applied sciences

MDPI

Article

Self-adaptive Artificial Bee Colony with a Candidate Strategy Pool

Yingui Huang [1,†], Ying Yu [1,†], Jinglei Guo [1,†] and Yong Wu [2,*,†]

[1] School of Computer Science, Central China Normal University, Wuhan 430079, China; yinguihuang960813@163.com (Y.H.); yuying@ccnu.edu.cn (Y.Y.); guojinglei@ccnu.edu.cn (J.G.)
[2] School of Automation, Wuhan University of Technology, Wuhan 430070, China
* Correspondence: wuyong@whut.edu.cn
† These authors contributed equally to this work.

Abstract: As a newly developed metaheuristic algorithm, the artificial bee colony (ABC) has garnered a lot of interest because of its strong exploration ability and easy implementation. However, its exploitation ability is poor and dramatically deteriorates for high-dimension and/or non-separable functions. To fix this defect, a self-adaptive ABC with a candidate strategy pool (SAABC-CS) is proposed. First, several search strategies with different features are assembled in the strategy pool. The top 10% of the bees make up the elite bee group. Then, we choose an appropriate strategy and implement this strategy for the present population according to the success rate learning information. Finally, we simultaneously implement some improved neighborhood search strategies in the scout bee phase. A total of 22 basic benchmark functions and the CEC2013 set of tests were employed to prove the usefulness of SAABC-CS. The impact of combining the five methods and the self-adaptive mechanism inside the SAABC-CS framework was examined in an experiment with 22 fundamental benchmark problems. In the CEC2013 set of tests, the comparison of SAABC-CS with a number of state-of-the-art algorithms showed that SAABC-CS outperformed these widely-used algorithms. Moreover, despite the increasing dimensions of CEC2013, SAABC-CS was robust and offered a higher solution quality.

Keywords: evolution; optimization; artificial bee colony; multi-strategy; self-adaptive mechanism; modified neighborhood operator

Citation: Huang, Y.; Yu, Y.; Guo, J.; Wu, Y. Self-adaptive Artificial Bee Colony with a Candidate Strategy Pool. *Appl. Sci.* **2023**, *13*, 10445. https://doi.org/10.3390/app131810445

Academic Editors: Agostino Forestiero and Antonio Fernández-Caballero

Received: 15 June 2023
Revised: 27 August 2023
Accepted: 4 September 2023
Published: 19 September 2023

1. Introduction

Optimization problems are omnipresent in industrial manufacturing and science activities. In general, these problems are complex and characterized by non-convexity, non-differentiability, discontinuity, etc. These kinds of problem are hard to handle with traditional methods in mathematics because they require strict limits on mathematical properties in optimization problems. In recent years, swarm algorithms (SAs) have received much attention as a powerful tool for solving these kinds of complex optimization problem. Since the need for SAs was recognized, a wide variety of SAs have been developed, often inspired by modeling the behaviors of organisms in the natural world, including the genetic algorithm (GA) [1,2], the firefly algorithm (FA) [3], the ant colony algorithm (ACO) [4], the differential evolution algorithm (DE) [5–7], the particle swarm algorithm (PSO) [8,9], and artificial bee colony (ABC) [10,11], etc.

The ABC algorithm, which replicates the tight collaborative activity of employed bees, onlooker bees, and scout bees in discovering suitable food sources, was initially described by Karaboga et al.[12] in 2005. Due to its straightforward design, few variables, and strong resilience, the ABC has attracted researcher interest and is applied in route planning, resource scheduling, and other related problems. However, it is very difficult to apply a single operator to perfectly solve all kinds of optimization problem. The ABC is no exception and faces the following challenges:

- In comparison to other SAs, ABC has a sluggish convergence, due to the 1-D update in the search equation. It is crucial to figure out how to increase the convergence speed to improve ABC's performance. One difficulty that should be addressed is how to increase the algorithm's convergence speed, while maintaining high performance, by enhancing the method;
- The problems that can be solved using the artificial bee colony algorithm are limited in variety, due to the simplicity and singularity of the ABC algorithm's updating method;
- According to certain pertinent literature studies [13], ABC has a significant capacity for exploration because of a single search equation in the evolution process. As a result, a popular area of research is how to improve exploitation, while maintaining exploration.

Many ABC variants have been designed to address these deficiencies, and the modified techniques can be divided into three groups: modifying search equations [14], assembling a multi-strategy [15], and hybridizing other metaheuristic search frameworks [16].

In order to resolve these difficult optimization issues, this study suggests a novel variation of ABC called SAABC-CS, which stands for self-adaptive ABC with a candidate strategy pool. Its main characteristics can be summed up as follows:

- Five alternative search methods are combined to generate a candidate strategy pool that improves ABC's exploitation capability, without sacrificing exploration capability. In addition, we include multi-dimensional updates in each strategy, which considerably increases the frequency of individual updates and boosts the convergence speed. A self-adaptive method is also suggested for choosing the right search technique. The knowledge from the previous information is used to adaptively update the selection probability of each strategy;
- Our approach, in contrast to other algorithms, performs quite well, without adding additional control parameters when applying each strategy, which is aligned to the artificial bee colony program's original intention—simplicity and effectiveness;
- By improving the method, we make it more useful for solving actual, practical issues in the real world.

The remaining part of this paper is divided into the following sections: The works pertaining to the fundamental ABC and its variations are detailed in Section 2. In Section 3, the suggested algorithm SAABC-CS is described. The effectiveness of our suggested approach and the analysis of the results of our algorithm in comparison to other algorithms are provided in Section 4. The last Section provides a summary of our work.

2. Related Work

2.1. ABC Algorithm

The initialization period, employed bee period, onlooker bee period, and scout bee period make up the primary foundation of the ABC. All bees have different responsibilities at different stages. It should be noted that each bee has its own food resources in each period, and the number of bees in each period is consistent. The food resource in this statement often represents a candidate solution to the optimization problem. The evolution framework of the ABC is seen in Figure 1.

Like all SAs, the ABC needs to go through an initialization stage before performing the other three stages of work cyclically. The following are the relevant contents of each phase:

(i) Initialization phase

In this phase, the entire population is randomly initialized, each individual represents a food resource and this is generated using Equation (1).

$$X_{i,j} = Lower + rand \cdot (Upper - Lower) \quad (1)$$

where $i = 1, 2, \cdots, SN$ and $j = 1, 2, \cdots, D$. D stands for the dimension of the optimization problems, while SN is the number of solutions in a swarm. $rand$ is a random number

belonging to (0,1), $X_{i,j}$ represents the element of jth dimension of the ith individual. *Lower* and $Upper$ denote the minimum value and maximum value in all dimensions of each individual, respectively.

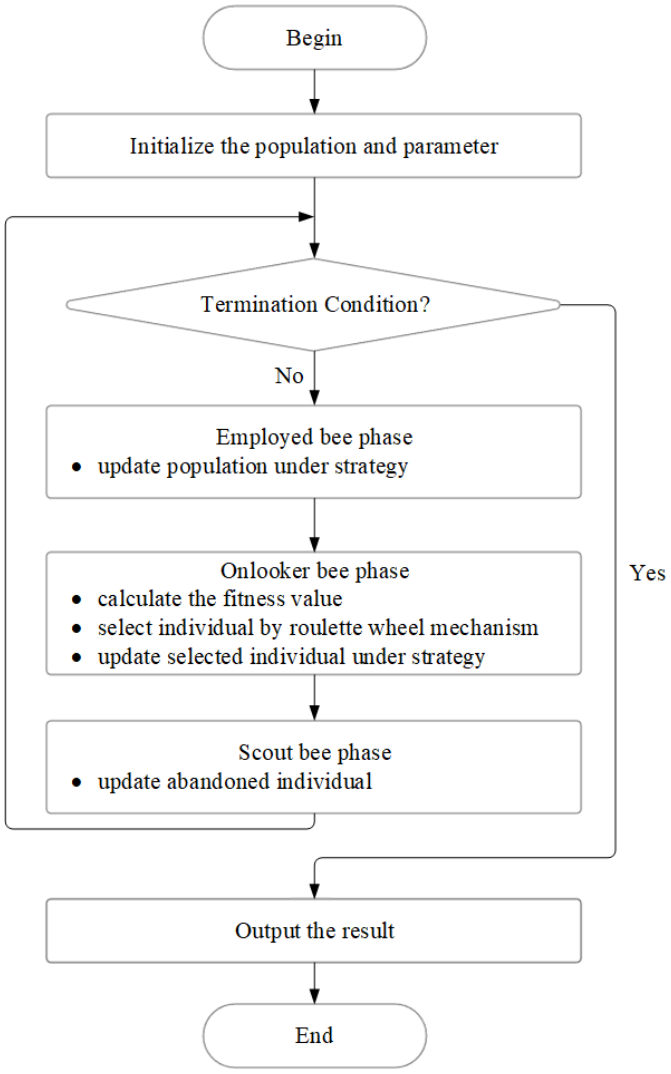


Figure 1. The ABC framework.

(ii) Employed bee phase

The evolution reaches the employed bee phase after startup. According to Equation (2), each employed bee searches the full search area for new food sources during this phase.

$$V_{i,j} = X_{i,j} + \phi_{i,j} \cdot (X_{i,j} - X_{k,j}) \tag{2}$$

The random number $\phi_{i,j}$ has the value $(-1,1)$. Unlike X_i, which belongs to the whole population, X_k is a randomly chosen solution. The value of $V_{i,j}$ is reinitialized using Equation (1) if it oversteps either the lower or higher barrier. V_i takes the place of X_i if its object function value is superior to that of X_i.

In addition, there are a number of updated individuals (NUI) for each solution. Thus, NUI is recorded using a 1·SN matrix. At the beginning, every element of NUI is initialized to zero. After that, once the X_i has successfully been replaced by V_i, the ith value of NUI is reset to zero. Otherwise, the ith value of NUI is increased by one. The NUI matrix has an effect on the subsequent scout bee phase.

(iii) Onlooker bee phase

The bee continues to search for new food sources during the onlooker bee phase. In contrast to the employed bee phase, the onlooker bee only has to seek in the vicinity of the chosen food resource, which is equivalent to expanding the utilization of the food resource. At this stage, not every food resource can be selected to search in its vicinity, but a probability search is carried out according to its fitness value. The calculation equation for the fitness value is as shown in Equation (3).

$$Fit_i = \begin{cases} \dfrac{1}{1+f(X_i)} & \text{if } f(X_i) \geq 0 \\ 1+abs(f(X_i)) & \text{otherwise} \end{cases} \tag{3}$$

where Fit_i and $f(X_i)$ are the fitness values of X_i and objective function result, correspondingly. $f(X_i)$ is calculated in the employed bee phase. Equation (4) determines the selection probability of each food resource.

$$p_i = \frac{Fit_i}{\sum_{j=1}^{SN} Fit_j} \tag{4}$$

After that, the onlooker bees use the classic roulette wheel selection strategy to select a food resource. Obviously, the larger the fitness value obtained, the greater the chance the food resource is selected. Equation (2) is also used as an update equation for the onlooker bees. In addition, the same process is used as for the employed bees after generating a candidate solution.

(iv) Scout bee phase

At scout bee phase, the element value NUI associated with each individual is checked. Once an individual's NUI exceeds a predefined value, it is believed that this food resource has been exhausted, which implies that the individual may be trapped at a local optima. As a result, in this circumstance, the employed bee is transformed into a scout bee to help Equation (1) generate new solutions.

2.2. ABC Variants

Although the ABC algorithm has a good optimization performance, it also has certain shortcomings, such as it being easy to fall into local optimum, the imbalance between exploration and exploitation, and a slow convergence speed. Due to the existing problems with the ABC, researchers have proposed many different methods to solve them. Most solutions can be divided into three distinct categories:

(1) Modifying the search equation

The performance of the ABC algorithm depends heavily on the solution search equation. In a basic ABC, the solution search equation does well in exploration but poorly in exploitation, since each individual X_k shown in Equation (2) is chosen randomly from the overall population. Thus, inspired by [17,18], Wang and Zhou et al. [19] proposed an ABC variant (KFABC). KFABC is based on knowledge fusion and its viability was tested against 32 benchmark functions. Lu et al. [20] designed Fast ABC (FABC), which made use of two extra alternative search equations for employed bees and onlooker bees, respectively. These two equations also utilized the bees' individual information and employed a Cauchy operator to equilibrize the global and local search capacities of individuals. In order to prove its effectiveness, the performance of FABC was compared with that of 10 benchmark functions and a genuine path planning issue. Gao et al. [21], inspired by differential evolution (DE), presented an improved search equation using a modified ABC (MABC). This variant enabled the bees to search around the best solutions found in the previous iteration, to improve the exploitation. A total of 28 benchmark functions were used in the comparison experiments. When compared to two ABC-based algorithms, the findings showed that MABC performed well when addressing complicated numerical optimization problems.The improved algorithm that Guo et al. [22] developed based on MABC is called

the global artificial bee colony search algorithm. Guo incorporated all the employed bees' historical best positions based on the information about food sources into the search equations to develop this algorithm. Yu et al. [23] proposed another form of ABC variant called the adaptive ABC (AABC). It adjusted the greedy degree of the original ABC using a novel greedy position update strategy and an adaptive control scheme. Using a set of benchmark functions, AABC outperformed the original ABC and subsequent ABC iterations in their tests.

(2) Hybridizing another metaheuristic search framework

Hybrid algorithms are mainly based on the combination of two or more metaheuristic algorithms, so that the advantages of one algorithm can be used to offset the deficiencies of other algorithms. This method could improve the optimization performance of an algorithm. The following are some examples of hybrid ABC algorithms that combined the ABC algorithm with other heuristic algorithms. Jadon et al. [24] proposed a hybridization of ABC and DE algorithms (HABCDE), to develop a more efficient algorithm than ABC or DE individually. Over twenty test problems and four actual optimization issues were used to evaluate the performance of HABCDE. Alqattan et al. [25] presented a hybrid particle movement ABC algorithm (HPABC). This algorithm adapted the particle moving process to improve the exploitation of the original ABC variant. The algorithm variant was provided, and seven benchmark functions were utilized to validate it. Chen et al. [26], on the other hand, introduced a simulated annealing algorithm into the employed bees' phase and proposed the simulated annealing-based ABC algorithm (SAABC). To improve algorithm exploitation, the simulated annealing algorithm was added in the employed bee search process. The experimental results were validated against a collection of numerical benchmark functions of varying size. This demonstrated that the SAABC algorithm outperformed the ABC and global best guided ABC algorithms in the majority of tests.

(3) Assembling multi-strategy

Multi-strategy search refers to the implementation of different search strategies in the different search stages of the ABC or for different food resources. In recent years, some algorithms that introduced multi-strategy search into ABC have been proposed, but their effectiveness varied. Gao et al. [27] formed a strategy pool using three distinct search strategies and adopted an adaptive selection mechanism to further enhance the performance of the algorithm. It was evaluated using a set of 22 benchmark functions and compared against other ABCs. In almost every case, the comparison findings revealed that the suggested method provided superior results. Song et al. [28] designed a novel algorithm called MFABC. MFABC improved the search ability of the ABC algorithm with a small population by fusing multiple search strategies for both employed bees and onlooker bees. MFABC's accuracy, stability, efficiency, and convergence rate were demonstrated experimentally on a set of benchmark functions. Chen et al. [29] proposed a new algorithm called self-adaptive differential artificial bee colony (sdABC) by incorporating multiple diverse search strategies and a self-adaptive mechanism into the original ABC algorithm. The sdABC technique was tested on 28 benchmark functions, including both common separable and difficult non-separable CEC2015 functions. The experimental findings suggested that sdABC obtained substantially better outcomes on both separable and non-separable functions than earlier ABC algorithms. In addition to the above ABC algorithm variants, Zhou et al. [30] developed a modified neighborhood search operator by utilizing an elite group, which is called MGABC. Their experiments employed 50 well-known test functions and one real-world optimization issue to validate the technique, which included 22 scalable basic test functions and 28 complicated CEC2013 test functions. The comparison included seven distinct and well-established ABC variations, and the findings suggested that the technique could obtain test results that were at least equivalent in test performance for most of the test functions.

Assessing these three improvement directions, the first is too simple and the second makes the algorithm extremely complicated. Thus, we choose the third direction as our

main interest. We based our research partially on prior work from other researchers. By assembling a multi-strategy search, a wider range of issues can be tackled and the outcomes are better.

3. The Proposed Algorithm SAABC-CS

3.1. Candidate Strategy Pool

In most cases, different problems have different characteristics, and they are hard to describe clearly in advance. Thus, problems are usually black boxes. Moreover, different update strategies for ABC have unique characteristics. It is unrealistic to rely on only one strategy to solve all problems. These observations make us reconsider how to select strategies or construct novel strategies to improve the robustness when facing different problems. Based on the motivations above, we selected five search strategies with different characteristics from the relevant literature [18,31] to construct our candidate strategy pool. In addition, we employed the binomial crossover method to enable the algorithm to find optimal solutions more effectively. Considering both exploration and exploitation during the entire evolution process, five strategies were selected and are described in detail, as follows:

(i) "rand":

$$V_{i,j} = X_{r1,j} + \phi_{i,j} \cdot (X_{r1,j} - X_{r2,j}) \tag{5}$$

(ii) "pbest-1":

$$V_{i,j} = X_{e,j} + \phi_{i,j} \cdot (X_{r1,j} - X_{r2,j}) \tag{6}$$

(iii) "pbest-2":

$$V_{i,j} = X_{e,j} + \phi_{i,j} \cdot (X_{r1,j} - X_{r2,j}) + \varphi_{i,j} \cdot (X_{r3,j} - X_{r4,j}) \tag{7}$$

(iv) "current-to-pbest":

$$V_{i,j} = X_{i,j} + \phi_{i,j} \cdot (X_{i,j} - X_{r1,j}) + \varphi_{i,j} \cdot (X_{e,j} - X_{i,j}) \tag{8}$$

(v) "pbest-to-rand":

$$V_{i,j} = X_{e,j} + \phi_{i,j} \cdot (X_{e,j} - X_{i,j}) \tag{9}$$

where X_{r1}, X_{r2}, X_{r3}, and X_{r4} are the different individuals selected randomly in the population, and they are all distinctive from X_i. A homogeneous random number between $[-1,1]$ is $\phi_{i,j}$. A solution from an elite group is represented by X_e. The top q·SN solutions are chosen to form the elite group, after all the individuals are sorted according to their fitness values. The size of the elite group is controlled by q, which is set at 0.1. $\varphi_{i,j}$ is a homogeneous random number between [0,1.5].

Figure 2 roughly depicts the behavior of each strategy, the individuals in ellipse are the elite individuals, and the remaining triangle icons represent other common individuals. The red circle in Figure 2 represents the new individual generated by the corresponding strategy. With the "rand" strategy, the position of the new individual in Figure 2a is between two different individuals, and it is close to the first random individual. Actually, its position falls within a circle with X_{r1} as the center and $|X_{r1} - X_{r2}|$ as the radius. The behavior of the "rand" strategy makes the algorithm focus more attention on a global search. Similarly, the position of the new individual in Figure 2b is between an elite individual and two different common individuals with the "pbest-1" strategy. This strategy leads the algorithm to learn the elite's information, while focusing on a global search. With the "pbest-2" strategy, the position of the new individual in Figure 2c is also in the center of the selected individuals, this makes our algorithm utilize more individual sampling information. As shown in Figure 2d, the position of the new individual is affected by the current individual, an elite individual, and a randomly selected individual in the "current-to-pbest" strategy. The position of the new individual in Figure 2e is based on the elite

individual and current individual, which comprehensively takes the current individual and elite individual into consideration.

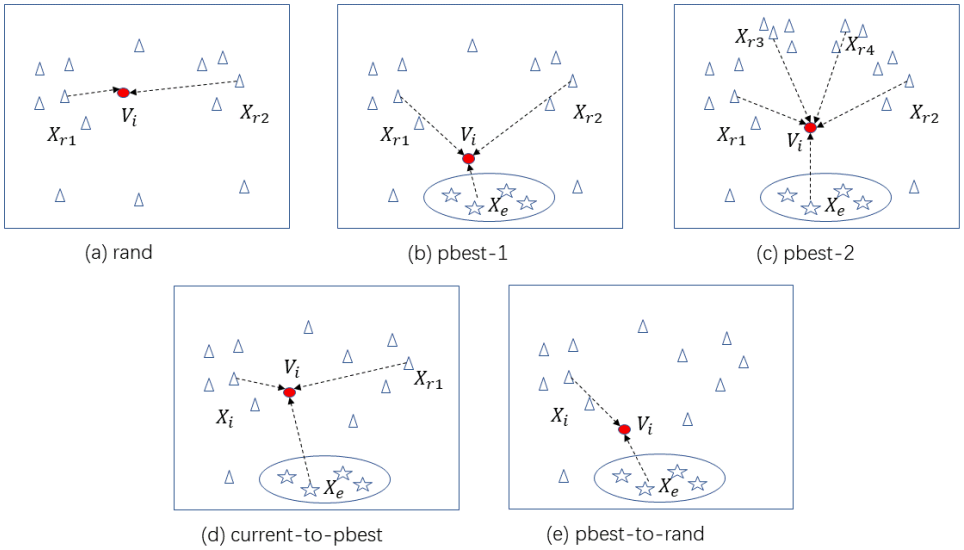


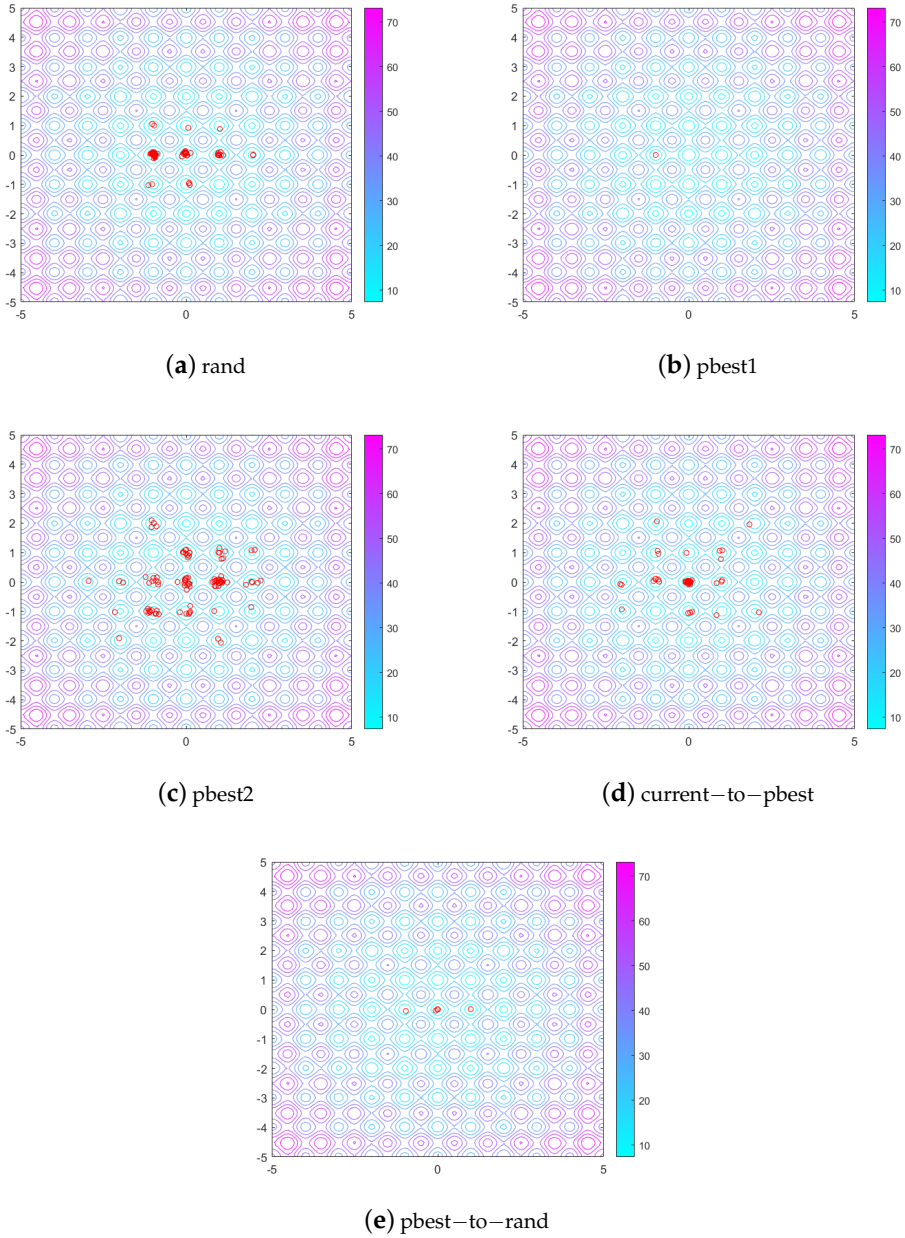
Figure 2. Schematic diagrams of five different strategies. (Stars represent elite individuals, triangles represent common individuals,red circles represent new individual,dashed arrows represent search direction).

In order to further reflect the different characteristics of the five strategies in seeking optimal solutions, we performed an experiment on the Rastrgin function [32] under the same conditions. The formula of Rastrigin is as follows, and its dimension was set to 2:

$$f(X) = 10 \cdot D + \sum_{i=1}^{D} [X_i^2 - 10 \cdot \cos(2 \cdot \pi \cdot X_i)] \tag{10}$$

where X is a 2-dimensional individual.

In this experiment, we obtained the two-dimensional individual distribution for the five strategies after 20 generations, and present the results in Figure 3. The initial population size of each strategy was 100. From Figure 3a, these individuals may be seen to disperse around local maxima, although the local maxima are still distant from the global maxima. Thus, it is obvious that the "ABC/rand" strategy has a strong exploration ability but weak exploitation ability. We can see clearly that all individuals converge around one local optimum in Figure 3b, but the local optimum is not the global optimum. Thus, it has a strong exploitation ability but weak exploration ability. The "ABC/pbest-2" strategy originates from "ABC/pbest-1", but with increased exploration ability. This modification causes most individuals to distribute around global optimum, with some individuals located around other local optima. The results in Figure 3c further demonstrate that the "ABC/pbest-2" strategy increased its exploration ability while keeping its exploitation ability. As for the results in Figure 3d, the "ABC/current-to-pbest" strategy uses the information of the current individual, a random different individual, and a random elite individual. Thus, it has a strong exploration ability during early generation and a strong exploitation ability during late generation. As we can see from Figure 3e, under the influence of "ABC/pbest-to-rand", the individuals mainly converged around the global optimum, with others also located near local optima. This was dominated by X_e but also uses the current individual information. Thus, it maintains a significant capacity for exploitation, while also having the opportunity to leave the local optima and go to a global or nearby one.



(**a**) rand (**b**) pbest1

(**c**) pbest2 (**d**) current−to−pbest

(**e**) pbest−to−rand

Figure 3. Five strategies' contour graphs based on the two-dimensional Rastrigin function. (Red dots represent individuals. Subfigures (**a**–**e**) represent individual distribution map of the five strategies in the current generation, respectively).

With the exception of the "ABC/rand" technique, the other four search methods all utilize the information of the elite group. The following two benefits result from using an elite group instead of the elite with the best fitness:

(1) In the first place, this allows the entire population to fully utilize the knowledge of the elite solution group during the evolution process and evolve in a better way.

(2) Second, the whole population is prone to becoming locked in local optima if the population only uses the present global optimal solution as the search traction. However, the population may evolve in numerous good directions and are provided better solutions by the elite group. As a result of using an elite group, it is simple for the population to move away from the local optima and reach the global or approximated optimal region.

Additionally, the original ABC search approach performs poorly for some issues with variable inseparability, since it only updates one variable at a time. Therefore, to update

many dimensions at once, these techniques combine mutation and crossover, as in GA. In this approach, using various update techniques inside the adaptive mechanism enhances the algorithm's efficiency, while simultaneously strengthening its robustness. Thus, to create a trial vector $U_{i,j}$, we apply a binomial crossover operator to $X_{i,j}$ and $V_{i,j}$.

$$U_{i,j} = \begin{cases} V_{i,j} & \text{if rand} \leq \text{M or j = k} \\ X_{i,j} & \text{Otherwise} \end{cases} \tag{11}$$

where $i = 1, 2, \ldots, SN$, $j = 1, 2, \ldots, D$. A number chosen at random between [1, D] called k is utilized to make certain that at least one element is updated. *rand* is an arbitrary number ranging from 0 to 1 with a uniform distribution.

In our algorithm, we also precisely apply the boundary correction technique to improve the outcome. If the jth dimension element of U_i is outside of the boundary, we make the following revisions:

$$U_{i,j} = \begin{cases} Lower & \text{if } U_{i,j} < Lower \\ Upper & \text{if } U_{i,j} > Upper \end{cases} \tag{12}$$

To join the following generation, we choose the superior source vector X_i over the trial vector U_i.

$$X_i^{G+1} = \begin{cases} U_i & \text{if } f(U_i) < f(X_i) \\ X_i & \text{otherwise} \end{cases} \tag{13}$$

The following values are set for the strategy's self-definition parameters: The elite community's size is q·SN. The dimension update is controlled by parameter M, which is set at 0.5.

3.2. Self-adaptive Mechanism

To maximize the algorithm's efficiency, we must choose a more appropriate approach in different phases of the algorithm, due to the distinctive characteristics of the aforementioned five alternative search strategies. As a result, we include an adaptive mechanism in our suggested algorithm, to choose the best strategy. The fundamental principle of self-adaptation is to dynamically modify the potential for choosing an appropriate approach, in accordance with the success information about producing superior solutions. The selection likelihood of one strategy increases when an exceptional solution is produced by this strategy. Additionally, any tactic has the chance to be picked out during the evolution, owing to the roulette selection system. Such a self-adaptive system can help the population move beyond the local ideal, as well as toward the optimal. The combination of this self-adaptive mechanism with the aforementioned five techniques is depicted in the flowchart in Figure 4.

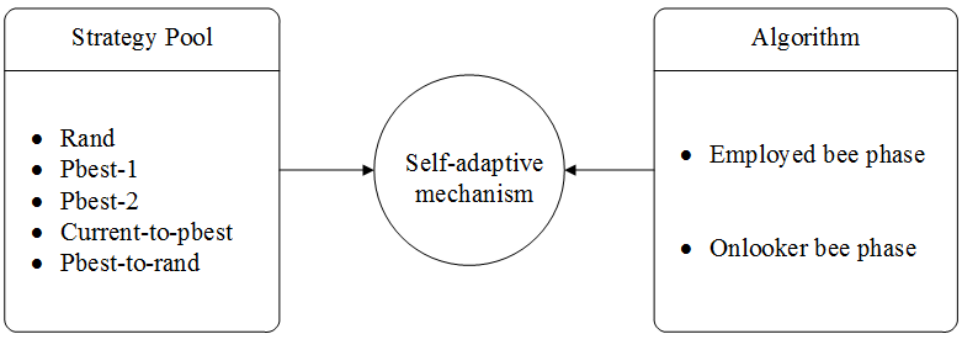


Figure 4. Flowchart of self-adaptive multiple strategies.

In the initialization phase, some variables are initialized by the self-adaptive mechanism. Prob is a 1×5 matrix, in which each element $Prob_i$ corresponds to the selection probability for the above $strategy_i$, and the sum of all elements is 1. In the beginning,

their selection probability is equal, to guarantee fairness. Two 1 × SN matrices sFlag and fFlag are used, to mark whether the candidate solution is better or worse than the original solution when using a corresponding strategy. SN is a measure of population density. If the new generated solution is better, the associated sFlag matrix element is set to 1 and the corresponding fFlag matrix element is set to 0, and vice versa. We also use two 5·LP matrices, sCounter and fCounter, to count the proportion of triumphs and failures of each generation in the LP generation, after updating using the corresponding strategy. LP represents a fixed interval, and we set this to 10 here. For every LP generation, we use sCounter and fCounter to update the Prob of each strategy. The statistical data information of sCounter and fCounter are the main source for updating the Prob value. Moreover, every time the selected strategy probability is updated, every element of sFlag, fFlag, sCounter, and fCounter must be reset to 0, to avoid affecting the next LP generations. The update equation of Prob is determined using the following Equation (14), and then the probability is normalized using Equation (15).

$$Prob_i = \begin{cases} \dfrac{\sum_{k=1}^{LP} sCounter[i][k]}{\sum_{k=1}^{LP} sCounter[i][k] + \sum_{k=1}^{LP} fCounter[i][k]} & \\ \qquad\qquad \sum_{k=1}^{LP} sCounter[i][k] \neq 0 & \\ 0.5 \cdot Prob_i \qquad \text{Otherwise} & \end{cases} \tag{14}$$

$$Prob_i = \frac{Prob_i}{\sum_{i=1}^{5} Prob(i)} \tag{15}$$

3.3. Scout Bee and Modified Neighborhood Search Operator

In this stage, we utilize the method proposed by Wang et al. in KFABC [19], adding two methods based on opposition-based learning(OBL) and the Cauchy approach, to generate two additional solutions. Then, we select the best solution from the random solutions, OBL solution, and Cauchy solution, to replace the abandoned solution. The random operator, the OBL operator, and Cauchy disturbance operator that produce the candidate solutions are described in Equations (1), (16) and (17).

$$OX_j = Lower + Upper - X_{a,j} \tag{16}$$

where the space's boundary is defined by Lower and Upper. j = 1, 2,..., D, and the abandoned solution is represented by X_a.

$$CX_j = X_{a,j} + Cauchy() \tag{17}$$

where $j = 1, 2, \ldots, D$, $Cauchy()$ return a value from the Cauchy distribution.

In addition , we use a neighborhood search operator in our method as a supplementary operator, which was suggested by Zhou et al. in MGABC [30]. The operator continues to use the data from the elite group solution and determines whether to employ the supplemental operator in this generation based on a certain possibility p (p is 0.1, as in MGABC [30]. The operator is shown in Equation (18).

$$TX_i = r1 \cdot X_i + r2 \cdot X_{e1} + r3 \cdot (X_{e2} - X_{e3}) \tag{18}$$

where three solutions from the elite group, X_{e1}, X_{e2}, and X_{e3}, were chosen at random and must be distinct from X_i. As positive numbers drawn at random from (0,1), r1, r2, and r3 must also satisfy the restriction that r1 + r2 + r3 = 1. If TX_i is superior to X_i, TX_i will take the place of X_i.

3.4. Framework of SAABC-CS

During the employed and onlooker bee phase, SAABC-CS employs five distinct search algorithms, four of which make use of knowledge from the elite group. We provide an adaptive mechanism based on prior knowledge to choose the best search technique, in order to make better use of these five tactics. To enhance the algorithm's efficiency and speed of convergence, we update the search technique used for the scout bee and add an additional neighborhood search operator. The pseudo-code for SAABC-CS is provided in Algorithms 1 and 2 and, the flowchart for it can be viewed in Figure 5, which help to better explain the entire process.

Algorithm 1: The pseudo-code of Modified neighborhood operator

```
for i = 1 to SN do
    if rand ≤ p then
        Generate a new solution TX_i by Equation (18) and evaluate it;
        FEs = FEs + 1;
        if f(TX_i) ≤ f(X) then
            Substitude X_i to TX_i;
        end
    end
end
```

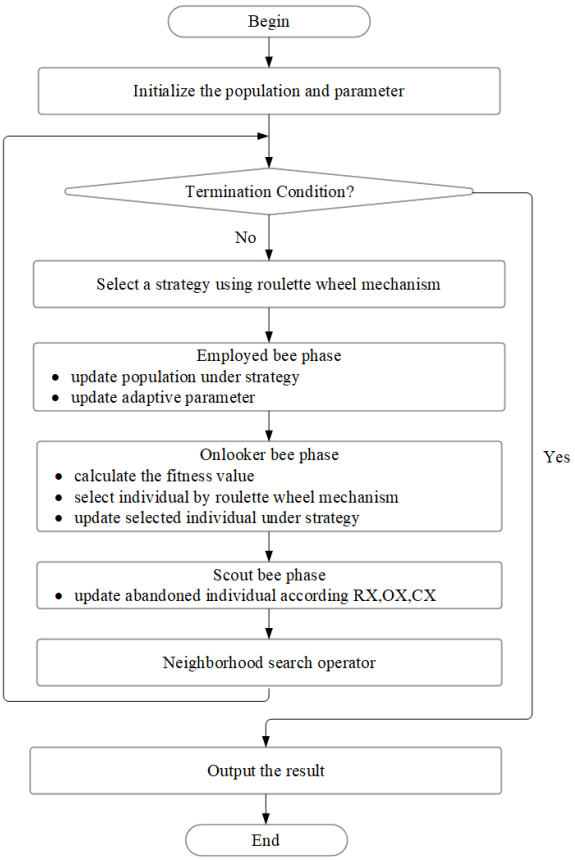


Figure 5. Flowchart of SAABC-CS.

Algorithm 2: The pseudo-code of SAABC-CS.

Randomly initialize and evaluate the population include SN food sources $X_1, X_2, \ldots, X_{SN}$ and set FEs = SN;
Initialize parameter Prob, sFlag, fFlag, sCounter, fCounter, MR, q, p, LP;
while $FEs \leq MaxFes$ **do**
    Select a strategy using roulette wheel selection mechanism;
    Select the elite Group belongs to top q·SN sorted by fitness value;
    `/* Employed bee phase */`
    for $i = 1$ *to* SN **do**
        Generate a candidate solution V_i using current strategy and evaluate it;
        if $f(V_i) \leq X_i$ **then**
            Substitude X_i to V_i;
            $trial_i = 0$, $sFlag_i = 1$, $fFlag_i = 0$;
        else
            $trial_i = trial_i + 1$, $sFlag_i = 0$, $fFlag_i = 1$;
        end
    end
    Update $sCounter_{j,k}$, $fCounter_{j,k}$, j represent the jth strategy and k is the generation number;
    `/* Onlooker bee phase */`
    Calculate the probability p_i accoring to Equation (3);
    Select the elite Group belongs to top q·SN sorted by fitness value;
    for $i = 1$ *to* SN **do**
        Choose a food source X_j by the roulette wheel selection mechanism;
        Generate a candidate solution V_i using current strategy and evaluate it;
        if $f(V_i) \leq X_i$ **then**
            Substitude X_i to V_i; $trial_i = 0$, $sFlag_i = 1$, $fFlag_i = 0$;
        else
            $trial_i = trial_i + 1$, $sFlag_i = 0$, $fFlag_i = 1$;
        end
    end
    Update $sCounter_{j,k}$,$fCounter_{j,k}$, j represent the jth strategy and k is the generation number;
    if *(generation mod LP)==0* **then**
        Update Prob using Equation (14) and (15);
        Reset sFlag, nFlag, sCounter, fCounter;
    end
    `/* Scout bee phase */`
    if $Max(trial) \geq limit$ **then**
        Generate three solutions RX, OX and CX by Equations (1), (16) and (17) respectively;
        Evaluate the three solutions and FEs = FEs+3;
        Select the best one from RX, OX and CX to replace X_i;
    end
    `/* Modified neighborhood operator */`
    Algorithm 1;
end

4. Experiments

4.1. Test Problems

We ran trials on 50 test issues that were separated into two sets of benchmarks, to demonstrate the efficacy of our suggested algorithm SAABC-CS. The first benchmark set included 22 basic functions, and the second benchmark set was referred to as CEC2013. The dimensions of the CEC2013 benchmarks were set as 30, 50, and 100. We used two values (Mean and Std) as metrics for algorithm comparison. "Mean" represents the average value of the optimal results obtained by the algorithm for the corresponding running times, and "Std" represents the corresponding variance. Experiment 1 not only verified the effectiveness of the strategy pool but also demonstrated the effectiveness of the self-adaptive method. Experiment 2 compared the performance of SAABC-CS with that of the other five algorithms in the CEC2013 function set. All algorithms designed in this section were utilized in MATLAB R2020a. Tables 1–4 report the compared results, in which the best result for each problem is marked in **bold**, and summarize the statistical findings. "+/=/−" indicate that SAABC-CS outperformed, was comparable to, or underperformed the compared algorithm in the test tasks.

Table 1. Results of five single strategies vs. multi-strategy ABC algorithms with self-adaptive/rand on basic 22 function (D = 30).

Function		ABC-Rand	ABC-Pbest-1	ABC-Pbest-2	ABC-Current-to-Pbest	ABC-Pbest-to-Rand	RABC-CS	SAABC-CS
F1	Mean	$2.248526 \times 10^{-120}$ +	$7.064515 \times 10^{-113}$ +	6.559319×10^{-98} +	$8.878308 \times 10^{-117}$ +	$8.129275 \times 10^{-176}$ +	$4.189678 \times 10^{-128}$ +	$\mathbf{2.259275 \times 10^{-176}}$
	Std	$1.131182 \times 10^{-119}$	$1.988608 \times 10^{-112}$	1.662559×10^{-97}	$4.071247 \times 10^{-116}$	0.000000×10^{0}	$9.079933 \times 10^{-128}$	0.000000×10^{0}
F2	Mean	4.119521×10^{-62} +	3.388025×10^{-90} +	3.362581×10^{-49} +	4.777346×10^{-59} +	1.004464×10^{-56} +	4.135273×10^{-65} +	8.188025×10^{-91}
	Std	8.821390×10^{-62}	6.589025×10^{-90}	6.925141×10^{-49}	8.332598×10^{-59}	2.159204×10^{-56}	6.097771×10^{-65}	7.509025×10^{-90}
F3	Mean	7.906320×10^{-95} +	9.642783×10^{-89} +	8.755489×10^{-83} +	3.004974×10^{-94} +	$5.845769 \times 10^{-125}$ +	$3.300380 \times 10^{-100}$ +	$6.201979 \times 10^{-130}$
	Std	2.782121×10^{-94}	5.162428×10^{-88}	2.132246×10^{-82}	1.363288×10^{-93}	$2.367933 \times 10^{-124}$	1.234790×10^{-99}	$2.101411 \times 10^{-129}$
F4	Mean	5.442075×10^{-54} +	1.176991×10^{-50} +	1.281191×10^{-45}+	7.984461×10^{-52} +	5.442075×10^{-54} +	2.471204×10^{-56}+	2.899439×10^{-80}
	Std	6.026015×10^{-54}	1.601701×10^{-50}	2.159118×10^{-45}	1.749531×10^{-51}	6.026015×10^{-54}	6.575316×10^{-56}	1.518933×10^{-80}
F5	Mean	2.466807×10^{1} +	2.575768×10^{1} +	2.523777×10^{1} +	2.543241×10^{0} +	2.076386×10^{1} +	2.402630×10^{1} +	2.352532×10^{0}
	Std	8.280997×10^{-2}	1.369552×10^{-1}	1.217680×10^{-1}	1.025347×10^{-1}	1.479236×10^{-1}	1.590104×10^{-1}	2.352992×10^{0}
F6	Mean	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$
	Std	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}
F7	Mean	9.141111×10^{-4} +	1.382338×10^{-3} +	1.491601×10^{-3} +	$\mathbf{1.119361 \times 10^{-4}}$ =	5.043735×10^{-4} +	1.060796×10^{-3} +	$\mathbf{1.119325 \times 10^{-4}}$
	Std	3.535380×10^{-4}	5.575281×10^{-4}	6.834976×10^{-4}	5.475349×10^{-4}	2.076206×10^{-4}	4.821712×10^{-4}	7.547842×10^{-4}
F8	Mean	$2.936318 \times 10^{-116}$ +	$1.231300 \times 10^{-109}$ +	$4.147755 \times 10^{-173}$ +	$3.719664 \times 10^{-114}$ +	1.126064×10^{-93} +	$8.024818 \times 10^{-125}$ +	$\mathbf{4.575380 \times 10^{-181}}$
	Std	$1.574936 \times 10^{-115}$	$3.518575 \times 10^{-109}$	0.000000×10^{0}	$9.643898 \times 10^{-114}$	4.557991×10^{-93}	$1.676287 \times 10^{-124}$	$2.053852 \times 10^{-180}$
F9	Mean	$1.600553 \times 10^{-177}$ +	$1.534562 \times 10^{-113}$ +	3.379257×10^{-99} +	$1.924197 \times 10^{-117}$ +	$2.223761 \times 10^{-122}$ +	$5.549013 \times 10^{-128}$ +	$\mathbf{4.336995 \times 10^{-185}}$
	Std	0.000000×10^{0}	$5.281619 \times 10^{-113}$	1.046764×10^{-98}	$7.117055 \times 10^{-117}$	$6.496603 \times 10^{-122}$	$2.377221 \times 10^{-127}$	$1.740039 \times 10^{-185}$
F10	Mean	$\mathbf{0.000000 \times 10^{0}}$ =	$5.224310 \times 10^{-280}$ +	$3.861741 \times 10^{-219}$ +	$3.696811 \times 10^{-293}$ +	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$=	$\mathbf{0.000000 \times 10^{0}}$
	Std	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}
F11	Mean	$\mathbf{1.687530 \times 10^{-9}}$ =	$\mathbf{1.687530 \times 10^{-9}}$ =	$\mathbf{1.687530 \times 10^{-9}}$ =	$\mathbf{1.687530 \times 10^{-9}}$ =	$\mathbf{1.687530 \times 10^{-9}}$ =	$\mathbf{1.687530 \times 10^{-9}}$ =	$\mathbf{1.687530 \times 10^{-9}}$
	Std	1.261982×10^{-24}	1.261982×10^{-24}	1.261982×10^{-24}	1.261982×10^{-24}	1.261982×10^{-24}	1.261982×10^{-24}	1.261982×10^{-24}
F12	Mean	1.163545×10^{3} +	9.456324×10^{2} +	5.080409×10^{3} +	1.932564×10^{3} +	2.887515×10^{3} +	1.853151×10^{3} +	$\mathbf{2.090269 \times 10^{2}}$
	Std	2.636002×10^{2}	3.413073×10^{2}	4.410889×10^{2}	4.605312×10^{2}	6.269367×10^{2}	5.195663×10^{2}	1.485411×10^{2}
F13	Mean	$\mathbf{0.000000 \times 10^{0}}$ =	6.160741×10^{0} +	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$
	Std	0.000000×10^{0}	1.150776×10^{1}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}
F14	Mean	4.440892×10^{-15} +	4.085621×10^{-15} +	3.967197×10^{-15} +	4.440892×10^{-15} +	4.440892×10^{-15} +	4.322468×10^{-15} +	$\mathbf{1.204045 \times 10^{-15}}$
	Std	0.000000×10^{0}	1.084034×10^{-15}	1.228336×10^{-15}	0.000000×10^{0}	0.000000×10^{0}	6.486338×10^{-16}	9.013523×10^{-16}
F15	Mean	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$=	$\mathbf{0.000000 \times 10^{0}}$=	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$0 =	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$
	Std	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}
F16	Mean	1.980643×10^{-7} +	3.821659×10^{-32} +	5.656174×10^{-11} +	4.240471×10^{-32} +	$\mathbf{2.617575 \times 10^{-32}}$=	2.984035×10^{-32} +	$\mathbf{2.615389 \times 10^{-32}}$
	Std	1.084843×10^{-6}	4.199724×10^{-32}	2.679150×10^{-10}	4.577028×10^{-32}	2.308711×10^{-32}	3.150553×10^{-32}	2.503190×10^{-32}
F17	Mean	3.949367×10^{-33} +	6.448967×10^{-33} +	1.063166×10^{-10} +	7.719422×10^{-10} +	5.149175×10^{-33} +	4.199327×10^{-33} +	$\mathbf{1.465313 \times 10^{-33}}$
	Std	4.750555×10^{-33}	8.215939×10^{-33}	5.062651×10^{-10}	3.540454×10^{-9}	6.479902×10^{-33}	5.820550×10^{-33}	7.798810×10^{-33}
F18	Mean	1.241186×10^{1} +	3.539242×10^{1} +	4.627517×10^{1} +	1.172911×10^{1} +	9.657119×10^{-6} +	6.866667×10^{0} +	$\mathbf{1.996023 \times 10^{-6}}$
	Std	8.701415×10^{0}	1.888478×10^{1}	4.945725×10^{1}	1.895325×10^{1}	5.289422×10^{-5}	1.015308×10^{1}	2.408278×10^{-6}
F19	Mean	4.052816×10^{-62} +	1.897842×10^{-57} +	3.779705×10^{-50} +	1.364857×10^{-59} +	1.997167×10^{-90} +	4.217641×10^{-65} +	$\mathbf{5.174038 \times 10^{-91}}$
	Std	5.462664×10^{-62}	3.765824×10^{-57}	4.345121×10^{-50}	1.995794×10^{-59}	3.848453×10^{-90}	1.137639×10^{-64}	1.715634×10^{-90}
F20	Mean	$\mathbf{1.161948 \times 10^{-28}}$ =	$\mathbf{1.161948 \times 10^{-28}}$ =	$\mathbf{1.161948 \times 10^{-28}}$ =	$\mathbf{1.161948 \times 10^{-28}}$ =	$\mathbf{1.161948 \times 10^{-28}}$ =	$\mathbf{1.161948 \times 10^{-28}}$ =	$\mathbf{1.161948 \times 10^{-28}}$
	Std	2.280406×10^{-44}	2.280406×10^{-44}	2.280406×10^{-44}	2.280406×10^{-44}	2.280406×10^{-44}	2.280406×10^{-44}	2.280406×10^{-44}
F21	Mean	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$ =
	Std	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}
F22	Mean	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$ =	$\mathbf{0.000000 \times 10^{0}}$ =
	Std	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}	0.000000×10^{0}
+/=/−		14/8/0	16/6/0	15/7/0	14/8/0	13/9/0	14/8/0	\

4.2. Effectiveness Analysis of the Proposed Strategy Pool and Self-adaptive Mechanism

In experiment 1, we wanted to probe the following two problems:

- Problem 1: Is it necessary to assemble the five different strategies?
- Problem 2: Is the self-adaptive mechanism required and are the results affected when the self-adaptive selection mechanism is replaced by a random selection mechanism?

To solve problem 1, each single strategy was embedded into the original ABC, to make a result comparison between each strategy and SAABC-CS. As for problem 2, we tested two different strategy selections. One was the random strategy selection mechanism, and the other was the self-adaptive selection mechanism. The ABC algorithms including various search methods mentioned below examined the efficacy of the strategy pool and the self-adaptive mechanism.

- ABC-rand: the original ABC with rand strategy;
- ABC-pbest-1: the original ABC with pbest-1 strategy;
- ABC-pbest-2: the original ABC with pbest-2 strategy;
- ABC-current-to-pbest: the original ABC with current-to-pbest strategy;
- ABC-pbest-to-rand: the original ABC with pbest-to-rand strategy;
- SAABC-CS: ABC with self-adpative selection mechanism in the strategy pool;
- RABC-CS: ABC with random selection mechanism in the strategy pool.

The fundamental settings for the seven algorithms listed above were as follows: the SN, D, limit, MaxFEs, and running times were set to 100, 30, 100, 5000·D, and 30, respectively. Table 1 displays the outcomes of the ABC using the RABC-CS and SAABC-CS single search techniques on the fundamental 22 functions. SAABC-CS in Table 1 outperformed ABC-rand, ABC-pbest-1, ABC-pbest-2, ABC-current-to-pbest, and ABC-pbest-to-rand on 14, 16, 15, 14, and 13 of the 22 test functions, respectively. This demonstrated that the combination of five techniques increased the test function accuracy. SAABC-CS outperformed RABC-CS, which chooses methods at random, on 14 functions, while being comparable for 8 of them. In this test suite, the self-adaptive selection mechanism performed better than the random selection method.

To determine the contribution of each strategy, we counted the use times of each strategy during the whole processes for two multimodal functions ($f14$, $f17$) and three unimodal functions ($f2$, $f3$, $f7$). The outcomes are displayed in Figure 6. As shown in the figure, the strategies with the highest frequency for f2, f3, f7, f14, and f17 were "pbest-1", "pbest-to-rand", "current-to-pbest", "pbest-2", and "rand", respectively. Among the five single strategies in Table 1, "pbest-1", "pbest-to-rand", "current-to-pbest", "pbest-2", and "rand" produced the best results for the f2, f3, f7, f14, and f17 functions, respectively. Taking function f7 as an example, "current-to-pbest" had the best performance and "pbest-to-rand" came second among the results of the five single techniques shown in Table 1. According to Figure 6, the suggested self-adaptive mechanism chose the strategy "current-to-pbest" most, followed by "pbest-to-rand". This phenomenon explains why the adaptive selection approach worked so well. The self-adaptive mechanism had the capability to adaptively choose the best approach in accordance with the requirements of the problem, so that the quality of the solutions was improved.

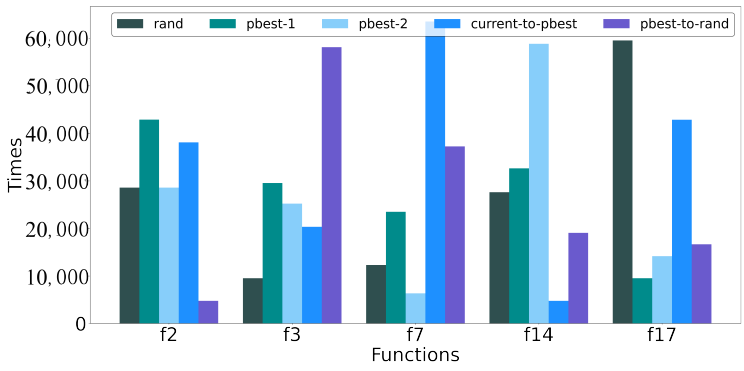


Figure 6. The frequency of strategies by function.

Table 2. Results of SAABC-CS vs. the other five ABC algorithms on CEC2013 function with D = 30.

Function		ABC	ABCNG	KFABC	SABC-GB	MGABC	SAABC-CS
F1	Mean	1.045919×10^{-12}+	1.193484×10^{-10} +	7.217691×10^{3} +	1.818989×10^{-13} +	**0.000000×10^{0}** =	**0.000000×10^{0}**
	Std	2.668884×10^{-13}	3.755755×10^{-10}	2.219269×10^{4}	1.016846×10^{-13}	0.000000×10^{0}	0.000000×10^{0}
F2	Mean	1.584525×10^{7} +	2.021825×10^{7} +	3.260723×10^{7} +	2.684755×10^{7} +	1.836199×10^{6} +	**6.459514×10^{5}**
	Std	3.053047×10^{6}	4.651604×10^{6}	5.793827×10^{6}	1.144070×10^{7}	6.914972×10^{5}	1.937672×10^{5}
F3	Mean	1.497846×10^{9} +	2.727830×10^{9} +	1.174722×10^{10} +	2.548842×10^{9} +	5.483760×10^{8} +	**5.004087×10^{7}**
	Std	5.525781×10^{8}	1.142430×10^{9}	2.631946×10^{9}	1.558031×10^{9}	6.431569×10^{8}	4.567237×10^{7}
F4	Mean	6.445392×10^{4} +	8.824601×10^{4} +	7.019652×10^{4} +	8.705039×10^{4} +	3.886162×10^{4} +	**9.954682×10^{3}**
	Std	8.712809×10^{3}	9.121559×10^{3}	1.868345×10^{1}	1.561130×10^{4}	4.226200×10^{3}	5.997636×10^{3}
F5	Mean	2.320121×10^{-10} +	7.825637×10^{-7} +	6.645994×10^{0} +	1.136868×10^{-13} +	8.293133×10^{1} +	**6.821210×10^{-14}**
	Std	9.315427×10^{-11}	1.813981×10^{-6}	1.137336×10^{1}	0.000000×10^{0}	2.104354×10^{2}	6.226885×10^{-14}
F6	Mean	**1.942986×10^{1}** −	2.343924×10^{1} −	9.750413×10^{1} +	2.043955×10^{1} −	4.370608×10^{1} −	4.797174×10^{1}
	Std	2.497262×10^{0}	1.306460×10^{0}	2.896130×10^{1}	1.866870×10^{0}	2.784367×10^{1}	3.105646×10^{1}
F7	Mean	1.102697×10^{2} +	1.233136×10^{2} +	9.123335×10^{1} +	1.380252×10^{2} +	1.797136×10^{2} +	**6.868069×10^{1}**
	Std	1.167065×10^{1}	1.915723×10^{1}	9.150610×10^{0}	1.887645×10^{1}	1.126540×10^{2}	5.374762×10^{1}
F8	Mean	**2.096425×10^{1}** =	**2.096832×10^{1}** =	2.118192×10^{1} +	2.107899×10^{1} +	2.107992×10^{1} +	**2.095631×10^{1}**
	Std	4.534480×10^{-2}	7.410263×10^{-2}	5.625405×10^{-2}	5.680530×10^{-2}	2.706550×10^{-2}	3.660565×10^{-2}
F9	Mean	3.053202×10^{1} −	2.836110×10^{1} −	3.278991×10^{1} +	3.176783×10^{1} +	2.785300×10^{1} −	3.136165×10^{1}
	Std	2.089771×10^{0}	1.122232×10^{0}	1.361819×10^{0}	2.538868×10^{0}	4.190502×10^{0}	7.213594×10^{0}
F10	Mean	6.111205×10^{0} +	1.573150×10^{1} +	7.155009×10^{1} +	8.302503×10^{0} +	2.671815×10^{-1} +	**2.607005×10^{-1}**
	Std	1.180270×10^{0}	5.412964×10^{0}	1.259467×10^{1}	2.664426×10^{0}	1.315116×10^{-1}	9.298362×10^{-2}
F11	Mean	6.400001×10^{-11} −	1.570459×10^{-9} −	2.694004×10^{2} −	**5.684342×10^{-14}** −	1.145733×10^{2} −	5.492158×10^{1}
	Std	1.219411×10^{-10}	3.338045×10^{-9}	4.685348×10^{2}	0.000000×10^{0}	4.619428×10^{1}	2.002800×10^{1}
F12	Mean	2.454513×10^{2} +	1.618361×10^{2} +	6.321330×10^{2} +	1.721923×10^{2} +	1.541547×10^{2} +	**1.493833×10^{2}**
	Std	3.657845×10^{1}	2.710955×10^{1}	3.283000×10^{2}	3.414756×10^{1}	6.149077×10^{1}	7.723931×10^{1}
F13	Mean	3.282447×10^{2} +	2.183721×10^{2} +	9.764368×10^{2} +	2.292618×10^{2} +	2.179081×10^{2} +	**1.307685×10^{2}**
	Std	2.198303×10^{1}	2.198283×10^{1}	2.735796×10^{2}	2.036628×10^{1}	4.048664×10^{1}	3.548699×10^{1}
F14	Mean	6.335722×10^{1} −	**4.171621×10^{1}** −	3.397454×10^{2} −	7.386230×10^{0} −	2.407904×10^{3} +	2.392425×10^{3}
	Std	3.525780×10^{1}	2.524628×10^{0}	4.087044×10^{2}	3.475162×10^{0}	9.732670×10^{2}	9.783598×10^{2}
F15	Mean	4.757794×10^{3} +	4.634326×10^{3} +	5.592640×10^{3} +	4.836574×10^{3} +	5.014348×10^{3} +	**3.790289×10^{3}**
	Std	2.395951×10^{2}	3.819604×10^{2}	4.371912×10^{2}	9.120662×10^{2}	1.700247×10^{3}	6.287230×10^{2}
F16	Mean	1.848622×10^{0} −	**1.682927×10^{0}** −	3.968684×10^{0} +	2.375125×10^{0} +	2.244456×10^{0} +	2.225754×10^{0}
	Std	1.879309×10^{-1}	2.462413×10^{-1}	9.809905×10^{-1}	5.279497×10^{-1}	1.364966×10^{0}	8.320054×10^{-1}
F17	Mean	3.342858×10^{1} −	**3.044152×10^{1}** −	1.298987×10^{2} +	3.046909×10^{1} −	9.441017×10^{1} +	7.149418×10^{1}
	Std	7.271755×10^{-1}	9.453500×10^{-3}	7.430159×10^{1}	7.645190×10^{-2}	1.605471×10^{1}	1.131667×10^{1}
F18	Mean	3.678532×10^{2} +	2.255822×10^{2} +	9.230954×10^{2} +	2.668022×10^{2} +	1.520744×10^{2} +	**8.645602×10^{1}**
	Std	3.185708×10^{1}	2.077005×10^{1}	5.075368×10^{1}	3.107039×10^{1}	4.527983×10^{1}	6.044520×10^{1}
F19	Mean	2.635527×10^{0} −	**6.979700×10^{-1}** −	4.030256×10^{5} +	8.452421×10^{-1} −	1.057998×10^{1} +	4.135579×10^{0}
	Std	4.297100×10^{-1}	2.661896×10^{-1}	3.467954×10^{5}	3.281864×10^{-1}	3.985207×10^{0}	1.752942×10^{0}
F20	Mean	1.441240×10^{1} +	1.425405×10^{1} +	1.500000×10^{1} +	1.466189×10^{1} +	1.451120×10^{1} +	**1.142710×10^{1}**
	Std	2.605160×10^{-1}	5.229196×10^{-1}	9.678038×10^{-11}	6.368384×10^{-1}	9.516934×10^{-3}	7.389737×10^{-1}
F21	Mean	**2.514810×10^{2}** −	2.891845×10^{2} −	2.742664×10^{3} +	3.661265×10^{2} +	3.143544×10^{2} −	3.574177×10^{2}
	Std	2.600244×10^{1}	7.660174×10^{1}	4.287399×10^{-13}	1.117488×10^{2}	4.539264×10^{1}	7.862236×10^{1}
F22	Mean	2.208385×10^{2} −	1.568746×10^{2} −	4.160514×10^{2} −	**1.086814×10^{2}** −	2.290799×10^{3} +	1.489146×10^{3}
	Std	3.851348×10^{1}	4.215301×10^{1}	1.020057×10^{2}	2.055194×10^{1}	9.996683×10^{2}	4.555037×10^{2}
F23	Mean	5.492901×10^{3} +	5.268257×10^{3} +	7.665719×10^{3} +	6.149480×10^{3} +	5.556585×10^{3} +	**4.022977×10^{3}**
	Std	2.729496×10^{2}	4.193448×10^{2}	2.084489×10^{2}	4.333990×10^{2}	1.489409×10^{3}	7.975158×10^{2}
F24	Mean	2.852008×10^{2} +	2.744805×10^{2} +	2.867909×10^{2} +	2.804525×10^{2} +	2.784434×10^{2} +	**2.517143×10^{2}**
	Std	7.092098×10^{0}	2.768032×10^{0}	8.840250×10^{0}	5.685422×10^{0}	8.978342×10^{0}	1.411454×10^{1}
F25	Mean	3.167588×10^{2} +	2.737370×10^{2} +	3.037339×10^{2} +	2.831343×10^{2} +	2.845589×10^{2} +	**2.758435×10^{2}**
	Std	5.437050×10^{0}	4.131749×10^{0}	2.941593×10^{0}	6.008858×10^{0}	1.317743×10^{1}	1.523675×10^{1}
F26	Mean	**2.012222×10^{2}** −	2.017343×10^{2} −	2.035708×10^{2} −	**2.012044×10^{2}** −	3.060999×10^{2} +	2.567051×10^{2}
	Std	1.401790×10^{-1}	4.463412×10^{-1}	1.315997×10^{0}	3.262038×10^{-1}	9.151166×10^{1}	7.811489×10^{1}
F27	Mean	4.035857×10^{2} −	4.637355×10^{2} −	1.521531×10^{3} +	**4.000881×10^{2}** −	1.065358×10^{3} +	9.942069×10^{2}
	Std	2.438432×10^{0}	1.879815×10^{2}	1.429126×10^{2}	1.966334×10^{-1}	1.509388×10^{2}	1.774184×10^{2}
F28	Mean	3.217317×10^{2} +	**3.000450×10^{2}** =	3.548491×10^{3} +	**3.000000×10^{2}** =	3.000000×10^{2} **=**	**3.000000×10^{2}**
	Std	9.928230×10^{1}	4.776208×10^{-2}	3.566458×10^{2}	7.129640×10^{-7}	3.259904×10^{-13}	1.906586×10^{-13}
+/=/−		16/1/11	15/2/11	25/0/3	19/1/8	23/2/3	\

Table 3. Results of SAABC-CS vs. other five ABC algorithm on CEC2013 function with D = 50.

Function		ABC	ABCNG	KFABC	SABC-GB	MGABC	SAABC-CS
F1	Mean	2.955858×10^{-12}+	3.148009×10^{-8} +	1.018143×10^{4} +	2.273737×10^{-13} +	**4.547474×10^{-14}** =	**4.547474×10^{-14}**
	Std	1.169250×10^{-12}	9.954543×10^{-8}	2.871279×10^{4}	0.000000×10^{0}	9.586916×10^{-14}	1.016846×10^{-13}
F2	Mean	3.284758×10^{7} +	4.618962×10^{7} +	1.708788×10^{9} +	4.637076×10^{7} +	1.971770×10^{6} +	**7.914809×10^{5}**
	Std	5.558586×10^{6}	5.790349×10^{6}	2.623811×10^{9}	1.114626×10^{7}	7.040444×10^{5}	4.080623×10^{7}
F3	Mean	7.841954×10^{9} +	1.847470×10^{10} +	2.851026×10^{10} +	1.762672×10^{10} +	8.447470×10^{8} +	**2.542380×10^{8}**
	Std	3.259891×10^{9}	6.121788×10^{9}	5.797317×10^{9}	6.695196×10^{9}	7.333293×10^{8}	2.087111×10^{8}
F4	Mean	1.274341×10^{5} +	1.642763×10^{5} +	2.234841×10^{5} +	1.694570×10^{5} +	8.924475×10^{4} +	**1.724965×10^{4}**
	Std	7.008384×10^{3}	1.489793×10^{4}	3.931104×10^{4}	2.075625×10^{4}	1.832230×10^{4}	8.075878×10^{3}
F5	Mean	2.861123×10^{-9} +	7.562626×10^{-6} +	3.134081×10^{4} +	2.046363×10^{-13} +	1.783583×10^{1} +	**1.136868×10^{-13}**
	Std	1.875903×10^{-9}	1.770777×10^{-5}	2.634133×10^{4}	5.084230×10^{-14}	5.640184×10^{1}	0.000000×10^{0}
F6	Mean	4.277263×10^{1} −	4.490482×10^{1} −	2.040206×10^{3} +	4.249225×10^{1} −	**4.220742×10^{1}** −	4.458985×10^{1}
	Std	3.912023×10^{0}	1.407079×10^{0}	5.172172×10^{3}	1.429593×10^{0}	3.628206×10^{0}	2.554739×10^{0}

Table 3. *Cont*.

Function		ABC	ABCNG	KFABC	SABC-GB	MGABC	SAABC-CS
F7	Mean	1.622092×10^{2} +	1.592080×10^{2} +	1.414640×10^{2} +	1.666456×10^{2} +	1.179465×10^{2} +	$\mathbf{5.708652 \times 10^{1}}$
	Std	1.449257×10^{1}	1.083587×10^{1}	8.759337×10^{0}	1.962182×10^{1}	2.313107×10^{1}	1.366854×10^{1}
F8	Mean	2.115074×10^{1} +	2.115255×10^{1} +	2.132399×10^{1} +	2.125988×10^{1} +	2.123293×10^{1} +	$\mathbf{2.114303 \times 10^{1}}$
	Std	4.048518×10^{-2}	3.072095×10^{-2}	4.592058×10^{-2}	2.544278×10^{-2}	3.146337×10^{-2}	3.860337×10^{-2}
F9	Mean	5.897591×10^{1} +	5.745982×10^{1} +	6.670232×10^{1} +	6.076537×10^{1} +	5.754785×10^{1} +	$\mathbf{4.684075 \times 10^{1}}$
	Std	1.893957×10^{0}	1.483945×10^{0}	1.662834×10^{0}	1.559200×10^{0}	7.347343×10^{0}	1.497032×10^{1}
F10	Mean	1.347232×10^{1} +	4.771359×10^{1} +	3.118192×10^{2} +	2.049230×10^{1} +	3.413696×10^{1} +	$\mathbf{2.517742 \times 10^{-1}}$
	Std	2.002235×10^{0}	1.767519×10^{1}	9.390496×10^{1}	3.725336×10^{0}	1.072055×10^{2}	1.011803×10^{-1}
F11	Mean	4.159881×10^{-4} −	1.963591×10^{-8} −	2.184322×10^{2} +	$\mathbf{5.684342 \times 10^{-14}}$ −	2.335787×10^{2} +	1.392530×10^{2}
	Std	1.312952×10^{-3}	6.206800×10^{-8}	4.393977×10^{1}	0.000000×10^{0}	6.621192×10^{1}	5.118038×10^{1}
F12	Mean	7.178075×10^{2} +	4.917360×10^{2} +	1.086795×10^{0} +	5.622186×10^{2} +	2.735126×10^{2} +	$\mathbf{1.388959 \times 10^{2}}$
	Std	6.373474×10^{1}	5.148087×10^{1}	1.126136×10^{1}	5.881967×10^{1}	4.636084×10^{1}	1.758279×10^{1}
F13	Mean	7.814717×10^{2} +	5.461840×10^{2} +	1.387675×10^{3} +	5.990427×10^{2} +	4.531404×10^{2} +	$\mathbf{3.072911 \times 10^{2}}$
	Std	6.938212×10^{1}	3.117511×10^{1}	8.629146×10^{1}	6.079101×10^{1}	7.781740×10^{1}	7.755705×10^{1}
F14	Mean	2.575095×10^{2} −	$\mathbf{1.101760 \times 10^{1}}$ −	8.542596×10^{2} −	2.786400×10^{1} −	5.347860×10^{3} +	4.776190×10^{3}
	Std	1.129973×10^{2}	4.446142×10^{0}	7.478447×10^{2}	1.652054×10^{1}	1.308783×10^{3}	1.761472×10^{3}
F15	Mean	9.827417×10^{3} +	8.907545×10^{3} +	1.171550×10^{4} +	1.086651×10^{4} +	1.137648×10^{4} +	$\mathbf{8.412513 \times 10^{3}}$
	Std	3.921616×10^{2}	4.483995×10^{2}	5.934273×10^{2}	1.734875×10^{3}	3.282637×10^{3}	1.422399×10^{3}
F16	Mean	2.701649×10^{0} −	$\mathbf{2.242826 \times 10^{0}}$ −	4.081204×10^{0} −	3.954293×10^{0} −	3.699498×10^{0} −	2.867917×10^{0}
	Std	9.149871×10^{-2}	2.512675×10^{-1}	1.280502×10^{0}	4.807582×10^{-1}	6.058890×10^{-1}	1.508652×10^{-1}
F17	Mean	6.261697×10^{1} −	5.086540×10^{1} −	3.152645×10^{2} +	$\mathbf{5.084679 \times 10^{1}}$ −	2.824810×10^{2} +	1.423049×10^{2}
	Std	2.765426×10^{0}	3.099526×10^{-2}	8.177475×10^{1}	1.352002×10^{-1}	6.248677×10^{1}	2.138828×10^{1}
F18	Mean	9.238280×10^{2} +	5.295994×10^{2} +	1.238050×10^{3} +	6.694845×10^{2} +	2.891465×10^{2} +	$\mathbf{1.295341 \times 10^{2}}$
	Std	4.428796×10^{1}	4.794994×10^{1}	9.605859×10^{0}	6.861874×10^{1}	1.143260×10^{2}	3.810039×10^{1}
F19	Mean	6.997984×10^{1} −	$\mathbf{1.608722 \times 10^{0}}$ −	8.691579×10^{4} +	1.969550×10^{0} −	3.695087×10^{1} +	1.020953×10^{1}
	Std	7.562652×10^{-1}	2.366303×10^{-1}	2.742856×10^{5}	5.667733×10^{-1}	2.164269×10^{1}	3.568900×10^{0}
F20	Mean	2.451305×10^{1} +	2.443238×10^{1} +	2.500000×10^{1} +	2.479311×10^{1} +	2.484394×10^{1} +	$\mathbf{1.906000 \times 10^{1}}$
	Std	6.179035×10^{-2}	1.913468×10^{-1}	3.626571×10^{-8}	4.626115×10^{-1}	4.935189×10^{-1}	4.442649×10^{-1}
F21	Mean	3.002346×10^{2} −	2.331136×10^{2} −	3.937298×10^{2}+	$\mathbf{2.000024 \times 10^{2}}$ −	1.007922×10^{3} +	7.091540×10^{0}
	Std	3.655648×10^{1}	5.858767×10^{1}	8.291218×10^{2}	4.499760×10^{-3}	1.475861×10^{2}	2.846257×10^{2}
F22	Mean	5.182953×10^{2} −	1.075276×10^{2} −	1.553978×10^{3} −	$\mathbf{6.506353 \times 10^{1}}$ −	5.641707×10^{3} +	3.417625×10^{3}
	Std	1.260235×10^{2}	5.535060×10^{1}	4.507743×10^{2}	5.047782×10^{1}	1.105454×10^{3}	6.778751×10^{2}
F23	Mean	1.159107×10^{4} +	1.063746×10^{4} +	1.287130×10^{4} +	1.272815×10^{4} +	1.166782×10^{0} +	$\mathbf{7.758915 \times 10^{3}}$
	Std	6.086164×10^{2}	7.287601×10^{2}	1.599906×10^{3}	1.319689×10^{3}	2.959131×10^{3}	7.102152×10^{2}
F24	Mean	3.744423×10^{2} +	3.510602×10^{2} +	4.166738×10^{2} +	3.583240×10^{2} +	3.667745×10^{2} +	$\mathbf{2.967712 \times 10^{2}}$
	Std	5.975679×10^{0}	5.223262×10^{0}	9.890606×10^{0}	1.308944×10^{1}	1.378969×10^{1}	1.829007×10^{1}
F25	Mean	4.374945×10^{2} +	3.483497×10^{0} +	3.759264×10^{2} +	3.555845×10^{2} +	3.658872×10^{0} +	$\mathbf{3.445000 \times 10^{2}}$
	Std	3.011667×10^{0}	4.612998×10^{0}	1.557984×10^{1}	1.369593×10^{1}	1.160330×10^{1}	3.316366×10^{1}
F26	Mean	$\mathbf{2.030621 \times 10^{2}}$ −	2.046948×10^{2} −	2.100765×10^{2} −	2.035836×10^{2} −	4.238846×10^{2} +	4.035065×10^{2}
	Std	5.837760×10^{-1}	1.270617×10^{0}	1.407218×10^{0}	6.566464×10^{-1}	8.086476×10^{1}	2.080650×10^{1}
F27	Mean	8.799567×10^{2} −	1.648409×10^{3} +	1.906722×10^{3} +	$\mathbf{7.090895 \times 10^{2}}$ −	1.828519×10^{3} +	1.277921×10^{3}
	Std	7.316749×10^{2}	4.377836×10^{2}	6.292964×10^{1}	6.907071×10^{2}	1.516850×10^{2}	2.410391×10^{2}
F28	Mean	$\mathbf{4.000096 \times 10^{2}}$ −	$\mathbf{4.000096 \times 10^{2}}$ −	9.274103×10^{3} +	$\mathbf{4.000000 \times 10^{2}}$ −	7.603380×10^{2} −	1.095181×10^{3}
	Std	8.593386×10^{-3}	4.890569×10^{-5}	2.656043×10^{3}	3.405390×10^{-12}	1.139489×10^{3}	1.554472×10^{3}
+/=/−		17/0/11	19/0/9	25/0/3	19/0/9	25/1/2	\

Table 4. Results of SAABC-CS vs. the other five ABC algorithms on CEC2013 function with D = 100.

Function		ABC	ABCNG	KFABC	SABC-GB	MGABC	SAABC-CS
F1	Mean	1.973604×10^{-11}+	4.128162×10^{-7} +	2.451098×10^{4} +	2.451098×10^{4} +	1.118425×10^{-4} +	$\mathbf{2.273737 \times 10^{-13}}$
	Std	3.802877×10^{-12}	7.287129×10^{-7}	5.990424×10^{4}	1.016846×10^{-13}	2.530758×10^{-4}	0.000000×10^{0}
F2	Mean	8.834536×10^{7} +	1.240612×10^{8} +	1.198907×10^{8} +	1.285546×10^{8} +	5.389145×10^{6} +	$\mathbf{1.785271 \times 10^{6}}$
	Std	1.184489×10^{7}	1.967212×10^{7}	1.284506×10^{7}	3.465274×10^{7}	1.021979×10^{6}	1.982445×10^{5}
F3	Mean	4.482419×10^{10} +	7.905068×10^{10} +	3.991188×10^{23} +	6.912422×10^{10} +	1.350896×10^{10} +	$\mathbf{1.334383 \times 10^{9}}$
	Std	9.027770×10^{9}	1.791850×10^{10}	1.262124×10^{24}	1.232375×10^{10}	8.226565×10^{9}	1.222671×10^{9}
F4	Mean	3.094007×10^{5} +	3.681476×10^{5} +	2.615840×10^{5} +	3.890927×10^{5} +	2.173122×10^{5} +	$\mathbf{3.952361 \times 10^{4}}$
	Std	1.303058×10^{4}	2.133351×10^{4}	5.347582×10^{2}	1.611143×10^{4}	6.106829×10^{4}	2.153741×10^{4}
F5	Mean	8.254081×10^{-7} +	2.557590×10^{-5} +	3.772101×10^{4} +	4.547474×10^{-13} +	8.518334×10^{-6} +	$\mathbf{1.591616 \times 10^{-13}}$
	Std	7.529267×10^{-7}	7.720090×10^{-5}	5.419016×10^{4}	8.038873×10^{-14}	2.693720×10^{-5}	6.226885×10^{-14}
F6	Mean	2.142622×10^{2} +	9.698993×10^{1} +	1.709420×10^{3} +	1.009548×10^{2} +	1.488648×10^{2} +	$\mathbf{8.023747 \times 10^{1}}$
	Std	2.430805×10^{1}	2.907379×10^{0}	6.999790×10^{2}	1.889868×10^{1}	4.056920×10^{1}	5.311755×10^{1}
F7	Mean	3.054039×10^{3} +	3.234884×10^{2} +	9.620188×10^{3} +	2.147341×10^{3} +	2.733054×10^{3} +	$\mathbf{1.237953 \times 10^{2}}$
	Std	9.462866×10^{2}	9.118325×10^{1}	3.860683×10^{3}	1.146374×10^{3}	5.572922×10^{3}	2.587804×10^{1}
F8	Mean	2.128398×10^{1} +	2.128740×10^{1} +	2.142877×10^{1} +	2.140472×10^{1} +	2.135731×10^{1} +	$\mathbf{2.123126 \times 10^{1}}$
	Std	3.115940×10^{-2}	3.286301×10^{-2}	3.175913×10^{-2}	2.885485×10^{-2}	2.906741×10^{-2}	2.846930×10^{-2}
F9	Mean	1.424419×10^{2} +	1.382971×10^{2} +	1.650088×10^{2} +	1.420014×10^{2} +	1.438865×10^{2} +	$\mathbf{1.225657 \times 10^{2}}$
	Std	1.991745×10^{0}	2.798468×10^{0}	4.284012×10^{0}	2.512077×10^{0}	9.806690×10^{0}	2.133002×10^{1}
F10	Mean	2.923123×10^{1} +	1.539758×10^{2} +	5.712395×10^{3} +	6.950258×10^{1} +	$\mathbf{7.182894 \times 10^{-2}}$ −	1.660239×10^{-1}
	Std	6.475596×10^{0}	2.389479×10^{1}	1.551510×10^{4}	1.957730×10^{1}	3.627664×10^{-2}	8.312484×10^{-2}
F11	Mean	6.092375×10^{-1} −	3.784635×10^{-11} −	8.378384×10^{2} +	$\mathbf{1.477929 \times 10^{-13}}$ −	5.465159×10^{2} +	2.845364×10^{2}
	Std	5.116787×10^{-1}	8.196940×10^{-11}	1.395362×10^{3}	3.113442×10^{-14}	1.112934×10^{2}	2.537027×10^{1}

Table 4. *Cont.*

Function		ABC	ABCNG	KFABC	SABC-GB	MGABC	SAABC-CS
F12	Mean	2.198673×10^3 +	1.517617×10^3 +	2.697353×10^3 +	1.955326×10^3 +	8.101987×10^2 +	$\mathbf{3.768888 \times 10^2}$
	Std	7.354249×10^1	9.996793×10^1	2.658300×10^2	2.338236×10^2	1.770356×10^2	4.834688×10^1
F13	Mean	2.503947×10^3 +	1.733014×10^3 +	3.265827×10^3 +	2.047888×10^3 +	1.215428×10^3 +	$\mathbf{6.376625 \times 10^2}$
	Std	7.441021×10^1	9.697171×10^1	1.733385×10^2	2.217272×10^2	1.815931×10^2	1.038178×10^2
F14	Mean	1.249470×10^3 −	$\mathbf{4.918872 \times 10^1}$ −	3.312805×10^3 −	1.564673×10^2 −	1.566715×10^4 +	1.164011×10^4
	Std	2.042146×10^2	1.145955×10^1	7.614969×10^2	3.933930×10^1	6.336429×10^3	1.692848×10^3
F15	Mean	2.139087×10^4 +	1.804621×10^4 +	1.986577×10^4 +	2.182511×10^4 +	1.969055×10^4 +	$\mathbf{1.546327 \times 10^4}$
	Std	7.133549×10^2	1.644285×10^3	9.934635×10^2	4.009070×10^3	4.438910×10^3	1.456028×10^3
F16	Mean	3.245700×10^0 −	$\mathbf{2.433718 \times 10^0}$ −	5.258963×10^0 +	4.341247×10^0 +	4.146268×10^0 +	3.412060×10^0
	Std	1.671285×10^{-1}	1.727265×10^{-1}	4.145646×10^{-1}	3.074977×10^{-1}	9.278684×10^{-1}	1.300912×10^0
F17	Mean	1.455926×10^2 −	$\mathbf{1.018400 \times 10^2}$ −	4.054080×10^2 −	$\mathbf{1.020771 \times 10^2}$ −	8.785634×10^2 +	4.476043×10^2
	Std	5.280956×10^0	1.063732×10^{-1}	7.338916×10^1	5.371392×10^{-1}	1.398662×10^2	4.923496×10^1
F18	Mean	3.032877×10^3 +	1.667279×10^3 +	3.006562×10^3 +	2.012308×10^3 +	1.132124×10^3 +	$\mathbf{3.217418 \times 10^2}$
	Std	1.321905×10^2	6.506401×10^1	2.211562×10^3	2.035596×10^2	3.637035×10^2	7.163749×10^1
F19	Mean	1.868629×10^1 −	$\mathbf{3.529313 \times 10^0}$ −	9.648661×10^2 +	4.940257×10^0 −	1.570489×10^2 +	3.911692×10^1
	Std	1.710116×10^0	6.629051×10^{-1}	8.586362×10^2	1.085188×10^0	6.816761×10^1	9.322297×10^0
F20	Mean	5.000000×10^1 +	5.000000×10^1 +	5.000000×10^1 +	5.000000×10^1 +	5.000000×10^1 +	$\mathbf{4.922633 \times 10^1}$
	Std	0.000000×10^0	0.000000×10^0	0.000000×10^0	0.000000×10^0	0.000000×10^0	1.729981×10^0
F21	Mean	4.266808×10^2 +	4.139761×10^2 +	6.277225×10^3 +	$\mathbf{3.755167 \times 10^2}$ −	4.00023×10^2 =	4.000230×10^2
	Std	2.009050×10^{-2}	4.003411×10^1	2.459363×10^3	6.893975×10^1	7.239635×10^{-2}	8.023434×10^{-12}
F22	Mean	1.826683×10^3 −	2.272431×10^2 −	2.441767×10^3 −	$\mathbf{2.082828 \times 10^2}$ −	1.642002×10^4 +	9.973881×10^3
	Std	2.498711×10^2	4.141507×10^1	1.709820×10^3	6.767953×10^1	3.497406×10^3	1.449154×10^3
F23	Mean	2.644424×10^4 +	2.296186×10^4 +	2.573263×10^4 +	2.612368×10^4 +	2.415738×10^4 +	$\mathbf{1.705627 \times 10^4}$
	Std	4.200315×10^2	1.326126×10^3	3.682603×10^2	2.487084×10^3	4.569038×10^3	2.441021×10^3
F24	Mean	6.252343×10^2 +	5.569459×10^2 +	6.226265×10^2 +	5.770593×10^2 +	5.814677×10^2 +	$\mathbf{4.186013 \times 10^2}$
	Std	1.190408×10^1	9.093040×10^0	4.713148×10^1	1.216896×10^1	1.995748×10^1	2.670765×10^1
F25	Mean	7.700743×10^2 +	5.390881×10^2 +	6.198248×10^2 +	5.628228×10^2 +	5.650778×10^2 +	$\mathbf{5.586434 \times 10^2}$
	Std	1.579620×10^1	1.433180×10^1	8.338052×10^0	7.234443×10^0	2.894703×10^1	5.314425×10^1
F26	Mean	$\mathbf{2.098754 \times 10^2}$ −	2.146916×10^2 −	7.292073×10^2 +	2.109556×10^2 −	6.618175×10^2 +	5.612805×10^2
	Std	9.910123×10^{-1}	2.330025×10^0	1.252176×10^1	1.674934×10^0	2.288813×10^1	5.874442×10^1
F27	Mean	2.891602×10^3 −	3.912540×10^3 +	4.681923×10^3 +	$\mathbf{1.216460 \times 10^3}$ −	3.961627×10^3 +	3.188342×10^3
	Std	1.942001×10^3	7.631826×10^1	8.980030×10^1	1.565963×10^3	2.252456×10^2	2.514344×10^2
F28	Mean	6.594381×10^3 +	3.911700×10^3 +	1.316997×10^4 +	3.765292×10^3 +	7.956850×10^3 +	$\mathbf{2.721174 \times 10^3}$
	Std	9.182439×10^2	8.363343×10^2	9.754517×10^2	1.311856×10^2	2.886238×10^3	1.197070×10^2
+/=/−		20/0/8	20/0/8	25/0/3	19/0/9	26/1/1	\

4.3. Comparison for CEC2013 Functions

In this section, we used the CEC2013 testing suite in experiment 2, to further illustrate the effectiveness of our algorithm SAABC-CS in handling complicated functions and high-dimensional issues. Since the CEC2013 test functions are more complicated than the 22 fundamental scalar functions, it was challenging to find the global best solution for CEC2013. We compared our algorithm SAABC-CS with the original ABC [12] and four additional cutting-edge ABC algorithms (ABCNG [33], KFABC [19], SABC-GB [15], and MGABC [30]). We measured the outcomes on D = 30, 50, and 100, to examine the comprehensive performance of these algorithms in several dimensions. For the fairness of the experiments, the methods were evaluated using the same parameters, SN = 100, limit = 1000, and MaxFEs = 10,000·D. The final statistical outcomes of the six related algorithms are encapsulated in Tables 2–4 based on 30 independent runs.

For D = 30, the results of the above ABC variations are reported in Table 2. For 21, 17, 25, 20, and 25 of the 28 test functions, our algorithm SAABC-CS outperformed or equaled the ABC, ABCNG, KFABC, SABC-GB, and MGABC in terms of outcomes. Additionally, SAABC-CS achieved the best results on 17 functions of F1–F5, F7, F8, F10, F12, F13, F15, F18, F20, F23-F25, and F28 when it was compared with the other algorithms. The statistical results of all algorithms on CEC2013 with the dimension D = 50 are shown in Table 3. SAABC-CS outperformed or equaled the ABC, ABCNG, KFABC, SABC-GB, and MGABC on 21, 17, 25, 20, and 25 out of 28 functions, respectively. The outcomes of all algorithms with the criterion D = 100 are displayed in Table 4. On 24, 20, 25, 19, and 25 out of 28 functions, SAABC-CS outperformed or equaled the ABC, ABCNG, KFABC, SABC-GB, and MGABC, with comparable or superior performance. Moreover, on F1–F9, F12–F13, F15–F20, F23–F25, and F28, SAABC-CS always achieved the best value out of all algorithms. As a result, SAABC-CS had the highest overall performance of the six algorithms that were examined.

To be more intuitive, we used the Friedman test [34] to rank all algorithms across all test problems, with the results shown in Table 5 and Figure 7. It is worth noting that the Friedman test uses the post hoc technique [35–38], and the lower the ranking, the better the overall performance of the algorithm. From the data, we can see the average rank of SAABC-CS was always the first for 30, 50, and 100 dimensions. Therefore, SAABC-CS's effectiveness was always superior to the others on all dimensions.

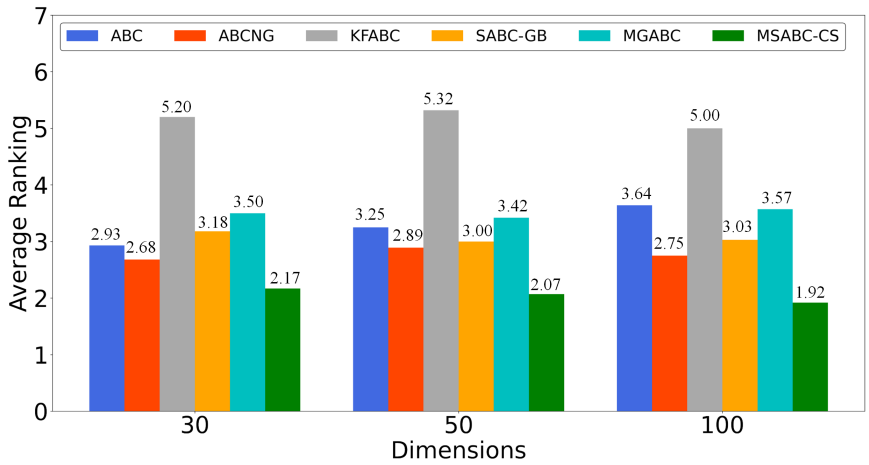


Figure 7. Illustration of Average Ranking.

Table 5. Average Ranking.

Function	D = 30	D = 50	D = 100
ABC	2.93	3.25	3.64
ABCNG	2.68	2.89	2.75
KFABC	5.20	5.32	5.00
SABC-GB	3.18	3.00	3.03
MGABC	3.50	3.42	3.57
SAABC-CS	**2.17**	**2.07**	**1.92**

We can further demonstrate the efficacy of our approach in finding the optimal for the complicated functions, based on the comparison findings mentioned above for the CEC2013 functions.

4.4. SAABC-CS for Practical Engineering Problems

In real-world engineering, parameter estimation of frequency-modulated (FM) sound waves [39] is frequently investigated. In this section, we performed tests to make comparisons between SAABC-CS and other ABC variants. We ran each algorithm 30 times independently and compared the results of the top outcomes of each algorithm.

4.4.1. Parameter Estimation for Frequency-Modulated (FM) Sound Waves

Frequency-modulated (FM) sound wave synthesis plays an important role in various modern music systems. To estimate the parameter of a FM synthesizer and to optimize the results, we solved a six-dimensional optimization problem, where we tried to optimize the vector $X = \{a_1, \omega_1, a_2, \omega_2, a_3, \omega_3\}$ given in Equation (19). The goal of this optimization problem was to generate a sound (19) similar to the target sound (20). This problem highly complex and multi-modal, having strong epistasis, with a minimum value $f(\vec{X_{sol}}) = 0$. The expressions for the estimated sound and the target sound waves are given as

$$y(t) = a_1 \cdot \sin\left(\omega_1 \cdot t \cdot \theta + a_2 \cdot \sin\left(\omega_2 \cdot t \cdot \theta + a_3 \cdot \sin\left(\omega_3 \cdot t \cdot \theta\right)\right)\right) \quad (19)$$

$$y_0(t) = 1.0 \cdot \sin\left(5.0 \cdot t \cdot \theta - 1.5 \cdot \sin\left(4.8 \cdot t \cdot \theta + 2.0 \cdot \sin\left(4.9 \cdot t \cdot \theta\right)\right)\right) \quad (20)$$

where $\theta = (2 \cdot \pi)/100$ and the parameters are defined in the range $[-6.4, 6.35]$. The fitness function is the summation of the square errors between the estimated wave (19) and the target wave (20), as follows:

$$minf(\vec{X}) = \sum_{t=0}^{100} (y(t) - y_0(t))^2 \tag{21}$$

4.4.2. Results of SAABC-CS Compared with Other Algorithms

Table 6 is a summary of the final results of our experiments, including the six parameters and the optimal cost. SAABC-CS obtained the minimum cost in solving the parameter estimation for frequency-modulated (FM) sound waves problem.

Table 6. SAABC-CS vs. other algorithms for parameter estimation of FM sound waves.

Algorithms	The Best Value of Six Parameters						Error Variance
	a_1	ω_1	a_2	ω_2	a_3	ω_3	
ABC	0.3559	0.0316	1.0678	0.1394	5.0670	4.9233	1.659715×10^1
ABCNG	0.5546	0.0627	1.5207	0.1437	4.3253	4.9361	1.460573×10^1
KFABC	0.1414	0.2657	0.1414	0.1414	0.1414	0.1414	2.901492×10^1
MGABC	0.6894	14.6861	0.7746	9.7434	1.0506	5.1230	1.226273×10^1
SABC-GB	0.7306	14.5462	0.6579	4.6106	5.7068	4.8736	1.177162×10^1
SAABC-CS	0.3370	4.7477	0.4820	0.2280	1.2430	3.2447	$\mathbf{1.024282 \times 10^1}$

5. Conclusions

In this paper, we proposed a new self-adaptive ABC algorithm with candidate strategies (SAABC-CS) to balance the exploration and exploitation of the evolution. Compared with the original ABC, SAABC-CS has three modifications, without adding any extra parameters: (1) five strategies are selected and assembled in a strategy pool; (2) a self-adaptive mechanism was designed to make the algorithm universal; (3) three neighbor mutations work together to enhance the scout phase. The aforementioned additions enhance ABC's overall performance by allowing it to tackle complicated issues with more features, while balancing its exploration and development capabilities.

Comprehensive experiments were performed on two groups of functions: 22 basic benchmark functions and CEC2013 test suites. The experiment results for the 22 basic test functions showed that SAABC-CS obtained a much better performance than an ABC with one strategy. Furthermore, the self-adaptive selection mechanism in SAABC-CS was well-turned to select an appropriate strategy for facing problems of a different nature. For the complex and difficult CEC2013 benchmark suite, SAABC-CS still achieved promising results and surpassed four state-of-the-art ABCs. With the increasing dimensions of CEC2013, the performance of SAABC-CS did not deteriorate. The method also produced positive results when it was used to tackle real-world engineering challenges, demonstrating that it can solve real-world optimization problems as well as test functions.

Although extensive experiments were conducted to demonstrate the performance of SAABC-CS, we hope to theoretically analyze the algorithm, inspired by the literature [40–42]. We also wish to extend the use of SAABC-CS to certain large and expensive problems in the future.

Author Contributions: Y.H.: Conceptualization, Methodology, Software, Validation, Formal analysis, Writing—Original Draft. Y.Y.: Conceptualization, Methodology, Writing—Original Draft, Supervision, Project administration. J.G.: Conceptualization, Methodology, Writing—Original Draft, Supervision, Project administration. Y.W.: Methodology, Supervision, Project administration. All authors have read and agreed to the published version of the manuscript.

Funding: This work was part funded by the National Natural Science Foundation of China (No. 61966019), and the Fundamental Research Funds for the Central Universities (No. CCNU20TS026).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data used to support the findings of this investigation are accessible upon request from the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Whitley, D. A genetic algorithm tutorial. *Stat. Comput.* **1994**, *4*, 65–85. [CrossRef]
2. Houck, C.R.; Joines, J.; Kay, M.G. A genetic algorithm for function optimization: A Matlab implementation. *Ncsu-Ie Tr* **1995**, *95*, 1–10.
3. Wang, H.; Zhou, X.; Sun, H.; Yu, X.; Zhao, J.; Zhang, H.; Cui, L. Firefly algorithm with adaptive control parameters. *Soft Comput.* **2017**, *21*, 5091–5102. [CrossRef]
4. Dorigo, M.; Birattari, M.; Stutzle, T. Ant colony optimization. *IEEE Comput. Intell. Mag.* **2006**, *1*, 28–39. [CrossRef]
5. Qin, A.K.; Huang, V.L.; Suganthan, P.N. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Trans. Evol. Comput.* **2008**, *13*, 398–417. [CrossRef]
6. Qin, A.K.; Suganthan, P.N. Self-adaptive differential evolution algorithm for numerical optimization. In Proceedings of the 2005 IEEE Congress on Evolutionary Computation, Scotland, UK, 2–5 September 2005; Volume 2, pp. 1785–1791.
7. Mallipeddi, R.; Suganthan, P.N.; Pan, Q.K.; Tasgetiren, M.F. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Appl. Soft Comput.* **2011**, *11*, 1679–1696. [CrossRef]
8. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the ICNN'95—International Conference on Neural Networks, Perth, Australia, 27 November–1 December 1995; Volume 4, pp. 1942–1948.
9. Shi, Y. Particle swarm optimization: Developments, applications and resources. In Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546), Seoul, Republic of Korea, 27–30 May 2001; Volume 1, pp. 81–86.
10. Karaboga, D.; Basturk, B. On the performance of artificial bee colony (ABC) algorithm. *Appl. Soft Comput.* **2008**, *8*, 687–697. [CrossRef]
11. Karaboga, D.; Akay, B. A comparative study of artificial bee colony algorithm. *Appl. Math. Comput.* **2009**, *214*, 108–132. [CrossRef]
12. Karaboga, D. *An Idea Based on Honey Bee Swarm for Numerical Optimization*; Technical Report, Technical Report-tr06; Engineering Faculty, Erciyes University: Kayseri, Turkey, 2005.
13. Bajer, D.; Zorić, B. An effective refined artificial bee colony algorithm for numerical optimisation. *Inf. Sci.* **2019**, *504*, 221–275. [CrossRef]
14. Kumar, D.; Mishra, K. Co-variance guided artificial bee colony. *Appl. Soft Comput.* **2018**, *70*, 86–107. [CrossRef]
15. Xue, Y.; Jiang, J.; Zhao, B.; Ma, T. A self-adaptive artificial bee colony algorithm based on global best for global optimization. *Soft Comput.* **2018**, *22*, 2935–2952. [CrossRef]
16. Cui, L.; Li, G.; Zhu, Z.; Lin, Q.; Wen, Z.; Lu, N.; Wong, K.C.; Chen, J. A novel artificial bee colony algorithm with an adaptive population size for numerical function optimization. *Inf. Sci.* **2017**, *414*, 53–67. [CrossRef]
17. Wang, H.; Wu, Z.; Rahnamayan, S.; Sun, H.; Liu, Y.; Pan, J.S. Multi-strategy ensemble artificial bee colony algorithm. *Inf. Sci.* **2014**, *279*, 587–603. [CrossRef]
18. Gao, W.; Liu, S.; Huang, L. A novel artificial bee colony algorithm based on modified search equation and orthogonal learning. *IEEE Trans. Cybern.* **2013**, *43*, 1011–1024. [PubMed]
19. Wang, H.; Wang, W.; Zhou, X.; Zhao, J.; Wang, Y.; Xiao, S.; Xu, M. Artificial bee colony algorithm based on knowledge fusion. *Complex Intell. Syst.* **2021**, *7*, 1139–1152. [CrossRef]
20. Lu, R.; Hu, H.; Xi, M.; Gao, H.; Pun, C.M. An improved artificial bee colony algorithm with fast strategy, and its application. *Comput. Electr. Eng.* **2019**, *78*, 79–88. [CrossRef]
21. Gao, W.; Liu, S. A modified artificial bee colony algorithm. *Comput. Oper. Res.* **2012**, *39*, 687–697. [CrossRef]
22. Guo, P.; Cheng, W.; Liang, J. Global artificial bee colony search algorithm for numerical function optimization. In Proceedings of the 2011 Seventh International Conference on Natural Computation, Shanghai, China, 26–28 July 2011; Volume 3, pp. 1280–1283.
23. Yu, W.J.; Zhan, Z.H.; Zhang, J. Artificial bee colony algorithm with an adaptive greedy position update strategy. *Soft Comput.* **2018**, *22*, 437–451. [CrossRef]
24. Jadon, S.S.; Tiwari, R.; Sharma, H.; Bansal, J.C. Hybrid artificial bee colony algorithm with differential evolution. *Appl. Soft Comput.* **2017**, *58*, 11–24. [CrossRef]
25. Alqattan, Z.N.; Abdullah, R. A hybrid artificial bee colony algorithm for numerical function optimization. *Int. J. Mod. Phys. C* **2015**, *26*, 1550109. [CrossRef]
26. Chen, S.M.; Sarosh, A.; Dong, Y.F. Simulated annealing based artificial bee colony algorithm for global numerical optimization. *Appl. Math. Comput.* **2012**, *219*, 3575–3589. [CrossRef]
27. Gao, W.f.; Huang, L.l.; Liu, S.y.; Chan, F.T.; Dai, C.; Shan, X. Artificial bee colony algorithm with multiple search strategies. *Appl. Math. Comput.* **2015**, *271*, 269–287. [CrossRef]

28. Song, X.; Zhao, M.; Xing, S. A multi-strategy fusion artificial bee colony algorithm with small population. *Expert Syst. Appl.* **2020**, *142*, 112921. [CrossRef]
29. Chen, X.; Tianfield, H.; Li, K. Self-adaptive differential artificial bee colony algorithm for global optimization problems. *Swarm Evol. Comput.* **2019**, *45*, 70–91. [CrossRef]
30. Zhou, X.; Lu, J.; Huang, J.; Zhong, M.; Wang, M. Enhancing artificial bee colony algorithm with multi-elite guidance. *Inf. Sci.* **2021**, *543*, 242–258. [CrossRef]
31. Zhu, G.; Kwong, S. Gbest-guided artificial bee colony algorithm for numerical function optimization. *Appl. Math. Comput.* **2010**, *217*, 3166–3173. [CrossRef]
32. Mühlenbein, H.; Schomisch, M.; Born, J. The parallel genetic algorithm as function optimizer. *Parallel Comput.* **1991**, *17*, 619–632. [CrossRef]
33. Xiao, S.; Wang, H.; Wang, W.; Huang, Z.; Zhou, X.; Xu, M. Artificial bee colony algorithm based on adaptive neighborhood search and Gaussian perturbation. *Appl. Soft Comput.* **2021**, *100*, 106955. [CrossRef]
34. Derrac, J.; García, S.; Molina, D.; Herrera, F. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm Evol. Comput.* **2011**, *1*, 3–18. [CrossRef]
35. Wang, B.C.; Li, H.X.; Li, J.P.; Wang, Y. Composite differential evolution for constrained evolutionary optimization. *IEEE Trans. Syst. Man, Cybern. Syst.* **2018**, *49*, 1482–1495. [CrossRef]
36. Wang, Y.; Li, J.P.; Xue, X.; Wang, B.C. Utilizing the correlation between constraints and objective function for constrained evolutionary optimization. *IEEE Trans. Evol. Comput.* **2019**, *24*, 29–43. [CrossRef]
37. Wang, Y.; Wang, B.C.; Li, H.X.; Yen, G.G. Incorporating objective function information into the feasibility rule for constrained evolutionary optimization. *IEEE Trans. Cybern.* **2015**, *46*, 2938–2952. [CrossRef]
38. Fan, Q.; Jin, Y.; Wang, W.; Yan, X. A performancE−driven multi-algorithm selection strategy for energy consumption optimization of sea-rail intermodal transportation. *Swarm Evol. Comput.* **2019**, *44*, 1–17. [CrossRef]
39. Das, S.; Suganthan, P.N. *Problem Definitions and Evaluation Criteria for CEC 2011 Competition on Testing Evolutionary Algorithms on Real World Optimization Problems*; Jadavpur University, Nanyang Technological University: Kolkata, India, 2010; pp. 341–359.
40. KhalafAnsar, H.M.; Keighobadi, J. Adaptive Inverse Deep Reinforcement Lyapunov learning control for a floating wind turbine. *Sci. Iran.* **2023**, *in press*. [CrossRef]
41. Keighobadi, J.; KhalafAnsar, H.M.; Naseradinmousavi, P. Adaptive neural dynamic surface control for uniform energy exploitation of floating wind turbine. *Appl. Energy* **2022**, *316*, 119132. [CrossRef]
42. Keighobadi, J.; Nourmohammadi, H.; Rafatania, S. Design and Implementation of GA Filter Algorithm for Baro-inertial Altitude Error Compensation. In Proceedings of the Conference: ICLTET-2018, Istanbul, Turkey, 21–23 March 2018; pp. 21–23.