Arab J Sci Eng (2014) 39:6149–6174
DOI 10.1007/s13369-014-1248-7

RESEARCH ARTICLE - COMPUTER ENGINEERING AND COMPUTER SCIENCE

A Novel Self-adaptive Differential Evolution Algorithm with Population Size Adjustment Scheme

Shuguang Zhao · Xu Wang · Liang Chen · Wu Zhu

Received: 2 December 2012 / Accepted: 31 August 2013 / Published online: 8 July 2014

Abstract It is well known that mutation scale factor, the crossover constant, and the population size are three main control parameters of the differential evolution (DE) algorithm. These parameters are of great importance to the efficiency of a DE algorithm. However, finding appropriate settings is a difficult task. In this work, a self-adaptive DE with population adjustment scheme (SAPA) is proposed to tune the size of offspring population. The novel algorithm involves two DE strategies and two population adjustment schemes. The performance of the SAPA algorithm is evaluated on a set of benchmark problems. Simulation results show that the proposed algorithm is better than, or at least comparable with, other classic or adaptive DE algorithms. Performance comparisons with some other well-known evolutionary algorithms from literatures are also presented.

Keywords Differential evolution (DE) · Self-adaptation · Global optimization · Population adjustment schemes

الخلاصـة

من المعروف جيداً أن عامل قياس الطفرة وثابت التعابر وحجم السكان هي ثلاث معلمات تحكّمٍ رئيسية في خوارزمية التطور التفاضلية. وتعد هذه المـعـلمات ذات أهمية عظيمـة لكفاءة خوارزمية التطور التفاضلية. ومـع ذلك ، فإن إيجاد الإعدادات المناسبة مهمة صعبة. وقد تم عرض تطور تفاضلي ذي تكيفٍ ذاتيٍ مع مخطط ضبط للسكان وذلك لضبط حجم تناسل السكان. وتشمل الخوارزمية الجديدة استراتيجيتي تطورٍ تفاضليٍ ومخطَّطي ضبط للسكان. وقد تم تقييم أداء خوارزمية التكيف الذاتي مع ضبط السكان على مجموعة من المسائل الإرشادية. وتُظهر نتائج المحاكاة أن الخوارزمية المقترحة أفضل من أو على الأقل مماثلة للخوارزميات الكلاسيكية أو التطورية التفاضلية التكيفية الأخرى. كما تم عرض مقارنات للأداء ببعض الخوارزميات التطورية الأخرى المعروفة من المراجعات الأدبية.

This paper was partially supported by National Natural Science Foundation of China (No. 61271114 and No. 61203325) and Innovation Program of Shanghai Municipal Education Commission (No. 14ZZ068).

S. Zhao · X. Wang (✉) · L. Chen
College of Information Science and Technology, Donghua University, Shanghai 201620, People's Republic of China
e-mail: daisy_wang1@aliyun.com

S. Zhao
e-mail: sg.zhao@126.com

X. Wang
College of Automotive Engineering,
Shanghai University of Engineering and Science,
Shanghai 201620, People's Republic of China

W. Zhu
School of International Relations and Public Affairs,
Fudan University, Shanghai 200433, People's Republic of China
e-mail: dtzhuwu@gmail.com

L. Chen
Guangxi Branch, Agricultural Bank of China, Nanning 530028,
People's Republic of China
e-mail: lawrence.cl@hotmail.com

1 Introduction

Differential evolution (DE) is perceived as a reliable and versatile population-based heuristic optimization technique, which exhibits remarkable performance in a wide variety of problems from diverse fields [1–3]. It has been successfully applied in diverse fields such as pattern recognition [4], data mining [5], combinatorial optimization [6], and multi-objective optimization [9]. Like other EAs [7,8], the performance of DE is quite dependent on the setting of control parameters such as the mutation factor F, the crossover probability C_r, and the population size NP according to both experimental studies Gamperle et al. [10] and theoretical analyses [11]. A good deal of research work has been undertaken so far to improve the ultimate performance of DE by tuning its control parameters. There is, however, no universal parameter setting solution suitable for various problems.

Owing to replace sensitive parameters in DE by less sensitive parameters, the adaptive and self-adaptive DE algorithms have shown more reliable convergence performance than the classic DE algorithms without parameter control [12–14]. Some adaptive algorithms [15,16] are developed based on the classic DE/rand/1/bin that is known to be robust, but less efficient in terms of convergence rate. Because no single mutation method can turn out to be the best for all problems, others [13,17] simultaneously implement DE/rand/1/bin and one or more greedy DE variants such as DE/current-to-best/1/ bin and dynamically update the probability of using each variant to generate offspring. Usually self-adaptation is used to tune F and C_r. In [17], for the first time, Qin et al. presented a self-adaptive variant of DE (SaDE), where trial vector generation strategies are gradually self-adapted by learning from their prior experiences in generating promising solutions. Lately, the previous work was extended [18]. In their proposed SaDE approach, the parameter F is approximated by a normal distribution with mean value 0.5 and standard deviation 0.3. And the range of C_r values is adjusted according to previous C_r values, which have generated trial vectors successfully entering the new population. Furthermore, in SaDE, four diverse characteristics of mutation strategies have been adopted so that they can exhibit distinct performance characteristics during different stages of the evolution. According to the idea of SaDE, an ensemble of mutation strategies and control parameters (EPSDE) was employed in [19]. A pool of mutation strategies competes to produce successful offspring population in this algorithm. Unlike the SaDE, the target vector of the EPSDE should be randomly re-initialized with a new mutation strategy from the respective pools or from the successful combinations stored with equal probability when the trial vector performs poorer. In order to avoid the need for problem-specific parameter tuning and also to improve the convergence characteristics of DE, an adaptive DE-variant, called JADE, was recently proposed [20]. The algorithm applies a new mutation strategy, referred by the authors as DE/current-to-*p*best, and utilizes an optional external archive to provide information of progress direction. Meanwhile, in JADE, the parameters F and C_r are also self-adaptively tuned according to a normal distribution and a Cauchy distribution, respectively. Simulation results show that JADE is better than, or at least comparable with, other classic or adaptive DE algorithms, the canonical PSO, and other EAs. Additionally, Brest et al. [21] proposed a self-adaptation scheme for the DE control parameters (jDE). Control parameters F and C_r are encoded into the individual and adjusted by two new parameters τ_1 and τ_2. In jDE algorithm, a set of F and C_r values was assigned to each individual in the population, augmenting the dimensions of each vector. The better values of these encoded control parameters lead to better individuals that in turn are more likely to survive and produce offspring and thus propagate these better parameter values.

Recently, various strategies for population size of DE have received particular research attention [22–25]. Teo [15] presented a DE algorithm with dynamic population sizing strategy based on self-adaption, while F and C_r are also self-adapted. In [15], there is a population size parameter π, which is initialized to a random value according to a uniform distribution between $[-0.5, 0.5]$, which may be negative or positive to allow for variations in population size that can decrease as well as increase. Consequently, a more suitable population size along with its parameter settings can be determined adaptively to match different search stages. In [26], an improved jDE algorithm with population size reduction (called a dynNP-DE algorithm) was used for large-scale global optimization on CEC 2008. The number of allowed fitness evaluations (FEs) is predefined, and dimension of problem is large. Therefore, small population size is highly recommended for DE algorithms. In dynNP-DE algorithm, there are the largest population size at the beginning of the evolutionary process and the smallest at the end. In the meantime, the population size is gradually descending. The concept behind this strategy is that of focusing the search in progressively smaller search spaces in order to inhibit the DE stagnation in an environment with high dimensionality. The population size reduction mechanism improves the algorithm's efficiency and robustness.

The three control parameters all depend on each other, and different optimized problems need different good parameter setting. Meanwhile, there are still no general guidelines for the parameters setting. In another word, question how to self-adaptively tune the parameters in DE remains open. Inspired by above considerations, we introduce a self-adaptive DE with population adjustment scheme (SAPA) in which the population size can be adjusted dynamically based upon the online solution-searching status. In our algorithm, two population adjustment schemes are designed to obtain the appropriate parameter of *NP* according to the desired population distribution. More specifically, on the one hand, in order to improve the diversity of the population and to enhance local search ability, population-increasing strategy is presented to generate better individuals. On the other hand, a population-decreasing scheme is applied to remove the redundant individuals and to save computational space in evolutionary process. Furthermore, a *trigger strategy monitor* is introduced to control the sensitivity of population strategy. The monitor will trigger population-decreasing or increasing strategy in accordance with the solution-searching status. Moreover, in this paper, two DE strategies (DE/current-to-best/1 strategy and DE/current-to-*p*best/1 strategy) are adopted to improve the search performance efficiently. In each iteration of the evolutionary process, only one strategy is active. To be specific, at an earlier stage of evolutionary process, DE/current-to-best/1 strategy is used to achieve fast convergence speed; with the increase of generation, to prevent strategy trapping into a local minimum, DE/current-to-*p*best/1 strategy is applied to search a relatively large region, which is biased toward promising progress directions. In addition, we compare SAPA with several recently proposed evolutionary algorithms and observe competitive experimental results on benchmark functions.

The rest of the paper is organized as follows. Section 2 describes the basic procedure of differential evolution and introduces various adaptive and self-adaptive DE. The new algorithm, SAPA, is elaborated in Sect. 3. Simulation results are presented in Sect. 4 for the comparison of SAPA with other evolutionary algorithms. Finally, concluding remarks are summarized in Sect. 5.

2 Differential Evolution Algorithms

This section provides an overview of differential evolution and introduces necessary notations and terminologies, which facilitate a better understanding of our new algorithm, as proposed in the next section.

2.1 Differential Evolution

Differential evolution (DE) starts with a randomly initiated population of *NP* *D*-dimensional real-valued parameter vectors, which represent the candidate solutions to the actual optimization problem. After initialization, the evolutionary operations of mutation, crossover, and selection are repeatedly employed to produce the optimal solution. If $G = 0, 1, 2, \ldots, G_{max}$ denotes the subsequent generations in DE, the *i*th individual at the current generation is

$$\mathbf{X}_{i,G} = (x^1_{i,G},\ x^2_{i,G}, \ldots,\ x^j_{i,G}), \qquad j = 1, 2, \ldots, D \tag{1}$$

where D defines the dimensionality of the search space. At the first generation ($G = 0$), the population should be initialized to cover the total search space as much as possible.

2.1.1 Mutation Operation

At each generation, for each individual $\mathbf{X}_{i,G}$, called as the *target vector*, DE employs *mutation operator* to produce a *mutant vector* $\mathbf{V}_{i,G}$. Specifically, three individuals $\mathbf{X}_{r_1,G}$, $\mathbf{X}_{r_2,G}$, and $\mathbf{X}_{r_3,G}$ are randomly extracted from the population. According to the DE logic, the *mutant vector* is generated by mutation:

$$\mathbf{V}_{i,G} = \mathbf{X}_{r_1,G} + F \cdot (\mathbf{X}_{r_2,G} - \mathbf{X}_{r_3,G}) \tag{2}$$

where F is a scale factor which controls the length of the exploration vector $(\mathbf{X}_{r_2,G} - \mathbf{X}_{r_3,G})$ and thus determines how far from point $\mathbf{X}_{i,G}$ the offspring should be generated. The scalar number F typically lies in the interval [0.4, 1]. The mutation scheme given in Eq. (2) is also known as DE/rand/1. Other variants of the mutation rule have been subsequently proposed in [27] and [17]:

1. "DE/best/1"

$$\mathbf{V}_{i,G} = \mathbf{X}_{best,G} + F \cdot (\mathbf{X}_{r_1,G} - \mathbf{X}_{r_2,G}); \tag{3}$$

2. "DE/rand/2"

$$\begin{aligned}\mathbf{V}_{i,G} = {} & \mathbf{X}_{r_1,G} + F \cdot (\mathbf{X}_{r_2,G} - \mathbf{X}_{r_3,G}) \\ & + F \cdot (\mathbf{X}_{r_4,G} - \mathbf{X}_{r_5,G});\end{aligned} \tag{4}$$

3. "DE/best/2"

$$\begin{aligned}\mathbf{V}_{i,G} = {} & \mathbf{X}_{best,G} + F \cdot (\mathbf{X}_{r_1,G} - \mathbf{X}_{r_2,G}) \\ & + F \cdot (\mathbf{X}_{r_3,G} - \mathbf{X}_{r_4,G});\end{aligned} \tag{5}$$

4. "DE/current-to-best/1"

$$\begin{aligned}\mathbf{V}_{i,G} = {} & \mathbf{X}_{i,G} + F \cdot (\mathbf{X}_{best,G} - \mathbf{X}_{i,G}) \\ & + F \cdot (\mathbf{X}_{r_1,G} - \mathbf{X}_{r_2,G});\end{aligned} \tag{6}$$

5. "DE/current-to-rand/1"

$$\begin{aligned}\mathbf{U}_{i,G} = {} & \mathbf{X}_{i,G} + K \cdot (\mathbf{X}_{r_1,G} \\ & - \mathbf{X}_{i,G}) + F' \cdot (\mathbf{X}_{r_2,G} - \mathbf{X}_{r_3,G}),\end{aligned} \tag{7}$$

6. "DE/rand-to-best/2"

$$\begin{aligned}\mathbf{U}_{i,G} = {} & \mathbf{X}_{i,G} + K \cdot (\mathbf{X}_{best,G} - \mathbf{X}_{i,G}) \\ & + F' \cdot (\mathbf{X}_{r_1,G} - \mathbf{X}_{r_2,G}) \\ & + F' \cdot (\mathbf{X}_{r_3,G} - \mathbf{X}_{r_4,G}),\end{aligned} \tag{8}$$

where $\mathbf{X}_{best,G}$ specifies the vector with the best fitness among the individuals. A *difference vector* is calculated between a pair of vectors $\mathbf{X}_{r_2,G}$ and $\mathbf{X}_{r_3,G}$ to determine a search direction. In the difference vector, the indices r_1, r_2, r_3, r_4 and r_5 are mutually different integers randomly generated from the running index, which are also different from the *target vector* index i.

2.1.2 Crossover Operation

After mutation stage, a "binary" crossover operation forms the *trial vector* $\mathbf{U}_{i,G}$ according to the *target vector* $\mathbf{X}_{i,G}$ and its corresponding *mutant vector* $\mathbf{V}_{i,G}$. The *trial vector* is given by

$$u^j_{i,G} = \begin{cases} v^j_{i,G}, & \text{if } rand_{i,j}[0, 1] \le C_r \ or \ j = j_{rand} \\ x^j_{i,G}, & \text{otherwise} \end{cases} \tag{9}$$

where C_r is crossover control parameter or factor within the range [0, 1] and denotes the probability of creating parameters for a trial vector from the mutant vector. $j_{rand} \in [1, D]$ indicates a randomly selected integer, which ensures at least one dimension of the trial vector $\mathbf{U}_{i,G}$ will differ from its associated target vector $\mathbf{X}_{i,G}$. Here we have described the binary crossover operation (*bin*). The other DE crossover operation that could also be used in optimizations is exponential (*exp*).

2.1.3 Selection Operation

The selection operator selects between the target and corresponding trial vectors. The fittest vector, i.e., vector with the BEST fitness value, becomes a member of the next generation. Assuming a minimization problem, the particle with a smaller value of fitness function indicates a better performance, while the particle with a larger value of fitness function correspond to worse performance. The selection operation is described as

$$\mathbf{X}_{i,G+1} = \begin{cases} \mathbf{U}_{i,G}, & \text{if } f(\mathbf{U}_{i,G}) \leq f(\mathbf{X}_{i,G}) \\ \mathbf{X}_{i,G}. & \text{otherwise} \end{cases} \quad (10)$$

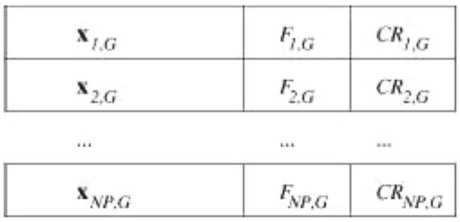
A schematic description of DE highlighting the working principles of the depicted variants is given in Algorithm 1.

Algorithm 1 The pseudo-code for the DE algorithm with DE/rand/1

```
1: Begin
2:     Initialization();
3:     while G ≤ $G_{max}$ do
4:        for $i = 1$ to $NP$ do
5:           Random select indexes $r_1 \neq r_2 \neq r_3 \neq i$
6:           $\mathbf{V}_{i,G} = \mathbf{X}_{r_1,G} + F \cdot (\mathbf{X}_{r_2,G} - \mathbf{X}_{r_3,G})$
7:           for $j = 1$ to $D$ do
8:              if $rand_{i,j}[0, 1] \leq C_r$ or $j = j_{rand}$ then
9:                 $u^j_{i,G} = v^j_{i,G}$
10:             else
11:                $u^j_{i,G} = x^j_{i,G}$
12:             end if
13:          end for
14:          if $f(\mathbf{U}_{i,G}) \leq f(\mathbf{X}_{i,G})$ then
15:             $\mathbf{X}_{i,G+1} = \mathbf{U}_{i,G}$
16:          else
17:             $\mathbf{X}_{i,G+1} = \mathbf{X}_{i,G}$
18:          end if
19:       end for
20:       G = G + 1
21:    end while
22: End
```

2.2 Adaptive DE Algorithms

Being fascinated by the potential and prospect of DE, many researchers are working on its improvement, which resulted in a wealth of variants of the basic DE algorithm. This section briefly reviews some recent adaptive and self-adaptive DE algorithms that dynamically update control parameters as the evolutionary search proceeds.



$\mathbf{x}_{1,G}$	$F_{1,G}$	$CR_{1,G}$
$\mathbf{x}_{2,G}$	$F_{2,G}$	$CR_{2,G}$
...	...	...
$\mathbf{x}_{NP,G}$	$F_{NP,G}$	$CR_{NP,G}$

Fig. 1 jDE Self-adapting: encoding aspect

2.2.1 The jDE Algorithm

Brest et al. [21] presented a new adaptive DE, jDE, which is based on the classic DE/-rand/1/bin. Similar to other schemes, jDE fixes the population size during the optimization, while adapting the control parameters F and C_r associated with each individual in Fig. 1.

The self-adaptive control parameters $F_{i,G+1}$ and $C_{r_{i,G+1}}$ are calculated as follows:

$$F_{i,G+1} = \begin{cases} F_{l,G} + rand_1 \cdot F_{u,G}, & \text{if } rand_2 < \tau_1 \\ F_{i,G}, & \text{otherwise} \end{cases} \quad (11)$$

$$C_{r_{i,G+1}} = \begin{cases} rand_3, & \text{if } rand_4 < \tau_2 \\ C_{r_{i,G}}, & \text{otherwise} \end{cases} \quad (12)$$

They produce control parameters F and C_r in a new vector. The quantities $rand_k$, $k \in \{1, 2, 3, 4\}$ represent uniform random values within the range [0, 1]. τ_1 and τ_2 are probabilities to adjust control parameters F and C_r, respectively. Constants τ_1, τ_2, $F_{l,G}$, $F_{u,G}$ are assigned fixed values to 0.1, 0.1, 0.1, 0.9, respectively. The control parameter values $F_{i,G+1}$ and $C_{r_{i,G+1}}$ are obtained before the mutation operation is performed. This means, they influence the mutation, crossover, and selection operations of the new vector.

2.2.2 The DEahcSPX Algorithm

Local search (LS) algorithms mainly explore a small neighborhood of a candidate solution in the search space until they find a locally optimal point or meet the terminal criteria. Typically in LS method, every candidate solution has more than one neighbor solution; the choice of which one to move to is taken using only information about the solutions in the neighborhood of the current one. If the choice of the neighboring solution is done by taking the one locally maximizing the criterion, the meta-heuristic is named *hillclimbing*.

In [28], Noman and Iba presented a crossover-based LS technique to adaptively adjust the length of the search (called search length) by a hill-climbing heuristic. By introducing the LS technique in original DE, they proposed an algorithm named Differential Evolution with Adaptive Hill Climbing Simplex Crossover (DEahcSPX). The algorithm took proper balance of the exploration abilities of DE and the exploitation abilities of a Local Searcher to achieve an high performance. More specifically, at each generation, the individual having best fitness value was selected and would undergo the LS process in which the simple rule of hill-climbing adaptively determines the best search length by taking feedback from the search and the best search length is added to the selected individual to generate a new individual. Owing of the simple hill-climbing mechanism, the adaptive LS does not add any additional complexity or any additional parameter to the original algorithm. The experiment results showed that the proposed new version of DE performs better, or at least comparably, than classic DE algorithm.

2.2.3 The JADE Algorithm

In JADE [20], a novel mutation strategy and an optional external archive are utilized to provide information of progress direction. This DE/current-to-*p*best strategy uses multiple best solutions to balance the greediness of the mutation and the diversity of the population, which is generated as follows:

$$\mathbf{V}_{i,G} = \mathbf{X}_{i,G} + F_i \cdot (\mathbf{X}^p_{best,G} - \mathbf{X}_{i,G}) \\ + F_i \cdot (\mathbf{X}_{r_1,G} - \tilde{\mathbf{X}}_{r_2,G}) \tag{13}$$

where $X^p_{best,G}$ is randomly selected as one of the top $100p\%$ individuals of the current population with $p \in (0, 1]$. $X_{i,G}$, $X^p_{best,G}$, and $X_{r_1,G}$ are chosen from the current population $\mathbf{P}$. $\tilde{X}_{r_2,G}$ is randomly selected from the union, $\mathbf{P} \cup \mathbf{A}$, while $\mathbf{A}$, an archive, is employed to store the recently explored inferior solutions, when compared to the current population. F_i denotes the scaling factor associated with the ith individual and it is updated dynamically in each generation as follows:

$$F_i = randc_i(\mu_F, 0.1) \tag{14}$$

$$C_{r_i} = randn_i(\mu_{C_r}, 0.1) \tag{15}$$

At each generation, F_i and C_{r_i} are, respectively, generated according to a Cauchy distribution and a Normal distribution with associated mean value μ_F and μ_{C_r}. The proposed two location parameters are initialized to be 0.5 and then updated at the end of each generation as:

$$\mu_F = (1 - c) \cdot \mu_F + c \cdot mean_L(S_F) \tag{16}$$

$$\mu_{C_r} = (1 - c) \cdot \mu_{C_r} + c \cdot mean_A(S_{C_r}) \tag{17}$$

where c is a positive constant $\in (0, 1)$; S_F and S_{C_r} denote the set of all successful mutation/crossover rates; mean$_A(\cdot)$ indicates the usual arithmetic mean, and mean$_L(\cdot)$ returns the Lehmer mean:

$$mean_L(S_F) = \frac{\sum_{i=1}^{|S_F|} F_i^2}{\sum_{i=1}^{|S_F|} F_i} \tag{18}$$

The procedure of JADE is described in Algorithm 2.

Algorithm 2 The pseudo-code for the JADE algorithm

```
1: Begin
2:   Initialization()
3:   $\mu_F = 0.5, \mu_{C_r} = 0.5, A = \emptyset$;
4:   while G ≤ G$_{max}$ do
5:     $S_F = \emptyset; S_{C_r} = \emptyset$;
6:     for $i = 1$ to $NP$ do
7:       Generate $C_{r_i} = randn_i(\mu_{C_r}, 0.1)$, $F_i = randc_i(\mu_F, 0.1)$
8:       $\mathbf{V}_{i,G} = \mathbf{X}_{i,G} + F_i \cdot (\mathbf{X}^p_{best,G} - \mathbf{X}_{i,G}) + F_i \cdot (\mathbf{X}_{r_1,G} - \tilde{\mathbf{X}}_{r_2,G})$
9:       for $j = 1$ to $D$ do
10:        if $rand_{i,j}[0, 1] \le C_r$ or $j = j_{rand}$ then
11:          $u^j_{i,G} = v^j_{i,G}$
12:        else
13:          $u^j_{i,G} = x^j_{i,G}$
14:        end if
15:      end for
16:      if $f(\mathbf{U}_{i,G}) \le f(\mathbf{X}_{i,G})$ then
17:        $\mathbf{X}_{i,G+1} = \mathbf{U}_{i,G}$;
18:        $\mathbf{X}_{i,G} \to A; C_{r_i} \to S_{C_r}, F_i \to S_F$
19:      else
20:        $\mathbf{X}_{i,G+1} = \mathbf{X}_{i,G}$
21:      end if
22:    end for
23:    Randomly remove solutions from A ($|A| \le NP$)
24:    Calculate and update $\mu_F, \mu_{C_r}$
25:    G = G + 1
26:  end while
27: End
```

2.2.4 The SaDE Algorithm

Qin et al. [18] presented the SaDE algorithm, where one trial vector generation strategy is chosen from the candidate pool ("DE/rand/1," "DE/rand/2," "DE/rand-to-best/2" and "DE/current-to-rand/1") according to the probability learned from its success rate in generating promising solutions within a certain number of previous generations, called the learning period (*LP*). More specifically, these probabilities are initially equal and then gradually self-adapted in the following manner:

$$p_{k,G} = \frac{S_{k,G}}{\sum_{k=1}^{K} S_{k,G}} \tag{19}$$

where $p_{k,G}, k = 1, 2, \ldots, K$ denotes the probability of applying the kth strategy, and K is the total number of strategies contained in the pool. $S_{k,G}$ is the success rate of the trial vector generated by the kth strategy and successfully entering the next generation:

$$S_{k,G} = \frac{\sum_{g=G-LP}^{G-1} ns_{k,g}}{\sum_{g=G-LP}^{G-1} ns_{k,g} + \sum_{g=G-LP}^{G-1} nf_{k,g}} + \varepsilon \tag{20}$$

where $ns_{k,g}$ and $nf_{k,g}$ record the number of trial vectors generated by the kth strategy that retain or discard in the selection operation in the last *LP* generations. The small constant value $\varepsilon = 0.01$ is used to avoid the possible null success rates. At each generation, for each solution in the current population, the parameters F_i and $C_{r_{i,k}}$ are independently calculated as:

$$F_i = randn_i(0.5, 0.3) \tag{21}$$

$$C_{r_{i,k}} = randn_i(CRm_k, 0.1) \tag{22}$$

where coefficients are, respectively, generated, for each individual, by sampling their values from a normal distribution. Nevertheless, CRm_k is initialized as 0.5. There are memories to store those C_r values with respect to the kth strategy which have generated trial vectors entering the next generation within the previous *LP* generations. During the first *LP* generations, CRm_k does not change. In every LP generations, CRm_k is overwritten by the median value of C_r values.

3 Self-adaptive DE with Population Adjustment Scheme

In this section, a new differential evolution algorithm is presented for balancing the local search and global search, which can improve the search performance efficiently. The novel algorithm involves two DE strategies and two population adjustment schemes:

1. Two DE strategies
   - DE/current-to-best/1 Strategy
   - DE/current-to-*p*best/1 Strategy
2. Two population size adjustment schemes
   - Population-Decreasing Strategy
   - Population-Increasing Strategy

The structure of the self-adaptive DE with population adjustment scheme (SAPA) is given in Algorithm 3. The algorithm performs the maximal number of iterations on the *NP* individuals. In each iteration, only one DE strategy is active and is applied to the mutation, crossover, and selection operations. Furthermore, a monitor is introduced to keep a track of the progress of individuals and regulate the sensitivity of population adjustment schemes. The DE strategies and population adjustment schemes are described as follow.

3.1 DE Strategies

Our proposed SAPA algorithm uses two DE strategies:

1. DE/current-to-best/1
2. DE/current-to-*p*best/1

Algorithm 3 The SAPA algorithm

```
1: Begin
2:   G=0
3:   Create a population of NP vectors randomly with $NP \in [Lbound, Ubound]$
4:   Evaluate the fitness values of the population
5:   while $G \le G_{max}$ do
6:     for $i = 1$ to $NP$ do
7:       $rn = unifrnd(0, 1)$
8:       if $rn > \varphi$ then
9:         //$\varphi$ is a time-varying variable
10:        $s = 1$;   // DE/current-to-best/1 strategy
11:      else
12:        $s = 2$;   // DE/current-to-pbest/1 strategy
13:      end if
14:      Take mutation operation on $\mathbf{X}_{i,G}$ by mutation strategy $s$
15:      Then take the crossover operation to generate trial vector ($\mathbf{U}_{i,G}$) as JADE does.
16:      if $f(\mathbf{U}_{i,G}) \le f(\mathbf{X}_{i,G})$ then
17:        $\mathbf{X}_{i,G+1} = \mathbf{U}_{i,G}, \mathbf{X}_{i,G} \rightarrow A$
18:      else
19:        $\mathbf{X}_{i,G+1} = \mathbf{X}_{i,G}$
20:      end if
21:    end for
22:    $[K_1, K_2, UM, LM]$ = Trigger Strategy Monitor()
23:    if ($K_1 == 1$) or (UM > $R$) then
24:      Population_Decreasing_Strategy()
25:      $K_1 = 0$
26:      UM = 0
27:    end if
28:    if ($K_2 == 1$) or (LM > $R$) then
29:      Population_Increasing_Strategy()
30:      Evaluate the new additional individuals
31:      $K_2 = 0$
32:      LM = 0
33:    end if
34:    G = G + 1
35:    Update $\varphi$
36:  end while
37: End
```

Specifically, DE/current-to-best/1 strategy has fast convergence speed by incorporating best solution information in the evolutionary search. A mutation vector is generated in the following manner:

$$\mathbf{V}_{i,G} = \mathbf{X}_{i,G} + F \cdot (\mathbf{X}_{best,G} - \mathbf{X}_{i,G}) + F \cdot (\mathbf{X}_{r_1,G} - \mathbf{X}_{r_2,G}) \tag{23}$$

where $\mathbf{X}_{best,G}$ specifies the vector with the best fitness among the individuals. A *difference vector* is calculated between a pair of vectors $\mathbf{X}_{r_1,G}$ and $\mathbf{X}_{r_2,G}$ to determine a search direction. In the difference vector, the indices r_1 and r_2 are different integers randomly generated from the running index, which are also different from the *target vector* index i.

On the other hand, in DE/current-to-*p*best/1 strategy, multiple best solutions are used to balance the greediness of the mutation and the diversity of the population, which is generated as follows:

$$\mathbf{V}_{i,G} = \mathbf{X}_{i,G} + F_i \cdot (\mathbf{X}^{p}_{best,G} - \mathbf{X}_{i,G}) + F_i \cdot (\mathbf{X}_{r_1^i,G} - \mathbf{X}'_{r_2^i,G}) \tag{24}$$

where $\mathbf{X}^{p}_{best,G}$ is randomly selected as one of the top $100p\%$ individuals of the current population with $p \in (0, 1]$. $\mathbf{X}_{i,G}$, $\mathbf{X}^{p}_{best,G}$ and $\mathbf{X}_{r_1^i,G}$ are chosen from the current population $\mathbf{P}$. $\mathbf{X}'_{r_2^i,G}$ is randomly selected from the union, $\mathbf{P} \cup \mathbf{A}$, while $\mathbf{A}$, an archive, is employed to store the recently explored inferior solutions.

The main idea of strategy selection is at an earlier stage of evolutionary process, DE/current-to-best/1 strategy is used to achieve fast convergence speed; with the increase of generation, to prevent strategy trapping into a local minimum, DE/current-to-*p*best/1 strategy is adopted to search a relatively large region, which is biased toward promising progress directions.

In each iteration of the evolutionary process, only one strategy is active, as presented in lines 7-11 in Algorithm 3. The DE/current-to-best/1 strategy is used when rn greater than φ; otherwise, DE/current-to-$p$best/1 strategy is adopted. The rn is a random number from the continuous uniform distribution on the interval (0,1). Notice $\varphi \in [0.1, 1]$ is a time-varying variable, which increases with every generation and can be express as:

$$\varphi = \frac{G}{G_{max}} * (\varphi_{max} - \varphi_{min}) + \varphi_{min} \tag{25}$$

where $\varphi_{min} = 0.1$, $\varphi_{max} = 1$, G denotes the generation counter. Specifically, at earlier stage, DE/current-to-best/1 strategy can be used more frequently because the random number can be easily greater than φ. On the contrary, at later stage, it is harder that the random number is greater than φ. So that DE/current-to-$p$best/1 strategy is applied more frequently with the increase of generation.

3.2 Trigger Strategy Monitor

Generally, the population size is a significant parameter for DE algorithm. An appropriate population size can greatly affect the effectiveness and efficiency of the optimization performance. However, there is no exact method to find a suitable population size for DE. On the one hand, it is obvious that if the population size is too small, the algorithm may suffer from premature convergence. On the other hand, if the population is too large, the computational cost may be too high in practice. Therefore, in this paper, adaptive population size scheme is proposed for DE to improve the efficiency of the algorithm.

In our method, the dynamic population size adjustment scheme depends on a *trigger strategy monitor*. This monitor will trigger population-decreasing or increasing strategy in accordance with the solution-searching status. Specifically, if a better solution can be found in one iteration process, it means that redundant individuals may have been existed, and population-decreasing strategy may be used according to corresponding probability. Moreover, if a better solution cannot be obtained in the evolutionary process, new individuals should be considered to add into population. Considering the typical learning situation in which the probability to obtain even better solutions decrease with the running of the algorithm, an increase of the population size is expected with the number of iterations. Furthermore, the population size is monitored to avoid breaking the upper and lower bound. The pseudo-code of the trigger strategy monitor is shown in Algorithm 4.

Algorithm 4 Trigger Strategy Monitor()

```
1: Begin
2:    //$\vartheta$ is the minimum fitness value of function evaluations in each
         previous generation
3:    //$fitbest$ is fitness value of the best solution in current genera-
         tion.
4:    if $fitbest$ is better than $\vartheta$ then
5:       if 0 < unifrnd(0,1) ≤ $P$ then
6:          $K_1 = 0$;
7:       else
8:          $K_1 = 1$;
9:       end if
10:      $\vartheta = fitbest$
11:   else
12:      if 0 < unifrnd(0,1) ≤ $Q$ then
13:         $K_2 = 0$;
14:      else
15:         $K_2 = 1$;
16:      end if
17:   end if
18:
19:   if $popsize \geq Ubound$ then
20:      UM = UM + 1
21:      LM = 0
22:   else if $popsize \leq Lbound$ then
23:      UM = 0
24:      LM = LM + 1
25:   end if
26: End
```

In Algorithm 4, *UM* and *LM* denote the upper bound monitor variable and the lower bound monitor variable. *Ubound* and *Lbound* are upper and lower bounds of the population size, respectively. The proposed K_1 and K_2 are the trigger variable, which are used to control the sensitivity of the dynamic population strategy. $P \in (0, 1]$ and $Q \in (0, 1]$ are the user-defined probabilities which determine whether the population adjustment schemes are adopted in this iteration or not.

More specifically, if there is an improvement in fitness in one generation, two optional population adjustment ways can be chosen, i.e., population maintain and population decrease method. The probability of taking maintain scheme is P, while the probability of adopting decrease scheme is $1 - P$. The trigger variable K_1 is set to 1 when unifrnd(0,1) is greater than P, that is leading to a population-decreasing scheme to remove poor individuals from current population. Furthermore, similar to above situation, if there is no improvement in fitness, population maintain and increase scheme will be used. The probability of them is Q and $1 - Q$, respectively. The population-increasing strategy is used when the trigger variable K_2 is set to 1.

Besides, when the population size is equal to the upper bound (*Ubound*) in consecutive generations, the upper bound monitor (UM) variable will be increased in each generation. This operator will be terminated when the process happens in R consecutive generations, i.e., the population-decreasing strategy will be used if UM $> R$. Alternatively, if the population size is equal to the lower bound (*Lbound*) in consecutive generations, the lower bound monitor (LM) variable will be increased in each generation. This process is continuously repeated until achieve a user-defined value R, i.e., the population-increasing strategy will be applied if LM $> R$.

3.3 Population Size Adjustment Scheme

The purpose of the adjustment is that population size can dynamically increase or decrease based on the online solution-searching status. In this paper, two population size adjustment schemes are applied:

1. Population-Decreasing Scheme
2. Population-Increasing Scheme

On the one hand, the objective of the population-decreasing scheme is to remove the redundant individuals and to save computational space. More exactly, in population-decreasing strategy, fitness function values of population are evaluated firstly. After that, every particle is arranged according to their fitness function values from small to large. The population reduction mechanism consists in deleting some individuals with worse performance from the current population. The mechanism of population reduction is presented in Algorithm 5.

Algorithm 5 Decreasing_Strategy()
```
1: Begin
2:     Evaluate the fitness values of the population
3:     Rearrange the population with its fitness function values from
           small to large
4:     $F(f(\mathbf{X}_1), ..., f(\mathbf{X}_{NP})) = (f_{min}, ..., f_{max})$
5:     $\delta_1 = \lfloor m\% \times NP \rfloor$
6:     Delete the last $\delta_1$ individuals from current population
7: End
```

In Algorithm 5, δ_1 is the number of individuals for elimination. Compared with the algorithm with small δ_1, the algorithm for population size reduction is not more computationally expensive when the value of δ_1 is very large. However, the diversity of population might be destroyed due to the too small population. Therefore, in order to keep a tradeoff between the population diversity and the search speed, we fix m on a small value, i.e., $m = 1$.

On the other hand, in order to improve the diversity of the population and to enhance local search ability, population-increasing scheme is introduced to generate better individuals. In other words, the purpose of our population-increasing scheme is to produce new individuals based on best individuals from present population to overcome the weakness of local exploration.

To begin with, the best δ_2 individuals $\mathbf{X}_{1,G}, \mathbf{X}_{2,G}, \ldots \mathbf{X}_{\delta_2,G}$ from the current population can be selected. Similar to population-decreasing strategy, $\delta_2 = \lceil m\% \times NP \rceil$ should not be too large, and coefficient m is also set to 1. After that, the idea of mutation with difference vectors in DE is applied into our population-increasing scheme, i.e., "mutation" operation is performed on each candidate best individuals. To create a new individual for every best candidate particle from current population, two other distinct parameter vector, say $\mathbf{X}_{r,G}$, $\mathbf{X}_{s,G}$ are sampled randomly from the current population. The indices r and s are mutually exclusive integers randomly chosen from the range $[1, NP]$, which are also different from the index i. The difference of these two parameter vector is scaled by a "scalar number" H and the scaled difference is added to the best chosen individual whence we obtain the new perturbation particle $\mathbf{X}_{b,G}$. We can express the process as

$$\mathbf{X}_{b,G} = \mathbf{X}_{i,G} + H \cdot (\mathbf{X}_{r,G} - \mathbf{X}_{s,G}) \quad (26)$$

The pseudocode of the population-increasing scheme is shown in Algorithm 6.

Algorithm 6 Increasing_Strategy()
```
1: Begin
2:     $\delta_2 = \lceil m\% \times NP \rceil$
3:     Select $\delta_2$ best individuals from $NP$
4:     for $i = 1$ to $\delta_2$ do
5:         Randomly generate two different vectors    $\mathbf{X}_{r,G} \neq \mathbf{X}_{s,G}$
               from $NP$
6:         $\mathbf{X}_{b,G} = \mathbf{X}_{i,G} + H \cdot (\mathbf{X}_{r,G} - \mathbf{X}_{s,G})$
7:         if $f(\mathbf{X}_{b,G}) \leq f(\mathbf{X}_{i,G})$ then
8:             Store $\mathbf{X}_{b,G}$ into BA (Best-Archive)
9:         end if
10:    end for
11:    Add all the individuals of BA into the current population
12:    Empty BA
13: End
```

The scalar number H typically lies in the interval [0.4, 1], and we use $H = 0.5$ in this paper as recommended in [21].

Moreover, the new particle from "mutation" operation should be compared with the best source individuals according to their fitness function values. Only the best ones are stored into a temporary memory, called Best-Archive (BA). Following the stage of "mutation" operation, all individuals of BA are added into the current population, and BA should be emptied at the end of every generation.

4 Experiments

4.1 Experiments Setup

The test suite that we have used for different experiments consists of 30 benchmark functions. The first ten test functions of the suite are functions commonly found in the literatures and the other benchmarks are proposed in the CEC 2005 special session on real-parameter optimization.

Our test suit is presented in Table 1, where N is the dimensionality of the problem at hand and f_{bias} is a function value of the global optimum. A more detailed description of these test functions can be found in [29]. In our test, f_{sph}, f_{ros} and f_1 to f_5 are unimodal functions; f_6 to f_{14}, and f_{ack} to f_{pn2} are multimodal functions (f_{13} and f_{14} are expanded multimodal functions); f_{15} to f_{20} are hybrid composition functions.

To evaluate the performance of the algorithms, we use the *solution error measure*, defined as $f(x) - f(x^*)$, where x is the best solution found by the algorithm in a run and x^* is the global optimum of the benchmark function. Meanwhile, the

Table 1 Benchmark functions

Functions	Name	Search space	f_{bias}
$f_{ack}(x)$	Ackley	$[-32, 32]^N$	0
$f_{grw}(x)$	Griewank	$[-600, 600]^N$	0
$f_{ras}(x)$	Rastrigin	$[-5, 5]^N$	0
$f_{sch}(x)$	Generalized Schwefel's Problem 2.26	$[-500, 500]^N$	0
$f_{sal}(x)$	Salomon	$[-100, 100]^N$	0
$f_{wht}(x)$	Whitely	$[-100, 100]^N$	0
$f_{pn1}(x)$	Generalized Penalized Function 1	$[-50, 50]^N$	0
$f_{pn2}(x)$	Generalized Penalized Function 2	$[-50, 50]^N$	0
$f_{sph}(x)$	Sphere	$[-100, 100]^N$	0
$f_{ros}(x)$	Rosenbrock	$[-100, 100]^N$	0
$f_1(x)$	Shifted Sphere	$[-100, 100]^N$	−450
$f_2(x)$	Shifted Schwefel's Problem 1.2	$[-100, 100]^N$	−450
$f_3(x)$	Shifted Rotated High Conditioned Elliptic	$[-100, 100]^N$	−450
$f_4(x)$	Shifted Schwefel's Problem 1.2 with Noise in Fitness	$[-100, 100]^N$	−450
$f_5(x)$	Schwefel's Problem 2.6 with Global Optimum on Bounds	$[-100, 100]^N$	−310
$f_6(x)$	Shifted Rosenbrock	$[-100, 100]^N$	390
$f_7(x)$	Shifted Rotated Griewank's Function without Bounds	$[0, 600]^N$	−180
$f_8(x)$	Shifted Rotated Ackley's Function with Global Optimum on Bounds	$[-32, 32]^N$	−140
$f_9(x)$	Shifted Rastrigin	$[-100, 100]^N$	−330
$f_{10}(x)$	Shifted Rotated Rastrigin	$[-5, 5]^N$	−330
$f_{11}(x)$	Shifted Rotated Weierstrass	$[-0.5, 0.5]^N$	90
$f_{12}(x)$	Schwefel's Problem 2.13	$[-\pi, \pi]^N$	−460
$f_{13}(x)$	Expanded Extended F8 plus F2	$[-3, 1]^N$	−130
$f_{14}(x)$	Shifted Rotated Expanded Scaffer's F6	$[-100, 100]^N$	−300
$f_{15}(x)$	Hybrid CF1	$[-5, 5]^N$	120
$f_{16}(x)$	Rotated Hybrid CF1	$[-5, 5]^N$	120
$f_{17}(x)$	Rotated Hybrid CF1 with Noise in Fitness	$[-5, 5]^N$	120
$f_{18}(x)$	Rotated Hybrid CF2	$[-5, 5]^N$	10
$f_{19}(x)$	Rotated Hybrid CF2 with a Narrow Basin for the Global Optimum	$[-5, 5]^N$	10
$f_{20}(x)$	Rotated Hybrid CF2 with the Global Optimum on the Bounds	$[-5, 5]^N$	10

Table 2 Experimental results of 30/50/100-dimensional problems $f_{ack}(x) - f_{20}(x)$, averaged over 30 independent runs

Dimension	$N = 30$		$N = 50$		$N = 100$	
Function	JADE Mean error ± SD	SAPA Mean error ± SD	JADE Mean error ± SD	SAPA Mean error ± SD	JADE Mean error ± SD	SAPA Mean error ± SD
$f_{ack}(x)$	2.66E−15 ± 0.00E+00 ≈	2.66E−15 ± 0.00E+00	5.74E−15 ± 1.22E−15+	**4.79E−15 ± 1.77E−15**	8.67E−01 ± 5.47E−01+	**4.90E−15 ± 2.68E−15**
$f_{grw}(x)$	0.00E+00 ± 0.00E+00 ≈	0.00E+00 ± 0.00E+00	4.92E−04 ± 2.69E−03+	**0.00E+00 ± 0.00E+00**	3.53E−03 ± 5.91E−03+	**1.72E−03 ± 4.24E−03**
$f_{ras}(x)$	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
$f_{sch}(x)$	0.00E+00 ± 0.00E+00 ≈	0.00E+00 ± 0.00E+00	3.94E+00 ± 2.16E+01+	**1.75E+00 ± 1.97E+01**	7.89E+00 ± 3.00E+01+	**4.10E+00 ± 2.09E+00**
$f_{sal}(x)$	2.03E−01 ± 1.82E−02+	**1.79E−01 ± 4.06E−02**	2.93E−01 ± 4.49E−02+	**2.09E−01 ± 3.05E−02**	7.06E−01 ± 8.27E−02+	**3.93E−01 ± 4.49E−02**
$f_{wht}(x)$	**3.06E+01 ± 2.26E+01**−	1.01E+02 ± 3.18E+01	**5.61E+01 ± 3.97E+01**−	2.87E+02 ± 1.00E+02	4.03E+02 ± 5.06E+02+	**2.50E+02 ± 2.63E+02**
$f_{pn1}(x)$	1.57E−32 ± 5.56E−48 ≈	1.57E−32 ± 5.56E−48	2.07E−03 ± 1.13E−02+	**9.42E−33 ± 2.78E−48**	4.25E−02 ± 8.59E−02+	**6.21E−03 ± 1.89E−02**
$f_{pn2}(x)$	1.34E−32 ± 5.56E−48 ≈	1.34E−32 ± 5.56E−48	1.34E−32 ± 5.56E−48 ≈	1.34E−32 ± 5.56E−48	1.90E−01 ± 5.94E−01+	**7.32E−04 ± 2.78E−03**
$f_{sph}(x)$	**1.98E−107 ± 1.08E−106**−	1.45E−69 ± 6.36E−69	**7.00E−188 ± 0.00E+00**−	1.39E−185 ± 0.00E+00	5.83E−188 ± 0.00E+00+	**1.45E−188 ± 0.00E+00**
$f_{ros}(x)$	2.65E−01 ± 1.01E+00+	**1.17E−31 ± 6.43E−31**	3.98E−01 ± 1.21E+00+	**2.65E−01 ± 1.01E+00**	1.06E+00 ± 1.79E+00+	**7.97E−01 ± 1.62E+00**
$f_1(x)$	0.00E+00 ± 0.00E+00 ≈	0.00E+00 ± 0.00E+00	0.00E+00 ± 0.00E+00 ≈	0.00E+00 ± 0.00E+00	6.56E−30 ± 1.67E−29+	**0.00E+00 ± 0.00E+00**
$f_2(x)$	1.16E−28 ± 1.15E−28+	**1.09E−29 ± 3.84E−29**	**5.38E−27 ± 5.68E−27**−	1.00E−26 ± 1.17E−26	1.90E−11 ± 9.51E−11+	**3.87E−15 ± 1.05E−14**
$f_3(x)$	8.42E+03 ± 7.26E+03+	**6.32E+03 ± 5.96E+03**	1.62E+04 ± 7.19E+03+	**1.51E+04 ± 6.34E+03**	2.22E+05 ± 6.71E+04+	**1.66E+05 ± 6.20E+04**
$f_4(x)$	1.98E−14 ± 7.65E−14+	**1.02E−27 ± 1.54E−27**	5.48E−01 ± 1.13E+00+	**1.93E−03 ± 3.21E−03**	6.55E+03 ± 2.38E+03+	**1.98E+03 ± 1.08E+03**
$f_5(x)$	8.59E−08 ± 5.23E−07+	**4.04E−09 ± 1.09E−09**	1.52E+03 ± 4.25E+02+	**6.08E+02 ± 3.80E+02**	3.81E+03 ± 8.06E+02+	**2.53E+03 ± 6.33E+02**
$f_6(x)$	1.02E+01 ± 2.96E+01+	**7.46E−01 ± 4.09E+00**	2.16E+00 ± 8.25E+00+	**1.32E−01 ± 7.27E−01**	**9.57E−01 ± 1.74E+00**−	1.59E+00 ± 1.98E+00
$f_7(x)$	8.07E−03 ± 7.42E−03+	**3.20E−03 ± 4.67E−03**	5.82E−03 ± 9.27E−03+	**3.19E−03 ± 7.30E−03**	5.12E−03 ± 7.67E−03+	**3.69E−03 ± 4.74E−03**
$f_8(x)$	2.09E+01 ± 1.68E−01 ≈	2.09E+01 ± 5.66E−02	2.12E+01 ± 2.64E−01 ≈	2.11E+01 ± 2.12E−01	2.11E+01 ± 4.28E−01 ≈	2.11E+01 ± 3.59E−01
$f_9(x)$	**0.00E+00 ± 0.00E+00**−	1.34E−11 ± 6.20E−12	**0.00E+00 ± 0.00E+00**−	4.89E−12 ± 2.60E−12	7.11E−17 ± 3.55E−16+	**5.92E−17 ± 3.24E−16**
$f_{10}(x)$	**2.44E+01 ± 6.09E+00**−	3.95E+01 ± 6.16E+00	4.67E+01 ± 8.14E+00+	**3.74E+01 ± 8.00E+00**	**1.45E+02 ± 1.45E+02**−	1.61E+02 ± 2.01E+01
$f_{11}(x)$	**2.53E+01 ± 1.65E+00**−	2.68E+01 ± 1.18E+00	5.26E+01 ± 1.76E+00 ≈	5.25E+01 ± 2.05E+00	1.27E+02 ± 3.41E+00 ≈	1.24E+02 ± 3.68E+00
$f_{12}(x)$	6.15E+03 ± 4.79E+03+	**6.01E+03 ± 5.19E+03**	1.29E+04 ± 1.32E+04+	**7.63E+03 ± 7.10E+03**	6.03E+04 ± 4.74E+04+	**5.32E+04 ± 3.95E+04**
$f_{13}(x)$	**1.49E+00 ± 1.09E−01**−	2.18E+00 ± 1.51E−01	**2.80E+00 ± 1.60E−01**−	4.12E+00 ± 2.41E−01	**6.64E+00 ± 2.52E−01**−	9.54E+00 ± 4.09E−01
$f_{14}(x)$	1.27E+01 ± 3.11E−01 ≈	1.26E+01 ± 2.17E−01	3.15E+01 ± 4.00E−01+	**2.20E+01 ± 2.65E−01**	4.55E+01 ± 5.43E−01 ≈	4.59E+01 ± 3.86E−01
$f_{15}(x)$	3.91E+02 ± 1.28E+02 ≈	3.93E+02 ± 7.39E+01	3.63E+02 ± 9.36E+01+	**3.26E+02 ± 9.80E+01**	2.36E+02 ± 4.57E+01+	**2.07E+02 ± 2.00E+01**
$f_{16}(x)$	1.01E+02 ± 1.24E+02+	**9.71E+01 ± 1.01E+02**	1.00E+02 ± 1.21E+02+	**6.89E+01 ± 3.16E+01**	5.76E+01 ± 5.44E+00 ≈	5.80E+01 ± 3.80E+00
$f_{17}(x)$	1.47E+02 ± 1.33E+02+	**1.26E+02 ± 6.42E+01**	**1.25E+02 ± 8.09E+01**−	1.39E+02 ± 3.24E+01	1.35E+02 ± 1.07E+01+	**1.21E+02 ± 1.05E+01**

Table 2 continued

Dimension	$N = 30$		$N = 50$		$N = 100$	
Function	JADE Mean error ± SD	SAPA Mean error ± SD	JADE Mean error ± SD	SAPA Mean error ± SD	JADE Mean error ± SD	SAPA Mean error ± SD
$f_{18}(x)$	9.04E+02 ± 1.03E+00 ≈	9.03E+02 ± 2.59E−01	9.22E+02 ± 6.12E+00 ≈	9.19E+02 ± 3.62E+00	1.08E+03 ± 2.27E+01 ≈	1.06E+03 ± 2.56E+01
$f_{19}(x)$	9.04E+02 ± 8.40E−01 ≈	9.03E+02 ± 2.99E−01	9.19E+02 ± 5.16E+00 ≈	9.18E+02 ± 3.13E+00	1.19E+03 ± 2.28E+01+	**1.05E+03 ± 1.78E+01**
$f_{20}(x)$	9.04E+02 ± 8.47E−01 ≈	9.03E+02 ± 6.68E−01	9.63E+02 ± 3.44E+00 +	**9.17E+02 ± 3.23E+00**	1.09E+03 ± 2.14E+01 ≈	1.06E+03 ± 2.01E+01
Total score	JADE	SAPA	JADE	SAPA	JADE	SAPA
+	11	*	17	*	20	*
−	6	*	6	*	3	*
≈	13	*	7	*	7	*

initial population is selected uniformly randomly between the defined bounds for each variable. For each algorithm and each test function, 30 independent runs are conducted with N*10,000 function evaluations (FES) as the termination criterion. N is dimensionality of the test functions domains.

In our experiments, we compared our SAPA algorithm with four state-of-the-art DE algorithms JADE, jDE, SaDE and DEahcSPX, and classic DE/$rand$/1/exp. We follow the parameter settings in original paper of jDE [21], JADE [20], SaDE [17,18], and DEahcSPX [28]. For the original DE algorithm with DE/$rand$/1/exp strategy, the parameters are set to be $F = 0.9$, $C_r = 0.9$ and P(Population size) = N(Dimension) as used in [28].

In the experiments, the strategy configuration of the SAPA algorithm is listed as follow: The *Ubound* and *Lbound* are set to 200 and 50, respectively. Initial population size is set to 100. The threshold variable of boundary R is set to 4. Considering literatures suggested, we set the parameters based on trial and error. All the experiments were performed on a computer with Core 2 2.26-GHz CPU, 2-GB memory, and Windows XP operating system.

4.2 Effect of SAPA on JADE

The results of this section are intended to show how the proposed SAPA scheme can improve the performance of JADE. In order to show the superiority of the novel presented SAPA, we compared it with JADE carrying out experiments on the test suite at dimension N = 30, 50 and 100, respectively. Table 2 reports the experimental results of the mean error and standard deviation of the solutions in 30 independent runs. The best result among those DEs is indicated by **Boldface** in the table.

Depending on the relative performance of SAPA and JADE, we divided the experiments into three classes: the low-dimensional problems ($N = 30$), the middle-dimensional problems ($N = 50$), and the high- dimensional problems ($N = 100$). Specifically, in the 30-dimensional problems, SAPA attains an equal performance in nearly half of the functions ($f_{ack} - f_{sch}$, f_{pn1}, f_{pn2}, f_1, f_8, f_{14}, f_{15}, $f_{18} - f_{20}$) and only in eleven cases achieves a significant performance improvement (f_{sal}, f_{ros}, $f_2 - f_7$, f_{12}, f_{16} and f_{17}). In general, SAPA produces the smallest errors, which indicate its ability to locate global minimizers. This is more evident for the unimodal functions $f_{ros}(x)$ and $f_1 - f_5$. This behavior also explains the ability of the proposed SAPA algorithm to improve JADE. The outstanding performance of SAPA should be due to its two greedy mutation strategies (DE/current-to-best/1, DE/current-to-$p$best/1), which leads to faster convergence than JADE. Meanwhile, SAPA still performs slightly better than JADE in multimodal functions in ranking. Especially, in hybrid composition functions $f_{15} - f_{20}$, which are much harder than others since each of

Table 3 Experimental results of 30-dimensional problems $f_{ack}(x) - f_{20}(x)$, averaged over 30 independent runs with 300,000 FES

Function	DE Mean error ± SD	DEahcSPX Mean error ± SD	jDE Mean error ± SD	SaDE Mean error ± SD	SAPA Mean error ± SD
$f_{ack}(x)$	2.00E−09 ± 2.62E−09+	2.66E−15 ± 0.00E+00 ≈	3.25E−15 ± 1.34E−15+	1.31E−01 ± 3.43E−01+	**2.66E−15 ± 0.00E+00**
$f_{grw}(x)$	2.93E−03 ± 4.52E−03+	1.96E−03 ± 4.89E−03+	0.00E+00 ± 0.00E+00≈	5.17E−03 ± 7.81E−03+	**0.00E+00 ± 0.00E+00**
$f_{ras}(x)$	1.54E+01 ± 7.69E+00+	3.10E+01 ± 1.98E+01+	0.00E+00 ± 0.00E+00≈	0.00E+00 ± 0.00E+00≈	**0.00E+00 ± 0.00E+00**
$f_{sch}(x)$	5.01E+02 ± 2.55E+02+	4.99E+02 ± 3.21E+02+	0.00E+00 ± 0.00E+00≈	0.00E+00 ± 0.00E+00≈	**0.00E+00 ± 0.00E+00**
$f_{sal}(x)$	1.92E−01 ± 3.98E−02≈	1.80E−01 ± 4.08E−02≈	1.89E−01 ± 3.05E−02≈	2.46E−01 ± 5.69E−02≈	1.79E−01 ± 4.06E−02
$f_{wht}(x)$	3.40E+02 ± 1.67E+02+	2.86E+02 ± 2.00E+02+	2.21E+01 ± 3.22E+01−	**2.00E+01 ± 1.94E+01**−	1.01E+02 ± 3.18E+01
$f_{pn1}(x)$	4.85E−02 ± 1.52E−01+	2.21E−02 ± 7.56E−02+	1.57E−32 ± 5.56E−48≈	1.57E−32 ± 5.56E−48≈	**1.57E−32 ± 5.56E−48**
$f_{pn2}(x)$	1.67E−01 ± 6.88E−01+	1.62E−31 ± 5.48E−31+	1.34E−32 ± 5.56E−48≈	1.09E−03 ± 3.35E−03+	**1.34E−32 ± 5.56E−48**
$f_{sph}(x)$	4.65E−17 ± 3.13E−17+	2.75E−31 ± 6.09E−31+	7.24E−62 ± 9.98E−62+	**3.56E−131 ± 1.17E−130**−	1.45E−69 ± 6.36E−69
$f_{ros}(x)$	1.42E+01 ± 5.76E+01+	3.89E+00 ± 1.95E+01+	2.19E+01 ± 2.40E+01+	3.32E+01 ± 3.50E+01+	**1.17E−31 ± 6.43E−31**
$f_1(x)$	3.97E−14 ± 2.51E−14+	0.00E+00 ± 0.00E+00≈	0.00E+00 ± 0.00E+00≈	0.00E+00 ± 0.00E+00≈	**0.00E+00 ± 0.00E+00**
$f_2(x)$	8.21E−02 ± 7.23E−02+	5.63E−05 ± 4.84E−05+	1.11E−06 ± 1.96E−06+	8.26E−06 ± 1.65E−05+	**1.09E−29 ± 3.84E−29**
$f_3(x)$	3.93E+06 ± 2.54E+06+	1.42E+06 ± 8.57E+05+	1.98E+05 ± 1.10E+05+	5.16E+05 ± 2.01E+05+	**6.32E+03 ± 5.96E+03**
$f_4(x)$	4.93E+01 ± 6.45E+01+	4.96E+00 ± 8.12E+00+	4.40E−02 ± 1.26E−01+	1.77E+02 ± 2.67E+02+	**1.02E−27 ± 1.54E−27**
$f_5(x)$	1.01E+03 ± 4.29E+02+	9.00E+02 ± 3.20E+02+	5.11E+02 ± 4.40E+02+	3.17E+03 ± 6.27E+02+	**4.04E−09 ± 1.09E−09**
$f_6(x)$	6.12E+01 ± 1.96E+02+	3.35E+00 ± 3.15E+00+	2.43E+01 ± 2.54E+01+	5.35E+01 ± 3.31E+01+	**7.46E−01 ± 4.09E+00**
$f_7(x)$	7.55E−03 ± 7.96E−03≈	6.59E−03 ± 4.38E−03≈	1.18E−02 ± 7.78E−03+	1.57E−02 ± 1.38E−02+	**3.20E−03 ± 4.67E−03**
$f_8(x)$	2.09E+01 ± 1.33E−01≈	2.09E+01 ± 1.12E−01≈	2.09E+01 ± 5.39E−02≈	2.09E+01 ± 4.79E−02≈	2.09E+01 ± 5.66E−02
$f_9(x)$	2.19E+01 ± 5.43E+00+	2.14E+01 ± 8.32E+00+	**0.00E+00 ± 0.00E+00**−	9.94E−02 ± 3.03E−01+	1.34E−11 ± 6.20E−12
$f_{10}(x)$	6.96E+01 ± 5.42E+01+	6.00E+01 ± 4.28E+01+	5.54E+01 ± 8.46E+00+	4.55E+01 ± 1.03E+01+	**3.95E+01 ± 6.16E+00**
$f_{11}(x)$	3.98E+01 ± 1.25E+00+	/	2.88E+01 ± 1.87E+00+	**1.92E+01 ± 2.53E+00**−	2.68E+01 ± 1.18E+00
$f_{12}(x)$	**3.80E+03 ± 3.93E+03**≈	/	9.97E+03 ± 8.53E+03+	8.20E+04 ± 1.96E+04+	6.01E+03 ± 5.19E+03
$f_{13}(x)$	3.19E+00 ± 1.18E+00+	/	**1.69E+00 ± 1.36E−01**−	3.76E+00 ± 5.42E−01+	2.18E+00 ± 1.51E−01
$f_{14}(x)$	1.28E+01 ± 4.62E−01+	/	1.29E+01 ± 2.46E−01+	1.31E+01 ± 1.69E−01+	**1.26E+01 ± 2.17E−01**
$f_{15}(x)$	4.04E+02 ± 7.41E+01≈	/	4.00E+02 ± 0.00E+00≈	**2.38E+02 ± 3.78E+01**−	3.93E+02 ± 7.39E+01
$f_{16}(x)$	1.29E+02 ± 1.53E+02≈	/	8.16E+01 ± 8.25E+00≈	**7.21E+01 ± 1.34E+01**≈	9.71E+01 ± 1.01E+02
$f_{17}(x)$	2.09E+02 ± 2.07E+02+	/	1.28E+02 ± 1.00E+01≈	**8.82E+01 ± 4.90E+01**−	1.26E+02 ± 6.42E+01
$f_{18}(x)$	9.09E+02 ± 2.82E+00+	/	9.04E+02 ± 6.16E+00≈	**8.23E+02 ± 1.63E+00**−	9.03E+02 ± 2.59E−01
$f_{19}(x)$	9.10e+02 ± 1.65e+00+	/	9.04E+02 ± 6.01E+00≈	**8.23E+02 ± 1.70E+00**−	9.03E+02 ± 2.99E−01
$f_{20}(x)$	9.07e+02 ± 1.19e+01≈	/	9.05E+02 ± 6.58E+00≈	**8.23E+02 ± 2.84E+00**−	9.03E+02 ± 6.68E−01
Total score	DE	DEahcSPX	jDE	SaDE	SAPA
+	23	15	13	15	*
−	0	0	3	8	*
≈	7	5	14	7	*

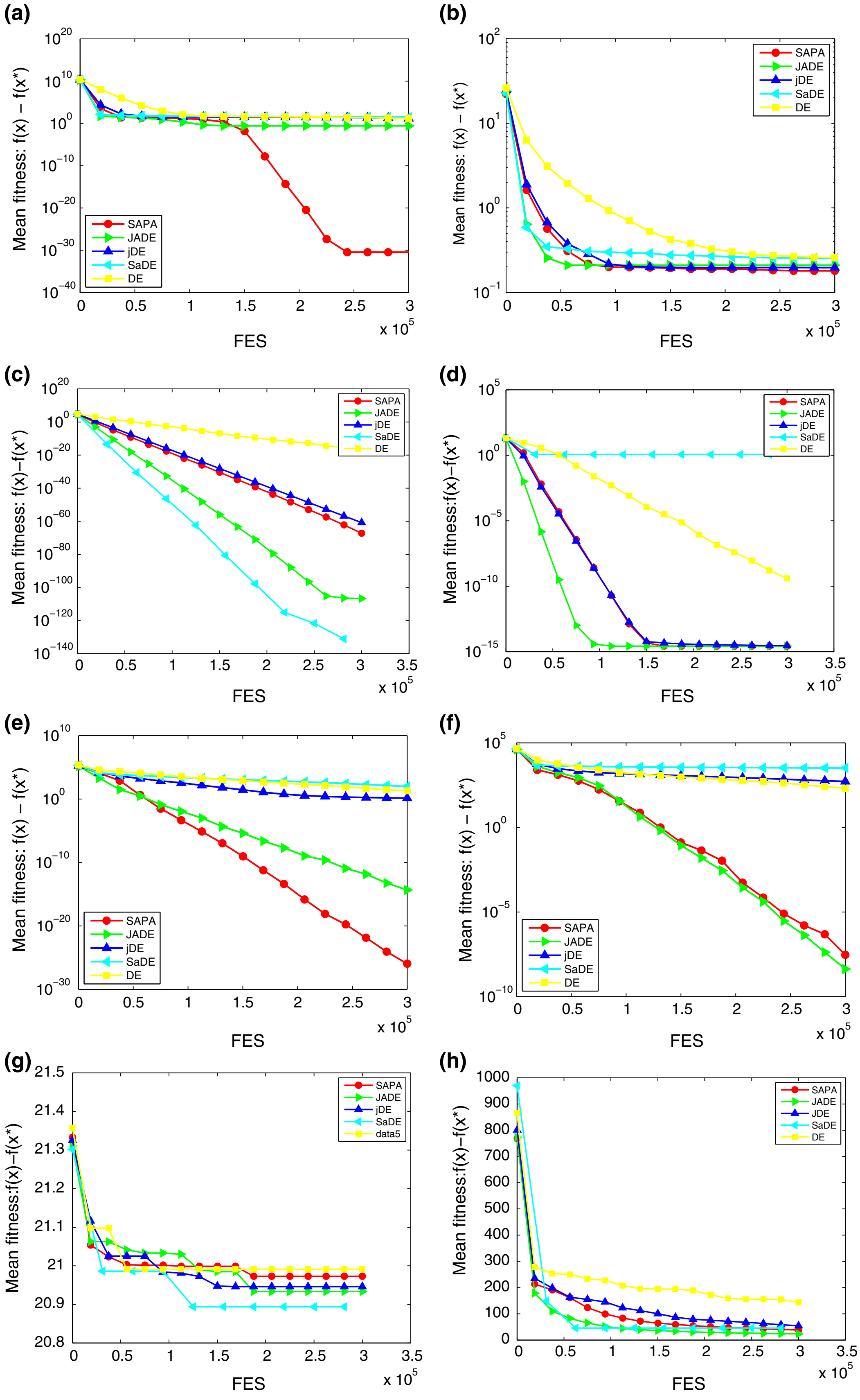


Fig. 2 Performance of the algorithms for eight 30-dimensional benchmark functions. **a** f_{ros}, **b** f_{sal}, **c** f_{sph}, **d** f_{ack}, **e** f_4, **f** f_5, **g** f_8, **h** f_{10}

them is composed of 10 sub-functions, SAPA obtains statistically significant better performance in two cases (f_{16}, f_{17}), and attains similar performance in the rest. Obviously, SAPA could balance exploration and exploitation on these benchmark functions by combining different mutation strategies with different control parameter settings.

In the cases of 50-dimensional problems, SAPA provides either substantial improvement in four unimodal, ten multimodal and three hybrid composition functions (f_{ros}, $f_3 - f_5$, f_{ack}, f_{sch}, f_{grw}, f_{sal}, f_{pn1}, f_6, f_7,f_{10}, f_{12}, f_{14} and f_{15}, f_{16}, respectively) or an equal performance (f_{ras}, f_{pn2}, f_1, f_8, f_{11}, f_{18}, f_{19}), while in six test functions the proposed SAPA deteriorates performance slightly (f_{wht}, f_{sch}, f_2, f_9, f_{13} and f_{17}). Furthermore, compare with 30- and 50-dimensional test function versions, SAPA achieves the best performance in the case of high- dimensional problems ($N = 100$). To be specific, SAPA supplies the best performance on the twenty functions, and only achieves a inferior performance on three multimodal functions. The results show that SAPA is effective and efficient even for high-dimensional problem optimization. Furthermore, on complicated or high- dimensional problems, SAPA exhibits more competitive power due to duly tuning population structure and right activating proper mutation strategy.

For a thorough comparison, the t test has been carried out to compare SAPA with its competitor algorithms in this paper, respectively. Table 2 presents the total score on every function of this two-tailed test with a significance level of 0.05 between the SAPA and JADE. Rows "+ (Better)," "= (Same)," and "− (Worse)" give the number of functions that the SAPA performs significantly better than, almost the same as, and significantly worse than the compared algorithm on fitness values in 30 runs, respectively. As confirmed in t test, the SAPA in general offers much better performance than the JADE algorithm. Moreover, the SAPA algorithm is useful for the high-dimensional problems, which can efficiently adjust the population structure and guide the evolution process toward more promising solutions. The results show that the SAPA algorithm has a very competitive performance. The convergence curves of SAPA and JADE shown in next part justify effectiveness of SAPA.

4.3 Comparison with Different DEs

In this section, we compare our SAPA algorithm with original DE algorithm and some state-of-the-art DE variants (i.e., DEahcSPX, jDE, SaDE). All the results of DEahcSPX are from [28]. The symbol "/" means no result. We first evaluate the performance of different DEs over the 30-dimensional version of our test suite. Table 3 shows the experimental results of 30-dimensional problems $f_{ack} - f_{20}$, averaged over 30 independent runs with 300,000 function evaluations (FES). Figure 2 illustrates the performance of the algorithms for four 30-dimensional benchmark functions.

From the Table 3 and Fig. 2, the SAPA algorithm provides the best performance on the $f_{ack} - f_{sal}$, f_{pn1}, f_{pn2}, $f_{ros} - f_8$, f_{10}, and f_{14}, then ranks the second on the f_{sph} and f_9, f_{11}–f_{13}, f_{15}, and $f_{17} - f_{20}$. The jDE algorithm offers the best performance on the f_9 and f_{13}. SaDE supplies the best performance on the f_{sph}, f_{wht}, f_{11}, f_{15}, and $f_{17} - f_{20}$. The algorithms can be sorted by average ranking into the following order: SAPA, jDE, SaDE, DEahcSPX, and DE. The best average ranking is obtained by the SAPA algorithm, which outperformed the other four algorithms.

More specifically, on the first seven unimodal functions (f_{sph}, f_{ros}, and $f_1 - f_5$), SaDE outperforms SAPA on function of f_{sph}, while SAPA performs better than original DE, DEahcSPX, jDE, SaDE on the rest of the unimodal functions. The outstanding performance of SAPA should be due to its dynamic population strategy, which leads to very fast convergence. On the second seventeen multimodal functions ($f_{ack} - f_{pn2}$, $f_6 - f_{14}$), SAPA achieves a significantly better performance on $f_{ack} - f_{sal}$, f_{pn1}, f_{pn2}, $f_6 - f_8$, f_{10}, and f_{14}, SaDE and jDE outperforms SAPA on test function f_{pn2} f_9, f_{11}, and f_{13}, respectively. SaDE performs best on all hybrid composition functions owing to multiple mutation strategies. Besides others cannot be better than SAPA on any test functions. Thus, SAPA is the winner on multimodal functions. This might be due to the fact that SAPA implements the adaptive population tuning scheme, which can help the algorithm to search the optimum as well as maintaining a higher convergence speed when dealing with multimodal rotated functions. It is clear that SAPA works best or second best in most cases and achieves overall better performance than other competitive algorithms.

We further evaluate the proposed SAPA on the 50-dimensional version of the set of benchmark functions. Tables 4 summarizes the experimental results of 50-dimensional problems $f_{ack} - f_{20}$, averaged over 30 independent runs with 500,000 FES. Figure 3 indicates the performance of the algorithms for four 50-dimensional benchmark functions.

From the Table 4 and Fig. 3, the SAPA provides the best performance on the $f_{ack} - f_{ras}$, f_{sal}, $f_{pn1} - f_8$, f_{10}, f_{12}, f_{14}, and $f_{18} - f_{20}$, then ranks the second on the f_{sch}, f_9, f_{11}, f_{13}, and $f_{15} - f_{17}$. jDE offers the best performance on the f_{sch}, f_{wht}, f_9, f_{13}, and f_{15}. The results show that SAPA have good ability of convergence speed.

To be specific, on the first seven unimodal functions (f_{sph}, f_{ros}, and $f_1 - f_5$), SAPA is the best among the five methods. On the second seventeen multimodal functions ($f_{ack} - f_{pn2}$, $f_6 - f_{14}$), although it worked slightly weaker on some functions, the SAPA in general offered much improved performance than all the compared DEs. It performs much better on the $f_{ack} - f_{ras}$, f_{sal}, f_{pn1}, f_{pn2}, $f_6 - f_8$, f_{10}, f_{12}, and f_{14}

Table 4 Experimental results of 50-dimensional problems $f_{ack}(x)$ – $f_{10}(x)$, averaged over 30 independent runs with 500,000 FES

Function	DE Mean error ± SD	DEahcSPX Mean error ± SD	jDE Mean error ± SD	SaDE Mean error ± SD	SAPA Mean error ± SD
$f_{ack}(x)$	2.69E−02 ± 7.63E−03+	1.23E−05 ± 8.16E−06+	5.86E−15 ± 1.08E−15+	1.18E+00 ± 5.99E−01+	**4.79E−15 ± 1.77E−15**
$f_{grw}(x)$	6.35E−02 ± 1.62E−01+	3.26E−03 ± 5.58E−03+	0.00E+00 ± 0.00E+00 ≈	1.15E−02 ± 1.94E−02+	**0.00E+00 ± 0.00E+00**
$f_{ras}(x)$	6.32E+01 ± 2.53E+01+	3.78E+01 ± 8.93E+00+	0.00E+00 ± 0.00E+00 ≈	7.20E−01 ± 9.02E−01+	**0.00E+00 ± 0.00E+00**
$f_{sch}(x)$	1.17E+03 ± 4.25E+02+	9.24E+02 ± 1.88E+02+	**1.81E−11 ± 0.00E+00**−	3.94E+00 ± 2.16E+01+	1.75E+00 ± 1.97E+01
$f_{sal}(x)$	1.78E+00 ± 2.00E−01+	6.80E−01 ± 1.40E−01+	2.26E−01 ± 4.49E−02 ≈	5.19E−01 ± 7.14E−02+	**2.09E−01 ± 3.05E−02**
$f_{wht}(x)$	1.63E+05 ± 3.89E+05+	1.61E+03 ± 1.43E+02+	**7.67E+01 ± 1.40E+02**−	2.18E+02 ± 2.20E+02−	2.87E+02 ± 1.00E+02
$f_{pn1}(x)$	2.87E−02 ± 6.83E−02+	3.12E−03 ± 1.28E−02+	9.42E−33 ± 2.78E−48 ≈	6.02E−02 ± 1.47E−01+	**9.42E−33 ± 2.78E−48**
$f_{pn2}(x)$	1.24E−01 ± 4.35E−01+	3.64E−03 ± 2.74E−03+	1.34E−32 ± 5.56E−48 ≈	1.15E−01 ± 4.04E−01+	**1.34E−32 ± 5.56E−48**
$f_{sph}(x)$	5.62E−02 ± 9.42E−02+	8.38E−09 ± 2.20E−08+	2.81E−74 ± 6.06E−74+	1.47E−118 ± 6.31E−118+	**1.39E−185 ± 0.00E+00**
$f_{ros}(x)$	1.45E+10 ± 2.63E+10+	2.03E+02 ± 2.12E+02+	3.46E+01 ± 2.52E+01+	6.46E+01 ± 4.11E+01+	**2.65E−01 ± 1.01E+00**
$f_1(x)$	1.75E−02 ± 1.24E−02+	1.16E−08 ± 1.48E−08+	0.00E+00 ± 0.00E+00 ≈	1.00E−29 ± 3.84E−29+	**0.00E+00 ± 0.00E+00**
$f_2(x)$	2.79E+04 ± 1.23E+04+	1.73E+03 ± 4.84E+02+	1.21E−02 ± 1.04E−02+	6.65E−02 ± 5.75E−02+	**1.00E−26 ± 1.17E−26**
$f_3(x)$	4.35E+08 ± 2.62E+08+	1.97E+07 ± 7.41E+06+	4.86E+05 ± 2.29E+05+	9.58E+05 ± 3.83E+05+	**1.51E+04 ± 6.34E+03**
$f_4(x)$	5.04E+04 ± 2.52E+04+	1.65E+04 ± 3.78E+03+	5.17E+02 ± 7.55E+02+	6.44E+03 ± 3.51E+03+	**1.93E−03 ± 3.21E−03**
$f_5(x)$	5.76E+03 ± 2.12E+03+	2.71E+03 ± 7.57E+02+	3.27E+03 ± 5.80E+02+	8.12E+03 ± 1.19E+03+	**6.08E+02 ± 3.80E+02**
$f_6(x)$	1.84E+03 ± 1.40E+03+	2.44E+02 ± 3.68E+02+	3.82E+01 ± 2.69E+01+	1.04E+02 ± 6.00E+01+	**1.32E−01 ± 7.27E−01**
$f_7(x)$	1.43E+00 ± 5.67E−02+	2.25E−02 ± 2.75E−02+	7.94E−03 ± 1.28E−02+	7.96E−03 ± 1.12E−02+	**3.19E−03 ± 7.30E−03**
$f_8(x)$	2.11E+01 ± 2.93E−02 ≈	2.11E+01 ± 3.63E−02 ≈	2.11E+01 ± 3.44E−02 ≈	2.11E+01 ± 3.80E−02 ≈	**2.11E+01 ± 2.12E−01**
$f_9(x)$	7.44E+01 ± 2.67E+01+	5.56E+01 ± 1.79E+01+	**0.00E+00 ± 0.00E+00**−	1.82E+00 ± 1.59E+00+	4.89E−12 ± 2.60E−12
$f_{10}(x)$	3.24E+02 ± 1.98E+01+	3.55E+02 ± 2.60E+01+	9.67E+01 ± 1.58E+01+	1.29E+02 ± 2.15E+01+	**3.74E+01 ± 8.00E+00**
$f_{11}(x)$	7.31E+01 ± 1.19E+00+	/	5.42E+01 ± 2.94E+00+	**3.95E+01 ± 4.07E+00**−	5.25E+01 ± 2.05E+00
$f_{12}(x)$	2.04E+04 ± 1.54E+04+	/	2.71E+04 ± 2.10E+04+	2.21E+04 ± 1.46E+04+	**7.63E+03 ± 7.10E+03**
$f_{13}(x)$	4.96E+00 ± 1.01E+00+	/	**3.06E+00 ± 1.93E−01**−	8.55E+00 ± 1.65E+00+	4.12E+00 ± 2.41E−01
$f_{14}(x)$	2.30E+01 ± 1.34E−01+	/	2.27E+01 ± 2.02E−01+	2.22E+01 ± 3.28E−01+	**2.20E+01 ± 2.65E−01**
$f_{15}(x)$	3.38E+02 ± 1.01E+02 ≈	/	**2.80E+02 ± 1.01E+02** ≈	3.91E+02 ± 5.49E+01−	3.26E+02 ± 9.80E+01
$f_{16}(x)$	2.25E+02 ± 7.87E+01+	/	8.12E+01 ± 1.15E+01+	9.70E+01 ± 7.44E+01+	6.89E+01 ± 3.16E+01
$f_{17}(x)$	2.89E+02 ± 4.93E+01+	/	1.77E+02 ± 2.09E+01+	**8.35E+01 ± 1.46E+01**−	1.39E+02 ± 3.24E+01
$f_{18}(x)$	9.23E+02 ± 3.82E+00+	/	9.23E+02 ± 1.98E+00+	9.85e+02 ± 1.04E+01+	**9.19E+02 ± 3.62E+00**
$f_{19}(x)$	9.21E+02 ± 4.57E+00+	/	9.18E+02 ± 3.55E+00 ≈	9.92E+02 ± 1.40E+01+	**9.18E+02 ± 3.13E+00**
$f_{20}(x)$	9.22E+02 ± 5.97E+00+	/	9.20E+02 ± 1.62E+00+	9.85E+02 ± 1.28E+01+	**9.17E+02 ± 3.23E+00**
Total score	DE	DEahcSPX	jDE	SaDE	SAPA
+	28	19	17	25	*
−	0	0	4	4	*
≈	2	1	9	1	*

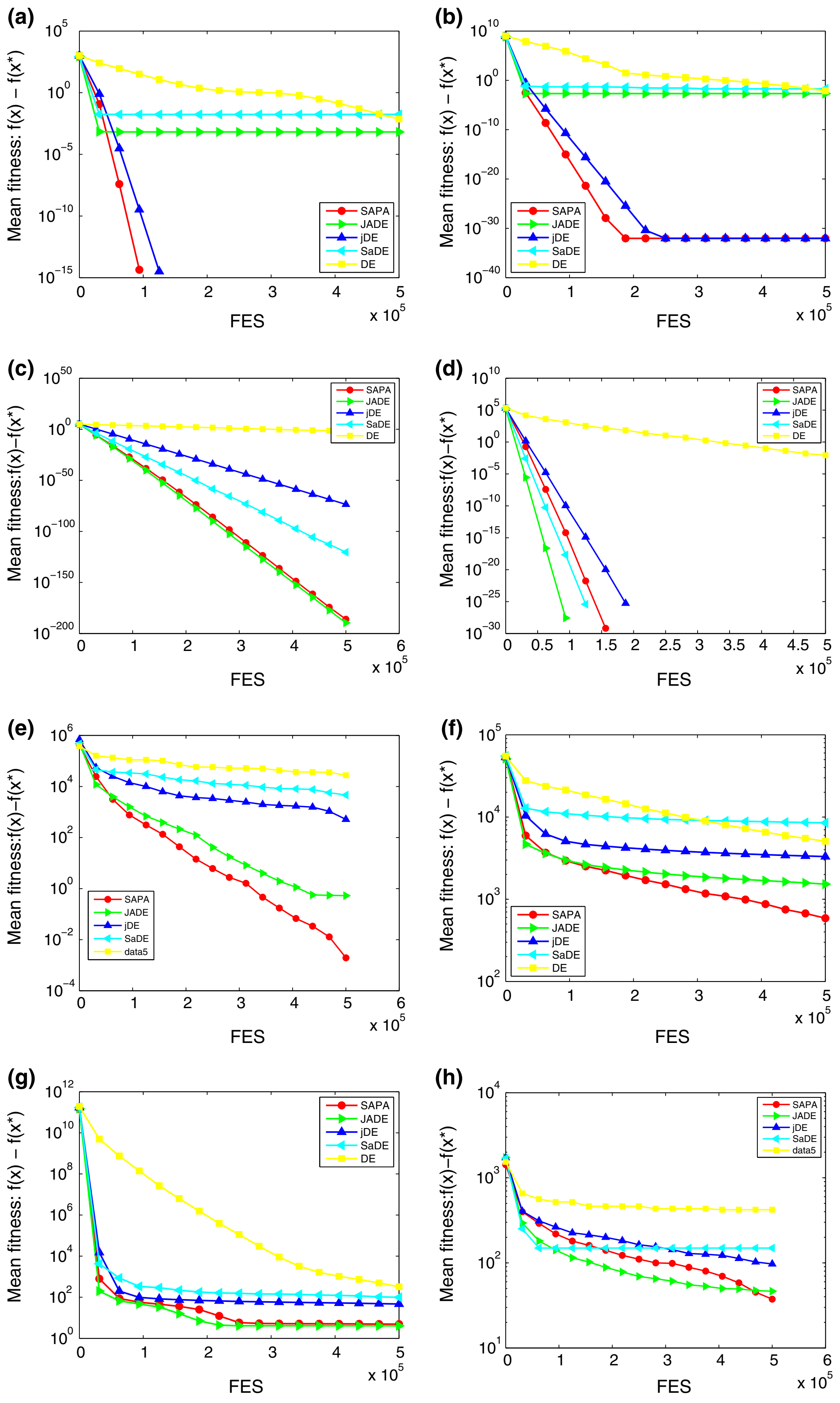


Fig. 3 Performance of the algorithms for eight 50-dimensional benchmark functions. **a** f_{grw}, **b** f_{pn1}, **c** f_{sph}, **d** f_1, **e** f_4, **f** f_5, **g** f_6, **h** f_{10}

Table 5 Experimental results of 100-Dimensional problems $f_{ack}(x) - f_{20}(x)$, averaged over 30 independent runs with 1,000,000 FES

Function	DE	DEahcSPX	jDE	SaDE	SAPA
	Mean error ± SD	Mean error ± SD	Mean error ± SD	Mean error ± SD	Mean error ± SD
$f_{ack}(x)$	7.88E+00 ± 7.16E−01+	1.43E+00 ± 2.37E−01+	1.23E−14 ± 2.45E−15+	2.76E+00 ± 4.37E−01+	**4.90E−15 ± 2.68E−15**
$f_{grw}(x)$	2.75E+01 ± 7.51E+00+	1.33E+00 ± 1.24E−01+	**0.00E+00 ± 0.00E+00**−	3.24E−02 ± 5.39E−02+	1.72E−03 ± 4.24E−03
$f_{ras}(x)$	7.23E+02 ± 5.24E+01+	3.49E+02 ± 5.45E+01+	0.00E+00 ± 0.00E+00≈	1.07E+01 ± 3.15E+00+	**0.00E+00 ± 0.00E+00**
$f_{sch}(x)$	2.50E+04 ± 2.64E+03+	2.22E+04 ± 2.94E+03+	**3.94E+00 ± 2.16E+01**≈	9.08E+01 ± 1.01E+02+	4.10E+00 ± 2.09E+00
$f_{sal}(x)$	1.09E+01 ± 6.85E−01+	2.51E+00 ± 4.21E−01+	4.69E−01 ± 4.66E−02+	1.51E+00 ± 3.18E−01+	**3.93E−01 ± 4.49E−02**
$f_{wht}(x)$	4.69E+15 ± 5.21E+15+	4.11E+10 ± 5.77E+10+	2.92E+02 ± 2.05E+02+	2.04E+03 ± 9.03E+02+	**2.50E+02 ± 2.63E+02**
$f_{pn1}(x)$	5.72E+05 ± 7.46E+05+	3.45E+00 ± 2.80E+00+	**4.71E−33 ± 1.39E−48**−	3.50E−02 ± 7.50E−02+	6.21E−03 ± 1.89E−02
$f_{pn2}(x)$	3.57E+06 ± 1.30E+06+	6.34E+01 ± 2.92E+01+	6.44E−32 ± 1.91E−33−	**2.04E−32 ± 7.22E−33**−	7.32E−04 ± 2.78E−03
$f_{sph}(x)$	3.23E+03 ± 1.77E+03+	4.61E+01 ± 7.23E+01+	1.51E−97 ± 3.10E−97+	1.28E−93 ± 3.76E−93+	**1.45E−188 ± 0.00E+00**
$f_{ros}(x)$	2.43E+08 ± 1.99E+08+	1.32E+05 ± 1.36E+05+	1.08E+02 ± 3.58E+01+	1.57E+02 ± 5.32E+01+	**7.97E−01 ± 1.62E+00**
$f_1(x)$	1.91E−27 ± 1.08E−27+	/	1.53E−29 ± 4.02E−29+	3.37E−30 ± 9.01E−30+	**0.00E+00 ± 0.00E+00**
$f_2(x)$	4.46E+03 ± 9.30E+02+	/	2.88E+01 ± 1.50E+01+	1.23E+02 ± 4.64E+01+	**3.87E−15 ± 1.05E−14**
$f_3(x)$	7.35E+06 ± 2.10E+06+	/	2.10E+06 ± 4.43E+05+	5.02E+06 ± 1.01E+06+	**1.66E+05 ± 6.20E+04**
$f_4(x)$	3.45E+04 ± 8.54E+03+	/	2.66E+04 ± 7.80E+03+	5.40E+04 ± 1.06E+04+	**1.98E+03 ± 1.08E+03**
$f_5(x)$	3.67E+03 ± 7.16E+02+	/	7.40E+03 ± 1.25E+03+	1.60E+04 ± 2.30E+03+	**2.53E+03 ± 6.33E+02**
$f_6(x)$	1.24E+02 ± 4.05E+01+	/	8.59E+01 ± 3.81E+01+	2.18E+02 ± 8.74E+01+	**1.59E+00 ± 1.98E+00**
$f_7(x)$	1.17E+04 ± 4.62E−12+	/	1.17E+04 ± 2.27E−12+	1.08E−02 ± 1.48E−02+	**3.69E−03 ± 4.74E−03**
$f_8(x)$	2.13E+01 ± 2.50E−02+	/	2.13E+01 ± 2.54E−02+	2.13E+01 ± 3.04E−02+	**2.11E+01 ± 3.59E−01**
$f_9(x)$	9.60E+01 ± 6.84E+01+	/	**0.00E+00 ± 0.00E+00**≈	1.06E+00 ± 1.17E+00+	5.92E−17 ± 3.24E−16
$f_{10}(x)$	8.21E+02 ± 2.61E+01+	/	2.04E+02 ± 2.93E+01+	2.97E+02 ± 3.09E+01+	**1.61E+02 ± 2.01E+01**
$f_{11}(x)$	1.61E+02 ± 1.72E+00+	/	1.29E+02 ± 3.65E+00+	**1.15E+02 ± 7.31E+00**−	1.24E+02 ± 3.68E+00
$f_{12}(x)$	6.30E+04 ± 1.01E+04+	/	1.28E+05 ± 1.53E+05+	4.32E+06 ± 4.09E+05+	**5.32E+04 ± 2.31E+04**
$f_{13}(x)$	6.91E+01 ± 4.12E+00+	/	**6.31E+00 ± 5.07E−01**−	3.32E+01 ± 1.87E+00+	9.54E+00 ± 4.09E−01
$f_{14}(x)$	4.76E+01 ± 1.23E−01+	/	4.66E+01 ± 4.42E−01+	4.72E+01 ± 2.53E−01+	**4.59E+01 ± 3.86E−01**
$f_{15}(x)$	2.44E+02 ± 8.71E+01+	/	**2.00E+02 ± 0.00E+00**−	4.82E+02 ± 2.38E+01+	2.07E+02 ± 2.00E+01
$f_{16}(x)$	2.45E+02 ± 6.33E+00+	/	8.20E+01 ± 8.97E+00+	8.16E+01 ± 8.07E+00+	**5.80E+01 ± 3.80E+00**
$f_{17}(x)$	2.73E+02 ± 6.33E+00+	/	1.88E+02 ± 1.02E+01+	9.09E+01 ± 9.43E+00+	**1.21E+02 ± 1.05E+01**
$f_{18}(x)$	**1.01e+03 ± 2.98e+00**−	/	1.04E+03 ± 1.72E+01−	1.11E+03 ± 1.18E+02+	1.06E+03 ± 2.56E+01
$f_{19}(x)$	**1.01E+03 ± 3.02E+00**−	/	1.04e+03 ± 1.41e+01−	1.06E+03 ± 1.50E+02+	1.05E+03 ± 1.78E+01
$f_{20}(x)$	**1.01E+03 ± 3.74E+00**−	/	1.05E+03 ± 1.95E+01≈	1.12E+03 ± 1.29E+02+	1.06E+03 ± 2.01E+01
Total score	DE	DEahcSPX	jDE	SaDE	SAPA
+	27	10	19	28	*
−	3	0	7	2	*
≈	0	0	4	0	*

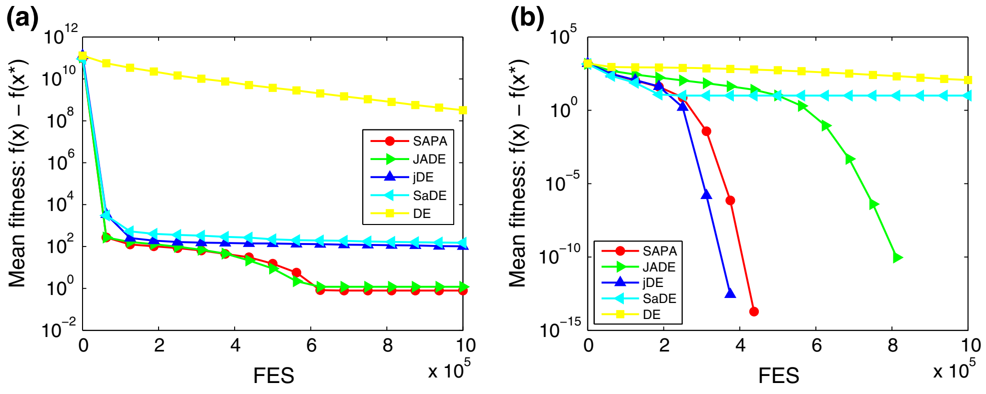


Fig. 4 Performance of the algorithms for two 100-dimensional benchmark functions. **a** f_{ros}, **b** f_{ras}

and attains slightly worse performances than the best solutions on the f_{sch}, f_{wht}, f_9, f_{11}, and f_{13}. The *t* test is also summarized in the last three rows of Table 4. In fact, SAPA performs better than DE, DEahcSPX, jDE, and SaDE on 28, 19, 17, and 25 functions, respectively. Thus, SAPA is better than other competitors in 50-dimensional problems.

Higher dimensional problems are typically harder to solve, and a common practice is to employ a larger population size. In order to test the ability of the SAPA in high-dimensional problems, we compared it with other DEs in 100-dimensional version of our test functions. In Table 5, we summarize the experimental results of 100-dimensional problems, averaged over 30 runs with 1,000,000 FES. Figure 4 illustrates the performance of the algorithms for two 100-dimensional benchmark functions.

Besides, similar to experiment in 100-dimensional problems, we also test our proposed algorithm with different DEs in 200-dimensional benchmark functions. However, some functions are up to 100 dimensionality. We test algorithms on a part of test suite. The experimental results of 200-dimensional problems $f_{ack} - f_{ros}$, averaged over 30 runs with 2,000,000 FES are summarized in Table 6. The performance graph of DEs for two 200-dimensional functions are plotted in Fig. 5.

From the Table 5 and Fig. 4, the SAPA provides the best performance on all the unimodal functions, f_{ack}, f_{ras}, f_{sal}, f_{wht}, f_6–f_8, f_{10}, f_{12}, f_{14} and f_{16}–f_{17}, and then ranks the second on the f_{grw}, f_{sch}, f_9, f_{11}, f_{13} and f_{15}. The jDE offers the best performance on the f_{grw}, f_{sch}, f_{pn1}, f_{13}, f_{15}, f_{18}, and f_{19}. SaDE achieves the best performance on the f_{pn2} and f_{11}. The results show that SAPA and jDE have good ability of convergence speed. Moreover, the results presented in Table 6 and Fig. 5 reveals that newly proposed SAPA algorithm is better than the other four DEs. More exactly, SAPA achieves a significantly better performance on unimodal functions of f_{sph}, f_{ros}, $f_2 - f_5$ and on multimodal functions of f_{grw}, f_{ras}, f_{sal}, f_{wht}, f_9 and f_{13}, then ranks the second on the f_{ack}, f_{sch}, f_{pn1}, f_{pn2}, f_1 and f_6.

In the last three rows of Table 6, *t* test is also presented. In fact, SAPA performs better than DE, DEahcSPX, jDE, and SaDE on 17, 10, 13, and 14 out of 17 test functions, respectively. Hence, SAPA exhibits the highest performance in high-dimensional problems. Comparing the results and the convergence graphs, among these DE algorithms, the jDE can converge to the best solution found so far very quickly though it is easy to get stuck in local optima. The SaDE has good global search ability and slow convergence speed. The DE and the DEahcSPX cannot perform well on all the functions. The SAPA has good local search ability and global search ability at the same time.

4.4 Comparison with Other Evolutionary Computation

This subsection provides a performance comparison between the proposed algorithm and some other hybrid GAs. Two GA models, minimal generation gap (MGG) [30] and generalized generation gap (G3) [31] have drawn much attention. Over the past few years, substantial research effort has been spent to develop more sophisticated crossover operations for GA and many outstanding schemes have been proposed, such as unimodal normal distribution crossover (UNDX) [32], simplex crossover (SPX) [33], and parent centric crossover (PCX) [31]. In relative literature, experimental results have shown that UNDX and SPX perform best with the MGG and PCX performs best with the G3 generational models. In this experiment therefore our proposed algorithm is compared with algorithms MGG+UNDX, G3+PCX, and MGG+SPX whose results are from [28]. The experimental results of 30-dimensional problems $f_{ack} - f_{10}$, averaged over 30 runs with 300,000 FES are shown in Table 7.

In our experiment, MGG model was set up with $P = 300$, $\lambda = 4$ offspring, generated from μ parents, where $\mu = 6$ is used for UNDX and $\mu = 3$ is used for SPX. For G3 model, $P = 100$, $\lambda = 2$, and $\mu = 3$ is used. From the Table 7, the MGG+SPX algorithm could not achieve the target accuracy

Table 6 Experimental results of 200-dimensional problems $f_{ack}(x) - f_{20}(x)$, averaged over 30 independent runs with 2,000,000 FES

Function	DE	DEahcSPX	jDE	SaDE	SAPA
	Mean error ± SD	Mean error ± SD	Mean error ± SD	Mean error ± SD	Mean error ± SD
$f_{ack}(x)$	1.33E+01 ± 1.43E−01+	7.78E+00 ± 2.69E−01+	**4.03E−13** ± **1.33E−12**−	5.31E+00 ± 6.64E−01+	1.79E+00 ± 2.45E−01
$f_{grw}(x)$	1.98E+03 ± 8.32E+01+	5.48E+01 ± 7.23E+00+	3.78E−02 ± 1.51E−01+	1.61E−01 ± 1.96E−01+	**5.74E−04** ± **3.14E−03**
$f_{ras}(x)$	1.37E+03 ± 5.87E+01+	1.48E+03 ± 7.63E+01+	2.94E−12 ± 1.34E−11+	5.65E+01 ± 7.54E+01+	**2.50E−13** ± **3.23E−13**
$f_{sch}(x)$	5.48E+04 ± 1.42E+03+	5.21E+04 ± 1.21E+03+	**3.94E+00** ± **2.16E+01**−	1.85E+03 ± 5.30E+02+	1.18E+01 ± 4.76E+01
$f_{sal}(x)$	2.95E+01 ± 1.26E+00+	1.34E+01 ± 3.22E−01+	1.03E+00 ± 2.07E−01+	4.24E+00 ± 8.51E−01+	**1.07E+00** ± **1.14E−01**
$f_{wht}(x)$	2.63E+18 ± 8.25E+17+	3.46E+13 ± 1.33E+13+	5.36E+03 ± 5.20E+03+	3.40E+04 ± 9.92E+03+	**4.99E+03** ± **4.85E+03**
$f_{pn1}(x)$	2.53E+08 ± 5.60E+07+	1.93E+01 ± 4.84E+00+	**1.55E−03** ± **6.25E−03**−	6.30E−02 ± 8.01E−02+	4.85E−02 ± 9.88E−02
$f_{pn2}(x)$	7.21E+08 ± 1.99E+08+	5.21E+04 ± 3.28E+04+	**8.35E−32** ± **8.86E−32**−	5.89E−01 ± 1.08E+00+	2.07E−01 ± 6.68E−01
$f_{sph}(x)$	2.00E+05 ± 3.12E+04+	6.76E+03 ± 1.44E+03+	3.48E−113 ± 1.40E−112+	2.94E−56 ± 1.60E−55+	**1.70E−145**± **6.86E−145**
$f_{ros}(x)$	2.63E+10 ± 2.84E+09+	1.32E+08 ± 1.25E+07+	2.94E+02 ± 6.54E+01+	3.71E+02 ± 7.75E+01+	**1.19E+00** ± **1.85E+00**
$f_1(x)$	5.90E+05 ± 1.70E+04+	/	5.70E+05 ± 1.56E+04+	**5.02E+05** ± **1.08E+04**−	5.15E+05 ± 1.23E+04
$f_2(x)$	2.24E+06 ± 1.83E+05+	/	2.34E+06 ± 1.85E+05+	2.04E+06 ± 1.92E+05+	**1.95E+06** ± **2.32E+05**
$f_4(x)$	2.59E+06 ± 2.64E+05+	/	2.79E+06 ± 1.75E+05+	2.46E+06 ± 1.68E+05+	**2.29E+06** ± **1.78E+05**
$f_5(x)$	1.14E+05 ± 4.19E+03+	/	1.05E+05 ± 4.18E+03+	1.09E+05 ± 3.91E+03+	**1.00E+05** ± **4.69E+03**
$f_6(x)$	2.27E+11 ± 1.21E+10+	/	2.26E+11 ± 1.37E+10+	**1.73E+11** ± **6.59E+09-**	1.80E+11 ± 7.15E+09
$f_9(x)$	3.35E+03 ± 6.78E+01+	/	3.35E+03 ± 9.67E+01+	3.11E+03 ± 3.56E+01≈	**3.10E+03** ± **3.75E+01**
$f_{13}(x)$	2.01E+02 ± 2.35E+00+	/	1.99E+02 ± 4.16E+00+	1.89E+02 ± 3.63E+00+	**1.81E+02** ± **2.24E+00**
Total score	DE	DEahcSPX	jDE	SaDE	SAPA
+	17	10	13	14	*
−	0	0	4	2	*
≈	0	0	0	1	*

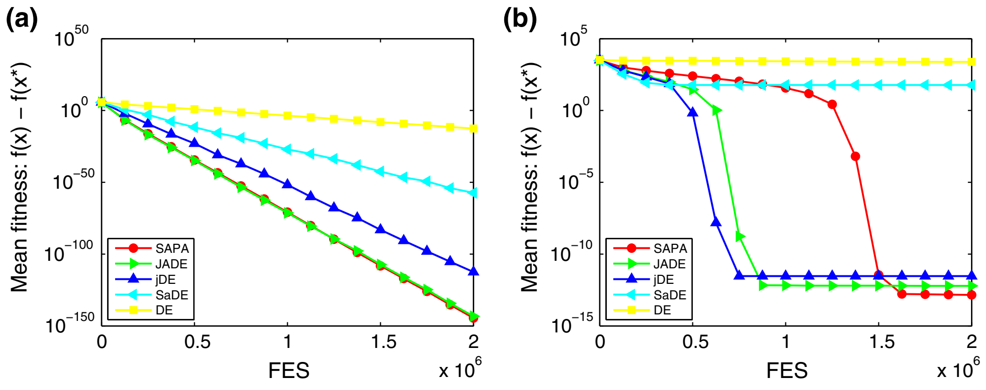


Fig. 5 Performance of the algorithms for two 200-dimensional benchmark functions. **a** f_{sph}, **b** f_{ras}

Table 7 Experimental results of 30-dimensional problems $f_{ack}(x) - f_{10}(x)$, averaged over 30 independent runs with 300,000 FES

Function	MGG+UNDX	G3+PCX	MGG+SPX	SAPA
	Mean error ± SD	Mean error ± SD	Mean error ± SD	Mean error ± SD
$f_{ack}(x)$	8.23E−07 ± 4.64E−07+	1.48E+01 ± 4.17E+00+	1.68E+00 ± 2.99E−01+	**2.66E−15 ± 0.00E+00**
$f_{grw}(x)$	2.96E−04 ± 1.48E−03+	1.07E−02 ± 1.30E−02+	1.09E+00 ± 2.24E−02+	**0.00E+00 ± 0.00E+00**
$f_{ras}(x)$	1.35E+00 ± 1.03E+00+	1.75E+02 ± 3.37E+01+	5.78E+00 ± 1.83E+00+	**0.00E+00 ± 0.00E+00**
$f_{sch}(x)$	4.12E+03 ± 1.72E+03+	4.04E+03 ± 1.09E+03+	8.70E+03 ± 2.41E+02+	**0.00E+00 ± 0.00E+00**
$f_{sal}(x)$	**1.50E−01 ± 4.95E−02−**	4.64E+00 ± 4.74E+00+	3.82E−01 ± 4.29E−02+	1.79E−01 ± 4.06E−02
$f_{wht}(x)$	4.28E+02 ± 3.82E+01+	7.90E+02 ± 1.27E+02+	3.28E+03 ± 2.77E+03+	**1.01E+02 ± 3.18E+01**
$f_{pn1}(x)$	4.93E−02 ± 3.50E−02+	4.35E+00 ± 6.94E+00+	2.57E−01 ± 6.90E−02+	**1.57E−32 ± 5.56E−48**
$f_{pn2}(x)$	4.39E−04 ± 2.20E−03+	1.50E+01 ± 1.58E+01+	2.29E+00 ± 3.72E−01+	**1.34E−32 ± 5.56E−48**
$f_{sph}(x)$	1.37E−11 ± 1.94E−11+	**3.58E−81 ± 1.36E−81**−	8.75E+00 ± 2.87E+00+	1.45E−69 ± 6.36E−69
$f_{ros}(x)$	2.81E+01 ± 1.23E+01+	4.18E+00 ± 9.68E+01+	1.38E+03 ± 6.45E+02+	**1.17E−31 ± 6.43E−31**
$f_1(x)$	2.83E−11 ± 3.33E−11+	3.52E−13 ± 1.22E−13+	4.71E+04 ± 4.21E+03+	**0.00E+00 ± 0.00E+00**
$f_2(x)$	1.41E+00 ± 7.15E−01+	4.14E−12 ± 1.21E−12+	3.96E+04 ± 3.89E+03+	**1.09E−29 ± 3.84E−29**
$f_3(x)$	8.76E+05 ± 2.98E+05+	**1.07E+03 ± 1.29E+03**−	7.16E+08 ± 1.34E+08+	6.32E+03 ± 5.96E+03
$f_4(x)$	5.01E+01 ± 3.62E+01+	9.35E+04 ± 2.66E+04+	4.45E+04 ± 3.73E+03+	**1.02E−27 ± 1.54E−27**
$f_5(x)$	1.67E+03 ± 6.01E+02+	8.13E+03 ± 2.65E+03+	3.34E+04 ± 2.11E+03+	**4.04E−09 ± 1.09E−09**
$f_6(x)$	1.79E+02 ± 2.38E+02+	1.34E+02 ± 2.48E+02+	1.56E+10 ± 1.47E+09+	**7.46E−01 ± 4.09E+00**
$f_7(x)$	7.26E−03 ± 8.19E−03+	2.01E−02 ± 1.85E−02+	1.02E+04 ± 4.71E+02+	**3.20E−03 ± 4.67E−03**
$f_8(x)$	2.09E+01 ± 5.62E−02 ≈	2.11E+01 ± 6.67E−12 ≈	2.10E+01 ± 4.06E−02 ≈	2.09E+01 ± 5.66E−02
$f_9(x)$	4.65E+01 ± 5.41E+01+	2.44E+02 ± 3.98E+01+	3.15E+02 ± 1.04E+01+	**1.34E−11 ± 6.20E−12**
$f_{10}(x)$	4.76E+01 ± 5.03E+01+	3.89E+02 ± 9.96E+01+	5.31E+02 ± 2.85E+01+	**3.95E+01 ± 6.16E+00**
Total score	MGG+UNDX	G3+PCX	MGG+SPX	SAPA
+	18	17	19	*
−	1	2	0	*
≈	1	1	1	*

levels for any test function. MGG+UNDX presents the best performance on the f_{sal}. The G3+PCX algorithm achieved a slightly better error average for some functions (f_{sph} and f_3) but is outperformed by SAPA for the other functions. So in general, it can be concluded from the *t* test in Table 7 that our SAPA algorithm exhibits overall better performance than the other algorithms in the table.

Table 8 Effects of DE strategies selection on 30-dimensional problems $f_1(x) - f_{20}(x)$, averaged over 30 independent runs

Function	SAPA	Best	Pbest	Pbb
	Mean error ± SD	Mean error ± SD	Mean error ± SD	Mean error ± SD
$f_1(x)$	**0.00E+00 ± 0.00E+00**	1.12E+03 ± 3.18E+02+	**0.00E+00 ± 0.00E+00**≈	1.08E+03 ± 4.52E+02+
$f_2(x)$	**1.09E−29 ± 3.84E−29**	5.38E+03 ± 1.10E+03+	2.90E−29 ± 3.02E−29+	4.44E+03 ± 1.14E+03+
$f_3(x)$	**6.32E+03 ± 5.96e+03**	2.72E+07 ± 1.93E+07+	7.08E+03 ± 3.75E+03≈	2.98E+07 ± 1.97E+07+
$f_4(x)$	**1.02E−27 ± 1.54E−27**	8.75E+03 ± 1.71E+03+	1.75E−22 ± 3.61E−22+	8.67E+03 ± 2.17E+03+
$f_5(x)$	**4.04E−09 ± 1.09E−09**	6.57E+03 ± 7.48E+02+	9.97E−09 ± 1.51E−08+	6.71E+03 ± 1.08E+03+
$f_6(x)$	**7.46E−01 ± 4.09E+00**	1.17E+10 ± 4.03E+09+	6.85E+00 ± 1.45E+01+	9.44E+09 ± 3.25E+09+
$f_7(x)$	**3.20E−03 ± 4.67E−03**	9.47E+02 ± 5.69E+02+	3.40E+03 ± 9.70E+02+	8.32E+02 ± 6.14E+02+
$f_8(x)$	**2.09E+01 ± 5.66E−02**	2.10E+01 ± 3.51E−02 ≈	**2.09E+01 ± 4.11E−02** ≈	2.09E+01 ± 4.88E−02 ≈
$f_9(x)$	**1.34E−11 ± 6.20E−12**	7.71E+01 ± 1.30E+01+	1.50E−11 ± 9.94E−12≈	7.94E+01 ± 1.50E+01+
$f_{10}(x)$	**3.95E+01 ± 6.16E+00**	2.08E+02 ± 3.03E+01+	4.12E+01 ± 1.32E+00+	1.95E+02 ± 2.24E+01+
$f_{11}(x)$	**2.68E+01 ± 1.18E+00**	**2.68E+01 ± 1.28E+00**≈	2.91E+01 ± 1.22E+00 +	2.66E+01 ± 2.10E+00+
$f_{12}(x)$	**6.01E+03 ± 5.19E+03**	5.59E+04 ± 1.11E+04+	6.87E+03 ± 6.06E+03≈	5.50E+04 ± 1.39E+04+
$f_{13}(x)$	**2.18E+00 ± 1.51E−01**	2.93E+00 ± 5.22E−01+	2.34E+00 ± 1.05E−01+	2.95E+00 ± 6.23E−01+
$f_{14}(x)$	**1.26E+01 ± 2.17E−01**	1.27E+01 ± 2.24E−01 ≈	**1.26E+01 ± 1.71E−01**≈	1.28E+01 ± 2.18E−01+
$f_{15}(x)$	3.93E+02 ± 7.39E+01	**3.06E+02 ± 1.76E+02**−	3.88E+02 ± 7.00E+01 ≈	2.91E+02 ± 1.71E+02–
$f_{16}(x)$	9.71E+01 ± 1.01E+02	2.48E+02 ± 3.45E+01+	**8.32E+01 ± 6.58E+01** ≈	3.53E+02 ± 6.06E+01+
$f_{17}(x)$	**1.26E+02 ± 6.42E+01**	3.10E+02 ± 3.26E+01+	1.30E+02 ± 8.01E+01 ≈	4.34E+02 ± 6.55E+01+
$f_{18}(x)$	9.03E+02 ± 2.59E−01	**8.50E+02 ± 1.04E+01**−	9.04E+02 ± 6.33E−01+	8.55E+02 ± 9.46E+00−
$f_{19}(x)$	9.03E+02 ± 2.99E−01	**8.53E+02 ± 9.53E+00**−	9.04E+02 ± 2.74E−01+	8.47E+02 ± 7.78E+00−
$f_{20}(x)$	9.03E+02 ± 6.68E−01	**8.52E+02 ± 1.17E+01**−	9.04E+02 ± 2.72E−01+	8.46E+02 ± 8.92E+00−
Total score				
+	*	13	11	15
−	*	4	0	4
≈	*	3	9	1

4.5 Effect of the DE Strategies Selection

In the SAPA algorithm, two DE strategies are chosen as the mutation strategies. This section intends to show the effect of the DE strategies selection. We test three SAPA variants which just adopt one mutation strategy to generate individuals or swap the sequence of mutation strategies. The legend "Best" denotes SAPA with DE/current-to-best/1 strategy, while "Pbest" denotes SAPA with DE/current-to-pbest/1 strategy. "Pbb" denotes SAPA variant with the opposed of DE strategies selection. The results are shown in Table 8. Figure 6 presents the convergence characteristics of the three algorithms on 30-dimensional problems, f_2, f_6, f_9 and f_{15}.

As we can see, SAPA performs best among the four algorithms. Specially, the convergence curves of SAPA and Pbest are close together, and so do Pbb and Best. It seems that the one used more frequently in later evolution stage decides the algorithm's convergence character. SAPA performs best on 15 out of 20 functions. Hence, DE strategies selection works well.

4.6 Performance of Trigger Monitor

In our proposed SAPA algorithm, a *trigger strategy monitor* is introduced to control the population strategy in accordance with the solution-searching status. The status monitor is described in Algorithm 4. In all the experiments, threshold R is set to 4 and P and Q are analyzed in the following. Moreover, the parameter m is also considered, which denotes the number of potential candidates for elimination or perturbation in Algorithms 5–6. An example of the change in population size for two 30-dimensional benchmark functions is shown in Fig. 7 in which the population size varies as iterations increases.

In this paper, P and Q are user-defined parameters, which determine whether the population adjustment schemes

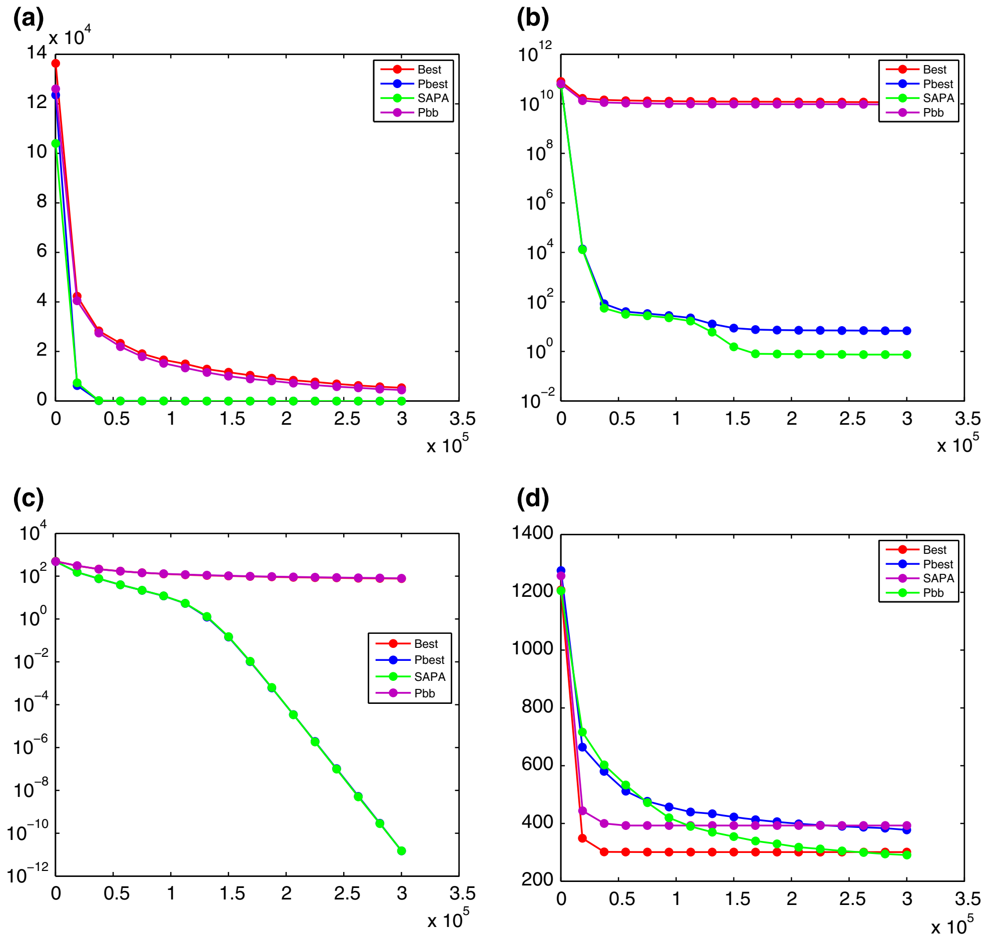


Fig. 6 Effect of the DE strategies selection for four 30-dimensional benchmark functions. **a** f_2, **b** f_6, **c** f_9, **d** f_{15}

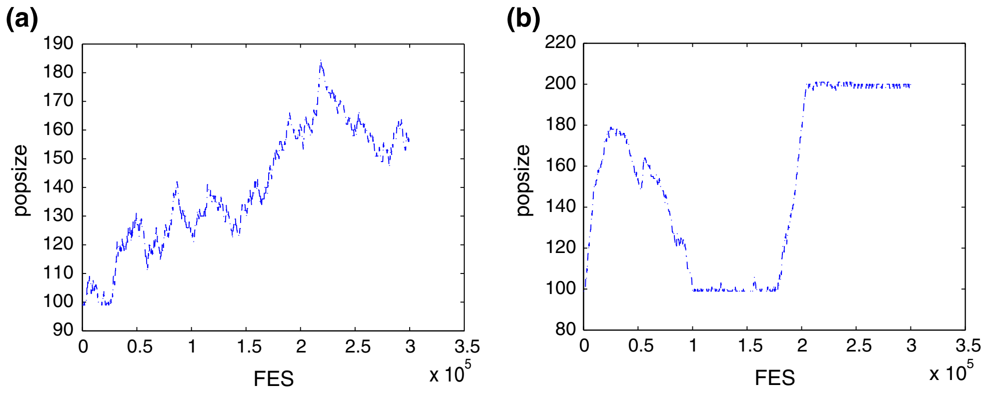


Fig. 7 Example of the change in population size for two 30-dimensional benchmark functions. **a** f_{sph}, **b** f_{12}

are adopted in this iteration or not. The range of their values is (0,1]. Once P and Q are set as one, the population size will be maintained in each generation. Setting big P and Q values will result in a low sensitivity of the SAPA, while small P and Q values will result in a higher efficiency of the population adjustment. On the other hand, coefficient m also influences the perturbation or elimination process substantially.

Table 9 shows the comparisons between SAPA with other three parameter settings of SAPA over 30-dimensional version of the 15 benchmark functions. It indicates that SAPA is still sensitive to adjustment of parameters. But, from Tables 3

Table 9 Effects of parameters on search accuracy of SAPA

Function	$P = 0.6,\ Q = 0.6, m = 1$ Mean error ± SD	$P = 0.8,\ Q = 0.8, m = 1$ Mean error ± SD	$P = 0.8, Q = 0.4, m = 5$ Mean error ± SD	$P = 0.4, Q = 0.8, m = 6$ Mean error ± SD
$f_1(x)$	0.00E+00 ± 0.00E+00	0.00E+00 ± 0.00E+00 ≈	0.00E+00 ± 0.00E+00 ≈	0.00E+00 ± 0.00E+00 ≈
$f_2(x)$	**1.09E−29 ± 3.84E−29**	1.16E−28 ± 1.26E−28+	2.32E−29 ± 4.51E−29+	2.11E−28 ± 1.36E−28+
$f_3(x)$	6.32E+03 ± 5.96E+03	7.49E+03 ± 6.76E+03+	**4.67E+03 ± 1.26E+03**−	5.11E+03 ± 2.31E+03−
$f_4(x)$	1.02E−27 ± 1.54E−27	**9.65E−28 ± 1.12E−28**−	2.63E−25 ± 9.82E−25+	1.98E−22 ± 1.66E−22+
$f_5(x)$	**4.04E−09 ± 1.09E−09**	8.56E−08 ± 5.52E−08+	9.80E−04 ± 1.52E−03+	8.32E−02 ± 1.98E−01+
$f_6(x)$	7.46E−01 ± 4.09E+00	**1.02E−02 ± 3.52E+00**−	1.26E+01 ± 3.41E+01+	1.29E+01 ± 3.42E+01+
$f_7(x)$	**3.20E−03 ± 4.67E−03**	6.46E−03 ± 3.64E−03+	4.26E−02 ± 1.26E−02+	7.96E−01 ± 2.85E−01+
$f_8(x)$	2.09E+01 ± 5.66E−02	2.09E+01 ± 1.68E−02 ≈	2.09E+01 ± 3.25E−02 ≈	2.09E+01 ± 4.32E−02 ≈
$f_9(x)$	**1.34E−11 ± 6.20E−12**	2.74E−10 ± 1.37E−10+	5.63E−10 ± 4.26E−10+	5.41E−10 ± 1.54E−11+
$f_{10}(x)$	3.95E+01 ± 6.16E+00	**3.32E+00 ± 1.55E+01**−	6.59E+01 ± 1.45E+01+	5.48E+01 ± 1.57E+01+
$f_{11}(x)$	2.68E+01 ± 1.18E+00	2.96E+01 ± 3.68E+00+	3.01E+01 ± 2.40E+00+	**1.62E+01 ± 1.03E+00**−
$f_{12}(x)$	6.01E+03 ± 5.19E+03	5.99E+03 ± 4.95E+03 ≈	**5.26E+03 ± 5.43E+03**−	6.36E+03 ± 2.39E+03+
$f_{13}(x)$	2.18E+00 ± 1.51E−01	**1.02E+00 ± 2.14E−01**−	2.11E+00 ± 2.23E−01 ≈	3.42E+00 ± 1.40E−01+
$f_{14}(x)$	**1.26E+01 ± 2.17E−01**	1.64E+01 ± 4.27E−01+	1.23E+01 ± 3.20E−01 ≈	1.02E+02 ± 3.24E−01+
$f_{15}(x)$	3.93E+02 ± 7.39E+01	4.02E+02 ± 9.57E+01+	3.89E+02 ± 7.61E+01 ≈	**3.31E+02 ± 1.52E+02**−

and 9, SAPA is still better or comparable with other comparison algorithms. In order to make a balance of the search accuracy and robustness, $P = 0.6$, $Q = 0.6$, and $m = 1$ are used as a representative parameter setting in our paper. This will prevent the instant elimination of a newborn individual and keep the status monitor high sensitivity.

4.7 Effect of the Population Size Adjustment Scheme

The population size adjustment scheme (PSA) is the crucial part of SAPA. In this section, in order to show the effect of PSA, we present the test results on the CEC benchmark functions at dimension 30, 50 and 100. In Table 10, the "without PSA" indicates the SAPA algorithm without the population size adjustment scheme. From Table 10, SAPA achieves better optimal solutions in absolute size on most functions. Meanwhile, in general, it can be concluded from the statistical analysis results that SAPA's performance is superior to that of the SAPA algorithm without PSA, which means PSA is effective to improve the algorithm's performance.

Specially, in the two expanded multimodal functions (f_{13}, f_{14}), the benefit from the PSA scheme is not evident. However, in $f_{15} - f_{20}$, the hybrid composition functions with a huge number of local minima, SAPA performs slightly better than the SAPA algorithm without PSA. Among the five unimodal functions ($f_1 - f_5$), SAPA obtains significantly better results on four 30-dimension, two 50-dimension and all five 100-dimension version functions. This behavior indicates the ability of PSA to improve SAPA to locate global minimizers. As the dimensionality increases, the benefit from PSA scheme seems to be apparent. In general, PSA keep a good balance between global search and local search, by which SAPA is substantially enhanced.

5 Conclusion

This paper proposed a SAPA algorithm. This algorithm was evaluated on a benchmark suite consisting of 10 functions carefully chosen from the literatures and 20 numerical optimization problems used in CEC2005 special session on real-parameter optimization.

The algorithm uses two DE's strategies, two newly proposed population strategy, and a method of trigger strategy monitor. To save computational space and time, the population-decreasing strategy is introduced to stochastically remove redundant individuals from population, while to improve the diversity of the population, the population-increasing strategy is applied to generate better individuals. Those population size adjustment schemes depend on a trigger strategy monitor, which controls the sensitivity of the dynamic population strategy.

The obtained results and statistical analysis give evidence that the SAPA algorithm is a highly competitive algorithm in 30, 50, 100, and 200-dimensional problems. To summarize the results of the tests, the SAPA algorithm presents significantly better results than the remaining algorithms, in most cases. In our future work, we will investigate the DE strategies selection further and study how to adaptively tune parameters P, Q and φ. Moreover, we will apply the pro-

Table 10 Effects of population size adjustment scheme on 30-Dimensional, 50-Dimensional, and 100-Dimensional Problems $f_1(x) - f_{20}(x)$, averaged over 30 independent runs

Function	30-Dimensional		50-Dimensional	
	SAPA Mean error ± SD	Without PSA Mean error ± SD	SAPA Mean error ± SD	Without PSA Mean error ± SD
$f_1(x)$	**0.00E+00 ± 0.00E+00**	**0.0E+00 ± 0.00E+00**≈	**0.00E+00 ± 0.00E+00**	**0.00E+00 ± 0.00E+00**≈
$f_2(x)$	**1.09E−29 ± 3.84E−29**	1.20E−28 ± 1.04E−28+	1.00E−26 ± 1.17E−26	**3.71E−27 ± 2.45E−27**−
$f_3(x)$	**6.32E+03 ± 5.96e+03**	9.21E+03 ± 7.28E+03+	**1.51E+04 ± 6.34E+03**	1.55E+04 ± 8.65E+03≈
$f_4(x)$	**1.02E−27 ± 1.54E−27**	3.05E−16 ± 7.62E−16+	**1.93E−03 ± 3.21E−03**	4.90E−01 ± 9.20E−01+
$f_5(x)$	**4.04E−09 ± 1.09E−09**	6.02E−09 ± 1.46E−08+	**6.08E+02 ± 3.80E+02**	1.62E+03 ± 4.90E+02+
$f_6(x)$	**7.46E−01 ± 4.09E+00**	5.56E+00 ± 1.73E+01+	**1.32E−01 ± 7.27E−01**	7.20E+00 ± 3.72E+01+
$f_7(x)$	**3.20E−03 ± 4.67E−03**	4.70E+03 ± 2.71E−12+	**3.19E−03 ± 7.30E−03**	6.20E+03 ± 2.93E−12+
$f_8(x)$	**2.09E+01 ± 5.66E−02**	**2.09E+01 ± 2.23E−01**≈	**2.11E+01 ± 2.12E−01**	**2.11E+01 ± 4.86E−02**≈
$f_9(x)$	1.34E−11 ± 6.20E−12	**0.00E+00 ± 0.00E+00**−	4.89E−12 ± 2.60E−12	**0.00E+00 ± 0.00E+00**−
$f_{10}(x)$	3.95E+01 ± 6.16E+00	**2.44E+01 ± 5.26E+00**−	**3.74E+01 ± 8.00E+00**	4.81E+01 ± 9.71E+00+
$f_{11}(x)$	2.68E+01 ± 1.18E+00	**2.55E+01 ± 1.63E+00**−	5.25E+01 ± 2.05E+00	**5.19E+01 ± 2.23E+00**≈
$f_{12}(x)$	**6.01E+03 ± 5.19E+03**	7.72E+03 ± 4.64E+03+	**7.63E+03 ± 7.10E+03**	1.33E+04 ± 9.12E+03+
$f_{13}(x)$	2.18E+00 ± 1.51E−01	**1.45E+00 ± 8.64E−02**−	4.12E+00 ± 2.41E−01	**2.77E+00 ± 2.06E−01**−
$f_{14}(x)$	1.26E+01 ± 2.17E−01	**1.23E+01 ± 3.50E−01**≈	2.20E+01 ± 2.65E−01	**2.18E+01 ± 3.70E−01**≈
$f_{15}(x)$	3.93E+02 ± 7.39E+01	**3.73E+02 ± 1.17E+02**≈	3.26E+02 ± 9.80E+01	**2.93E+02 ± 9.81E+01**≈
$f_{16}(x)$	**9.71E+01 ± 1.01E+02**	1.15E+02 ± 1.39E+02≈	**6.89E+01 ± 3.16E+01**	1.03E+02 ± 1.22E+02+
$f_{17}(x)$	**1.26E+02 ± 6.42E+01**	1.44E+02 ± 1.40E+02≈	1.39E+02 ± 3.24E+01	**1.34E+02 ± 9.55E+01**≈
$f_{18}(x)$	**9.03E+02 ± 2.59E−01**	9.04E+02 ± 3.67E−01≈	**9.19E+02 ± 3.62E+00**	9.20E+02 ± 5.32E+00≈
$f_{19}(x)$	**9.03E+02 ± 2.99E−01**	9.04E+02 ± 6.60E−01≈	**9.18E+02 ± 3.13E+00**	9.23E+02 ± 4.27E+00≈
$f_{20}(x)$	**9.03E+02 ± 6.68E−01**	9.04E+02 ± 7.15E−01≈	**9.17E+02 ± 3.23E+00**	**9.17E+02 ± 2.50E+00**≈
Total score				
+	*	7	*	7
−	*	4	*	3
≈	*	9	*	10

Function	100-Dimensional	
	SAPA Mean error ± SD	Without PSA Mean error ± SD
$f_1(x)$	**0.00E+00 ± 0.00E+00**	5.15E−30 ± 1.54E−29+
$f_2(x)$	**3.87E−15 ± 1.05E−14**	1.59E−12 ± 7.89E−12+
$f_3(x)$	**1.66E+05 ± 6.20E+04**	2.41E+05 ± 6.91E+04+
$f_4(x)$	**1.98E+03 ± 1.08E+03**	6.56E+03 ± 2.98E+03+
$f_5(x)$	**2.53E+03 ± 6.33E+02**	4.27E+03 ± 8.41E+02+
$f_6(x)$	**1.59E+00 ± 1.98E+00**	1.86E+00 ± 2.02E+00≈
$f_7(x)$	**3.69E−03 ± 4.74E−03**	1.17E+04 ± 5.29E−12+
$f_8(x)$	**2.11E+01 ± 3.59E−01**	2.12E+01 ± 3.69E−01≈
$f_9(x)$	**5.92E−17 ± 3.24E−16**	1.18E−16 ± 4.51E−16+
$f_{10}(x)$	1.61E+02 ± 2.01E+01	**1.49E+02 ± 1.52E+01**−
$f_{11}(x)$	**1.24E+02 ± 3.68E+00**	1.27E+02 ± 4.74E+00≈
$f_{12}(x)$	5.32E+04 ± 3.95E+04	**5.28E+04 ± 4.34E+04**≈
$f_{13}(x)$	9.54E+00 ± 4.09E−01	**6.65E+00 ± 3.28E−01**−
$f_{14}(x)$	4.59E+01 ± 3.86E−01	**4.55E+01 ± 5.41E−01**≈
$f_{15}(x)$	**2.07E+02 ± 2.00E+01**	2.57E+02 ± 1.01E+02+
$f_{16}(x)$	**5.80E+01 ± 3.80E+00**	5.82E+01 ± 2.27E+00≈

Table 10 continued

$f_{17}(x)$	**1.21E+02 ± 1.05E+01**	1.29E+02 ± 1.13E+01+
$f_{18}(x)$	**1.06E+03 ± 2.56E+01**	1.08E+03 ± 2.61E+01≈
$f_{19}(x)$	**1.05E+03 ± 1.78E+01**	1.08E+03 ± 3.57E+01≈
$f_{20}(x)$	**1.06E+03 ± 2.01E+01**	1.08E+03 ± 3.48E+01≈
Total score		
+	*	9
−	*	2
≈	*	9

posed algorithm to solve some real-world problems. We will also want to verify the potential of the SAPA algorithm for multi-objective optimization.

References

1. Zou, D.; Liu, H.; Gao, L.; Li, S.: A novel modified differential evolution algorithm for constrained optimization problems. Comput. Math. Appl. **61**(6), 1608–1623 (2011)
2. Hu, C.; Yan, X.: A hybrid differential evolution algorithm integrated with an ant system and its application. Comput. Math. Appl. **62**(1), 32–43 (2011)
3. Sayah, S.; Hamouda, A.; Zehar, K.: Economic dispatch using improved differential evolution approach: a case study of the Algerian electrical network. Arabian J. Sci. Eng. **38**(3), 715–722 (2013)
4. Das, S.; Sil, S.: Kernel-induced fuzzy clustering of image pixels with an improved differential evolution algorithm. Inf. Sci. **180**(8), 1237–1256 (2010)
5. Das, S.; Abraham, A.; Konar, A.: Automatic clustering using an improved differential evolution algorithm. IEEE Trans. Syst. Man Cybern. Part A **38**(1), 218–237 (2008)
6. Tasgetiren, M.F.; Suganthan, P.N.; Pan, Q.-K.: An ensemble of discrete differential evolution algorithms for solving the generalized traveling salesman problem. Appl. Math. Comput. **215**(9), 3356–3368 (2010)
7. Tang, Y.; Wang, Z.; Fang, J.: Controller design for synchronization of an array of delayed neural networks using a controllable probabilistic PSO. Inf. Sci. **181**, 4715–4732 (2011)
8. Tang, Y.; Wang, Z.; Fang, J.: Feedback learning particle swarm optimization. Appl. Soft Comput. **11**, 4713–4725 (2011)
9. Abido, M.A.; Al-Ali, N.A.: Multi-objective optimal power flow using differential evolution. Arabian J. Sci. Eng. **37**(4), 991–1005 (2012)
10. Gamperle, R.; Muller, S.D.; Koumoutsakos, P.: A parameter study for differential evolution, in Proceedings of Advanced Intelligent System, Fuzzy Systems, Evolutionary Computation, Crete, Greece, pp. 293–298 (2002)
11. Zhang, J.; Sanderson, A.C.: An approximate Gaussian model of differential evolution with spherical fitness functions. In: Proceedings of IEEE Congress on Evolutionary Computation, Singapore, pp. 2220–2228 (2007)
12. Huang, V.L.; Qin, A.K.; Suganthan, P.N.: Self-adaptive differential evolution algorithm for constrained real-parameter optimization. In Proceedings of IEEE Congress on Evolutionary Computation, Vancouver, BC, Canada, pp. 17–24 (2006)
13. Brest, J.; Zumer, V.; Maucec, M.S.: Self-adaptive differential evolution algorithm in constrained real-parameter optimization. In: Proceedings of IEEE Congress on Evolution Computation, Vancouver, BC, Canada, pp. 215–222 (2006)
14. Brest, J.; Boskovic, B.; Greiner, S.; Zumer, V.; Maucec, M.S.: Performance comparison of self-adaptive and adaptive differential evolution algorithms, soft computing—a fusion of foundations. Methodol. Appl. **11**(7), 617–629 (2007)
15. Teo, J.: Exploring dynamic self-adaptive populations in differential evolution. Soft computing—a fusion of foundations. Methodol. Appl. **10**(8), 673–686 (2006)
16. Yang, Z.; Tang, K.; Yao, X.: Self-adaptive differential evolution with neighborhood search. In: Proceedings of IEEE Congress on Evolution Computation, Hong Kong, China, pp. 1110–1116 (2008)
17. Qin, A.K.; Suganthan, P.N.: Self-adaptive differential evolution algorithm for numerical optimization. In: Proceedings of IEEE Congress on Evolutionary Computation, Edinburgh, UK, pp. 1785–1791 (2005)
18. Qin, A.K.; Huang, V.L.; Suganthan, P.N.: Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE Trans. Evol. Comput. **13**(2), 398–417 (2009)
19. Mallipeddi, R.; Mallipeddi, S.; Suganthan, P.: Ensemble strategies with adaptive evolutionary programming. Inf. Sci. **180**(9), 1571–1581 (2010)
20. Zhang, J.Q.; Sanderson, A.C.: JADE: adaptive differential evolution with optional external archive. IEEE Trans. Evol. Comput. **13**(5), 945–958 (2009)
21. Brest, J.; Greiner, S.; Boscovic, B.; Mernik, M.; Zumer, V.: Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. IEEE Trans. Evol. Comput. **10**(6), 646–657 (2006)
22. Rahnamayan, S.; Tizhoosh, H.R.; Salama, M.M.A.: A novel population initialization method for accelerating evolutionary algorithms. Comput. Math. Appl. **53**, 1605–1614 (2007)
23. Jansen, T.; Jong, K.D.; Wegener, I.: On the choice of the offspring population size in evolutionary algorithms. Evol. Comput. **13**(4), 413–440 (2005)
24. Tan, K.C.; Lee, T.H.; Khor, E.F.: Evolutionary algorithms with dynamic population size and local exploration for multiobjective optimization. IEEE Trans. Evol. Comput. **5**(6), 565–588 (2001)
25. Eiben, A.E.; Marchiori, E.; Valko, V.A.: Evolutionary algorithms with on-the-fly population size adjustment. In: Proceedings of the 8th International Conference on Parallel Problem Solving from Nature. Lecture Notes in Computer Science, vol. 3242, pp. 41–50 (2004)
26. Brest, J.; Maucec, M.S.: Population size reduction for the differential evolution algorithm. Appl. Intell. **29**(3), 228–247 (2008)

27. Epitropakis, M.G.; Tasoulis, D.K.; Pavlidis, N.G.: Enhancing differential evolution utilizing proximity-based mutation operators. IEEE Trans. Evol. Comput. **15**(1), 99–119 (2011)
28. Nasimul, N.; Iba, H.: Accelerating differential evolution using an adaptive local search. IEEE Trans. Evol. Comput. **12**(1), 107–125 (2008)
29. Suganthan, P.N.; Hansen, N.; Liang, J.J.; Deb, K.; Chen, Y.-P.; Auger, A.; Tiwari, S.: Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization, Nanyang Technol. Univ., Singapore, IIT Kanpur, Kanpur, India, Tech. Rep. KanGAL (2005)
30. Satoh, H.; Yamamura, M.; Kobayashi, S.: Minimal generation gap model for GAs considering both exploration and exploitation. In: Proceedings of IIZUKA96, Iizuka, Fukuoka, Japan, pp. 494–497 (1996)
31. Deb, K.; Anand, A.; Joshi, D.: A computationally efficient evolutionary algorithm for real-parameter optimization. Evol. Comput. **10**(4), 371–395 (2002)
32. Ono, I.; Kita, H.; Kobayashi, S.: Advances in Evolutionary Computing. New York: Springer, ch. A Real-Coded Genetic Algorithm Using the Unimodal Normal Distribution Crossover, pp. 213–237 (2003)
33. Tsutsui, S.; Yamamura, M.; Higuchi, T.: Multi-parent recombination with simplex crossover in real coded genetic algorithms. In: Proceedings of Genetic and Evolution Computation, Orlando, Florida, USA, pp. 657–664 (1999)