Information Sciences 604 (2022) 142–169

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

An adaptive clonal selection algorithm with multiple differential evolution strategies

Yi Wang [a,*,1], Tao Li [b,1], Xiaojie Liu [b], Jian Yao [a]

[a] School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan 411201, China
[b] College of Cybersecurity, Sichuan University, Chengdu 610065, China

ARTICLE INFO

Article history:
Received 20 April 2021
Received in revised form 18 April 2022
Accepted 23 April 2022
Available online 4 May 2022

Keywords:
Clonal selection algorithm
Artificial immune system
Mutation operator
Differential evolution operator
Optimization

ABSTRACT

Clonal selection algorithms have provided significant insights into numerical optimization problems. However, most mutation operators in conventional clonal selection algorithms have semi-blindness and lack an effective guidance mechanism, which has thus become one of the important factors restricting the performance of algorithms. To address these problems, this study develops an improved clonal selection algorithm called an adaptive clonal selection algorithm with multiple differential evolution strategies (ADECSA) with three features: (1) an adaptive mutation strategy pool based on its historical records of success is introduced to guide the immune response process effectively; (2) an adaptive population resizing method is adopted to speed up convergence; and (3) a premature convergence detection method and a stagnation detection method are proposed to alleviate premature convergence and stagnation problems in the evolution by enhancing the diversity of the population. Experimental results on a wide variety of benchmark functions demonstrate that our proposed method achieves better performance than both state-of-the-art clonal selection algorithms and differential evolution algorithms. Especially in the comparisons with other clonal selection algorithms, our proposed method outperforms at least 23 out of 30 benchmark functions from the CEC2014 test suite.

1. Introduction

In recent years, optimization problems have attracted great attention from researchers, and many nature-inspired computation algorithms have been proposed, such as the genetic algorithm (GA) [1], artificial immune algorithm(AIA) [2], particle swarm optimization (PSO) [3], ant colony algorithm (ACA) [4] and differential evolution (DE) [5].

Clonal selection algorithms (CSAs), which are based on the clonal selection principle, are a special kind of AIA. The core components of CSAs are cloning, mutation and selection operators. Cloning is used to generate new individuals centered on higher affinity values. Mutation produces perturbations on clones and maintains the diversity of the population. Selection is responsible for removing individuals with lower affinity. Therefore, CSAs have excellent performance in solving complex optimization problems and have been widely applied in numerous fields [6–8].

It is well known that mutation operators play an important role in CSAs. Many experimental studies have found that most mutation operators in conventional CSAs have great randomness and lack an effective guidance mechanism; thus, the semi-blindness in mutation operators causes the antibody evolution to be a trial-and-error process [9,10]. A few efforts have been made to introduce learning mechanisms into mutation operators. For example, [11] adopted a mutation operator based

* Corresponding author.
E-mail address: wangyi@hnust.edu.cn (Y. Wang).
[1] Yi Wang and Tao Li contributed equally.

https://doi.org/10.1016/j.ins.2022.04.043
0020-0255/© 2022 Elsevier Inc. All rights reserved.

on the Baldwinian learning mechanism [12] to alleviate the semi-blind search caused by random perturbation. [13] used multiple DE-like mutation strategies to simulate the immune response.

Although the experimental results show that learning mechanisms can improve the search ability of CSAs, the selection of learning strategies and the settings of control parameters are still not addressed well in most CSAs. Different learning strategies and control parameters are suitable for different application scenarios. Inappropriate selection may cause premature convergence or stagnation. Consequently, we need to adopt an adaptive mechanism that adjusts learning strategies and parameters automatically to meet the searching requirement of antibodies in different optimization problems or different evolutionary stages in an optimization problem.

Motivated by the above discussions, we propose an efficient clonal selection algorithm called the Adaptive Clonal Selection Algorithm with Multiple Differential Evolution Strategies (ADECSA). It introduces differential evolution strategies and an adaptive parameter control mechanism to guide the evolution process in the immune system, which can effectively alleviate the limitation of searching ability in numerical optimization problems caused by the semi-blindness in mutation operators. Overall, the main contributions of the paper are summarized as follows.

1. An adaptive strategy pool with three differential evolution strategies is presented, from which different individuals automatically choose their appropriate strategies and set associated parameters based on their historical records of success.
2. A linear population size reduction method is used to accelerate convergence speed.
3. A premature convergence detection method and a stagnation detection method are proposed, which are based on population distribution. Once convergence or stagnation is identified in the evolution, the diversity of the population is enhanced to alleviate the above problems.
4. Our proposed method provides a new paradigm for CSAs to solve the semi-blindness in mutation operators. Moreover, the components in the framework, such as the mutation strategy pool with adaptive parameters, linear population size reduction, premature convergence detection and stagnation detection, can be easily replaced by other advanced methods to obtain better performance.

The rest of the paper is organized as follows: Section 2 reviews some related work, including the differential evolution algorithms and the clonal selection algorithms. In Section 3, the ADECSA is discussed in detail. Section 4 presents the comparison experiments and the analyzes results. Finally, some conclusions and suggestions for future work are drawn in Section 5.

2. Related work

In this section, we review related work in two areas: differential evolution algorithms and clonal selection algorithms.

2.1. Differential evolution algorithm

Differential evolution [14], proposed by Storn and Price, is a simple and effective evolutionary algorithm for numerous real-world problems. The classical DE mainly includes four steps: initialization, mutation, crossover and selection, which are described in Algorithm 1. It begins with the initial population randomly generated by a uniform distribution and ensures that the population covers the entire solution space as much as possible. Then, the population will go through a number of iterations of the evolution process. In each generation, a mutant vector with respect to each individual, the so-called target vector, is generated by a mutation operation. After mutation, each pair of the target vector and the corresponding mutant vector use a crossover operation to produce a trial vector. Finally, in the selection operation, if the fitness value of the new trial vector is better than that of its target vector, then the new trial vector will replace its corresponding target vector and enter into the next evolution process.

Algorithm 1: Differential evolution algorithm

```
Input: individual population size $N$
Output: the best individual in the population
1: Initialization: Randomly initialize a population $P$ with a fixed size $N$ ;
2: while termination criteria are not met do
3:    Mutation: Generate a mutant individual $v_i^g$ for each target individual $x_i^g$ by the mutation operation;
4:    Crossover: Generate a trial individual $u_i^g$ for each pair of the target individual $x_i^g$ and the corresponding mutant individual $v_i^g$;
5:    Selection: Select the better individual $x_i^{g+1}$ between a target individual $x_i^g$ and its corresponding trial individual $u_i^g$ into the next generation;
      $g = g + 1$;
6: end while
```

The performance of DE is heavily dependent on the selected mutation strategy and the associated parameters. Unsuitable selection may significantly degrade its performance. Therefore, we focus on mutation strategy selection and parameter control in DE and provide a brief review.

(1) *Mutation strategy selection*: The classical DE algorithms include a series of mutation strategies, such as "DE/rand/1", "DE/rand/2", "DE/best/1", "DE/best/2", "DE/current-rand/1" and "DE/current-to-pbest/1 with an archive". The details of the strategies are shown below.

DE/rand/1 :

$$v_i^g = x_{r1}^g + F \cdot \left(x_{r2}^g - x_{r3}^g\right) \tag{1}$$

DE/rand/2 :

$$v_i^g = x_{r1}^g + F \cdot \left(x_{r2}^g - x_{r3}^g\right) + F \cdot \left(x_{r4}^g - x_{r5}^g\right) \tag{2}$$

DE/best/1 :

$$v_i^g = x_{best}^g + F \cdot \left(x_{r1}^g - x_{r2}^g\right) \tag{3}$$

DE/best/2 :

$$v_i^g = x_{best,g} + F \cdot \left(x_{r1}^g - x_{r2}^g\right) + F \cdot \left(x_{r3}^g - x_{r4}^g\right) \tag{4}$$

DE/current-rand/1 :

$$v_i^g = x_i^g + K \cdot \left(x_{r1}^g - x_i^g\right) + F \cdot \left(x_{r2}^g - x_{r3}^g\right) \tag{5}$$

DE/current-to-pbest/1 with an archive :

$$v_i^g = x_i^g + F \cdot \left(x_{best,p}^g - x_i^g\right) + F \cdot \left(x_{r1}^g - \tilde{x}_{r2}^g\right) \tag{6}$$

where N is the population size, the indices $r1, r2, r3, r4$, and $r5$ are mutually exclusive and randomly selected from the population set $\{1, 2, \ldots, N\} \setminus \{i\}, g$ is the current generation number, $x_{best,p}^g$ is randomly selected from the top $p\%$ individuals in the current generation, $\boldsymbol{P}$ represents the current population, and $\boldsymbol{A}$ is an external archive that consists of inferior solutions. $\tilde{x}_{r2}^g$ is randomly selected from the union set $\boldsymbol{P} \cup \boldsymbol{A}$. The parameters K and F are scaling factors used to control the scaled magnitudes of the difference vectors, and v_i^g is a mutant vector generated from x_i^g in generation g.

Compared with "DE/rand/1" and "DE/best/1", "DE/rand/2" and "DE/best/2" use two difference vectors to enhance the diversity. "DE/current-rand/1" is used to solve rotated multimodal problems due to its rotation invariance. "DE/target-to-best/1" adopts the best individual information to speed up convergence. "DE/current-to-pbest/1 with an archive" utilizes the information of multiple superior individuals and archived inferior individuals to balance the exploration and exploitation ability. Li et al. [15] proposed two variants of "DE/rand/2" and "DE/best/2", which introduced an elite guidance mechanism to improve the convergence speed without losing randomness. Wu et al. [16] designed a neighborhood mutation strategy based on DE/current-to-best/1, which used local neighborhood mutation to replace large-scale global mutation. In [17], a ranking-based mutation operator is proposed, in which the ranking of each individual was decided by its fitness and diversity contribution.

(2) *Parameter control*: Extensive studies have been carried out to configure the control parameters of DE, such as the scaling factor F, crossover rate CR and population size N. The automatic parameter-setting methods can be divided into three types: deterministic parameter control, adaptive parameter control and self-adaptive parameter control. Deterministic parameter control [18] automatically sets parameters according to a deterministic rule without considering any feedback from the evolutionary process. Adaptive parameter control [19] adjusts parameters based on the current search state. Self-adaptive parameter control [20] encodes control parameters as part of individuals, which undergo mutation, crossover and selection as well as individuals.

Recently, research on adaptive parameter control techniques has drawn intensive attention. According to the number of mutation strategies, DE variations are usually categorized as DE with adaptive parameters and a single strategy and DE with adaptive parameters and multiple strategies[21]. SHADE [22] is one of the most classical DE algorithms in the former ones, which employed a parameter adaptation mechanism based on success-history memory of parameter settings to improve JADE. L-SHADE [23] is an improved version of SHADE, where a linear population size reduction was introduced to decrease the population size and accelerate convergence. Subsequently, LSHADE-EpSin[24] improved the scaling factor of L-SHADE by using an ensemble of two sinusoidal adjustments. iL-SHADE [25] extended L-SHADE with a new memory update mechanism. jSO [26] developed a weighted mutation strategy to improve the performance of iL-SHADE. Based on the above DE variants, Meng et al. [27] proposed a mutation strategy with a hierarchical archive, an adaptation parameter control with grouping strategy and parabolic population size reduction to improve the performance of DE.

In the latter, CoDE [28] used three suitable mutation strategies (DE/rand/1, DE/rand/2, DE/current-to-rand/1) with control parameter settings to generate trial vectors. Wu et al. [29] proposed a multi-population ensemble DE named MPEDE to develop an ensemble of three strategies, in which the population was divided into four subpopulations, and each subpopulation was assigned to suitable mutation strategies. Li et al. [30] proposed an improved MPEDE that adopted a strategy ranking based grouping method, a sharing mechanism, and a new mutation strategy. In [31], three elite-guided mutation schemes with different adaptive parameter control strategies were applied to superior, medium and inferior subpopulations. Recently, high-level ensembles of multiple DE variants have attracted great attention [32]. Wu et al. [33] proposed a multipopulation-based framework to derive an ensemble DE, which integrated JADE, CoDE and EPSDE [34]. Ref [35] incorporated LSHADE-EpSin and LSHADE-RSP [36] into a DE variant.

2.2. Clonal selection algorithm

Burnet et al. [37] proposed a famous clonal selection theory in 1959, which explains the basic features of an adaptive immune response to an antigenic stimulus. The theory holds that only cells recognizing a nonself antigen can be selected to proliferate. The clonal selection process mainly consists of affinity proportional reproduction and affinity maturation. When cells are stimulated by antigens, some cells with higher affinities are selected to proliferate and then cloned in proportion to their affinities. To enhance the diversity of antibodies, these clones are subjected to an affinity maturation process in inverse proportion to the antigenic affinity. Inspired by clonal selection theory, a well-known clonal selection algorithm named CLONALG (CLONal selection ALGorithm)[38] was proposed by De Castro et al., which has been widely used in pattern recognition and numerical optimization. CLONALG simulates the process of antibody affinity maturation by selection, clonal proliferation and mutation. The CLONALG algorithm is described in Algorithm 2.

Algorithm 2: CLONALG algorithm

```
Input: antibody population size $N$,
    selection antibody size $n$,
    clonal scale $N_c$,
    replacement proportion in the antibody pool $N_r$
Output: the best antibody(antibodies) in the antibody pool $M$
1: Initialization: Randomly initialize an antibody pool $M$ with a fixed size $N$, which includes a memory antibody pool $m$
   and a remaining antibody pool $r$;
2: while termination criteria are not met do
3:     Select an antigen: Randomly present an antigen to the antibody pool $M$ from the antigen pool;
4:     Evaluation: Determine the affinities of all antibodies in the antibody pool $M$ against the antigen;
5:     Selection: Select $n$ antibodies with the best affinities from the antibody pool $M$;
6:     Cloning: Clone these $n$ selected antibodies proportionally to their affinities;
7:     Mutation: Mutate all the clones inversely proportional to their affinities;
8:     Updating: Calculate the affinities of the mutated antibodies, and then select the antibody with highest affinity from
   the mutated antibodies as a candidate memory antibody. If the affinity of the candidate is higher than that of the best
   antibody in the antibody pool $M$, then it will replace the best antibody. In addition, it can also update the antibody
   pool $M$ in a batched manner;
9:     Replacement: $N_r$ antibodies with the lowest affinities in the remaining antibody pool $r$ will be replaced with new
   random antibodies;
10: end while
```

In recent decades, CSAs have been widely studied, and a large number of their variations have been proposed. Many algorithms focus on designing new operators, modifying the operators and combining them with other search techniques. For example, Yan et al. [39] adopted an arithmetic crossover operator and an adaptive nonuniform mutation operator related to the number of evolutions to improve the search ability. Zhou et al. [40] adopted an affinity-based mutation rate, which gradually decreases with evolutionary algebra. Qiao et al.[41] used a mapped hyperplane correlated with the objective space to obtain uniformly distributed individuals and aimed to improve the population diversity. Li et al. [42] designed four heuristic initialization operators and new mutation operators for a specific flexible job shop scheduling problem to enhance the diversity and the exploitation ability and combined them with a simulated annealing method to improve the exploration ability. Liu et al. [43] integrated the CSA with a variable neighborhood search to improve the exploitation ability and reduce the probability of falling into a local optimum. Fefelova et al. [44] combined a CSA with a differential evolution algorithm to improve the prediction accuracy of the tertiary protein structure.

The mutation operator is an important factor that affects the performance of conventional CSAs. Numerous studies found that the hypermutation operator has great randomness and causes the algorithm to become a parallel hill climbing algorithm. Consequently, learning mechanisms in the hypermutation operator for promoting algorithm performance have attracted tremendous interest from researchers. For example, Gong et al.[11] adopted a Baldwinian learning mechanism to alleviate semi-blindness in the hypermutation operator caused by randomness and improve the evolution process. [13] incorporated Baldwinian learning with orthogonal learning into the clonal selection algorithm to guide the antibody evolution process. Zhang et al. [10] employed combinatorial recombination to improve the diversity of the evolution population and introduced an adaptive mutation strategy based on success history to enhance its search ability. Luo et al. [45] used Gaussian and DE-like mutation operators to speed up the convergence for dynamic multimodal optimization problems.

Compared with previous studies, the main differences are evident in four aspects. First, the methods of strategy selection are different. Previous studies have assigned a corresponding strategy to each clone in a fixed way, while our study assigns a strategy to each clone based on the success probability in a certain number of previous generations. Second, our study adopts a more advanced parameter control mechanism. Third, our study incorporates a linear population size reduction method to accelerate convergence speed and reduce function evaluation times. Finally, our study proposes a premature convergence detection method and a stagnation detection method to identify premature convergence and stagnation in the population, which have not been used in previous studies.

3. The proposed ADECSA method

In this study, we propose an improved clonal selection algorithm named ADECSA for numerical optimization problems. To discuss the algorithm more clearly, we describe some definitions, including a mutation strategy pool, an adaptive parameter control mechanism, linear population size reduction, a premature convergence detection method, a stagnation detection method, and a system framework for ADECSA.

3.1. Definition

To better understand ADECSA in the rest of the paper, some immunological terms are defined as follows.

(1) Antigens

An antigen is a substance that induces immune responses in the immune system. Here, the antigen is the problem to be solved. Take the following optimization problem as an example.

$$\underset{x\in\Omega}{Minimize}\ f(\boldsymbol{x}),\quad \Omega\in R^D \tag{7}$$

where $f(x)$ is an objective function, x is a variable vector in the feasible solution space Ω, and D is the dimension of the objective function.

(2) Antibodies

Antibodies are immunoglobulins that can bind to specific antigens. In this paper, antibodies represent candidate solutions of an antigen, denoted by

$$\boldsymbol{AB}(g)=\left\{ab_i^g|ab_i^g=\left(x_{i,1}^g,x_{i,2}^g,\ldots,x_{i,D}^g\right),\ i\leqslant N\right\} \tag{8}$$

where $\boldsymbol{AB}(g)$ is an antibody population in the gth generation, ab_i^g is the ith antibody in the gth generation, g is the evolutionary generation, and N is the antibody population size.

(3) Affinity

Affinity is used to measure the binding strength between antigens and antibodies. Here, the affinity of an antibody corresponds to the value of the objective function for a given antibody.

In addition, all the symbols involved in this section are listed in Table 1. It is important to note that the boldface expressions indicate matrices, and the expressions with subscripts represent vectors.

3.2. Mutation strategy pool

To effectively simulate the immune response in different optimization problems or different evolutionary stages in an optimization problem, our algorithm adopts an adaptive strategy pool with three differential evolution strategies based on their own distinct advantages. The involved strategies are described as follows, and more details can be found in Section 2.1.

(1) DE/rand/1

$$v_i^g=x_{r1}^g+F\cdot\left(x_{r2}^g-x_{r3}^g\right) \tag{9}$$

This strategy has been widely used in differential evolution algorithms due to its stronger exploration ability. All the antibodies are randomly selected from the population in the strategy; thus, this strategy has a strong exploration ability but a slow convergence rate. It is always adopted to solve multimodal problems.

(2) DE/current-to-pbest/1 with an archive

$$v_i^g=x_i^g+F\cdot\left(x_{best,p}^g-x_i^g\right)+F\cdot\left(x_{r1}^g-\tilde{x}_{r2}^g\right) \tag{10}$$

This strategy is a variant of the classic “DE/current-to-best” strategy. It uses archived inferior solutions and multiple best solutions to provide the information of the evolutionary direction; thus, it can effectively improve the performance of ”DE/current-to-best” by increasing the population diversity in multimodal problems.

Table 1
The description of symbols.

Symbols	Description
$\boldsymbol{AB}(g)$	A set of antibodies in the *gth* generation
g	The evolutionary generations
$f(\cdot)$	The affinity function
N_{init}	The initial antibody population size
N_{min}	The minimum population size
N_c	The clonal scale
r	The replacement proportion in the antibody pool
K	The strategy pool size
H	The historical memory pool size
FES	The number of function evaluations
R_P^C	The clonal proliferation operator
R_E^D	The differential mutation operator
R_S^C	The clonal selection operator

(3) DE/current-rand/1

$$v_i^g = x_i^g + F \cdot \left(x_{r1}^g - x_i^g\right) + F \cdot \left(x_{r2}^g - x_{r3}^g\right) \tag{11}$$

This strategy is suitable for solving rotated multimodal problems due to its rotation invariance.

Here, each clonal antibody adopts a separate mutation strategy in the strategy pool to create a mutant antibody. It is worth noting that only the "DE/rand/1" and "DE/current-to-pbest/1 with an archive" strategies are used with a binomial crossover operation, while the DE/current-rand/1 strategy is not needed.

3.3. Adaptive parameter control mechanism

It is well-known that the scaling factor F in the mutation strategies and the crossover probability CR in the binomial crossover operation are two important factors affecting the algorithm convergence and the population diversity. The scaling factor F is a control parameter that is used to scale the difference vectors. A larger value of F can increase the disturbance of the current solution by the difference vectors, and improve the population diversity and the global search ability of the algorithm. Nevertheless, since the mutant vectors are widely distributed in the search space, the population has low convergence speed. In contrast, a lower value of F can reduce the influences from the difference vector and make the exploration near the current solution, thereby enhanceing the local search ability of the algorithm and accelerating its convergence. The crossover probability CR is used to determine how many components of the trial vectors are inherited from the mutant vectors. With the value of CR increasing, more components are inherited, which is beneficial to speed up convergence.

Therefore, our proposed algorithm adopts an adaptive parameter control mechanism inspired by LSHADE-EpSin [24] where each individual has its own scaling factor F and crossover probability CR. The main difference between our algorithm and LSHADE-EpSin is that our algorithm adopts a strategy pool with multiple strategies, while LSHADE-EpSin uses a single DE strategy. In our algorithm, each mutation strategy is assigned its independent historical memory pool. Specifically, each mutation strategy k maintains a historical memory $\boldsymbol{M}_k^F$ with H entries for storing historical mean values of F, and a historical memory $\boldsymbol{M}_k^{CR}$ with H entries for storing historical mean values of CR. The last elements are set to fixed values equal to 0.9 during the evolution process. In the initialization phase, other elements in $\boldsymbol{M}_k^F$ and $\boldsymbol{M}_k^{CR}$ are set to 0.5, and the structures of the historical memories are shown in Fig. 1.

To adjust the scaling factor $F_{i,k}$ and the crossover probability $CR_{i,k}$ of individual x_i^g in the kth mutation strategy, the antibody evolution process is divided into two stages: the first half of generations $g_1 \in \left[1, \frac{G_{max}}{2}\right]$ and the second half of generations $g_2 \in \left(\frac{G_{max}}{2}, G_{max}\right]$, where G_{max} is the maximum number of allowed generations. The method is described in Algorithm 3, and some important details are explained as follows.

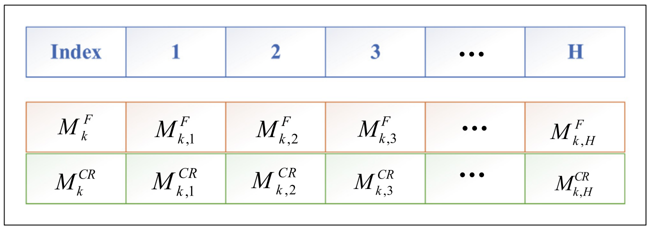


Fig. 1. The structures of historical memory M_k^F and M_k^{CR}.

Algorithm 3: Adaptive parameter control

```
Input: historical memory pool size $H$,
clonal scale $N_c$
Output: control parameters $F_{i,k}$ and $CR_{i,k}$ of individual $x_i^g$ in the kth mutation strategy
1: if $g \in g_1$
2:   for $j = 1$ to $N_c$ do
3:     Set $r_i = random(1, H)$;
4:     if $p_m < 0.5$ then
5:       Set $F_{i,k} = DeSinusoidal()$;/* non-adaptive sinusoidal decreasing adjustment, Eq. 20 */
6:     else
7:       Set $F_{i,k} = InSinusoidal()$;/* adaptive sinusoidal increasing adjustment Eq. 21 */
8:     end if
9:     Set $CR_{i,k} = randn\left(M_{k,r_i}^{CR}, 0.1\right)$;
10:  end for
11: else
12:  for $j = 1$ to $N_c$ do
13:    Set $r_i = random(1, H)$;
14:    Set $F_{i,k} = randc\left(M_{k,r_i}^{F}, 0.1\right)$;
15:    Set $CR_{i,k} = randn\left(M_{k,r_i}^{CR}, 0.1\right)$;
16:  end for
17: end if
```

In the first half of generations g_1, the scaling factor $F_{i,k}$ is adjusted by two sinusoidal methods with equal probability. The first method is nonadaptive sinusoidal decreasing adjustment, expressed as follows.

$$F_{i,k} = 0.5 \cdot \left(sin(2\pi \cdot freq \cdot g_1 + \pi) \cdot \frac{G_{max} - g_1}{G_{max}} + 1 \right) \tag{12}$$

where g_1 is the number of current generations, and $freq$ is a fixed value that is used to control the frequency of the sinusoidal function. Here, $freq$ is set to 0.5.

Another sinusoidal method is an adaptive sinusoidal increasing adjustment method. Unlike the first method, it adopts an adaptive mechanism based on historical records of success to adjust the frequency control parameter of the sinusoidal function, which is shown below.

$$F_{i,k} = 0.5 \cdot \left(sin(2\pi \cdot freq_{i,k} \cdot g_1) \cdot \frac{g_1}{G_{max}} + 1 \right) \tag{13}$$

where $freq_{i,k}$ is the frequency control parameter of the ith antibody in mutation strategy k, which is computed as

$$freq_{i,k} = randc\left(M_{k,r_i}^{freq}, 0.1\right) \tag{14}$$

$$r_i = rand(1, H); \tag{15}$$

In this equation, H is the historical memory size, r_i is an index variable that is randomly selected from [1, H], and $randc(\mu, \sigma)$ is a Cauchy distribution function with location parameter μ and scale location σ. In addition, the crossover probability $CR_{i,k}$ is adapted in the same way as the second half of generations g_2, which will be explained next.

In the second half of generations g_2, the scaling factor $F_{i,k}$ and the crossover probability $CR_{i,k}$ of individual x_i^g using the kth mutation strategy are computed according to the following equations.

$$F_{i,k} = randc\left(M_{k,r_i}^{F}, 0.1\right) \tag{16}$$

$$CR_{i,k} = randn\left(M_{k,r_i}^{CR}, 0.1\right) \tag{17}$$

where $randc(\mu, \sigma)$ is a Cauchy distribution function with location parameter μ and scale location σ, $randn(\mu, \sigma)$ is a normal distribution function with mean μ and variance σ, and $\boldsymbol{M}_{k,r_i}^{F}$ and $\boldsymbol{M}_{k,r_i}^{CR}$ are the r_ith elements in the historical memories $\boldsymbol{M}_k^F$ and $\boldsymbol{M}_k^{CR}$, respectively. Here, it is worth noting that if $F_{i,k} > 1$, then it will be truncated to 1, and if $F_{i,k} \leqslant 0$, then it will be regenerated according to Eq. (16). Similarly, if $CR_{i,k} > 1$, then it will be set to 1, and if $CR_{i,k} \leqslant 0$, then it will be set to 0.

When a new antibody is better than its parent antibody in each generation, then its $F_{i,k}, CR_{i,k}$ and $freq_{i,k}$ values will be recorded in S_k^F, S_k^{CR} and S_k^{freq}. At the end of the generation, the z_kth elements of the historical memories $\boldsymbol{M}_k^F, \boldsymbol{M}_k^{CR}$ and $\boldsymbol{M}_k^{freq}$, termed $\boldsymbol{M}_{k,z_k}^F, \boldsymbol{M}_{k,z_k}^{CR}$ and $\boldsymbol{M}_{k,z_k}^{freq}$, will be updated as follows. Note that if none of the new antibodies is better than its parent antibody, the historical memories $\boldsymbol{M}_k^F, \boldsymbol{M}_k^{CR}$ and $\boldsymbol{M}_k^{freq}$ will not be updated.

$$M_{k,z_k}^F = \begin{cases} mean_L\left(S_k^F\right) & S_k^F \neq \varnothing \\ M_{k,z_k}^F & otherwise \end{cases} \tag{18}$$

$$M_{k,z_k}^{CR} = \begin{cases} mean_L\left(S_k^{CR}\right) & S_k^{CR} \neq \varnothing \\ M_{k,z_k}^{CR} & otherwise \end{cases} \tag{19}$$

$$M_{k,z_k}^{freq} = \begin{cases} mean_L\left(S_k^{freq}\right) & S_k^{freq} \neq \varnothing \\ M_{k,z_k}^{freq} & otherwise \end{cases} \tag{20}$$

where the index z_k is initialized to 1 and then recursively increases. When $z_k > H$, then it is set to 1. $meanL(\cdot)$ is a weighted Lehmer mean function and is computed as follows.

$$mean_L(S_k) = \frac{\sum_{i=1}^{|S_k|} w_{i,k} \cdot (S_{k,i})^2}{\sum_{i=1}^{|S_k|} w_{i,k} \cdot S_{k,i}} \tag{21}$$

$$w_{i,k} = \frac{\triangle f_{i,k}}{um_{i=1}^{|S_k|} \triangle f_{i,k}} \tag{22}$$

$$\triangle f_{i,k} = \mid (f(u_{ik}) - f(x_{ik})) \mid \tag{23}$$

where $\triangle f_{i,k}$ is an absolute value of the difference between the affinity of the new antibody $f(u_{ik})$ and the affinity of its parent antibody $f(x_{ik})$.

3.4. Linear population size reduction

Antibody population size plays an important role in affecting performance. The smaller population size implies a faster convergence speed but simultaneously increases the chances of premature convergence in the antibody population. On the other hand, the larger population size can fully explore the search space but slows down the convergence speed. Thus, we adopted a linear population size reduction designed in L-SHADE, which linearly reduces the antibody population according to a function of the number of affinity function evaluations. The antibody population size in the next generation $g + 1$ is computed as follows.

$$N_{g+1} = round\left(\left(\frac{N_{min} - N_{init}}{FES_{max}}\right) \cdot FES + N_{init}\right) \tag{24}$$

where N_{min} is the minimum population size, N_{init} is the initial antibody population size, FES is the current number of function evaluations, and FES_{max} is the maximum number of function evaluations. It can be seen that the antibody population linearly decreases as the number of function evaluations increases. When $N_{g+1} < N_g$, we remove the $(N_g - N_{g+1})$ worst antibodies and adjust the external archive size.

In addition, the best n_{gw} antibodies in the antibody pool adopt Gaussian walks to enhance the exploitation ability of ADECSA when the antibody population size reduces to the predefined threshold N_g. These antibodies are performed for T_g generations according to the following equations.

$$\hat{ab}_i^g = Gaussian(ab_{best}^w, \sigma) + (r_1 \cdot ab_{best}^w - r_2 \cdot ab_i^g) \tag{25}$$

where Gaussian($\cdot$) is a normal distribution function with mean ab_{best}^w and variance σ, ab_{best}^w is the best antibody in the Gaussian walks, σ is computed according to Eq. (26), r_1 and r_2 are uniform distribution random numbers between 0 and 1, and ab_i^g is the ith antibody in the Gaussian walks.

$$\sigma = \left|\frac{log(FES)}{FES} \cdot (ab_i^g - ab_{best}^w)\right| \tag{26}$$

Obviously, as the current number of function evaluations increases, the variance σ decreases. Thus, Gaussian walks have a stronger exploration ability at the initial stage of evolution and a stronger exploitation ability at the later stage of evolution. Once the Gaussian walks are finished, the new antibodies are selected as candidates to replace the best antibodies in the antibody pool. If the affinities of the candidates are lower than those of the best antibodies, then the best antibodies will be replaced.

3.5. Premature convergence and stagnation detection

Premature convergence and stagnation are important factors that significantly affect the overall performance of CSAs. Premature convergence is caused by the loss of population diversity. In the early evolution stage, antibodies are generated whose affinities are much higher than the average affinity of the current population. After “survival of the fittest” selection, these antibodies occupy most of the current population, resulting in reduced the population diversity and evolutionary ability; thus, the population easily falls into a local optimum. Stagnation is another situation in which the antibody population remains diverse and is unconverged, but the CSA is still unable to find any better solutions. Therefore, we propose a premature convergence detection method and a stagnation detection method based on population distribution, and then give the corresponding strategy to deal with these situations.

(1) Premature convergence detection

Our algorithm adopts the Corriveau et al. formula [46] to measure population diversity (27).

$$D_{PW}^{N} = \frac{\frac{2}{N(N-1)}\sum_{i=2}^{N}\sum_{j=1}^{i-1}\sqrt{\sum_{k=1}^{D}\left(x_{i,k}-x_{j,k}\right)^{2}}}{NMDF} \tag{27}$$

where D is the dimension of the antibodies, N is the antibody population size, and $NMDF$ is a normalization factor, which is the maximum distance between two antibodies in the population thus far during the evolution process.

D_{PW}^{N} is a mean of the pairwise distance among individuals in the current population. The smaller D_{PW}^{N} is, the less population diversity there is. In theory, when $D_{PW}^{N}=0$, then the population has completely converged. Here, we adopt a small value δ_c to judge whether the population has converged. If D_{PW}^{N} is not greater than threshold δ_c, then the population has converged, and the variable c_g is set to 1. Otherwise, the population has not converged, and the variable c_g is set to 0.

$$c_g = \begin{cases} 1 & \text{if } D_{PW}^{N} \leqslant \delta_c, \\ 0 & \text{otherwise} \end{cases} \tag{28}$$

(2) Stagnation detection

When stagnation occurs in the population, the population diversity remains little changed. Our algorithm uses the value of D_{PW}^{N} to measure population diversity. If D_{PW}^{N} has not changed during several successive generations, we consider that the population has stagnated. Specifically, suppose that the variable s is the number of successive generations where the value of D_{PW}^{N} is unchanged, and it is calculated as follows.

$$s = \begin{cases} s+1 & \text{if } D_{PW,g}^{N} = D_{PW,g-1}^{N}, \\ 0 & \text{otherwise} \end{cases} \tag{29}$$

When s is not less than the given threshold δ_s, the population has stagnated and the variable s_g is set to 1. Otherwise, s_g is set to 0.

$$s_g = \begin{cases} 1 & \text{if } s \geqslant \delta_s, \\ 0 & \text{otherwise} \end{cases} \tag{30}$$

(3) Enhancement of population diversity

To alleviate the above phenomena, ADECSA increases the population diversity by reinitializing some antibodies with the worst affinities. Specifically, the antibodies in the antibody pool are sorted ascending by affinity, and the last d antibodies are then replaced as follows.

$$ab_{i,j}^{g} = ab_{min,j} + rand(0,1)\cdot\left(ab_{max,j}-ab_{min,j}\right), i=1,2,\ldots,N_P, j=1,2,\ldots,D \tag{31}$$

where $ab_{i,j}^{g}$ represents the jth attribute of the antibody ab_i^g, $ab_{min,j}$ and $ab_{max,j}$ denote the minimum and maximum values of the jth attribute of the antibodies, respectively, and $rand(0,1)$ generates a uniform distribution random number in the range [0,1].

In addition, during the evolution process, the number of replaced antibodies d is computed using Eq. (32), which replaces relatively more antibodies in the early stage and fewer antibodies in the late stage.

$$d = 10^{-\frac{FES}{FES_{max}}}\cdot r\cdot N \tag{32}$$

where *FES* is the current number of function evaluations, FES_{max} is the maximum number of function evaluations, *r* is the replacement proportion in the antibody pool, and *N* is the population size.

3.6. System framework of the ADECSA

The ADECSA algorithm employs three components, including the mutation strategy pool with adaptive parameters, linear population size reduction, premature convergence detection and stagnation detection. The general framework of ADECSA is shown in Fig. 2, where the dotted rectangles represent our improved steps. The pseudocode is described in Algorithm4, and some critical steps are explained as follows.

Algorithm4: ADE-CSA

```
Input: initial population size $N_{init}$, minimum population size $N_{min}$, clonal scale $N_c$, historical memory pool size $H$,
  replacement proportion in the antibody pool $r$, diversity threshold $\delta_c$, stagnation threshold $\delta_s$, population size for
  Gaussian walks $N_g$, iteration number of Gaussian walks $T_g$, maximum number of function evaluations $FES_{max}$
Output: the best antibody in the antibody population
1: Set $g = 1$;
2: Set $\boldsymbol{M}_k^F = \left\{M_{k,1}^F, M_{k,2}^F, \ldots, M_{k,H}^F\right\}$, and $M_{k,i}^F = \{0.5, 0.5, \ldots, 0.5, 0.9\}$;
3: Set $\boldsymbol{M}_k^{CR} = \left\{M_{k,1}^{CR}, M_{k,2}^{CR}, \ldots, M_{k,H}^{CR}\right\}$, and $M_{k,i}^{CR} = \{0.5, 0.5, \ldots, 0.5, 0.9\}$;
4: Set $\boldsymbol{M}_k^{freq} = \left\{M_{k,1}^{freq}, M_{k,2}^{freq}, \ldots, M_{k,H}^{freq}\right\}$, and $M_{k,i}^{freq} = \{0.5, 0.5, \ldots, 0.5\}$;
5: Set $z^F = \{z_1^F, z_2^F, \ldots, z_K^F\}$, and $z_k^F = 1$;          /* $z_k^F$ is the index for updating $M_{k,i}^F$*/
6: Set $z^{CR} = \{z_1^{CR}, z_2^{CR}, \ldots, z_K^{CR}\}$, and $z_k^{CR} = 1$;      /* $z_k^{CR}$ is the index for updating $M_{k,i}^{CR}$ */
7: Set $z^{freq} = \left\{z_1^{freq}, z_2^{freq}, \ldots, z_K^{freq}\right\}$, and $z_k^{freq} = 1$; * $z_k^{freq}$ is the index for updating $M_{k,i}^{freq}$ */
8: Set $FES = N_{init}$;
9: Randomly initialize an antibody population $\boldsymbol{AB}(g) = \left\{ab_1^g, ab_2^g, \ldots, ab_{N_{init}}^g\right\}$;
10: while $FES <= FES_{max}$ do
11:   Set $S^F = \left(S_1^F, S_2^F, \ldots, S_k^F\right)$, and $S_k^F = \varnothing$;
12:   Set $S^{CR} = \left(S_1^{CR}, S_2^{CR}, \ldots, S_k^{CR}\right)$, and $S_k^{CR} = \varnothing$;
13:   Set $S^{freq} = \left(S_1^{freq}, S_2^{freq}, \ldots, S_k^{freq}\right)$, and $S_k^{freq} = \varnothing$;
14:   for $i = 1$ to $N$ do
15:     Generate a clonal set $\boldsymbol{X_i}(g)$ by clonal proliferation operator $R_P^C$;
16:     if $g \in g_1$ then
17:       for $j = 1$ to $N_c$ do
18:         Set $r_i = random(1, H)$;
19:         Set $F_{i,k} = Sinusoidal()$;      /*sinusoidal decreasing or increasing adjustment*/
20:         Set $CR_{i,k} = randn\left(M_{k,r_i}^{CR}, 0.1\right)$;
21:       end for
22:     else
23:       for $j = 1$ to $N_c$ do
24:         Set $r_i = random(1, H)$;
25:         Set $F_{i,k} = randc\left(M_{k,r_i}^F, 0.1\right)$;
26:         Set $CR_{i,k} = randn\left(M_{k,r_i}^{CR}, 0.1\right)$;
27:       end for
28:     end if
29:     if mod(g, ng) == 0 then
30:       update $p_{i,k}^g$ by Eq. (36);
31:       $ns_{i,k}$ = 0;
32:       $nf_{i,k}$ = 0;
33:     end if
34:         Generate a mutated set $\boldsymbol{Y_i}(g)$ by mutation operator $R_E^D$;
35:     Generate the next population $\boldsymbol{AB}(g+1)$ by clone selection operator $R_S^C$;
```

(continued on next page)

a (*continued*)

Algorithm 4: ADE-CSA

```
36:     if $f(y_{ib}^g) < f(ab_i^g)$ then
37:       $S_k^{CR} = S_k^{CR} \cup CR_{b,k}$;
38:       if $g \in g_1$ then
39:         $S_k^{freq} = S_k^{freq} \cup freq_{b,k}$;
40:       else
41:         $S_k^F = S_k^F \cup F_{b,k}$;
42:       end if
43:       $ns_{i,k} = ns_{i,k} + 1$;
44:     else
45:       $nf_{i,k} = nf_{i,k} + 1$;
46:     end if
47:   end for
48:   for $k = 1$ to $K$ do
49:     if $S_k^F \neq \emptyset$ and $S_k^{CK} \neq \emptyset$ then
50:       if $g \in g_1$ then
51:         $M_{k,z_k^{freq}}^{freq} = mean_L\left(S_k^{freq}\right)$;
52:         $z_k^{freq} = z_k^{freq} \bmod H + 1$;
53:       else
54:         $M_{k,z_k^F}^F = mean_L\left(S_k^F\right)$;
55:         $z_k^F = z_k^F \bmod (H-1) + 1$;
56:       end if
57:       $M_{k,z_k^{CR}}^{CR} = mean_A\left(S_k^{CR}\right)$;
58:       $z_k^{CR} = z_k^{CR} \bmod (H-1) + 1$;
59:     end if
60:   end for
61:   $FES = FES + N_c * N$;
62:   Apply linear population size reduction;
63:   Apply premature convergence and stagnation detection;
64:   if $N <= N_g$ for the first time then
65:     Apply Gaussian walks;
66:     $FES = FES + N_{gw} * T_g$;
67:   end if
68:   $g = g + 1$;
69:   $FES = FES + N * r$;
70: end while
```

(1) Initialization (Line 9)

Randomly initialize an antibody population $\boldsymbol{AB}$ with a fixed size N.

$$\boldsymbol{AB}(g) = \left\{ab_i^g | ab_i^g = \left(x_{i,1}^g, x_{i,2}^g, \ldots, x_{i,D}^g\right),\ i \leqslant N\right\} \tag{33}$$

(2) Clonal proliferation (Line 15)

Each antibody in $\boldsymbol{AB}$ proliferates to produce N_c antibodies by the clonal proliferation operator R_P^C, and the clonal set of the antibody ab_i^g is denoted as below.

$$\boldsymbol{X}_i(g) = R_P^C(ab_i^g) = \underbrace{\left\{x_{i1}^g, x_{i2}^g, \ldots, x_{iN_c}^g\right\}}_{N_c} = \underbrace{\left\{ab_i^g, ab_i^g, \ldots, ab_i^g\right\}}_{N_c} \tag{34}$$

where N_c is the clonal scale and equals the strategy pool size K. Here, there are three differential evolution strategies; thus, $N_c = K = 3$.

After clonal proliferation, the clonal set of the antibody population $\boldsymbol{AB}$ can be obtained.

$$\boldsymbol{X}(g) = \{\boldsymbol{X}_1(g), \boldsymbol{X}_2(g), \ldots, \boldsymbol{X}_N(g)\} = \left\{R_P^C(ab_1^g), R_P^C(ab_2^g), \ldots, R_P^C(ab_N^g)\right\} \tag{35}$$

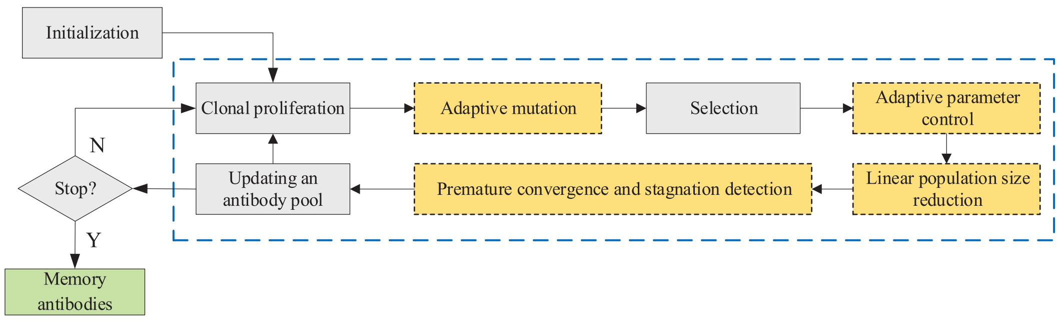

Initialization
Clonal proliferation
Adaptive mutation
Selection
Adaptive parameter control
Linear population size reduction
Premature convergence and stagnation detection
Updating an antibody pool
Stop?
N
Y
Memory antibodies

Fig. 2. The general framework of ADECSA.

(3) Adaptive mutation (Lines 16–34)

In ADECSA, the generation strategy of each mutated antibody is selected from the mutation strategy pool based on its success probability of improved antibodies in the previous ng generations. Specifically, each antibody ab_i^g has two variables: $ns_{i,k}$ and $nf_{i,k}$.The variable $ns_{i,k}$ is used to record the number of mutated antibodies generated by the kth strategy that are better than its parent, and the variable $nf_{i,k}$ is used to record the number of inferior mutated antibodies generated by the kth strategy during the ng generations. The probability of choosing the kth strategy for ab_i^g is computed by

$$p_{i,k}^g = \frac{ps_{i,k}^g}{\sum_{j=1}^{K} ps_{i,j}^g} \tag{36}$$

$$ps_{i,k}^g = \frac{ns_{i,k}}{ns_{i,k} + nf_{i,k}} \tag{37}$$

where K is the strategy pool size. Initially, the probability of the "DE/current-to-pbest/1 with an archive" scheme is set to 0.5, and the probabilities of the other schemes are set to 0.25.

The mutated set of the antibody ab_i^g is denoted $\boldsymbol{Y}_i(g)$, and the mutated set of the antibody population $\boldsymbol{AB}$ is $\boldsymbol{Y}(g)$.

$$\boldsymbol{Y}_i(g) = \underbrace{\left\{R_E^D\left(x_{i1}^g\right), R_E^D\left(x_{i2}^g\right), \ldots, R_E^D\left(x_{iN_c}^g\right)\right\}}_{N_c} = \underbrace{\left\{y_{i1}^g, y_{i2}^g, \ldots, y_{iN_c}^g\right\}}_{N_c} \tag{38}$$

$$\boldsymbol{Y}(g) = \{\boldsymbol{Y}_1(g), \boldsymbol{Y}_2(g), \ldots, \boldsymbol{Y}_N(g)\} = \left\{R_E^D(\boldsymbol{X}_1(g)), R_E^D(\boldsymbol{X}_2(g)), \ldots, R_E^D(\boldsymbol{X}_N(g))\right\} \tag{39}$$

If the mutant vector element $y_{ij,d}^g$ is outside the solution space boundaries, then the following formula will be applied.

$$y_{ij,d}^g = \begin{cases} \frac{ab_{min,d} + y_{ij,d}^g}{2} & if\ y_{ij,d}^g < ab_{min,d}, \\ \frac{ab_{max,d} + y_{ij,d}^g}{2} & if\ y_{ij,d}^g > ab_{max,d} \end{cases} \tag{40}$$

where $y_{ij,d}^g$ is the dth attribute of the mutated antibody y_{ij}^g and $ab_{min,d}$ and $ab_{max,d}$ are the minimum and maximum values of the dth attribute of the antibodies, respectively.(4) Selection (Line 35)

The ADECSA algorithm selects the best mutated antibody y_{ib}^g in the mutated set $\boldsymbol{Y}_i(g)$ and then compares it with its parent antibody ab_i^g; it chooses the antibody with the lowest affinity between them into the next population $\boldsymbol{AB}(g+1)$.

$$R_S^C\left(\boldsymbol{Y}_i(g) \cup ab_i^g\right) = \begin{cases} y_{ib}^g & if\ f\left(y_{ib}^g\right) \leqslant f\left(ab_i^g\right), \\ ab_i^g & otherwise \end{cases} \tag{41}$$

$$\boldsymbol{AB}(g+1) = R_S^C(\boldsymbol{Y}(g) \cup \boldsymbol{AB}(g)) = \left\{R_S^C\left(\boldsymbol{Y}_1(g) \cup ab_1^g\right), R_S^C\left(\boldsymbol{Y}_2(g) \cup ab_2^g\right), \ldots, R_S^C\left(\boldsymbol{Y}_N(g) \cup ab_N^g\right)\right\} \tag{42}$$

(5) Adaptive parameter control (Lines 36–60)

At the end of the generation, the historical memory $\boldsymbol{M}_k^F, \boldsymbol{M}_k^{CR}$ and $\boldsymbol{M}_k^{freq}$ will be updated based on the successful antibodies.

(6) Linear population size reduction (Line 62)

ADECSA linearly reduces the population size to speed up convergence.

(7) Premature convergence and stagnation detection (Line 63)

ADECSA adopts the population distribution to detect whether there is premature convergence or stagnation in the population. If detected, then it enhances the diversity of the population to alleviate these problems.

4. Experimental results

To verify the effectiveness of the ADECSA, our experimental study is conducted in the following aspects. First, we compare the performance of the ADECSA with state-of-the-art CSA and DE variants. Second, convergence comparisons are performed between ADECSA and other CSA and DE variants. Third, the influence of the antibody population size on the algorithm performance is studied by sensitivity analysis. Fourth, we analyze the benefit of each component in ADECSA on algorithm performance. Fifth, the effect of each mutation strategy is investigated during the evolutionary process.

4.1. Experimental setup

The CEC2014 [47] and CEC2017 [48] test suites are used to verify the performance of ADECSA, which are categorized into unimodal functions, simple multimodal functions, hybrid functions and composition functions, and the details are shown in Table 2.

In the experiments, ADECSA is compared with three CSA variants and five DE variants, including CLONALG, BCSA [11], MSHCSA [10], CoDE, MPEDE, L-SHADE, iL-SHADE and LSHADE-EpSin. ADECSA is implemented based on the open-source Wekaclassalgos package [2] in WEKA, and the corresponding parameters are set as follows: the initial antibody population size $N_{init} = 12 * D$ for CEC2014 and $N_{init} = 20 * D$ for CEC2017, the minimum population size $N_{min} = 4$, the clonal scale $N_c = 2$ for 10D, 30D and 50D, $N_c = 1$ for 100D, the historical memory pool size $H = 10$, the replacement proportion in the antibody pool $r = 10\%$, the diversity threshold $\delta_c = 0.001$, the stagnation threshold δ_s equals the current antibody population size, the population size for Gaussian walks $N_g = 20$, and the iteration number of Gaussian walks $T_g = 250$. The parameter sensitivity analysis of the initial antibody population size is given in Section 4.5. The parameters of the compared algorithms are the same as recommended in the original references and shown in Table 3.

To ensure the fairness and objectivity of the experiments, all the algorithms use the same termination criterion. Specifically, when the current number of function evaluations FES exceeds the maximum number of function evaluations $FES_{max} = D * 10,000$, or the algorithm finds the optimal solution, then the algorithm stops. D is the dimension of the objective function. In addition, we use the mean error to evaluate the algorithm performance, which is the absolute difference between the best value found in each algorithm and the optimal value. Moreover, we repeat each Algorithm30 times on the CEC2014 and CEC2017 benchmark functions to reduce the deviation from randomization.

4.2. Comparison with clonal selection algorithms

We compare the performance of ADECSA with a clonal selection algorithm and its variants, including CLONALG, BCSA and MSHCSA, on the CEC2014 benchmark suite. The reasons we choose these algorithms for comparison are as follows. CLONALG is a classical clonal selection algorithm and has been widely used in numerical optimization. As its variants, BCSA, MSHCSA and ADECSA modify the mutation operator by introducing learning mechanisms to improve the performance of CLONALG. Specifically, BCSA incorporates the Baldwinian learning mechanism into the clonal selection algorithm to alleviate the semi-blindness in the hypermutation operator caused by randomness. The MSHCSA uses combinatorial recombination and an adaptive mutation strategy based on success history to enhance the population diversity and search ability.

Tables 4–6 show the mean errors and standard deviations on CEC2014 benchmark functions with 30D, 50D and 100D, respectively. For a fair comparison, we directly cite the experimental results of MSHCSA in [10]. Due to the lack of detailed information on MSHCSA in each run, Wilcoxon's signed rank test was not used to evaluate the statistical significance between ADECSA and the compared CSA variants. The symbols "-", "+" and "≈" indicate that the compared algorithm is worse than, better than, or similar to ADECSA. The best results are in bold, and the last row records statistical information compared with ADECSA.

From Table 4, we can observe that the overall performance of ADECSA is obviously better than that of other compared algorithms on 30 benchmark functions with 30D, which outperforms other algorithms on 23 out of 30 benchmark functions. Specifically, on the unimodal functions (F_1-F_3), MSHCSA finds all the global optimal solutions and achieves the best performance, but ADECSA is also very competitive and obtains the optimal solutions on functions F_2 and F_3. On the simple multimodal functions (F_4-F_{16}), ADECSA is superior to other algorithms on all the functions except F_5. Both the ADECSA and MSHCSA find the optimal value on function F_7, but only the ADECSA achieves the optimal solution on function F_8 among all the compared algorithms. On the complex hybrid functions (F_{17}-F_{22}) and composition functions (F_{23}-F_{30}), ADECSA still exhibits the best performance on all the functions. In addition, the overall performance of CLONALG and BCSA is obviously lower than that of MSHCSA and ADECSA on all the functions except function F_{26}.

For 50D problems in Table 5, ADECSA achieves the best performance on 27 out of 30 benchmark functions. For the relatively simple unimodal and multimodal functions, ADECSA performs worse than MSHCSA only on functions F_1, F_4 and F_5.

[2] https://github.com/fracpete/wekaclassalgos

Table 2
The details of CEC2014 and CEC2017 test suites.

Function Type	CEC2014	CEC2017
Unimodal Functions	$F_1 - F_3$	$F_1 - F_3$
Simple Multimodal Functions	$F_4 - F_{16}$	$F_4 - F_{10}$
Hybrid Functions	$F_{17} - F_{22}$	$F_{11} - F_{20}$
Composition Functions	$F_{23} - F_{30}$	$F_{21} - F_{30}$

Table 3
The parameter settings of the involved algorithms.

Algorithms	Parameters
CLONALG	$N = 30, N_c = 4, r = 10\%, p_m = 1/D$
BCSA	$N = 30, N_c = 4, r = 10\%, p_l = 0.8, s = 0.8, p_m = 1/D$
MSHCSA	$N = 10 * D, N_c = 1, m = 0.3 * D, q = 0.2, M_{axage} = 6$
CoDE	$N = 30$
MPEDE	$N = 250, \lambda_1 = \lambda_2 = \lambda_3 = 0.2, ng = 20$
L-SHADE	$N_{init} = 18 * D, N_{min} = 4, p = 0.11, r^{arc} = 2.6, H = 6$
iL-SHADE	$N_{init} = 12 * D, N_{min} = 4, p = 0.10.2, r^{arc} = 2.6, H = 6$
LSHADE-EpSin	$N_{init} = 18 * D, N_{min} = 4, G_{ls} = 250, H = 5, \mu freq = 0.5$
ADECSA	$N_{init} = 12 * D$ or $20 * D, N_{min} = 4, N_c = 2$ or $1, H = 10, r = 10\%, \delta_c = 0.001, N_g = 20, T_g = 250$

Table 4
The mean errors and standard deviation means(stds) of ADECSA and other CSAs on CEC2014 benchmark functions with 30D.

CEC2014-30D		CLONALG	BCSA	MSHCSA	ADECSA
Unimodal Functions	F_1	8.02E+06(1.38E+06)	6.15E+06(1.12E+06)	**0.00E+00(0.00E+00)**	4.39E-09(1.38E-08)
	F_2	7.25E+07(7.11E+06)	2.93E+08(4.74E+07)	**0.00E+00(0.00E+00)**	**0.00E+00(0.00E+00)**
	F_3	6.12E+04(9.57E+03)	4.50E+02(7.31E+01)	**0.00E+00(0.00E+00)**	**0.00E+00(0.00E+00)**
Simple Multimodal Functions	F_4	7.26E+01(2.18E+01)	1.69E+02(1.58E+01)	1.18E+00(5.44E-01)	**4.76E-09(1.22E-08)**
	F_5	2.09E+01(4.45E-02)	2.04E+01(4.58E-02)	**2.00E+01(5.33E-05)**	2.01E+01(8.74E-02)
	F_6	2.62E+01(2.23E+00)	1.82E+01(1.15E+00)	2.41E+01(9.47E-01)	**1.74E-02(9.35E-02)**
	F_7	1.66E+00(8.77E-02)	3.59E+00(1.62E+00)	**0.00E+00(0.00E+00)**	**0.00E+00(0.00E+00)**
	F_8	1.82E+02(2.66E+01)	2.28E+01(3.36E+00)	5.24E+00(1.18E+00)	**0.00E+00(0.00E+00)**
	F_9	2.49E+02(2.11E+01)	1.16E+02(8.88E+00)	2.57E+01(5.10E+00)	**1.38E+01(3.03E+00)**
	F_{10}	3.97E+03(3.83E+02)	2.97E+02(2.53E+02)	2.17E+01(8.32E+00)	**6.94E-04(3.74E-03)**
	F_{11}	4.26E+03(4.03E+02)	3.10E+03(3.96E+02)	2.66E+03(3.09E+02)	**1.13E+03(2.58E+02)**
	F_{12}	1.97E+00(2.76E-01)	5.51E-01(6.78E-02)	4.21E-01(9.78E-02)	**1.34E-01(4.46E-02)**
	F_{13}	3.71E-01(5.00E-02)	5.57E-01(6.09E-02)	1.97E-01(3.21E-02)	**1.15E-01(2.14E-02)**
	F_{14}	2.18E-01(2.67E-02)	2.97E-01(5.01E-02)	1.79E-01(2.76E-02)	**1.72E-01(2.69E-02)**
	F_{15}	1.89E+01(1.18E+00)	2.49E+01(2.40E+00)	3.39E+00(6.21E-01)	**2.08E+00(3.59E-01)**
	F_{16}	1.26E+01(2.08E-01)	1.08E+01(2.68E-01)	1.10E+01(4.18E-01)	**8.23E+00(6.87E-01)**
Hybrid Functions	F_{17}	2.02E+05(5.80E+04)	1.98E+03(2.17E+02)	1.07E+03(1.56E+02)	**2.44E+02(1.30E+02)**
	F_{18}	1.05E+06(3.29E+05)	7.67E+01(1.14E+01)	1.78E+01(3.32E+00)	**7.42E+00(2.48E+00)**
	F_{19}	1.21E+01(9.34E-01)	1.15E+01(1.93E+00)	1.25E+01(1.01E+00)	**2.19E+00(6.09E-01)**
	F_{20}	7.77E+03(2.75E+03)	7.26E+01(8.78E+00)	1.20E+01(1.44E+00)	**2.95E+00(1.29E+00)**
	F_{21}	5.91E+04(1.58E+04)	9.92E+02(1.74E+02)	3.29E+02(9.73E+01)	**1.22E+02(7.56E+01)**
	F_{22}	5.60E+02(1.15E+02)	2.75E+02(1.10E+02)	6.23E+01(3.75E+01)	**4.76E+01(4.88E+01)**
Composition Functions	F_{23}	3.20E+02(9.12E-01)	3.17E+02(1.89E-01)	3.14E+02(0.00E+00)	**2.00E+02(0.00E+00)**
	F_{24}	2.31E+02(2.81E+00)	2.42E+02(3.63E+00)	2.10E+02(1.12E+01)	**2.00E+02(5.26E-09)**
	F_{25}	2.09E+02(1.49E+00)	2.07E+02(6.76E-01)	**2.00E+02(5.28E-04)**	**2.00E+02(0.00E+00)**
	F_{26}	**1.00E+02(4.65E-02)**	1.01E+02(6.59E-02)	**1.00E+02(2.81E-02)**	**1.00E+02(1.97E-02)**
	F_{27}	4.06E+02(1.08E+00)	4.34E+02(8.48E+01)	9.05E+02(1.85E+02)	**2.07E+02(2.49E+01)**
	F_{28}	3.31E+03(3.01E+02)	9.69E+02(2.75E+01)	3.67E+02(4.63E-01)	**2.00E+02(0.00E+00)**
	F_{29}	2.69E+04(6.36E+03)	1.82E+03(7.68E+02)	2.14E+02(6.42E-01)	**2.00E+02(0.00E+00)**
	F_{30}	1.24E+04(2.53E+03)	2.28E+03(4.21E+02)	3.25E+02(4.51E+01)	**2.00E+02(4.98E-08)**
	-	29	30	23	
	+	0	0	2	
	≈	1	0	5	

Moreover, it shows outstanding performance on all complex hybrid and composition functions. Similar to the benchmark functions with 30D, CLONALG and BCSA are obviously inferior to MSHCSA and ADECSA on all functions except function F_{26}.

Table 6 shows the comparison results on each benchmark function with 100D. From the table, we can observe that ADECSA is at least as good as the other comparison algorithms on 26 functions. For unimodal and multimodal functions, ADECSA has lower accuracy than MSHCSA on functions F_1, F_4 and F_5 and outperforms CLONALG and BCSA on all the functions. For complex hybrid and composition functions, ADECSA performs better than or comparable to other comparison algorithms on all the functions except function F_{19}.

Table 5
The mean errors and standard deviation means(stds) of ADECSA and other CSAs on CEC2014 benchmark functions with 50D.

CEC2014-50D		CLONALG	BCSA	MSHCSA	ADECSA
Unimodal Functions	F_1	1.10E+07(1.48E+06)	3.37E+07(1.48E+07)	**0.00E+00(0.00E+00)**	1.05E+03(1.87E+03)
	F_2	8.22E+07(7.73E+06)	9.44E+08(2.47E+08)	2.51E-07(1.77E-07)	**3.31E-08(9.65E-08)**
	F_3	7.06E+04(9.44E+03)	3.43E+03(1.33E+03)	**0.00E+00(0.00E+00)**	**0.00E+00(0.00E+00)**
Simple Multimodal Functions	F_4	8.41E+01(2.23E+01)	3.13E+02(1.86E+02)	**2.72E+01(8.12E-01)**	5.30E+01(4.83E+01)
	F_5	2.11E+01(3.44E-02)	2.05E+01(6.93E-02)	**2.00E+01(2.44E-04)**	2.02E+01(1.35E-01)
	F_6	5.37E+01(3.09E+00)	3.81E+01(4.68E+00)	5.09E+01(1.79E+00)	**3.46E-02(1.29E-01)**
	F_7	1.84E+00(6.66E-02)	1.22E+01(1.26E+01)	**0.00E+00(0.00E+00)**	**0.00E+00(0.00E+00)**
	F_8	3.87E+02(5.05E+01)	5.95E+01(3.56E+01)	3.26E+01(3.80E+00)	**0.00E+00(0.00E+00)**
	F_9	5.05E+02(4.52E+01)	2.45E+02(3.24E+01)	7.68E+01(8.82E+00)	**2.28E+01(9.51E+00)**
	F_{10}	7.14E+03(4.45E+02)	1.24E+03(1.19E+03)	1.51E+03(2.24E+02)	**4.77E-02(1.65E-01)**
	F_{11}	7.36E+03(4.06E+02)	7.23E+03(1.16E+03)	5.79E+03(4.10E+02)	**3.64E+03(5.63E+02)**
	F_{12}	1.85E+00(1.88E-01)	6.74E-01(1.76E-01)	6.82E-01(1.02E-01)	**1.84E-01(7.37E-02)**
	F_{13}	4.29E-01(4.51E-02)	5.68E-01(6.68E-02)	2.84E-01(2.99E-02)	**1.73E-01(2.49E-02)**
	F_{14}	2.62E-01(2.55E-02)	3.36E-01(3.96E-02)	2.22E-01(4.02E-02)	**1.81E-01(2.54E-02)**
	F_{15}	4.24E+01(2.18E+00)	2.60E+02(4.62E+02)	9.19E+00(1.24E+00)	**5.16E+00(5.11E-01)**
	F_{16}	2.22E+01(2.56E-01)	1.95E+01(5.03E-01)	2.03E+01(3.24E-01)	**1.66E+01(1.01E+00)**
Hybrid Functions	F_{17}	5.39E+05(1.40E+05)	1.05E+06(3.96E+05)	2.64E+03(2.99E+02)	**4.70E+02(1.87E+02)**
	F_{18}	2.48E+06(3.68E+05)	3.70E+02(3.95E+01)	7.78E+01(1.02E+01)	**2.17E+01(6.91E+00)**
	F_{19}	2.28E+01(1.74E+00)	3.51E+01(4.25E+00)	1.96E+01(1.56E+00)	**8.45E+00(1.40E+00)**
	F_{20}	1.49E+04(4.35E+03)	6.54E+02(1.26E+02)	4.89E+01(5.76E+00)	**8.63E+00(2.46E+00)**
	F_{21}	2.97E+05(8.62E+04)	4.40E+03(5.62E+02)	1.50E+03(2.48E+02)	**4.06E+02(1.49E+02)**
	F_{22}	1.20E+03(1.82E+02)	1.10E+03(2.35E+02)	4.63E+02(1.32E+02)	**1.09E+02(8.99E+01)**
Composition Functions	F_{23}	3.54E+02(1.52E+00)	3.51E+02(2.16E+00)	3.37E+02(2.85E-13)	**2.00E+02(0.00E+00)**
	F_{24}	2.96E+02(3.33E+00)	2.98E+02(1.40E+01)	2.64E+02(3.30E-01)	**2.00E+02(2.45E-13)**
	F_{25}	2.23E+02(4.64E+00)	2.18E+02(2.09E+00)	**2.00E+02(1.59E-02)**	**2.00E+02(0.00E+00)**
	F_{26}	**1.00E+02(8.56E-02)**	1.01E+02(1.06E-01)	**1.00E+02(2.83E-02)**	**1.00E+02(2.45E-02)**
	F_{27}	7.62E+02(5.92E+02)	1.32E+03(2.00E+02)	1.73E+03(3.70E+01)	**2.00E+02(8.34E-10)**
	F_{28}	7.34E+03(7.14E+02)	1.54E+03(1.40E+02)	3.57E+02(4.41E-01)	**2.00E+02(0.00E+00)**
	F_{29}	1.18E+05(1.57E+04)	1.39E+05(5.37E+04)	2.29E+02(8.98E-01)	**2.00E+02(0.00E+00)**
	F_{30}	3.33E+04(7.84E+03)	1.40E+04(1.43E+03)	9.58E+02(1.77E+02)	**2.00E+02(0.00E+00)**
	-	29	30	23	
	+	0	0	3	
	≈	1	0	4	

Overall, the best performer is ADECSA, followed by MSHCSA, BCSA and CLONALG. The reason the first three algorithms outperform CLONALG is that they improve the hypermutation operator in CLONALG by introducing the Baldwinian learning mechanism or the differential evolution mechanism. These mechanisms use different information between individuals to guide the evolution of the antibody population. ADECSA adopts linear population size reduction and premature convergence and stagnation detection and hence obtains the best performance on most benchmark functions.

4.3. Comparison with differential evolution algorithms

To further investigate the effectiveness of ADECSA, we compare ADECSA with five state-of-the-art differential evolution algorithms, including CoDE, MPEDE, L-SHADE, iL-SHADE and LSHADE-EpSin in this part of the experiments. CoDE and MPEDE adopt multiple mutation strategies. CoDE randomly combines three suitable mutation strategies (DE/rand/1, DE/rand/2, DE/current-to-rand/1) with control parameter settings ($[F = 1.0, CR = 0.1]$, $[F = 1.0, CR = 0.9]$, $[F = 0.8, CR = 0.2]$) to generate trial vectors. MPEDE divides the population into three smaller indicator subpopulations with equal size and one larger reward subpopulation. Then, each indicator subpopulation is assigned to different mutation strategies, and the reward subpopulation is assigned to the best mutation strategy in the previous generations. The last three algorithms are improved SHADE variants that use a single mutation strategy. LSHADE-EpSin is ranked as the joint winner with iL-SHADE in the CEC 2016 competition on real-parameter single objective optimization. More details of these algorithms can be found in Section 2.1.

Table 7 and Table 8 summarize the overall comparison results between ADECSA and other compared algorithms on CEC2014 and CEC2017 functions with 10D, 30D, 50D and 100D, and the detailed comparison results are shown in Tables S1-S8 in the supplementary material file due to space. Wilcoxon's signed rank test at the 0.05 significance level is conducted, and the symbols "-", "+" and "≈" indicate that the compared algorithm is worse than, better than, or similar to ADECSA, respectively. The overall comparison results are divided into five parts. The first four parts are the results on unimodal functions, simple multimodal functions, hybrid functions, and composition functions. The last part is the overall results on all the functions.

According to the results in Table 7 and Tables S1-S4, we observe that ADECSA performs significantly better than CoDE, MPEDE, L-SHADE and iL-SHADE, and slightly better than LSHADE-EpSin on most functions.

Table 6
The mean errors and standard deviation means(stds) of ADECSA and other CSAs on CEC2014 benchmark functions with 100D.

CEC2014-100D		CLONALG	BCSA	MSHCSA	ADECSA
Unimodal Functions	F_1	4.33E+07(3.61E+06)	1.57E+08(3.16E+07)	**9.95E+04(4.26E+04)**	1.40E+05(4.94E+04)
	F_2	9.43E+07(5.43E+06)	2.57E+09(4.08E+08)	2.46E+00(1.21E+00)	**4.60E-03(4.99E-03)**
	F_3	1.30E+05(8.62E+03)	1.11E+04(1.13E+03)	6.59E-07(3.64E-07)	**0.00E+00(0.00E+00)**
Simple Multimodal Functions	F_4	2.28E+02(2.39E+01)	6.90E+02(5.22E+01)	**8.44E+01(6.18E-01)**	1.82E+02(2.97E+01)
	F_5	2.13E+01(1.89E-02)	2.07E+01(2.67E-02)	**2.00E+01(1.87E-03)**	2.05E+01(1.74E-01)
	F_6	1.32E+02(4.48E+00)	8.77E+01(2.62E+00)	1.29E+02(2.61E+00)	**5.07E-01(6.34E-01)**
	F_7	1.85E+00(4.43E-02)	2.04E+01(2.54E+00)	**0.00E+00(0.00E+00)**	**0.00E+00(0.00E+00)**
	F_8	1.00E+03(5.81E+01)	1.04E+02(1.43E+01)	2.36E+02(1.64E+01)	**8.78E-06(1.78E-05)**
	F_9	1.26E+03(1.09E+02)	6.51E+02(3.09E+01)	3.17E+02(1.82E+01)	**4.78E+01(1.66E+01)**
	F_{10}	1.52E+04(7.59E+02)	1.74E+03(2.25E+02)	1.11E+04(5.54E+02)	**1.13E+00(5.21E-01)**
	F_{11}	1.55E+04(6.28E+02)	1.60E+04(4.84E+02)	1.69E+04(5.54E+02)	**1.01E+04(1.10E+03)**
	F_{12}	1.76E+00(1.53E-01)	8.36E-01(5.94E-02)	1.33E+00(1.34E-01)	**3.31E-01(4.74E-02)**
	F_{13}	4.35E-01(2.88E-02)	6.02E-01(3.47E-02)	3.67E-01(3.12E-02)	**3.05E-01(2.58E-02)**
	F_{14}	2.95E-01(1.87E-02)	3.48E-01(2.73E-02)	3.12E-01(9.89E-02)	**2.19E-01(1.63E-02)**
	F_{15}	1.32E+02(4.80E+00)	2.26E+02(5.78E+01)	3.00E+01(1.84E+00)	**1.54E+01(9.47E-01)**
	F_{16}	4.55E+01(4.31E-01)	4.16E+01(4.29E-01)	4.35E+01(4.63E-01)	**3.84E+01(6.95E-01)**
Hybrid Functions	F_{17}	2.25E+06(3.75E+05)	2.48E+07(5.60E+06)	5.87E+03(6.98E+02)	**2.99E+03(5.52E+02)**
	F_{18}	3.48E+06(2.98E+05)	1.38E+08(2.08E+07)	2.82E+02(9.44E+01)	**1.46E+02(2.39E+01)**
	F_{19}	6.19E+01(1.60E+01)	1.61E+02(1.11E+01)	**4.31E+01(3.35E-01)**	8.89E+01(1.33E+00)
	F_{20}	3.28E+04(6.66E+03)	4.25E+04(8.94E+03)	2.18E+02(1.49E+01)	**3.16E+01(5.79E+00)**
	F_{21}	1.48E+06(3.18E+05)	1.04E+07(2.59E+06)	4.76E+03(3.41E+02)	**6.30E+02(2.55E+02)**
	F_{22}	2.51E+03(2.09E+02)	3.23E+03(1.91E+02)	1.53E+03(3.35E+02)	**9.52E+02(2.50E+02)**
Composition Functions	F_{23}	3.79E+02(3.24E+00)	3.98E+02(7.98E+00)	3.45E+02(1.05E-12)	**2.00E+02(0.00E+00)**
	F_{24}	4.79E+02(1.25E+01)	4.16E+02(3.20E+00)	3.84E+02(2.91E+00)	**2.00E+02(2.45E-13)**
	F_{25}	2.93E+02(1.46E+01)	2.71E+02(5.81E+00)	2.01E+02(1.70E-01)	**2.00E+02(0.00E+00)**
	F_{26}	2.03E+02(1.71E-01)	1.91E+02(4.51E+01)	**1.00E+02(2.89E-02)**	**1.00E+02(2.45E-02)**
	F_{27}	3.70E+03(1.09E+03)	2.57E+03(6.73E+01)	3.71E+03(5.61E+01)	**2.00E+02(7.83E-10)**
	F_{28}	1.90E+04(1.98E+03)	3.58E+03(7.13E+02)	4.04E+02(5.30E+00)	**2.00E+02(0.00E+00)**
	F_{29}	1.89E+05(2.87E+04)	8.77E+05(2.16E+05)	2.43E+02(4.22E+00)	**2.00E+02(0.00E+00)**
	F_{30}	2.54E+05(4.46E+04)	5.80E+04(8.87E+03)	2.85E+03(2.73E+02)	**2.00E+02(0.00E+00)**
	-	29	30	24	
	+	1	0	4	
	≈	0	0	2	

- For 3 unimodal functions, ADECSA performs similarly to all the compared algorithms in solving 10D and 30D problems. When the dimensionality is greater than 30, ADECSA outperforms CoDE and MPEDE on most functions and is comparable to L-SHADE, iL-SHADE but is worse than LSHADE-EpSin on functions F_1 and F_2.
- For 13 simple multimodal functions, ADECSA shows better performance than CoDE, MPEDE, L-SHADE and LSHADE-EpSin and comparable performance to iL-SHADE.
- For 6 hybrid functions, ADECSA is also superior to CoDE, MPEDE and L-SHADE and performs slightly worse than LSHADE-EpSin. Compared with iL-SHADE, ADECSA shows inferior performance on 10D and 30D problems, but with the dimensionality increasing, the performance of ADECSA is gradually better than that of iL-SHADE on 2 functions with 50D and 5 functions with 100D.
- For 8 composition functions, ADECSA outperforms all the competitors on most functions with 10D, 30D, 50D and 100D. Significantly, the comparison results indicate that the ADECSA is more suitable than other algorithms for solving complex problems.

From the comparison in Table 8 and Tables S5-S8, we observe that ADECSA outperforms the other comparison algorithms on most benchmark functions, especially on relatively complex hybrid and composition functions.

- For 3 unimodal functions, ADECSA, L-SHADE, iL-SHADE and LSHADE-EpSin find the global optimal solutions on all the functions with 10D and 30D, while CoDE and MPEDE fail to obtain the global optimal solutions on function F_2 with 30D and function F_1 with 10D, respectively. For higher dimensions, ADECSA performs significantly better than CoDE and comparably to MPEDE but performs worse than L-SHADE, iL-SHADE and LSHADE-EpSin.
- For 7 simple multimodal functions, ADECSA is superior to CoDE and MPEDE and comparable to LSHADE-EpSin but inferior to L-SHADE and iL-SHADE.
- For 10 hybrid functions, ADECSA performs better than other algorithms on most functions except CoDE and L-SHADE on 10D.
- For 10 composition functions, ADECSA exhibits better performance than the other comparison algorithms on 10D, 50D and 100D problems.

Table 7
Results of Wilcoxon's signed rank test at the 0.05 significance level between ADECSA and other DE algorithms on CEC2014 functions.

Groups	vs.ADECSA	CoDE				MPEDE				L-SHADE				iL-SHADE				LSHADE-EpSin			
		10D	30D	50D	100D	10D	30D	50D	100D	10D	30D	50D	100D	10D	30D	50D	100D	10D	30D	50D	100D
Unimoda	-	0	1	3	3	0	0	2	2	0	0	0	0	0	0	1	1	0	0	0	0
Functions	+	0	0	0	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	2	2
	≈	3	2	0	0	3	3	0	0	3	3	2	2	3	3	1	1	3	3	1	1
Simple	-	9	9	10	9	11	10	11	9	4	3	5	8	1	1	3	4	3	4	4	7
Multimodal	+	1	1	2	3	0	0	0	2	2	1	1	4	1	3	4	6	0	0	3	1
Functions	≈	3	3	1	1	2	3	2	2	7	9	7	1	11	9	6	3	10	9	6	5
Hybrid	-	1	4	5	5	6	4	5	6	0	1	4	6	1	0	2	5	0	1	1	0
Functions	+	3	1	1	0	0	0	1	0	2	1	0	0	3	3	0	0	0	2	2	3
	≈	2	1	0	1	0	2	0	0	4	4	2	0	2	3	4	1	6	3	3	3
Composition	-	7	8	8	7	7	8	8	8	6	8	7	6	4	7	8	6	0	1	2	0
Functions	+	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	≈	1	0	0	0	1	0	0	0	2	0	0	2	4	1	0	2	8	7	6	8
All	-	17	22	26	24	24	22	27	25	10	12	16	20	6	8	14	16	3	6	7	7
Functions	+	4	2	3	4	0	0	1	3	4	2	3	5	4	6	5	7	0	2	7	6
	≈	9	6	1	2	6	8	2	2	16	16	11	5	20	16	11	7	27	22	16	17

Table 8
Results of Wilcoxon's signed rank test at the 0.05 significance level between ADECSA and other DE algorithms on CEC2017 functions.

Groups	vs.ADECSA	CoDE				MPEDE				L-SHADE				iL-SHADE				LSHADE-EpSin			
		10D	30D	50D	100D	10D	30D	50D	100D	10D	30D	50D	100D	10D	30D	50D	100D	10D	30D	50D	100D
Unimoda	-	0	0	2	3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Functions	+	0	0	0	0	0	0	1	1	0	0	1	3	0	0	1	3	0	0	1	2
	≈	3	3	1	0	3	3	2	1	3	3	2	0	3	3	2	0	3	3	2	1
Simple	-	4	3	4	6	6	4	5	5	1	0	0	2	0	0	0	2	1	1	3	1
Multimodal	+	0	1	0	1	0	1	0	1	0	4	4	5	1	4	5	5	0	1	1	2
Functions	≈	3	3	3	0	1	2	2	1	6	3	3	0	6	3	2	0	6	5	3	4
Hybrid	-	1	7	10	10	8	8	9	10	2	5	8	9	1	1	5	7	3	2	3	2
Functions	+	7	0	0	0	1	1	0	0	5	0	0	0	0	1	1	0	1	1	1	0
	≈	2	3	0	0	1	1	1	0	3	5	2	1	9	8	4	3	6	7	6	8
Composition	-	5	5	8	10	4	5	9	9	6	1	4	5	3	2	5	4	4	0	5	2
Functions	+	1	2	2	0	2	1	1	0	1	5	4	3	0	3	2	3	1	1	1	1
	≈	4	3	0	0	4	4	0	1	3	4	2	2	7	5	3	3	5	9	4	7
All	-	10	15	24	29	18	17	23	25	9	6	12	16	4	3	10	13	8	3	11	5
Functions	+	8	3	2	1	3	3	2	2	6	9	9	11	1	8	9	11	2	3	4	5
	≈	12	12	4	0	9	10	5	3	15	15	9	3	25	19	11	6	20	24	15	20

Generally, the experimental results show that ADECSA has the best overall performance compared with other algorithms. ADECSA performs slightly worse than L-SHADE, iL-SHADE and LSHADE-EpSin on unimodal functions with 50D and 100D, while it outperforms all other algorithms on more complex composition functions. In addition, although ADECSA adopts an improved adaptive parameter control mechanism inspired by LSHADE-EpSin, it still shows better performance than LSHADE-EpSin.

4.4. Convergence analysis

The convergence comparisons between ADECSA and other CSA and DE variants are conducted on CEC2017 benchmark functions with 50D in this part. Figs. 3-5, show the mean fitness errors with the number of function evaluations, where the symbol (log) represents the log value of the mean fitness error. From the figures, we can observe that ADECSA converges faster than CLONALG and BCSA on all the functions except functions F_5, F_{10}, F_{21} and F_{26}. Compared with DE variants, ADECSA usually has a slightly slower convergence speed at the early stage of evolution, but then it speeds up the convergence and obtains the best or competitive solutions on most functions at the end of evolution. For example, on unimodal and simple multimodal functions F_2, F_3 and F_9, the ADECSA converges slightly more slowly when the number of function evaluations is less than 200,000, but ultimately, it finds the global optimal solutions. On all the hybrid functions except function F_{13}, ADECSA converges faster than CoDE, MPEDE, L-SHADE, iL-SHADE and LSHADE-EpSin when the number of function evaluations is greater than 300,000. Similar phenomena can also be observed for most composition functions. In addition, we find that DE variants with fixed population sizes, such as CoDE and MPEDE, usually converge faster at the early stage of evolution but evolve more slowly due to premature convergence or stagnation at the later stage on most functions.

4.5. Parameter analysis

The initial antibody population size N_{init} is an important parameter in the clonal selection algorithm. Therefore, we analyze the influence of parameter N_{init} on the algorithm performance of ADECSA in this section. We conduct the experiments on the benchmark functions with 50D from CEC2014 and CEC2017, and evaluate the ADECSA performance in relation to different values of N_{init} based on Wilcoxon's signed rank test. Here, N_{init} takes the values {10*D, 12*D, 14*D, 16*D, 18*D, 20*D}, and other parameters are set to the default values.

Table 9 and Table 10 provide the mean error on each benchmark function with 50D from CEC2014 and CEC2017, respectively. From the above tables, it can be seen that the performance of ADECSA is sensitive to the initial antibody population size N_{init}. ADECSA with $N_{init} = 12 * D$ shows the best performance on the CEC2014 benchmark functions with 50D. Moreover, ADECSA with $N_{init} = 20 * D$ outperforms ADECSA with all the other values of N_{init} on the CEC2017 functions with 50D. Therefore, the parameter N_{init} is set to 12*D for CEC2014 functions and 20*D for CEC2017 functions in our experiments.

In addition, we usually hold that the larger the value of N_{init} is, the better the diversity in the antibody population. Thus, a larger antibody population is beneficial for solving relatively complex hybrid and composition functions, and a smaller population is beneficial for solving simple unimodal and multimodal functions. However, we observe some interesting phenomena. For example, as the value of N_{init} increases, ADECSA shows better performance on multimodal function F_9 from CEC2014 with 50D. When solving hybrid function F_{13} and composition function F_{29} from CEC2017 with 50D, ADECSA obtains the best performance with $N_{init} = 10 * D$. ADECSA does not strongly influence the performance on CEC2014 functions $F_3, F_7, F_{14}, F_{22} \sim F_{25}$ and $F_{27} \sim F_{30}$, and CEC2017 functions $F_2 \sim F_4, F_7, F_9, F_{16}, F_{17}, F_{23}, F_{24}, F_{28}$ and F_{30}. As a result, it can be concluded that N_{init} affects the performance related to the characteristic of functions but not in a manner closely related to the type of functions.

4.6. Influence of each component in ADECSA

The main components in ADECSA are a mutation strategy pool with adaptive parameters, linear population size reduction, and premature convergence and stagnation detection. To investigate the influence of the main components in ADECSA on the algorithm performance, we compare ADECSA with its three variants derived by removing different components on CEC2014 benchmark functions with 50D. The variants, termed ADECSA-noAMS, ADECSA-noLPR and ADECSA-noPSD, are ADECSA without an adaptive mutation strategy pool, ADECSA without linear population size reduction, and ADECSA without premature convergence and stagnation detection, respectively. Here, ADECSA-noAMS employs a hypermutation operator instead of the adaptive mutation strategy pool and sets $N_{init} = 50$ and $N_{min} = 20$ to obtain better performance.

Table 11 shows the mean error on each benchmark function with 50D. The Wilcoxon's signed rank test results are given in the last row. The improved mean error of the component is the difference between the mean error of the corresponding variant and that of ADECSA. The larger the improved mean error is, the stronger the positive influence. From the table, it is clearly that the overall performance of the ADECSA is significantly improved by the component of the adaptive mutation strategy pool, followed by the component of linear population size reduction and the component of premature convergence and stagnation detection. The component of the adaptive mutation strategy pool can significantly improve the algorithm performance on all the functions except the composition functions. The component of linear population size reduction is useless for improving the algorithm performance on unimodal functions, but it shows an obvious effect on composition functions. In addition, we find that only the component of the adaptive mutation strategy pool can effectively promote the algo-

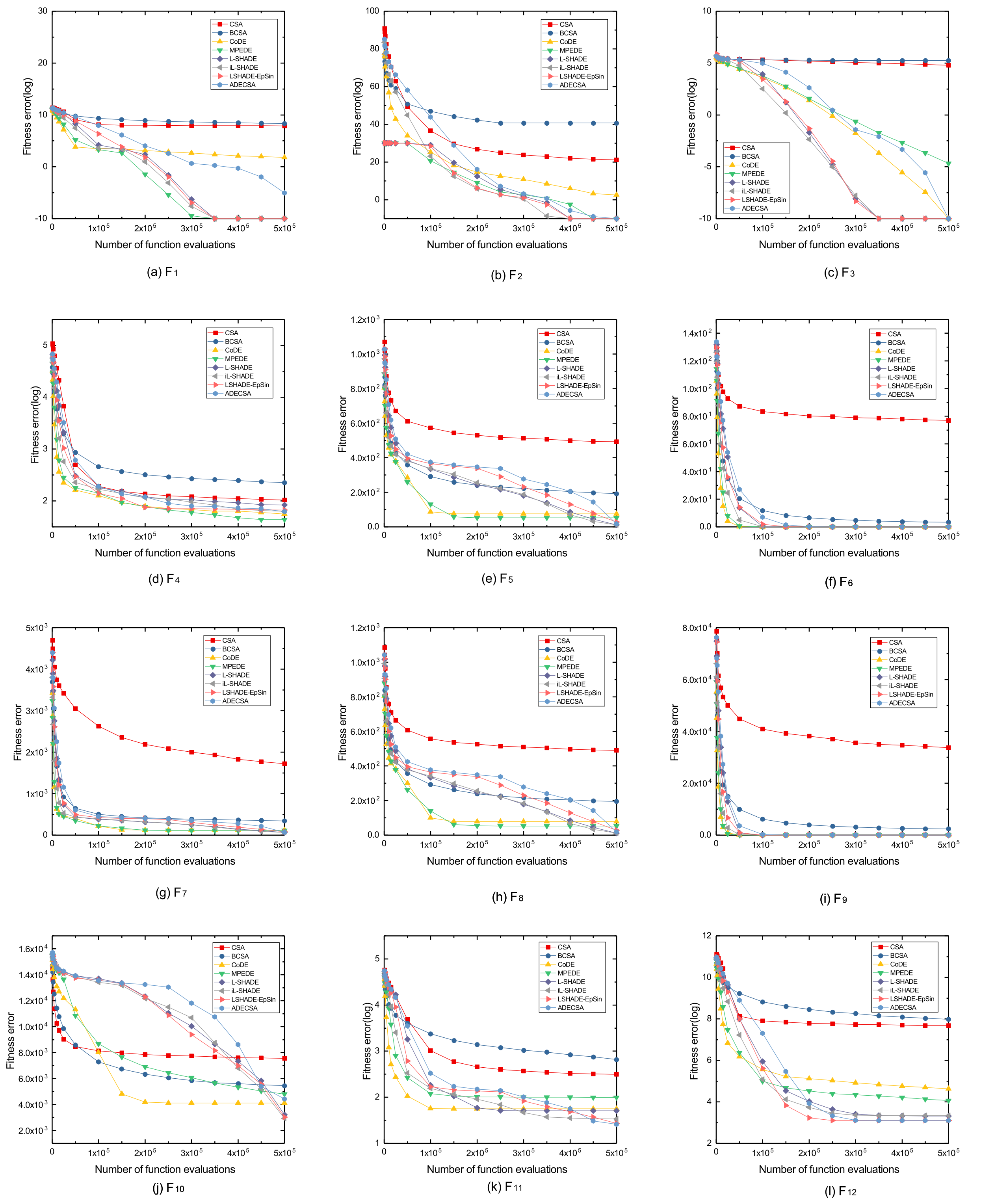


Fig. 3. Convergence curves of ADECSA and other compared algorithms on CEC2017 benchmark functions $F_1 \sim F_{12}$ with 50D.

rithm performance on functions $F_1 \sim F_3, F_7, F_9$ and F_{17}. A similar situation also occurs in the component of linear population size reduction on functions $F_4, F_{23} \sim F_{25}$ and $F_{27} \sim F_{30}$. By comparison, the component of premature convergence and stagnation detection cannot show significant advantages, but it is acceptable. Further improvement of this component will be researched in our future work.

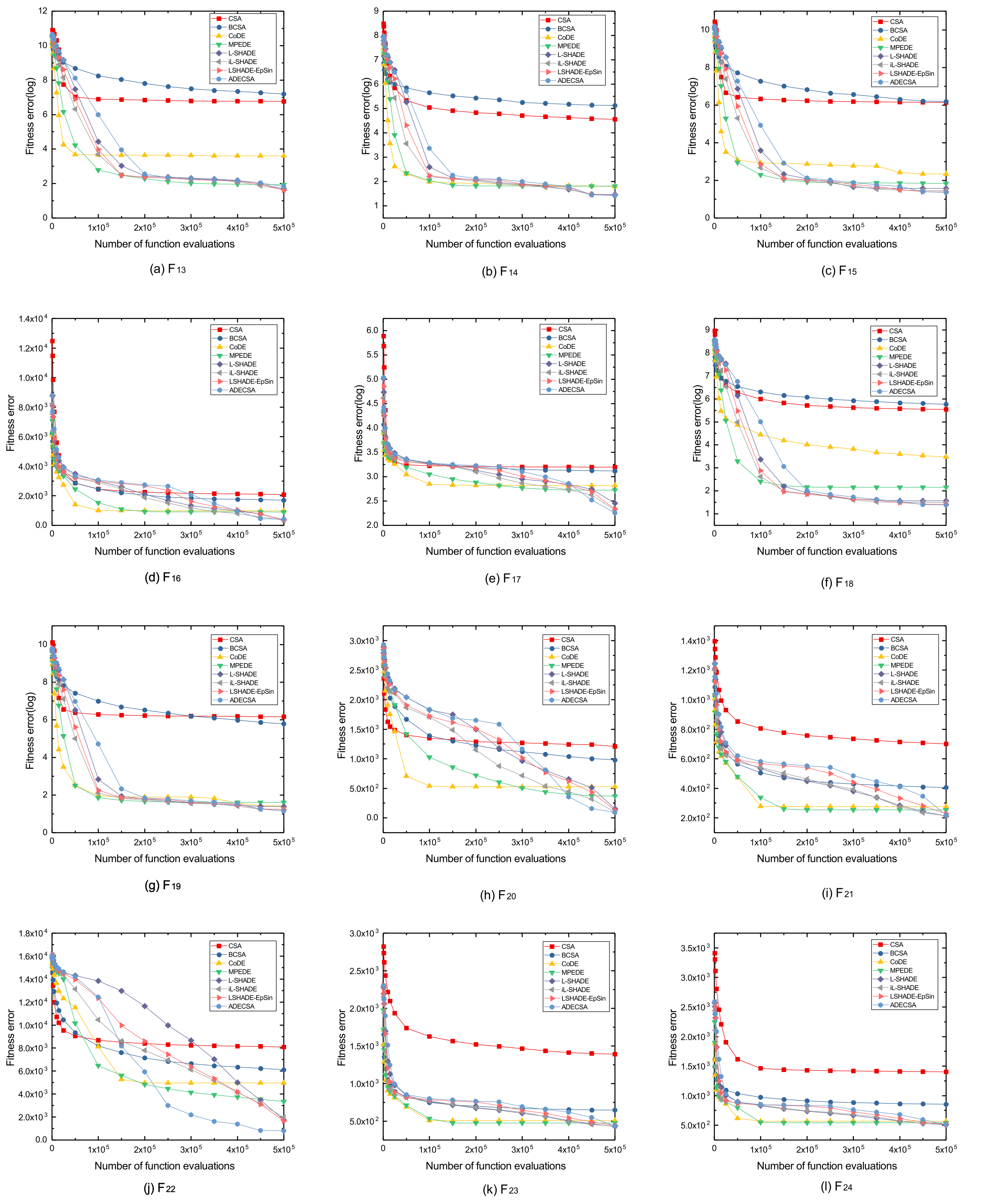


Fig. 4. Convergence curves of ADECSA and other compared algorithms on CEC2017 benchmark functions $F_{13} \sim F_{24}$ with 50D.

4.7. Evolution of mutation strategies

ADECSA includes three different mutation strategies: (1) DE/rand/1, (2) DE/current-to-pbest/1 with an archive, and (3) DE/current-rand/1. To investigate the effect of each mutation strategy during the evolutionary process, we analyze the prob-

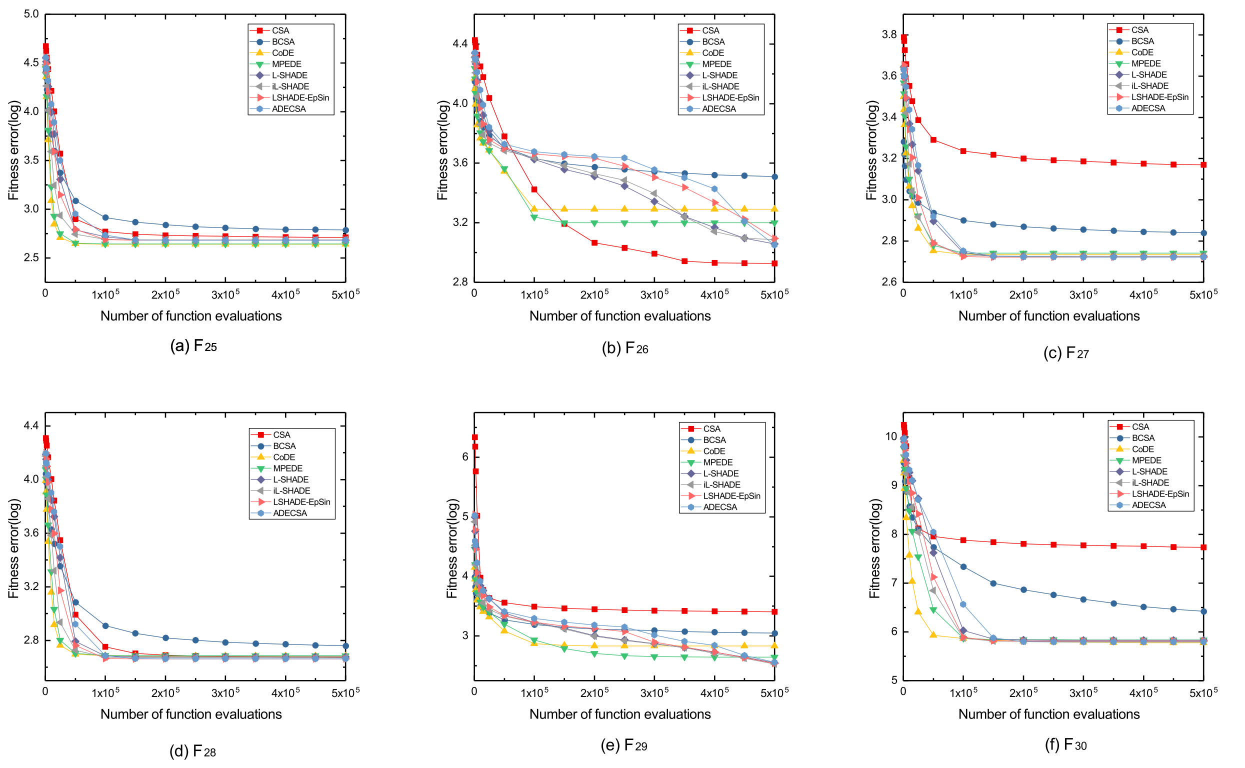


Fig. 5. Convergence curves of ADECSA and other compared algorithms on CEC2017 benchmark functions $F_{25} \sim F_{30}$ with 50D.

ability of each mutation strategy used to generate new individuals, and draw the corresponding probability curves. To make our comparison more objective, we choose six 100D functions $F_3, F_8, F_{10}, F_{22}, F_{25}$ and F_{28} from the different groups of CEC2014. From Fig. 6, we can see that during the population evolution, the probability of the strategy "DE/current-to-pbest/1 with an archive" generally increases, while the probabilities of the strategies "DE/rand/1" and "DE/current-rand/1" decrease except on function F_{10}. Overall, the strategy "DE/current-to-pbest/1 with an archive" dominates the whole evolutionary process, and the strategy "DE/current-rand/1" exerts the least effect.

4.8. Discussion

In summary, the experimental results show that the ADECSA performs better than the CSA and DE variants on most benchmark functions. However, there are still three aspects that need to be discussed.

- The performance of ADECSA on unimodal functions.
- The convergence behavior of the ADECSA at the early stage of evolution.
- The effect of premature convergence and stagnation detection in ADECSA.

From Table 8 and Tables S5-S8, we find that when solving some 50D and 100D CEC2017 unimodal functions, ADECSA cannot perform well compared with DE variants, such as L-SHADE, iL-SHADE and LSHADE-EpSin. From Fig. 3. (a)-(c), we also observe that L-SHADE, iL-SHADE and LSHADE-EpSin converge quickly and find the global optimal solutions when the number of function evaluations is less than 400,000. ADECSA has a slower convergence speed, although it finally obtains the global or approximate optimal solutions. This phenomenon may be caused by the selection mechanism of the mutation strategy adopted in ADECSA. Improper selection of mutation strategy cannot effectively guide antibodies to promising regions and results in a waste of the effective number of function evaluations.

The convergence comparison results in Figs. 3–5 show that ADECSA and compared DE variants with adaptive population size converge more slowly than DE variants with fixed population size at the early stage of evolution for most functions. The populations adopted by CoDE and MPEDE are smaller than those of the algorithms with adaptive population sizes. When the number of function evaluations is fixed, the smaller the population size is, the more evolutionary generations there are. Therefore, CoDE and MPEDE more easily generate better individuals at the early stage of evolution.

In the component analysis, we find that the component of the premature convergence and stagnation detection has less impact on the algorithm performance compared with two other components. In ADECSA, we adopt a fixed diversity thresh-

Table 9
The mean errors and standard deviation means(stds) of ADECSA in relation to different initial antibody population sizes over CEC2014 benchmark functions with 50D.

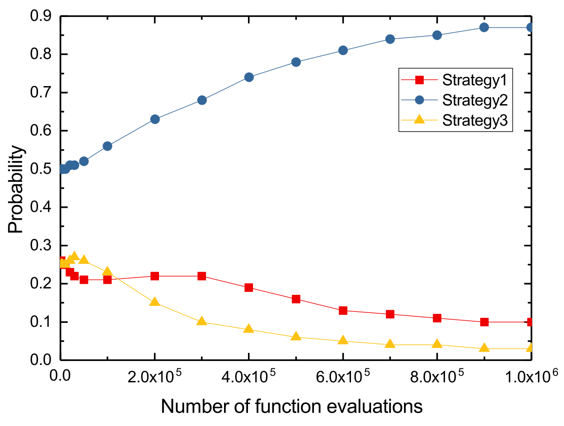
CEC2014		10*D	14*D	16*D	18*D	20*D	12*D
Unimodal Functions	F_1	5.04E+03 (4.82E+03)-	1.07E+02 (3.21E+02)+	1.40E+00 (3.16E+00)+	4.18E-01 (1.02E+00)+	6.63E-02 (7.04E-02)+	1.05E+03 (1.87E+03)
	F_2	0.00E+00 (0.00E+00)+	2.66E-07 (7.42E-07)-	7.74E-06 (2.33E-05)-	8.61E-06 (1.41E-05)-	5.79E-06 (1.21E-05)-	3.31E-08 (9.65E-08)
	F_3	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)
Simple Multimodal Functions	F_4	5.92E+01 (4.77E+01)≈	6.96E+01 (4.36E+01)≈	7.00E+01 (4.30E+01)-	7.55E+01 (3.90E+01)-	7.67E+01 (3.89E+01)-	5.30E+01 (4.83E+01)
	F_5	2.02E+01 (1.13E-01)≈	2.02E+01 (1.25E-01)≈	2.03E+01 (1.18E-01)≈	2.03E+01 (1.49E-01)≈	2.04E+01 (1.05E-01)-	2.02E+01 (1.35E-01)
	F_6	5.19E-02 (1.56E-01)≈	2.29E-05 (2.14E-05)≈	1.75E-02 (9.31E-02)+	2.72E-04 (2.92E-04)+	1.77E-02 (9.31E-02)+	3.46E-02 (1.29E-01)
	F_7	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)
	F_8	0.00E+00 (0.00E+00)≈	2.25E-08 (1.21E-07)≈	2.67E-07 (1.23E-06)-	8.46E-07 (1.03E-06)-	3.65E-05 (1.19E-04)-	0.00E+00 (0.00E+00)
	F_9	2.34E+01 (6.71E+00)≈	1.82E+01 (6.09E+00)≈	1.79E+01 (4.76E+00)+	1.77E+01 (6.06E+00)+	1.64E+01 (4.08E+00)+	2.28E+01 (9.51E+00)
	F_{10}	3.33E-03 (6.40E-03)+	1.18E+00 (3.01E+00)-	1.30E+00 (2.82E+00)-	2.68E+00 (4.23E+00)-	3.34E+00 (3.84E+00)-	4.77E-02 (1.65E-01)
	F_{11}	3.38E+03 (6.49E+02)≈	4.06E+03 (8.90E+02)≈	4.48E+03 (8.74E+02)-	4.45E+03 (6.81E+02)-	4.70E+03 (7.88E+02)-	3.64E+03 (5.63E+02)
	F_{12}	1.86E-01 (9.01E-02)≈	2.32E-01 (9.32E-02)-	2.88E-01 (1.15E-01)-	2.79E-01 (1.02E-01)-	3.13E-01 (9.46E-02)-	1.84E-01 (7.37E-02)
	F_{13}	1.71E-01 (2.32E-02)≈	1.82E-01 (2.71E-02)≈	1.83E-01 (2.26E-02)≈	1.87E-01 (2.37E-02)≈	1.85E-01 (1.87E-02)-	1.73E-01 (2.49E-02)
	F_{14}	1.93E-01 (2.67E-02)≈	1.85E-01 (2.28E-02)≈	1.82E-01 (2.89E-02)≈	1.75E-01 (3.12E-02)≈	1.86E-01 (2.70E-02)≈	1.81E-01 (2.54E-02)
	F_{15}	5.03E+00 (4.22E-01)≈	5.57E+00 (6.18E-01)-	5.77E+00 (6.12E-01)-	6.17E+00 (5.86E-01)-	6.14E+00 (5.93E-01)-	5.16E+00 (5.11E-01)
	F_{16}	1.65E+01 (1.03E+00)≈	1.70E+01 (6.87E-01)≈	1.73E+01 (8.37E-01)-	1.70E+01 (9.82E-01)≈	1.75E+01 (8.36E-01)-	1.66E+01 (1.01E+00)
Hybrid Functions	F_{17}	6.19E+02 (2.55E+02)-	3.75E+02 (1.79E+02)≈	3.70E+02 (1.63E+02)+	3.19E+02 (1.35E+02)+	3.55E+02 (1.37E+02)+	4.70E+02 (1.87E+02)
	F_{18}	2.86E+01 (1.30E+01)-	1.85E+01 (5.44E+00)≈	1.69E+01 (4.61E+00)+	1.57E+01 (5.56E+00)+	1.60E+01 (4.47E+00)+	2.17E+01 (6.91E+00)
	F_{19}	8.52E+00 (1.44E+00)≈	9.19E+00 (9.55E-01)-	9.44E+00 (1.06E+00)-	9.77E+00 (7.98E-01)-	9.57E+00 (8.71E-01)-	8.45E+00 (1.40E+00)
	F_{20}	9.00E+00 (2.51E+00)≈	7.64E+00 (2.28E+00)≈	6.54E+00 (1.98E+00)+	6.54E+00 (2.03E+00)+	5.99E+00 (1.54E+00)+	8.63E+00 (2.46E+00)
	F_{21}	4.15E+02 (1.41E+02)≈	3.34E+02 (1.11E+02)+	3.21E+02 (1.15E+02)+	2.73E+02 (1.44E+02)+	2.95E+02 (1.30E+02)+	4.06E+02 (1.49E+02)
	F_{22}	1.35E+02 (8.15E+01)≈	1.37E+02 (9.52E+01)≈	1.27E+02 (9.18E+01)≈	1.24E+02 (1.06E+02)≈	1.01E+02 (7.55E+01)≈	1.09E+02 (8.99E+01)
Composition Functions	F_{23}	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)
	F_{24}	2.00E+02 (1.41E-12)≈	2.00E+02 (1.91E-12)≈	2.00E+02 (1.51E-12)≈	2.00E+02 (1.37E-13)≈	2.00E+02 (1.79E-11)≈	2.00E+02 (2.45E-13)
	F_{25}	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)
	F_{26}	1.00E+02 (3.20E-02)≈	1.00E+02 (2.26E-02)≈	1.00E+02 (2.92E-02)≈	1.00E+02 (2.24E-02)≈	1.00E+02 (1.75E-02)-	1.00E+02 (2.45E-02)
	F_{27}	2.00E+02 (6.95E-06)≈	2.00E+02 (3.00E+01)≈	2.00E+02 (5.13E+01)≈	2.00E+02 (4.85E+01)≈	2.00E+02 (1.15E-11)≈	2.00E+02 (8.34E-10)
	F_{28}	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)
	F_{29}	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)
	F_{30}	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)
	-	3	5	9	8	12	
	+	2	2	7	7	7	
	≈	25	23	14	15	11	

Table 10
The mean errors and standard deviation means(stds) of ADECSA in relation to different initial antibody population sizes over CEC2017 benchmark functions with 50D.

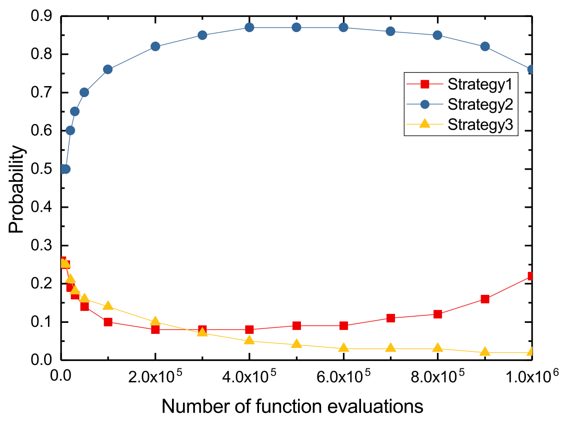
CEC2017		10*D	12*D	14*D	16*D	18*D	20*D
Unimodal Functions	F_1	7.53E-10 (4.06E-09)+	8.16E-09 (2.48E-08)+	3.11E-07 (1.11E-06)+	1.16E-06 (1.75E-06)+	7.21E-06 (1.34E-05)≈	9.21E-06 (1.98E-05)
	F_2	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)
	F_3	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)
Simple Multimodal Functions	F_4	5.17E+01 (4.31E+01)≈	4.75E+01 (3.79E+01)≈	5.31E+01 (4.38E+01)≈	6.59E+01 (4.33E+01)≈	5.19E+01 (4.16E+01)≈	6.22E+01 (4.69E+01)
	F_5	2.73E+01 (1.11E+01)-	2.28E+01 (8.85E+00)-	1.77E+01 (5.05E+00)≈	1.80E+01 (3.82E+00)≈	1.95E+01 (5.36E+00)≈	1.79E+01 (7.49E+00)
	F_6	1.60E-09 (8.61E-09)+	0.00E+00 (0.00E+00)+	3.24E-09 (9.32E-09)+	3.08E-08 (5.56E-08)+	1.10E-07 (1.63E-07)≈	2.07E-07 (2.93E-07)
	F_7	7.22E+01 (4.56E+00)≈	7.21E+01 (5.95E+00)≈	7.21E+01 (3.55E+00)≈	7.04E+01 (3.41E+00)≈	7.23E+01 (3.65E+00)≈	7.22E+01 (3.64E+00)
	F_8	2.60E+01 (6.25E+00)-	2.30E+01 (7.09E+00)≈	1.88E+01 (3.46E+00)≈	1.90E+01 (4.60E+00)≈	1.81E+01 (5.12E+00)≈	1.95E+01 (4.59E+00)
	F_9	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)
	F_{10}	3.29E+03 (5.54E+02)+	3.69E+03 (9.29E+02)+	3.85E+03 (6.90E+02)+	4.11E+03 (8.00E+02)≈	4.36E+03 (7.73E+02)≈	4.42E+03 (7.71E+02)
Hybrid Functions	F_{11}	2.85E+01 (4.93E+00)-	2.72E+01 (4.21E+00)≈	2.58E+01 (4.60E+00)≈	2.54E+01 (4.10E+00)≈	2.60E+01 (3.96E+00)≈	2.60E+01 (3.76E+00)
	F_{12}	1.91E+03 (2.77E+02)-	1.75E+03 (3.87E+02)-	1.62E+03 (3.81E+02)-	1.34E+03 (3.76E+02)≈	1.36E+03 (3.62E+02)≈	1.27E+03 (2.72E+02)
	F_{13}	4.76E+01 (3.17E+01)+	5.06E+01 (3.22E+01)≈	5.15E+01 (3.21E+01)≈	5.56E+01 (3.17E+01)≈	5.24E+01 (3.50E+01)≈	6.55E+01 (3.58E+01)
	F_{14}	3.03E+01 (2.53E+00)-	2.96E+01 (3.18E+00)-	2.83E+01 (3.18E+00)≈	2.81E+01 (3.26E+00)≈	2.74E+01 (2.79E+00)≈	2.69E+01 (1.99E+00)
	F_{15}	3.00E+01 (7.45E+00)-	2.71E+01 (6.44E+00)-	2.44E+01 (2.89E+00)≈	2.45E+01 (3.29E+00)≈	2.30E+01 (1.89E+00)≈	2.30E+01 (2.58E+00)
	F_{16}	3.96E+02 (1.69E+02)≈	4.40E+02 (1.77E+02)≈	3.41E+02 (1.41E+02)≈	4.28E+02 (1.78E+02)≈	4.07E+02 (1.68E+02)≈	4.14E+02 (1.67E+02)
	F_{17}	1.93E+02 (9.99E+01)≈	2.26E+02 (1.08E+02)≈	1.99E+02 (1.02E+02)≈	2.32E+02 (1.35E+02)≈	1.99E+02 (1.33E+02)≈	1.80E+02 (1.06E+02)
	F_{18}	3.10E+01 (4.50E+00)-	2.77E+01 (3.57E+00)-	2.62E+01 (2.64E+00)-	2.44E+01 (1.84E+00)≈	2.46E+01 (2.25E+00)≈	2.44E+01 (1.94E+00)
	F_{19}	1.68E+01 (3.11E+00)-	1.58E+01 (2.71E+00)-	1.53E+01 (2.88E+00)-	1.52E+01 (3.19E+00)≈	1.55E+01 (3.06E+00)≈	1.41E+01 (2.39E+00)
	F_{20}	1.44E+02 (9.93E+01)≈	1.56E+02 (8.91E+01)-	1.06E+02 (9.64E+01)≈	1.43E+02 (9.95E+01)≈	9.65E+01 (8.32E+01)≈	9.31E+01 (6.22E+01)
Composition Functions	F_{21}	2.26E+02 (7.64E+00)-	2.24E+02 (7.66E+00)≈	2.20E+02 (8.85E+00)≈	2.19E+02 (4.25E+00)≈	2.21E+02 (7.00E+00)≈	2.18E+02 (3.61E+00)
	F_{22}	1.68E+03 (1.83E+03)≈	1.77E+03 (2.08E+03)≈	1.40E+03 (2.02E+03)≈	2.12E+03 (2.40E+03)-	9.28E+02 (1.87E+03)≈	8.18E+02 (1.83E+03)
	F_{23}	4.38E+02 (8.63E+00)≈	4.35E+02 (8.38E+00)≈	4.35E+02 (8.68E+00)≈	4.36E+02 (1.05E+01)≈	4.35E+02 (6.75E+00)≈	4.37E+02 (8.20E+00)
	F_{24}	5.14E+02 (5.79E+00)≈	5.11E+02 (5.37E+00)≈	5.12E+02 (5.34E+00)≈	5.13E+02 (6.58E+00)≈	5.11E+02 (4.65E+00)≈	5.12E+02 (4.21E+00)
	F_{25}	4.82E+02 (4.00E+00)-	4.83E+02 (4.98E+00)-	4.81E+02 (2.90E+00)≈	4.81E+02 (2.89E+00)-	4.81E+02 (2.08E+00)-	4.80E+02 (1.67E-02)
	F_{26}	1.22E+03 (8.15E+01)-	1.15E+03 (8.21E+01)≈	1.14E+03 (5.17E+01)≈	1.13E+03 (5.33E+01)≈	1.14E+03 (5.92E+01)≈	1.12E+03 (7.51E+01)
	F_{27}	5.37E+02 (2.23E+01)-	5.30E+02 (1.73E+01)≈	5.23E+02 (8.97E+00)≈	5.32E+02 (2.49E+01)≈	5.29E+02 (1.85E+01)≈	5.28E+02 (1.73E+01)
	F_{28}	4.59E+02 (9.17E-14)≈	4.60E+02 (7.28E+00)≈	4.59E+02 (1.74E-13)≈	4.59E+02 (2.07E-13)≈	4.59E+02 (2.27E-13)≈	4.59E+02 (1.37E-13)
	F_{29}	3.51E+02 (1.93E+01)+	3.57E+02 (1.84E+01)≈	3.62E+02 (1.91E+01)≈	3.59E+02 (2.12E+01)≈	3.65E+02 (1.97E+01)≈	3.62E+02 (1.98E+01)
	F_{30}	6.43E+05 (5.58E+04)≈	6.36E+05 (6.66E+04)≈	6.35E+05 (4.71E+04)≈	6.18E+05 (4.63E+04)≈	6.40E+05 (6.99E+04)≈	6.23E+05 (3.69E+04)
	-	12	8	3	2	1	
	+	5	3	3	2	0	
	≈	13	19	24	26	29	

Table 11
Comparison results between ADECSA and its variants on CEC2014 benchmark functions with 50D.

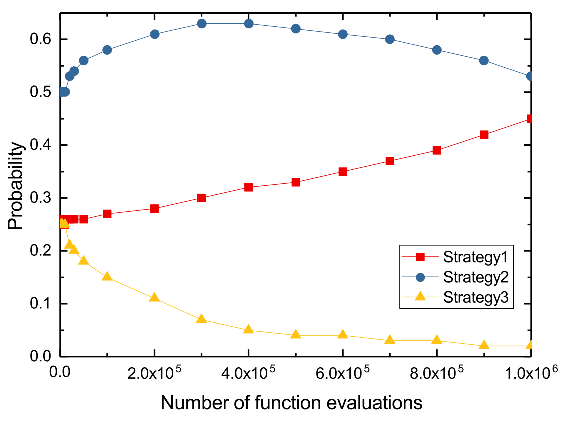
50D		ADECSA-noAMS	ADECSA-noLPR	ADECSA-noPSD	ADECSA
Unimodal Functions	F_1	9.54E+06 (1.45E+06)-	7.69E+02 (1.09E+03)≈	1.05E+03 (1.76E+03)≈	1.05E+03 (1.87E+03)
	F_2	8.37E+07 (5.00E+06)-	1.60E-09 (8.62E-09)+	0.00E+00 (0.00E+00)+	3.31E-08 (9.65E-08)
	F_3	5.93E+04 (6.01E+03)-	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)
Simple Multimodal Functions	F_4	7.27E+01 (2.63E+01)≈	8.21E+01 (3.04E+01)-	6.02E+01 (4.65E+01)≈	5.30E+01 (4.83E+01)
	F_5	2.11E+01 (3.20E-02)-	2.09E+01 (1.88E-01)-	2.02E+01 (1.19E-01)≈	2.02E+01 (1.35E-01)
	F_6	5.98E+01 (2.38E+00)-	4.80E-02 (1.82E-01)-	3.49E-02 (1.29E-01)-	3.46E-02 (1.29E-01)
	F_7	1.83E+00 (4.82E-02)-	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)
	F_8	4.83E+02 (5.67E+01)-	8.34E+01 (1.00E+01)-	0.00E+00 (0.00E+00)≈	0.00E+00 (0.00E+00)
	F_9	6.00E+02 (1.33E+02)-	2.10E+02 (2.98E+01)+	2.04E+01 (8.27E+00)≈	2.28E+01 (9.51E+00)
	F_{10}	6.96E+03 (4.24E+02)-	3.61E+03 (2.67E+02)-	4.37E-01 (1.70E+00)-	4.77E-02 (1.65E-01)
	F_{11}	7.34E+03 (4.11E+02)-	1.02E+04 (1.00E+03)-	3.65E+03 (9.03E+02)≈	3.64E+03 (5.63E+02)
	F_{12}	1.76E+00 (1.65E-01)-	2.05E+00 (7.51E-01)-	1.83E-01 (6.87E-02)≈	1.84E-01 (7.37E-02)
	F_{13}	4.09E-01 (3.64E-02)-	2.97E-01 (2.94E-02)-	1.84E-01 (2.18E-02)-	1.73E-01 (2.49E-02)
	F_{14}	2.54E-01 (1.91E-02)-	2.49E-01 (2.94E-02)-	1.81E-01 (2.55E-02)≈	1.81E-01 (2.54E-02)
	F_{15}	4.15E+01 (1.65E+00)-	2.21E+01 (1.33E+00)-	5.31E+00 (6.65E-01)≈	5.16E+00 (5.11E-01)
	F_{16}	2.27E+01 (2.67E-01)-	2.14E+01 (4.14E-01)-	1.70E+01 (5.81E-01)-	1.66E+01 (1.01E+00)
Hybrid Functions	F_{17}	4.43E+05 (1.06E+05)-	5.65E+02 (2.00E+02)≈	4.78E+02 (1.73E+02)≈	4.70E+02 (1.87E+02)
	F_{18}	2.52E+06 (4.14E+05)-	6.87E+01 (2.71E+01)-	2.59E+01 (7.33E+00)-	2.17E+01 (6.91E+00)
	F_{19}	2.20E+01 (1.81E+00)-	1.15E+01 (3.28E-01)-	8.47E+00 (1.35E+00)≈	8.45E+00 (1.40E+00)
	F_{20}	1.10E+04 (3.50E+03)-	3.02E+01 (1.01E+01)-	7.92E+00 (2.54E+00)≈	8.63E+00 (2.46E+00)
	F_{21}	2.49E+05 (7.50E+04)-	8.51E+02 (2.76E+02)-	4.08E+02 (1.21E+02)≈	4.06E+02 (1.49E+02)
	F_{22}	1.08E+03 (1.44E+02)-	4.23E+02 (1.02E+02)-	8.85E+01 (7.21E+01)≈	1.09E+02 (8.99E+01)
Composition Functions	F_{23}	2.00E+02 (0.00E+00)≈	3.44E+02 (2.33E-13)-	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)
	F_{24}	2.00E+02 (0.00E+00)≈	2.67E+02 (2.43E+00)-	2.00E+02 (3.21E-11)≈	2.00E+02 (2.45E-13)
	F_{25}	2.00E+02 (0.00E+00)≈	2.05E+02 (1.66E-01)-	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)
	F_{26}	1.04E+02 (1.79E+01)-	1.00E+02 (3.52E-02)-	1.00E+02 (2.33E-02)-	1.00E+02 (2.45E-02)
	F_{27}	2.00E+02 (0.00E+00)≈	3.17E+02 (2.08E+01)-	2.03E+02 (1.80E+01)≈	2.00E+02 (8.34E-10)
	F_{28}	2.00E+02 (0.00E+00)≈	1.14E+03 (3.38E+01)-	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)
	F_{29}	2.00E+02 (6.96E-06)≈	8.25E+02 (4.60E+01)-	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)
	F_{30}	2.00E+02 (8.35E-11)≈	8.69E+03 (4.79E+02)-	2.00E+02 (0.00E+00)≈	2.00E+02 (0.00E+00)
	-	22	24	6	
	+	0	2	1	
	≈	8	4	23	



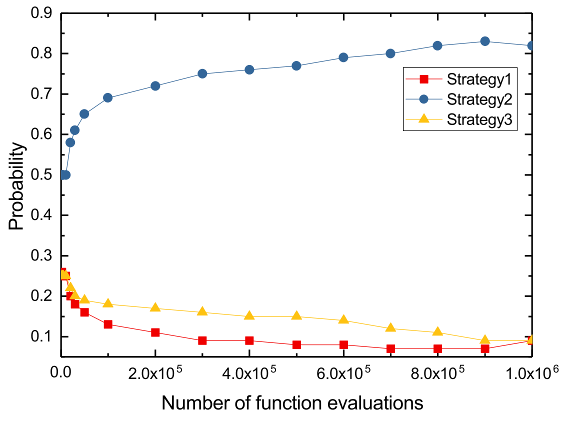
(a) F3



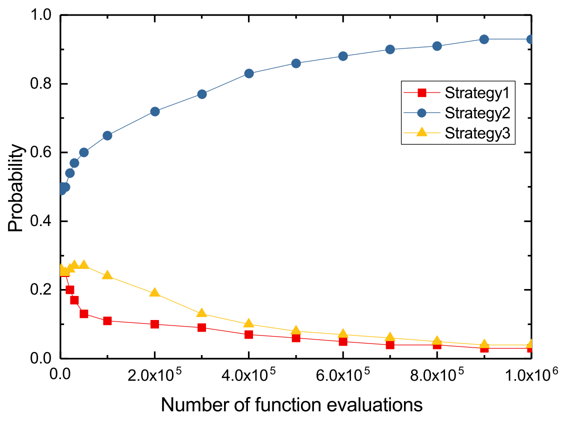
(b) F8



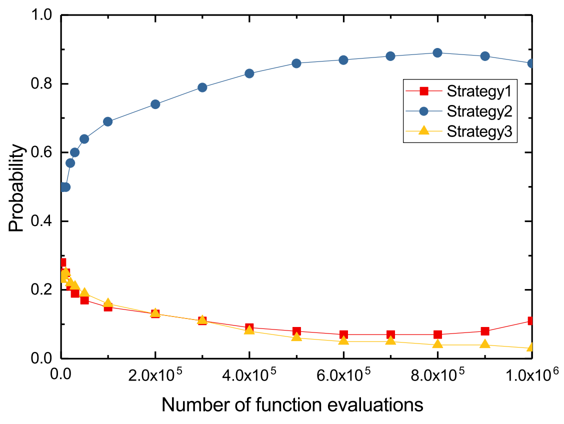
(c) F10



(d) F22



(e) F25



(f) F28

Fig. 6. Probability curves of applying each mutation strategy on CEC2014 benchmark functions with 100D.

old δ_c to judge whether the population has converged. In fact, the population diversity gradually decreases during the evolution process. Therefore, the diversity threshold δ_c needs to be designed as an adaptive parameter in our future work.

5. Conclusions

In this paper, an adaptive clonal selection algorithm with multiple differential evolution strategies is proposed, which is used to solve the semi-blindness in the hypermutation mechanism of clonal selection algorithms. Compared to CLONALG and its variants, the proposed method adopts a strategy pool and an adaptive parameter control mechanism based on historical records of success to guide the evolution of the antibody population and improve the search ability of the algorithm. It also incorporates an adaptive population resizing method to speed up convergence. Moreover, it employs the premature convergence detection method and the stagnation detection method to find premature convergence and stagnation during population evolution and then alleviates these problems by enhancing the diversity of the population. In addition, we carry out comprehensive experiments in six aspects to verify the effectiveness of ADECSA. The results show that ADECSA shows better or competitive performance compared with other state-of-the-art clonal selection algorithms and differential evolution algorithms.

In future work, we will explore how to further improve the component of premature convergence and stagnation detection to promote the algorithm performance. We will also extend our algorithm into multiobjective problems and apply it to solve real-world engineering optimization problems. In addition, since the evolution of each antibody is relatively independent, we can parallelize the algorithm to reduce execution time in the future.

CRediT authorship contribution statement

Yi Wang: Conceptualization, Methodology, Software, Writing - original draft. **Tao Li:** Conceptualization, Methodology, Software, Supervision. **Xiaojie Liu:** Software, Writing - review & editing. **Jian Yao:** Validation, Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work is supported by National Key Research and Development Program of China (Grant No. 2020YFB1805400), National Natural Science Foundation of China (Grant No. U19A2068, No. U1736212, No. 62032002, and No. 62102146).

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, athttps://doi.org/10.1016/j.ins.2022.04.043.

References

[1] G.D 'Angelo, F. Palmieri, GGA: a modified genetic algorithm with gradient-based local search for solving constrained optimization problems, Information Sciences 547 (2021) 136–162.
[2] B. Etaati, Z. Ghorrati, M.M. Ebadzadeh, A full-featured cooperative coevolutionary memory-based artificial immune system for dynamic optimization, Applied Soft Computing 117 (2022) 108389.
[3] E.H. Houssein, A.G. Gad, K. Hussain, P.N. Suganthan, Major advances in particle swarm optimization: theory, analysis, and application, Swarm and Evolutionary Computation 63 (2021) 100868.
[4] G. Rivera, C.A.C. Coello, L. Cruz-Reyes, E.R. Fernandez, C. Gomez-Santillan, N. Rangel-Valdez, Preference incorporation into many-objective optimization: an ant colony algorithm based on interval outranking, Swarm and Evolutionary Computation 69 (2022) 101024.
[5] Z. Wang, Z. Chen, Z. Wang, J. Wei, X. Chen, Q. Li, Y. Zheng, W. Sheng, Adaptive memetic differential evolution with multi-niche sampling and neighborhood crossover strategies for global optimization, Information Sciences 583 (2022) 121–136.
[6] Y. Wang, T. Li, Local feature selection based on artificial immune system for classification, Applied Soft Computing 87 (2020) 105989.
[7] K. Ajmera, T.K. Tewari, Vms-mcsa: virtual machine scheduling using modified clonal selection algorithm, Cluster Computing 24 (4) (2021) 3531–3549.
[8] M. Pantourakis, S. Tsafarakis, K. Zervoudakis, E. Altsitsiadis, A. Andronikidis, V. Ntamadaki, Clonal selection algorithms for optimal product line design: a comparative study, European Journal of Operational Research 298 (2) (2022) 585–595.
[9] L. Huang, Y. Ding, M. Zhou, Y. Jin, K. Hao, Multiple-solution optimization strategy for multirobot task allocation, IEEE Transactions on Systems, Man, and Cybernetics: Systems 50 (11) (2018) 4283–4294.
[10] W. Zhang, K. Gao, W. Zhang, X. Wang, Q. Zhang, H. Wang, A hybrid clonal selection algorithm with modified combinatorial recombination and success-history based adaptive mutation for numerical optimization, Applied Intelligence 49 (2) (2019) 819–836.
[11] M. Gong, L. Jiao, L. Zhang, Baldwinian learning in clonal selection algorithm for optimization, Information Sciences 180 (8) (2010) 1218–1236.
[12] G.E. Hinton, S.J. Nowlan, et al, How learning can guide evolution, Adaptive Individuals in Evolving Populations: Models and Algorithms 26 (1996) 447–454.
[13] Y. Peng, B.-L. Lu, Hybrid learning clonal selection algorithm, Information Sciences 296 (2015) 128–146.
[14] R. Storn, K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, Journal of Global optimization 11 (4) (1997) 341–359.

[15] Y. Li, S. Wang, B. Yang, An improved differential evolution algorithm with dual mutation strategies collaboration, Expert Systems with Applications 153 (2020) 113451.
[16] W. Deng, S. Shang, X. Cai, H. Zhao, Y. Song, J. Xu, An improved differential evolution algorithm and its application in optimization problem, Soft Computing 25 (7) (2021) 5277–5298.
[17] J. Cheng, Z. Pan, H. Liang, Z. Gao, J. Gao, Differential evolution algorithm with fitness and diversity ranking-based mutation operator, Swarm and Evolutionary Computation 61 (2021) 100816.
[18] S. Khalfi, A. Draa, G. Iacca, A compact compound sinusoidal differential evolution algorithm for solving optimisation problems in memory-constrained environments, Expert Systems with Applications 186 (2021) 115705.
[19] Z. Meng, C. Yang, Two-stage differential evolution with novel parameter control, Information Sciences 596 (2022) 321–342.
[20] L. Zhu, Y. Ma, Y. Bai, A self-adaptive multi-population differential evolution algorithm, Natural Computing 19 (1) (2020) 211–235.
[21] R.D. Al-Dabbagh, F. Neri, N. Idris, M.S. Baba, Algorithmic design issues in adaptive differential evolution schemes: review and taxonomy, Swarm and Evolutionary Computation 43 (2018) 284–311.
[22] R. Tanabe, A. Fukunaga, Success-history based parameter adaptation for differential evolution, in: 2013 IEEE Congress on Evolutionary Computation, IEEE, 2013, pp. 71–78.
[23] R. Tanabe, A.S. Fukunaga, Improving the search performance of shade using linear population size reduction, in: 2014 IEEE Congress on Evolutionary Computation, IEEE, 2014, pp. 1658–1665.
[24] N.H. Awad, M.Z. Ali, P.N. Suganthan, R.G. Reynolds, An ensemble sinusoidal parameter adaptation incorporated with l-shade for solving cec2014 benchmark problems, in: 2016 IEEE Congress on Evolutionary Computation, IEEE, 2016, pp. 2958–2965.
[25] J. Brest, M.S. Maučec, B. Bošković, il-shade: improved l-shade algorithm for single objective real-parameter optimization, in: 2016 IEEE Congress on Evolutionary Computation, IEEE, 2016, pp. 1188–1195.
[26] J. Brest, M.S. Maučec, B. Bošković, Single objective real-parameter optimization: algorithm jso, in: 2017 IEEE Congress on Evolutionary Computation, IEEE, 2017, pp. 1311–1318.
[27] Z. Meng, J.-S. Pan, Hard-de: hierarchical archive based mutation strategy with depth information of evolution for the enhancement of differential evolution on numerical optimization, IEEE Access 7 (2019) 12832–12854.
[28] Y. Wang, Z. Cai, Q. Zhang, Differential evolution with composite trial vector generation strategies and control parameters, IEEE Transactions on Evolutionary Computation 15 (1) (2011) 55–66.
[29] G. Wu, R. Mallipeddi, P.N. Suganthan, R. Wang, H. Chen, Differential evolution with multi-population based ensemble of mutation strategies, Information Sciences 329 (2016) 329–345.
[30] X. Li, L. Wang, Q. Jiang, N. Li, Differential evolution algorithm with multi-population cooperation and multi-strategy integration, Neurocomputing 421 (2021) 285–302.
[31] L. Deng, C. Li, R. Han, L. Zhang, L. Qiao, Tpde: a tri-population differential evolution based on zonal-constraint stepped division mechanism and multiple adaptive guided mutation strategies, Information Sciences 575 (2021) 22–40.
[32] G. Wu, R. Mallipeddi, P.N. Suganthan, Ensemble strategies for population-based optimization algorithms–a survey, Swarm and Evolutionary Computation 44 (2019) 695–711.
[33] G. Wu, X. Shen, H. Li, H. Chen, A. Lin, P.N. Suganthan, Ensemble of differential evolution variants, Information Sciences 423 (2018) 172–186.
[34] R. Mallipeddi, P.N. Suganthan, Q.-K. Pan, M.F. Tasgetiren, Differential evolution algorithm with ensemble of parameters and mutation strategies, Applied Soft Computing 11 (2) (2011) 1679–1696.
[35] X. Wang, C. Li, J. Zhu, Q. Meng, L-shade-e: ensemble of two differential evolution algorithms originating from l-shade, Information Sciences 552 (2021) 201–219.
[36] V. Stanovov, S. Akhmedova, E. Semenkin, Lshade algorithm with rank-based selective pressure strategy for solving cec 2017 benchmark problems, in: 2018 IEEE Congress on Evolutionary Computation, IEEE, 2018, pp. 1–8.
[37] F.M. Burnet, The clonal selection theory of acquired immunity, Cambridge University Press, Cambridge, U.K., 1959.
[38] L.N. De Castro, F.J. Von Zuben, Learning and optimization using the clonal selection principle, IEEE Transactions on Evolutionary Computation 6 (3) (2002) 239–251.
[39] X. Yan, P. Li, K. Tang, L. Gao, L. Wang, Clonal selection based intelligent parameter inversion algorithm for prestack seismic data, Information Sciences 517 (2020) 86–99.
[40] C. Zhou, X. Liu, F. Xu, W. Chen, Sliding mode switch control of adjustable hydro-pneumatic suspension based on parallel adaptive clonal selection algorithm, Applied Sciences 10 (5) (2020) 1852.
[41] J. Qiao, F. Li, S. Yang, C. Yang, W. Li, K. Gu, An adaptive hybrid evolutionary immune multi-objective algorithm based on uniform distribution selection, Information Sciences 512 (2020) 446–470.
[42] J.-Q. Li, Z.-M. Liu, C. Li, Z.-X. Zheng, Improved artificial immune system algorithm for type-2 fuzzy flexible job shop scheduling problem, IEEE Transactions on Fuzzy Systems 29 (11) (2020) 3234–3248.
[43] J. Liu, Z. Zhang, F. Chen, S. Liu, L. Zhu, A novel hybrid immune clonal selection algorithm for the constrained corridor allocation problem, Journal of Intelligent Manufacturing (2020) 1–20.
[44] I. Fefelova, A. Fefelov, V. Lytvynenko, O. Ohnieva, S. Smailova, Prediction of native protein conformation by a hybrid algorithm of clonal selection and differential evolution, in: Lecture Notes in Computational Intelligence and Decision Making, Springer, 2021, pp. 314–330.
[45] W. Luo, X. Lin, T. Zhu, P. Xu, A clonal selection algorithm for dynamic multimodal function optimization, Swarm and Evolutionary Computation 50 (2019) 100459.
[46] G. Corriveau, R. Guilbault, A. Tahan, R. Sabourin, Review and study of genotypic diversity measures for real-coded representations, IEEE Transactions on Evolutionary Computation 16 (5) (2012) 695–710.
[47] J.J. Liang, B.Y. Qu, P.N. Suganthan, Problem definitions and evaluation criteria for the cec 2014 special session and competition on single objective real-parameter numerical optimization, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore 635 (2013) 490.
[48] G. Wu, R. Mallipeddi, P.N. Suganthan, Problem definitions and evaluation criteria for the cec 2017 competition on constrained real-parameter optimization, National University of Defense Technology, Changsha, Hunan, PR China and Kyungpook National University, Daegu, South Korea and Nanyang Technological University, Singapore, Technical Report.