsiung, Chaotic Cuckoo Search Algorithm for Solving Unmanned Combat Aerial Vehicle Path Planning Problems, in: Proceedings of the 2019 11th International Conference on Machine Learning and Computing, ACM, 2019, pp. 224–230.

[8] X.-S. Yang, S. Deb, Engineering Optimisation by Cuckoo Search (2010). arXiv:1005.2908.

[9] X.-S. Yang, S. Deb, Multiobjective cuckoo search for design optimization, Computers & Operations Research 40 (6) (2013) 1616–1624. doi:10.1016/j.cor.2011.09.026.

[10] M. Shehab, A. T. Khader, M. A. Al-Betar, A survey on applications and variants of the cuckoo search algorithm, Applied Soft Computing 61 (2017) 1041–1059.

[11] A. B. Mohamad, A. M. Zain, N. E. Nazira Bazin, Cuckoo search algorithm for optimization problems—a literature review and its applications, Applied Artificial Intelligence 28 (5) (2014) 419–448.

[12] S. Walton, O. Hassan, K. Morgan, M. Brown, Modified cuckoo search: a new gradient free optimisation algorithm, Chaos, Solitons & Fractals 44 (9) (2011) 710–718.

[13] G.-G. Wang, S. Deb, A. H. Gandomi, Z. Zhang, A. H. Alavi, Chaotic cuckoo search, Soft Computing 20 (9) (2016) 3349–3362.

[14] D. Rodrigues, L. A. Pereira, T. Almeida, J. P. Papa, A. Souza, C. C. Ramos, X.-S. Yang, BCS: A binary cuckoo search algorithm for feature selection, in: 2013 IEEE International Symposium on Circuits and Systems (ISCAS), IEEE, 2013, pp. 465–468.

[15] E. Valian, S. Mohanna, S. Tavakoli, Improved cuckoo search algorithm for feedforward neural network training, International Journal of Artificial Intelligence & Applications 2 (3) (2011) 36–43.

[16] M. Marichelvam, An improved hybrid Cuckoo Search (IHCS) metaheuristics algorithm for permutation flow shop scheduling problems, International Journal of Bio-Inspired Computation 4 (4) (2012) 200–205.

[17] A. Ouaarab, B. Ahiod, X.-S. Yang, Discrete cuckoo search algorithm for the travelling salesman problem, Neural Computing and Applications 24 (7-8) (2014) 1659–1669.

[18] S. I. Boushaki, N. Kamel, O. Bendjeghaba, A new quantum chaotic cuckoo search algorithm for data clustering, Expert Systems with Applications 96 (2018) 358–372.

[19] L. Yang, J. Qi, D. Song, J. Xiao, J. Han, Y. Xia, Survey of robot 3D path planning algorithms, Journal of Control Science and Engineering 2016 (2016) 5.

[20] M. N. Zafar, J. Mohanta, Methodology for path planning and optimization of mobile robots: A review, Procedia Computer Science 133 (2018) 141–152.

[21] L. Yang, J. Qi, J. Xiao, X. Yong, A literature review of UAV 3D path planning, in: Proceeding of the 11th World Congress on Intelligent Control and Automation, IEEE, 2014, pp. 2376–2381.

[22] T. T. Mac, C. Copot, D. T. Tran, R. De Keyser, Heuristic approaches in robot path planning: A survey, Robotics and Autonomous Systems 86 (2016) 13–28.

[23] Y. Volkan Pehlivanoglu, O. Baysal, A. Hacioglu, Path planning for autonomous UAV via vibrational genetic algorithm, Aircraft Engineering and Aerospace Technology 79 (4) (2007) 352–359.

[24] N. Shahidi, H. Esmaeilzadeh, M. Abdollahi, C. Lucas, Memetic algorithm based path planning for a mobile robot, in: International Conference on Computational Intelligence, Citeseer, 2004.

[25] J. L. Foo, J. Knutzon, V. Kalivarapu, J. Oliver, E. Winer, Path planning of unmanned aerial vehicles using B-splines and particle swarm optimization, Journal of Aerospace Computing, Information, and Communication 6 (4) (2009) 271–290.

[26] S. Rafi, A. Kumar, G. Singh, An improved particle swarm optimization method for multirate filter bank design, Journal of the Franklin Institute 350 (4) (2013) 757 – 769. doi:10.1016/j.jfranklin.2013.01.006.

[27] I. Hasircioglu, H. R. Topcuoglu, M. Ermis, 3-D path planning for the navigation of unmanned aerial vehicles by using evolutionary algorithms, in: Proceedings of the 10th annual conference on Genetic and evolutionary computation, ACM, 2008, pp. 1499–1506.

[28] C.-T. Cheng, K. Fallahi, H. Leung, K. T. Chi, Cooperative path planner for UAVs using ACO algorithm with gaussian distribution functions, in: 2009 IEEE International Symposium on Circuits and Systems, IEEE, 2009, pp. 173–176.

[29] I. Hassanzadeh, K. Madani, M. A. Badamchizadeh, Mobile robot path planning based on shuffled frog leaping optimization algorithm, in: 2010 IEEE International Conference on Automation Science and Engineering, IEEE, 2010, pp. 680–685.

[30] P. K. Mohanty, D. R. Parhi, Optimal path planning for a mobile robot using cuckoo search algorithm, Journal of Experimental & Theoretical Artificial Intelligence 28 (1-2) (2016) 35–52. doi:10.1080/0952813X.2014.971442.

[31] G. Wang, L. Guo, H. Duan, H. Wang, L. Liu, M. Shao, A hybrid metaheuristic DE/CS algorithm for UCAV three-dimension path planning, The Scientific World Journal 2012.

[32] Y. Gigras, K. Gupta, K. Choudhury, A comparison between Bat algorithm and Cuckoo search for path planning, International Journal of Innovative Research in Computer and Communication Engineering 3 (5) (2015) 4459–4466.

[33] M. Saraswathi, G. B. Murali, B. Deepak, Optimal Path Planning of Mobile Robot Using Hybrid Cuckoo Search-Bat Algorithm, Procedia Computer Science 133 (2018) 510 – 517, international Conference on Robotics and Smart Manufacturing (RoSMa2018). doi:10.1016/j.procs.2018.07.064.

[34] J. Wang, X. Shang, T. Guo, J. Zhou, S. Jia, C. Wang, Optimal Path Planning Based on Hybrid Genetic-Cuckoo Search Algorithm, in: 2019 6th International Conference on Systems and Informatics (ICSAI), IEEE, 2019, pp. 165–169.

[35] G.-G. Wang, H. E. Chu, S. Mirjalili, Three-dimensional path planning for UCAV using an improved bat algorithm, Aerospace Science and Technology 49 (2016) 231–238.

[36] X. Luan, F. Gong, Z. Wei, B. Yin, Y. Sun, Using Ant Colony Optimization and Cuckoo Search in AUV 3D Path Planning, in: Software Engineering and Information Technology: Proceedings of the 2015 International Conference on Software Engineering and Information Technology (SEIT2015), World Scientific, 2016, pp. 208–212.

[37] G. Wang, L. Guo, H. Duan, L. Liu, H. Wang, A bat algorithm with mutation for UCAV path planning, The Scientific World Journal 2012.

[38] S. Zhang, Y. Zhou, Z. Li, W. Pan, Grey wolf optimizer for unmanned combat aerial vehicle path planning, Advances in Engineering Software 99 (2016) 121–136.

[39] E. Tuba, I. Strumberger, D. Zivkovic, N. Bacanin, M. Tuba, Mobile robot path planning by improved brain storm optimization algorithm, in: 2018 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2018, pp. 1–8.

[40] M. Subotic, M. Tuba, N. Bacanin, D. Simian, Parallelized Cuckoo Search Algorithm for Unconstrained Optimization, in: Proceedings of the 5th WSEAS Congress on Applied Computing Conference, and Proceedings of the 1st International Conference on Biologically Inspired Computation, BICA'12, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 2012, p. 151–156.

[41] J.-F. Chang, J. F. Roddick, J.-S. Pan, S.-C. Chu, A parallel particle swarm optimization algorithm with communication strategies, Journal of Information Science and Engineering (2005) 809–818.

[42] F. Neri, E. Mininno, G. Iacca, Compact Particle Swarm Optimization, Information Sciences 239 (2013) 96–121. `doi:10.1016/j.ins.2013.03.026`.

[43] G. R. Harik, F. G. Lobo, D. E. Goldberg, The compact genetic algorithm, IEEE Transactions on Evolutionary Computation 3 (4) (1999) 287–297.

[44] E. Mininno, F. Neri, F. Cupertino, D. Naso, Compact differential evolution, IEEE Transactions on Evolutionary Computation 15 (1) (2010) 32–54.

[45] J.-S. Pan, T.-K. Dao, et al., A Compact Bat Algorithm for Unequal Clustering in Wireless Sensor Networks, Applied Sciences 9 (10) (2019) 1973.

[46] X. Xue, J.-S. Pan, A Compact Co-Evolutionary Algorithm for sensor ontology meta-matching, Knowledge and Information Systems 56 (2) (2018) 335–353.

[47] T.-T. Nguyen, J.-S. Pan, T.-K. Dao, An Improved Flower Pollination Algorithm for Optimizing Layouts of Nodes in Wireless Sensor Network, IEEE Access 7 (2019) 75985–75998.

[48] T.-K. Dao, T.-S. Pan, T.-T. Nguyen, S.-C. Chu, A Compact Articial Bee Colony Optimization for Topology Control Scheme in Wireless Sensor Networks, Journal of Network Intelligence 6 (2) (2015) 297–310.

[49] J. Abela, A Parallel Genetic Algorithm for Solving the School Timetabling Problem, Tech. rep., Division of Information Technology, C.S.I.R.O (1991).

[50] T.-K. Dao, T.-S. Pan, J.-S. Pan, et al., Parallel bat algorithm for optimizing makespan in job shop scheduling problems, Journal of Intelligent Manufacturing 29 (2) (2018) 451–462.

[51] L. Kong, J.-S. Pan, P.-W. Tsai, S. Vaclav, J.-H. Ho, A balanced power consumption algorithm based on enhanced parallel cat swarm optimization for wireless sensor network, International Journal of Distributed Sensor Networks 11 (3) (2015) 729680.

[52] S.-C. Chu, J. F. Roddick, J.-S. Pan, Ant colony system with communication strategies, Information Sciences 167 (1-4) (2004) 63–76.

[53] X. S. Yang, S. Deb, Cuckoo search: Recent advances and applications, Neural Computing and Applications 24 (1) (2014) 169–174. doi:10.1007/s00521-013-1367-1.

[54] M. Tuba, M. Subotic, N. Stanarevic, Modified cuckoo search algorithm for unconstrained optimization problems, in: Proceedings of the 5th European conference on European computing conference, World Scientific and Engineering Academy and Society (WSEAS), 2011, pp. 263–268.

[55] M. Mareli, B. Twala, An adaptive cuckoo search algorithm for optimisation, Applied Computing and Informatics 14 (2) (2018) 107 – 115. doi:10.1016/j.aci.2017.09.001.

[56] X. Li, M. Yin, Modified cuckoo search algorithm with self adaptive parameter method, Information Sciences 298 (2015) 80 – 97. doi:10.1016/j.ins.2014.11.042.

[57] Z. Cui, B. Sun, G. Wang, Y. Xue, J. Chen, A novel oriented cuckoo search algorithm to improve DV-Hop performance for cyber–physical systems, Journal of Parallel and Distributed Computing 103 (2017) 42 – 52, special Issue on Scalable Cyber-Physical Systems. doi:10.1016/j.jpdc.2016.10.011.

[58] M. K. Naik, R. Panda, A novel adaptive cuckoo search algorithm for intrinsic discriminant analysis based face recognition, Applied Soft Computing 38 (2016) 661 – 675. doi:10.1016/j.asoc.2015.10.039.

[59] X. Li, M. Yin, A particle swarm inspired cuckoo search algorithm for real parameter optimization, Soft Computing 20 (4) (2016) 1389–1413.

[60] H. Chiroma, T. Herawan, I. Fister, I. Fister, S. Abdulkareem, L. Shuib, M. F. Hamza, Y. Saadi, A. Abubakar, Bio-inspired computation: Recent development on the modifications of the cuckoo search algorithm, Applied Soft Computing 61 (2017) 149 – 173. doi:10.1016/j.asoc.2017.07.053.

[61] H. Duan, Y. Yu, X. Zhang, S. Shao, Three-dimension path planning for UCAV using hybrid meta-heuristic ACO-DE algorithm, Simulation Modelling Practice and Theory 18 (8) (2010) 1104–1115.

[62] P. Larrañaga, J. A. Lozano, Estimation of distribution algorithms: A new tool for evolutionary computation, Vol. 2, Springer Science & Business Media, 2001.

[63] I. N. Bronshtein, K. A. Semendyayev, Handbook of mathematics, Springer Science & Business Media, 2013.

[64] J. C. Mason, D. C. Handscomb, Chebyshev polynomials, Chapman and Hall/CRC, 2002.

[65] W. J. Cody, Rational Chebyshev Approximations for the Error Function, Mathematics of Computation 23 (107) (1969) 631–637. doi:10.2307/2004390.

[66] M. Abramowitz, I. Stegun, Error function and Fresnel integrals, Handbook of mathematical functions with formulas, graphs, and mathematical tables 9 (1972) 297–309.

[67] J.-S. Pan, P.-C. Song, S.-C. Chu, Y.-J. Peng, Improved Compact Cuckoo Search Algorithm Applied to Location of Drone Logistics Hub, Mathematics 8 (3) (2020) 333. doi:10.3390/math8030333.

[68] J. Momin, X.-S. Yang, A literature survey of benchmark functions for global optimization problems, Journal of Mathematical Modelling and Numerical Optimisation 4 (2) (2013) 150–194. doi:10.1504/IJMMNO.2013.055204.

[69] X. Yao, Y. Liu, G. Lin, Evolutionary programming made faster, IEEE Transactions on Evolutionary Computation 3 (2) (1999) 82–102.

[70] M. Naik, M. R. Nath, A. Wunnava, S. Sahany, R. Panda, A new adaptive cuckoo search algorithm, in: 2015 IEEE 2nd International Conference on Recent Trends in Information Systems (ReTIS), IEEE, 2015, pp. 1–5.

[71] H.-G. Beyer, S. Finck, HappyCat–A simple function class where well-known direct search algorithms do fail, in: International Conference on Parallel Problem Solving from Nature, Springer, 2012, pp. 367–376.

[72] A. I. Oyman, Convergence behavior of evolution strategies on ridge functions, Ph.D. thesis, Universität Dortmund (1999).

[73] R. Eberhart, J. Kennedy, Particle swarm optimization, in: Proceedings of the IEEE international conference on neural networks, Vol. 4, Citeseer, 1995, pp. 1942–1948.

[74] X.-S. Yang, A New Metaheuristic Bat-Inspired Algorithm, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 65–74. doi:10.1007/978-3-642-12538-6_6.

[75] X.-S. Yang, Flower Pollination Algorithm for Global Optimization, in: J. Durand-Lose, N. Jonoska (Eds.), Unconventional Computation and Natural Computation, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 240–249.

[76] S. Mirjalili, SCA: A Sine Cosine Algorithm for solving optimization problems, Knowledge-Based Systems 96 (2016) 120–133. doi:10.1016/j.knosys.2015.12.022.

[77] S. Arora, S. Singh, Butterfly optimization algorithm: a novel approach for global optimization, Soft Computing 23 (3) (2019) 715–734.

CRediT authorship contribution statement

Pei-Cheng Song: Conceptualization, Methodology, Software, Writing original draft. **Jeng-Shyang Pan**: Data curation, Methodology. **Shu-Chuan Chu**: Data curation, Methodology.

We the undersigned declare that this manuscript is original, has not been published before and is not currently being considered for publication elsewhere.

We wish to confirm that there are no known conflicts of interest associated with any of the suggested reviewers and there has been no significant financial support for this work that could have influenced its outcome.

We confirm that the manuscript has been read and approved by all named authors and that there are no other persons who satisfied the criteria for authorship but are not listed. We further confirm that the order of authors listed in the manuscript has been approved by all of us.

We confirm that we have given due consideration to the protection of intellectual property associated with this work and that there are no impediments to publication, including the timing of publication, with respect to intellectual property. In so doing we confirm that we have followed the regulations of our institutions concerning intellectual property.

We understand that the Corresponding Author is the sole contact for the Editorial process (including Editorial Manager and direct communications with the office). Dr. Chu is responsible for communicating with the other authors about progress, submissions of revisions and final approval of proofs. We confirm that we have provided a current, correct email address which is accessible by the Corresponding Author.

Pei-Cheng Song Pei-cheng Song 13/11/2019

Jeng-Shyang Pan Jeng-Shyang Pan 13/11/2019

Shu-Chuan Chu Shu-Chuan Chu 13/11/2019