Front. Comput. Sci.,2015, 9(4): 623–635
DOI 10.1007/s11704-015-4178-y

RESEARCH ARTICLE

Cuckoo search with varied scaling factor

Lijin WANG[1,2], Yilong YIN (✉)[1,3], Yiwen ZHONG[2]

1 School of Computer Science and Technology, Shandong University, Jinan 250101, China
2 College of Computer and Information Science, Fujian Agriculture and Forestry University, Fuzhou 350002, China
3 School of Computer Science and Technology, Shandong University of Finance and Economics, Jinan 250014, China

Abstract Cuckoo search (CS), inspired by the obligate brood parasitic behavior of some cuckoo species, iteratively uses Lévy flights random walk (LFRW) and biased/selective random walk (BSRW) to search for new solutions. In this study, we seek a simple strategy to set the scaling factor in LFRW, which can vary the scaling factor to achieve better performance. However, choosing the best scaling factor for each problem is intractable. Thus, we propose a varied scaling factor (VSF) strategy that samples a value from the range [0,1] uniformly at random for each iteration. In addition, we integrate the VSF strategy into several advanced CS variants. Extensive experiments are conducted on three groups of benchmark functions including 18 common test functions, 25 functions proposed in CEC 2005, and 28 functions introduced in CEC 2013. Experimental results demonstrate the effectiveness of the VSF strategy.

Keywords cuckoo search algorithm, uniform distribution, random sampling, scaling factor, function optimization problems

1 Introduction

Optimization is an important tool in decision science and in physical systems [1]. Mathematically speaking, optimization can be formulated as the minimization or maximization of objective functions subject to constraints on their variables. Meta-heuristic algorithms are general optimization methods widely employed to find solutions in scientific applications. Many meta-heuristic algorithms have been designed using nature-inspired analogical models, and these are suitable for global optimization. Genetic algorithms (GA) [2] are a model or abstraction of biological evolution based on Darwin's theory of natural selection. The differential evolution algorithm (DE) is vector-based evolutionary method based on natural selection theory [3]. Ant colony optimization (ACO) simulates the behavior of real ants for searching food sources [4]. Particle swarm optimization (PSO) is inspired by the swarming behavior of flocking birds or schooling fish in nature [5, 6]. The artificial bee colony algorithm (ABC) is enlightened by food-searching behavior of bees [7]. The bat algorithm (BA) models the echolocation behavior of bats [8]. The firefly algorithm (FA) imitates the flashing patterns and behavior of fireflies [9]. The harmony search (HS) model is inspired by the observation aiming at searching for a perfect state of harmony [10]. Biogeography-based optimization (BBO) is based on the geographical distribution of biological organisms [11].

Cuckoo search (CS) is another nature-inspired algorithm based on the obligate brood parasitic behavior of some cuckoo species in combination with the Lévy flights behavior of some birds and fruit files [12, 13]. CS iteratively uses Lévy flights random walk (LFRW) and biased/selective random walk (BSRW) to search for new solutions. LFRW uses a mutation operator to generate new solutions based on the best solution obtained so far. In BSRW, first, a trial solution is generated with a mutation of the current solution and a differential step-size from two stochastically selected solutions. Second, a new solution is made by a crossover operator from the current and the trial solutions. After each random walk, a

Received April 17, 2014; accepted January 21, 2015

E-mail: ylyin@sdu.edu.cn

greedy strategy selects a better solution from the current and new generated solutions according to their fitness.

CS is a simple yet very promising global optimization technique in terms of the above two random-walks. In general, the random-walk system-equation of a meta-heuristic algorithm is formulated using a step-size with a scaling factor, and a solution. In LFRW, the step-size is drawn from the Lévy distribution. According to the standard implementation of CS [13], for each problem, the scaling factor is set to the same constant value, typically 0.01, which comes from the factor whose typical length-scale is divided by 100. Unfortunately, a typical length-scale depends on candidate optimization problems, and this results in the problem that the scaling factor is sensitive to the given optimization problems. In other words, for different optimization problems, different scaling factor values should be set instead of a same constant value. Generally speaking, the scaling factor can be constant or varied. Therefore, it is necessary to study the scaling factor parameter of CS, which is similar to parameters studies of evolutionary algorithms, e.g., PSO [14], and DE [15]. Valian et al., [16, 17] proposed an improved CS algorithm (ICS) using a varied scaling factor based on the current iteration value, maximum iteration value, and maximum and minimum scaling factors. However, ICS employs the same value for each optimization problem. Moreover, the performance of ICS may deteriorate with an increase in the maximum scaling factor value [17]. Therefore, the maximum and the minimum scaling factors, which are also sensitive to the given optimization problems, are difficult to obtain.

In this study, we utilize a simple strategy to set the value of the scaling factor in LFRW, considering a varied scaling factor (VSF) value. VSF uses a completely random strategy to sample a value that is drawn from a uniform distribution on the range [0, 1] at each iteration process. In addition, the proposed varied scaling factor is also integrated into other advanced CS variants. The major advantages of our approach are as follows: (i) since the scaling factor is set to a constant value, typically 0.01, a varied scaling factor is generally more suitable for optimization problems; (ii) our approach does not destroy the structure of CS, therefore it is still very simple; (iii) it does not increase the complexity of CS; (iv) it can be easily integrated into other CS variants. We investigate CS with varied scaling factors on 18 common benchmark functions [18–20], 25 functions proposed in CEC 2005 [21], and 28 functions introduced in CEC 2013 [22]; our results demonstrate the promise of a varied scaling factor.

Section 2 describes the standard CS algorithm. Section 3 analyzes CS with a varied scaling factor. Section 4 reports the experimental results. Section 5 introduces related work. Section 6 concludes this paper.

2 Cuckoo search algorithm

CS, developed recently by Yang and Deb [12, 13], is a simple yet very promising population-based stochastic search technique. In general, when CS is used to solve an objective function $f(x)$ with the solution space $[x_{j,\min}, x_{j,\max}]$, $j = 1, 2, \ldots, D$, a nest represents a candidate solution $X = (x_1, \ldots, x_D)$.

In the initialization phase, CS initializes solutions that are randomly sampled from solution space by

$$x_{i,j,0} = x_{i,j,\min} + r \times (x_{i,j,\max} - x_{i,j,\min}), i = 1, 2, \ldots, NP, \quad (1)$$

where r represents a uniformly distributed random variable on the range [0,1], and NP is the population size.

After initialization, CS goes into an iterative phase where two random walks, Lévy flights random walk and biased/selective random walk, are employed to search for new solutions. After each random walk process, CS selects the better solutions according to their fitness using the greedy strategy. At the end of each iteration, the best solution is updated.

2.1 Lévy flights random walk

LFRW is a random walk whose step-size is drawn from the Lévy distribution. At generation $G(G > 0)$, LFRW can be formulated as follows.

$$X_{i,G+1} = X_{i,G} + \alpha \oplus \text{Lévy}(\beta), \quad (2)$$

where $\alpha > 0$ is a step-size that is related to the scale of the problem.

In CS, LFRW is employed to search for new solutions close to the best solution obtained so far. Therefore, the step-size can be obtained by the following equation [13].

$$\alpha = \alpha_0 \times (X_{i,G} - X_{\text{best}}), \quad (3)$$

where α_0 is a scaling factor (generally, $\alpha_0 = 0.01$), and X_{best} represents the best solution obtained so far.

The product $\oplus$ means entry-wise multiplications. Lévy(β) is a random number, which is drawn from a Lévy distribution for large steps

$$\text{Lévy}(\beta) \sim \mu = t^{-1-\beta}, 0 < \beta \leqslant 2. \quad (4)$$

In implementation, Lévy(β) can be calculated as follows [13]:

$$\text{Lévy}(\beta) \sim \frac{\phi \times \mu}{|\nu|^{\frac{1}{\beta}}}, \quad (5)$$

$$\phi = \left(\frac{\Gamma(1+\beta) \times \sin\left(\frac{\pi \times \beta}{2}\right)}{\Gamma\left(\frac{1+\beta}{2} \times \beta \times 2^{\frac{\beta-1}{2}}\right)} \right)^{\frac{1}{\beta}}, \quad (6)$$

where β is a constant and set to 1.5 in the standard software implementation of CS [13], μ and ν are random numbers drawn from a normal distribution with mean of 0 and standard deviation of 1, and Γ is a *gamma function*.

Obviously, Eq. (2) can be reformulated as

$$X_{i,G+1} = X_{i,G} + \alpha_0 \frac{\phi \times \mu}{|\nu|^{\frac{1}{\beta}}}(X_{i,G} - X_{\text{best}}). \quad (7)$$

2.2 Biased/selective random walk

BSRW is used to discover new solutions sufficiently far from the current best solution using far field randomization [12]. First, a trial solution is built using a mutation of the current solution and a differential step size from two stochastically selected solutions. Second, a new solution is generated by a crossover operator from the current and the trial solutions. BSRW can be formulated as follows:

$$x_{i,j,G+1} = \begin{cases} x_{i,j,G} + r \times (x_{m,j,G} - x_{n,j,G}), \text{ with } p_a; \\ x_{i,j,G}, \text{ with the remaining probability}, \end{cases} \quad (8)$$

where m and n are random indexes, r is a random number on the range [0,1], and p_a is a fraction probability.

3 CS with varied scaling factor

In LFRW, CS owns a scaling factor of the step-size. According to the standard implementation of CS, for each optimization problem, the scaling factor is set to the same constant value, commonly 0.01, which comes from the factor whose typical length-scale is divided by 100. However, to the best of our knowledge, the typical length-scale is dependent on candidate optimization problems, and this results in making the scaling factor sensitive to the given optimization problems. To ensure this, we experimentally investigate the performance of CS with different scaling factor values, and the normalized function error are plotted in Fig. 1 where F_{ros} and F_{ras}, listed in APPENDIX A, are widely used in the evolutionary computation community [19], and F_6, F_7, and F_{10} were proposed in the CEC 2005 competition. The function error is defined as $(f(x) - f(x^*))$, where x^* is the global optimum of the function, and x is the best solution obtained by the algorithm in a given run. Each benchmark function at $D = 30$ is tested with the different scaling factor values sampled from 0.001 to 0.01 with a step size of 0.001, from 0.02 to 0.1 with a step size of 0.01, and from 0.2 to 1 with a step size of 0.1. The population size and the maximum function evaluations are D and $10000 \times D$, respectively, and the algorithm is performed 50 times.

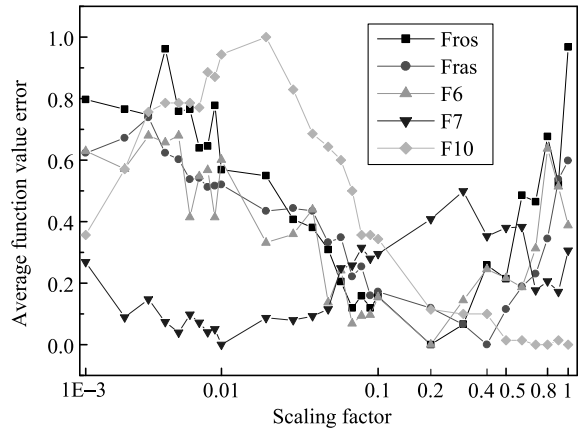

Fig. 1 The performances of CS with different scaling factor values

Figure 1 says that the performance of CS is sensitive to the scaling factor for the same benchmark function. For example, for F_7, CS can obtain good performance around 0.01, and bad performance far away from 0.01. In contrast, for F_{10}, CS brings promising performance far away from 0.01. Furthermore, it is observed that the scaling factor values, suitable for the different benchmark functions obtain the best performance by CS, are different. For F_7, the scaling factor is 0.01, and the scaling values are 0.2, 0.4, and 0.2 for F_{ros}, F_{ras}, and F_6, respectively.

Our investigation results further motivate us to analyze a reasonable scaling factor value for each optimization function. The scaling factor value can be constant or varied. Inspired by this, we seek a reasonable scaling factor value by considering a varied value for each optimization problem.

However, as we observe from Fig. 1, it is difficult to fit a curve to a kind of distribution that presents the relationship between the scaling factor and its performance for each optimization problem. Even if it can be done, the distribution presenting the relationship will be complex, resulting in increasing the complexity of CS. Therefore, a simple yet common approach is to employ the uniform distribution with a range between 0 and a positive value U at each iteration process. However, in the case of $U > 1$, Lévy flights may become too aggressive, which makes new solutions jump out of the search space, resulting in wasted evaluations. The scaling factor in this paper is similar to the mutation scale factor F of DE, whose effective range is usually between 0.4 and 1 [15]. Therefore, the upper bound of the range U is set to 1, and

Eq. (7) can be rewritten as follows:

$$X_{i,G+1} = X_{i,G} + F\frac{\phi \times \mu}{|\nu|^{\frac{1}{\beta}}}(X_{i,G} - X_{\text{best}}), \tag{9}$$

where F presents a varied scaling factor drawn from a uniform distribution on the range [0,1].

According to the above description, CS with a varied scaling factor (VCS) can be summarized in Algorithm 1.

Algorithmic 1 VCS

```
G ← 0;
Nest_0 = (X_{i,0}, ..., X_{NP,0}) ←InitializeSolution();
Fitness←Evaluation(Nest_0);
FES = NP;
BestX ←FindBestSolutionByFitness();
while FES < MaxFES do
    G ← G + 1;
    for i ← 1 to NP do
        F ←Sampling randomly from the range [0,1]
        newX_{i,G} ←Generating new solution with Eq.(9)
        X_{i,G} ←EvaluatingAndSelecting(newX_{i,G}, X_{i,G−1});
        FES = FES+1;
    end for
    for i ← 1 to NP do
        newX_{i,G} ←Generating new solution with Eq.(8);
        X_{i,G} ←EvaluatingAndSelecting(newX_{i,G}, X_{i,G});
        FES = FES + 1;
    end for
    BestX ←FindAndUpdateBestSolution();
end while
```

4 Experimental verifications

In this section, we experimentally verify the effectiveness of CS with a varied scaling factor (VCS), and apply the varied scaling factor strategy to some improved CS versions. Three groups of benchmark functions are used in simulation. In our experimental studies, we use the average and standard deviation of the function error to compare the performance of our algorithm. The maximum number of fitness evaluations ($FES_{\max}$) is set to 10000×D, where D is the dimension of the function, and set to 30 for all benchmark functions. The minimum function error that each algorithm can find is also recorded in different runs, and the average and the standard deviation of the error are calculated. The notation $AVG_{Er}\pm SD_{Er}$ is used in the different tables. According to the Wilcoxon signed-rank test, the comparison results between algorithms are summarized as "–/=/+" in the last row of the tables. Each algorithm is performed 25 times and averaged results are presented for each benchmark function respectively, and the p_a is set to 0.25. In addition, in order to make a fair comparison, we employ the boundary-handling method used in the literature [23] for all mentioned algorithms.

4.1 Comparison results on common benchmark functions

A suite of 18 benchmark functions is used to analyze the performance of VCS. These 18 benchmark functions are widely used in the evolutionary computation community [18–20]; they are described in Appendix A. A more detailed description of them can be found in [18–20].

Table 1 lists the average function error of 18 benchmark functions obtained by CS and VCS. It is clear that VSF improves the performance of CS significantly according to the average function error. Moreover, VCS can obtain the global optimal solution for F_{grw} and F_{ste} although VCS is similar to CS according to the Wilcoxon signed rank test at $\alpha = 0.05$. In all, in terms of "–/=/+", this clearly shows CS with a varied scaling factor outperforms CS with a constant scaling factor.

Table 1 Average function error obtained by CS and VCS for common benchmark functions at $D = 30$

Fun	CS		VCS
	$AVG_{Er}\pm SD_{Er}$		$AVG_{Er}\pm SD_{Er}$
F_{sph}	1.21E–30±2.56E–30	+	**4.53E–49**±7.48E–49
F_{ros}	1.32E+01±1.13E+01	+	**4.67E+00**±2.33E+00
F_{ack}	1.93E–11±9.55E–11	+	**7.25E–15**±7.11E–16
F_{grw}	4.93E–04±2.46E–03	=	**0.00E+00**±0.00E+00
F_{ras}	2.56E+01±5.61E+00	+	**1.78E+01**±6.96E+00
F_{sch}	1.57E+03±1.93E+02	+	**5.61E+02**±3.38E+02
F_{sal}	3.76E–01±8.79E–02	+	**2.24E–01**±4.36E–02
F_{pn1}	4.23E–15±2.11E–14	+	**1.57E–32**±5.59E–48
F_{pn2}	2.21E–27±1.01E–26	+	**1.35E–32**±4.07E–05
F_{pow}	2.97e–04±3.32e–04	+	**7.55E–05**±5.59E–48
F_{zak}	6.91e–05±3.62e–05	+	**1.52E–05**±8.15E–06
F_{lvy}	1.07e–02±2.97e–02	+	**1.50E–32**±8.38E–48
F_{sus}	7.57e–32±7.76e–32	+	**2.73E–50**±3.34E–50
F_{dap}	6.67e–01±3.80e–07	+	6.67E–01±8.69E–15
F_{scw}	7.41e–15±3.84e–15	+	**8.29E–28**±8.16E–28
F_{sce}	2.25e+00±1.33e+00	+	**2.74E–05**±1.81E–05
F_{ste}	8.00e–02±2.77e–01	=	**0.00E+00**±0.00E+00
F_{qua}	1.18e–02±5.35e–03	+	**6.17E–03**±1.70E–03
–/=/+	0/2/16		

Note: "+" and "=" show VCS is better than, or similar to CS according to the Wilcoxon signed rank test at $\alpha = 0.05$, respectively.

Figure 2 shows the convergence curves of CS and VCS. In this we can see that VCS converges faster than CS.

4.2 Comparison results on benchmark functions presented in CEC 2005 competition

In this section, 25 benchmark functions proposed in the CEC 2005 special session on real-parameter optimization [21] are used to study the performance of VCS. These 25 benchmark

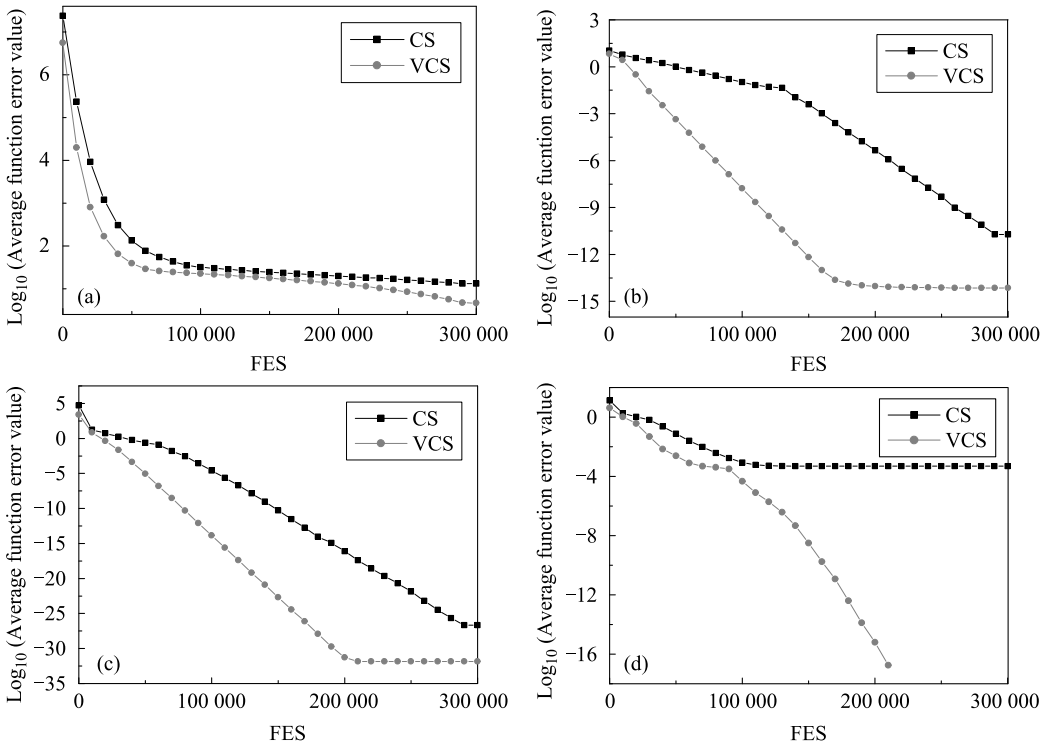

Fig. 2 The convergence curves of CS and VCS on (a) F_{ros}, (b) F_{ack}, (c) F_{pn2}, and (d) F_{grw}

Table 2 Average function error obtained by CS and VCS for CEC 2005 benchmark functions at $D = 30$

Fun	CS		VCS
	$AVG_{Er} \pm SD_{Er}$		$AVG_{Er} \pm SD_{Er}$
F_1	7.60E–30±1.45E–29	+	**0.00E+00**±0.00E+00
F_2	**9.55E–03**±1.34E–02	–	1.44E–02±1.27E–02
F_3	**2.17E+06**±5.67E+05	–	3.34E+06±9.76E+05
F_4	1.40E+03±8.84E+02	+	**4.45E+02**±2.74E+02
F_5	2.98E+03±6.46E+02	+	**1.84E+03**±7.73E+02
F_6	1.78E+01±2.07E+01	=	**1.61E+01**±2.65E+01
F_7	**2.74E–04**±5.46E–04	–	3.46E–03±6.55E–03
F_8	2.09E+01±5.96E–02	=	2.09E+01±4.42E–02
F_9	2.56E+01±6.18E+00	+	**1.74E+01**±6.33E+00
F_{10}	1.63E+02±3.77E+01	+	**9.87E+01**±1.50E+01
F_{11}	2.93E+01±1.48E+00	+	**2.71E+01**±1.52E+00
F_{12}	**1.84E+04**±4.94E+03	–	2.65E+04±1.33E+04
F_{13}	**5.91E+00**±6.26E–01	–	7.22E+00±1.68E+00
F_{14}	1.29E+01±1.50E–01	+	**1.27E+01**±2.42E–01
F_{15}	**2.75E+02**±7.72E+01	=	3.17E+02±1.01E+02
F_{16}	1.83E+02±4.71E+01	+	**1.31E+02**±2.47E+01
F_{17}	2.13E+02±4.38E+01	+	**1.89E+02**±4.40E+01
F_{18}	9.10E+02±2.06E+00	+	**9.03E+02**±2.14E+01
F_{19}	9.06E+02±2.16E+01	+	**9.03E+02**±2.15E+01
F_{20}	9.02E+02±3.00E+01	=	9.02E+02±2.15E+01
F_{21}	5.00E+02±9.08E–13	=	5.00E+02±1.55E–13
F_{22}	9.61E+02±2.59E+01	+	**9.19E+02**±1.66E+01
F_{23}	5.46E+02±5.61E+01	+	**5.34E+02**±2.16E–04
F_{24}	2.00E+02±6.00E–14	=	2.00E+02±1.52E–12
F_{25}	2.13E+02±1.15E+00	+	**2.12E+02**±9.53E–01
–/=/+	5/6/14		

Note: "+", "–" and "=" show VCS is better than, worse than or similar to CS according to the Wilcoxon signed rank test at $\alpha = 0.05$, respectively.

functions can be divided into four classes: unimodal functions F_1–F_5, basic multimodal functions F_6–F_{12}, expanded multimodal functions F_{13}–F_{14}, and hybrid composition functions F_{15}–F_{25}. A detailed definition of these benchmark functions can be found in the literature [21]. The results are presented in Table 2.

For unimodal functions F_1–F_5, in terms of the average function error listed in Table 2, VSF enhances the performance of CS for F_1, F_4, and F_5. For basic multimodal functions F_6–F_{12}, in terms of the average function error, VCS outperforms CS on 3 out of 7, is tied for performance on 2 out of 7, and has worse performance on 2 out of 7. For expanded multimodal functions F_{13}–F_{14}, VCS is equivalent to CS with the help of the average function error. For hybrid composition functions F_{15}–F_{25}, VCS outperforms CS on 7 out of 11 functions, and offers equivalent performance for 4 out of 11 functions. In summary, for 25 CEC2005 benchmark functions, VCS wins 14 functions, ties 6 functions, and loses 5 functions. This offers strong evidence that the varied scaling factor value is more appropriate than the constant scaling factor value for most optimization problems.

Figure 3 shows the convergence curves of CS and VCS on some functions proposed in CEC 2005. It can be observed that, for the functions on which VCS outperforms than CS, VCS also has the faster convergence performance, as seen in Fig. 3.

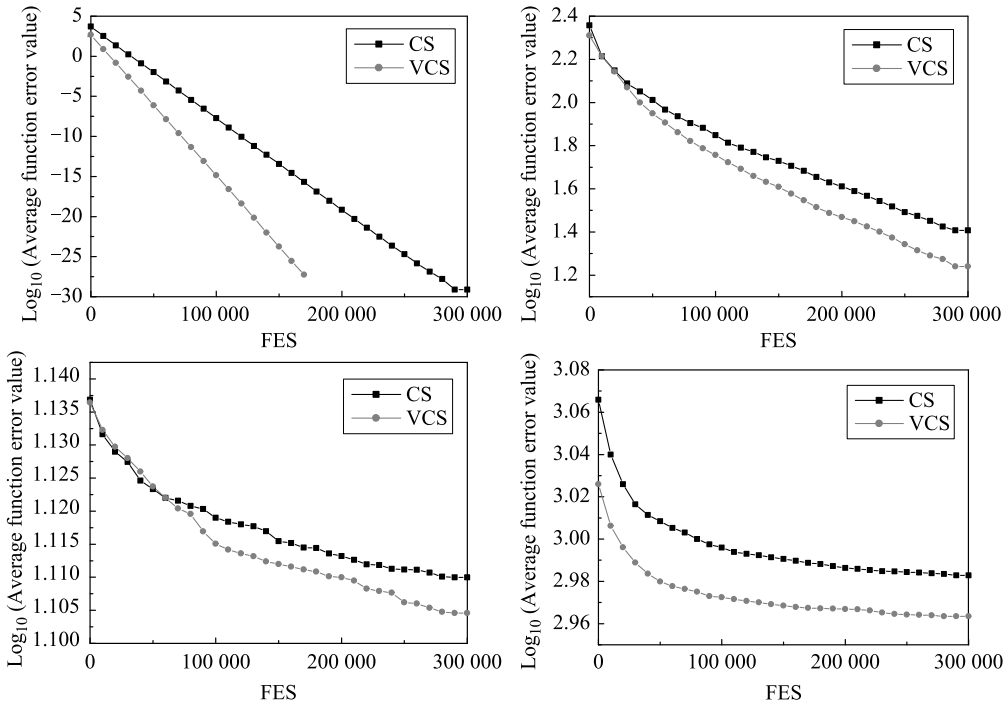

Fig. 3 The convergence curves of CS and VCS on (a) F_1, (b) F_9, (c) F_{14}, and (d) F_{22}

4.3 Comparison results on benchmark functions presented in CEC 2013 competition

In this section, we further verify the varied scaling factor strategy on another 28 benchmark functions proposed in CEC 2013 special session on real-parameter optimization [22]. These 28 benchmark functions include 5 unimodal functions F_1–F_5, 15 basic multimodal functions F_6–F_{20}, and 8 composition functions F_{21}–F_{28}. More detailed descriptions of these benchmark functions can be found in the literature [22].

Table 3 presents the average function error for each algorithm. For unimodal functions, VCS outperforms CS on 4 out of 5 functions. For basic multimodal functions, VCS exhibits higher performance than CS on 8 out of 15 functions, is equivalent to with CS on 4 out of 15 functions, and shows less performance than CS on 3 out of 15 functions. For composition functions, VCS is better on 3, and worse on 1 out of 8 functions, and VCS shows equivalent performance on 4 out of 8 functions. In terms of "–/=/+", VCS wins, ties, and loses CS on 15, 8, and 5 functions, respectively. It also demonstrates that the varied scaling factor value is more reasonable than the constant value for CS to solve most benchmark functions proposed in CEC 2013.

Figure 4 shows the convergence curves of CS and VCS on some functions proposed in CEC 2013. The same conclusions as in Fig. 3 can be drawn from Fig. 4, e.g., VCS outperforms CS in terms of the convergence curve shown in Fig. 4.

Table 3 Average function error obtained by CS and VCS for CEC 2013 benchmark functions at $D = 30$

Fun	CS		VCS
	$AVG_{Er} \pm D_{Er}$		$AVG_{Er} \pm SD_{Er}$
F_1	1.28E–29±3.36E–29	+	**0.00E+00**±0.00E+00
F_2	**2.74E+06**±6.41E+05	–	4.98E+06±1.94E+06
F_3	2.76E+07±2.63E+07	+	**3.94E+06**±9.79E+06
F_4	2.79E+04±4.11E+03	+	**1.67E+04**±3.82E+03
F_5	9.12E–17±1.89E–16	+	**0.00E+00**±0.00E+00
F_6	1.61E+01±1.78E+01	=	**1.20E+01**±6.88E+00
F_7	9.50E+01±1.51E+01	+	**6.32E+01**±1.50E+01
F_8	2.10E+01±3.76E–02	=	**2.09E+01**±3.69E–02
F_9	3.02E+01±1.43E+00	+	**2.85E+01**±1.47E+00
F_{10}	**6.32E–03**±4.71E–03	=	1.52E–02±1.12E–02
F_{11}	3.20E+01±6.89E+00	+	**2.03E+01**±8.95E+00
F_{12}	1.59E+02±2.83E+01	+	**9.76E+01**±1.28E+01
F_{13}	2.02E+02±2.60E+01	+	**1.34E+02**±1.97E+01
F_{14}	**2.01E+03**±2.57E+02	–	2.56E+03±7.06E+02
F_{15}	**4.17E+03**±2.62E+02	–	5.00E+03±2.59E+02
F_{16}	**1.73E+00**±2.82E–01	–	1.94E+00±2.00E–01
F_{17}	1.32E+02±1.71E+01	+	**1.13E+02**±2.03E+01
F_{18}	1.94E+02±2.98E+01	+	**1.65E+02**±9.94E+00
F_{19}	8.53E+00±1.69E+00	=	**7.85E+00**±1.55E+00
F_{20}	1.26E+01±5.93E–01	+	**1.20E+01**±4.03E–01
F_{21}	**2.24E+02**±5.34E+01	=	2.55E+02±6.28E+01
F_{22}	**2.40E+03**±3.37E+02	–	2.78E+03±9.00E+02
F_{23}	**5.37E+03**±2.84E+02	=	5.42E+03±3.50E+02
F_{24}	2.80E+02±1.34E+01	+	**2.59E+02**±1.47E+01
F_{25}	3.09E+02±6.36E+00	+	**2.94E+02**±6.85E+00
F_{26}	2.00E+02±2.91E–02	=	2.00E+02±4.89E–02
F_{27}	1.06E+03±2.37E+02	+	**1.03E+03**±5.91E+01
F_{28}	3.00E+02±7.40E–10	=	3.00E+02±6.96E–14
–/=/+	5/8/15		

Note: "+", "–" and "=" show VCS is better than, worse than or similar to CS according to the Wilcoxon signed rank test at $\alpha = 0.05$, respectively.

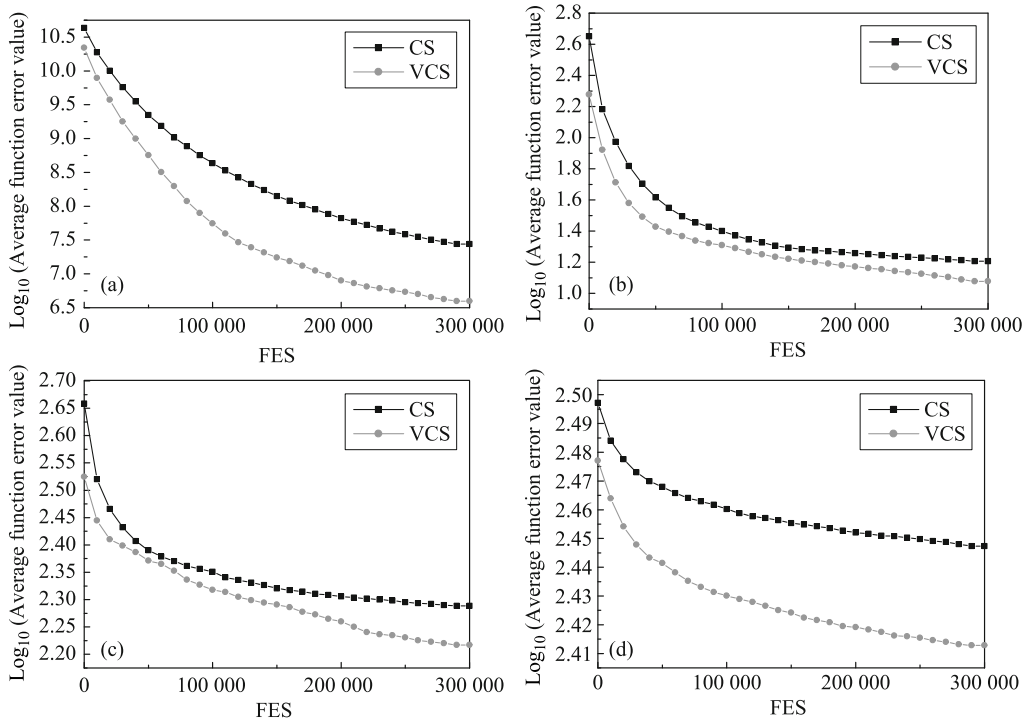

Fig. 4 The convergence curves of CS and VCS on (a) F_3, (b) F_6, (c) F_{18}, and (d) F_{24}

Table 4 Average function error obtained by CS and VCS for common benchmark functions at $D = 10$, $D = 50$, $D = 100$, and $D = 200$

Fun	$D = 10$			$D = 50$			$D = 100$			$D = 200$		
	CS		VCS	CS		VCS	CS		VCS	CS		VCS
F_{sph}	4.81E–26	+	**5.37E–57**	3.92E–17	+	**4.61E–23**	6.68E–19	+	**1.31E–21**	1.53E–20	+	**7.58E–21**
F_{ros}	5.22E–01	+	**1.40E–01**	4.46E+01	+	**3.38E+01**	1.42e+02	+	**1.12E+02**	4.33e+02	+	**3.84E+02**
F_{ack}	3.08E–11	+	**3.55E–15**	3.55E–02	+	**2.51E–12**	5.22E–01	+	**1.40E–01**	1.72e+00	+	**2.33E–11**
F_{grw}	3.18E–02	=	**2.56E–02**	2.16E–11	+	**0.00E+00**	8.36e–01	+	**9.65E–12**	**3.73E–16**	–	2.96E–04
F_{ras}	3.34E+00	+	**1.18E+00**	**8.52E+01**	–	1.04E+02	**1.66E+02**	–	2.32e+02	**3.03E+02**	–	3.67e+02
F_{sch}	7.46E+01	+	**3.64E–14**	**4.93E+03**	–	5.81E+03	**1.15E+04**	–	1.48e+04	**2.59E+04**	–	3.39e+04
F_{sal}	9.99E–02	=	9.99E–02	6.60E–01	+	**4.00E–01**	1.38E+00	+	**8.46E–01**	3.06E+00	+	**1.84E+00**
F_{pn1}	5.80E–19	+	**4.71E–32**	1.00E–03	+	**3.45E–18**	3.19E–04	+	**2.32E–10**	1.35E–02	+	**1.33E–02**
F_{pn2}	3.26E–23	+	**1.35E–32**	5.74E–14	+	**9.08E–22**	2.44E–06	+	**1.81E–19**	2.39E+01	+	**4.64E–01**
F_{pow}	**3.45E–15**	–	1.85E–13	2.76E–03	+	**1.34E–03**	2.70E–03	+	**1.53E–03**	**5.58E–03**	–	6.82E–03
F_{zak}	2.30e–16	+	**3.95E–24**	**5.11E+00**	–	9.65E+00	**8.18E+01**	–	1.18E+02	**4.02E+02**	–	5.08e+02
F_{lvy}	1.57E–20	+	**1.50E–32**	1.16E–02	+	**1.93E–21**	1.29E–01	+	**3.06E–21**	9.42E–01	+	**2.82E–21**
F_{sus}	1.18E–27	+	**1.58E–58**	8.55E–18	+	**6.23E–24**	3.16E–19	+	**5.32E–22**	1.62E–20	+	**4.99E–21**
F_{dap}	**2.18E–01**	=	3.18E–01	6.67E–01	=	6.67E–01	1.75E+00	+	**6.67E–01**	2.19E+01	+	**1.64E+01**
F_{scw}	5.31E–12	+	**4.08E–33**	5.07e–08	+	**3.03E–13**	1.68E–10	+	**2.26E–12**	2.10E–13	+	**1.32E–13**
F_{sce}	2.77E–06	+	**1.39E–12**	3.27E+00	+	**1.25E–01**	1.23E+01	+	**3.52E+00**	1.86E+01	+	**1.59E+01**
F_{ste}	0.00E+00	=	0.00E+00	0.00E+00	=	0.00E+00	1.28E+00	+	**4.00E–02**	1.14E+01	+	**4.40E–01**
F_{qua}	2.76E–03	+	**1.54E–03**	2.15E–02	+	**1.38E–02**	9.49E–02	+	**5.19E–02**	3.23E–01	+	**1.77E–01**
–/=/+	2/4/12			3/2/13			3/0/15			5/0/13		

Note: "+", "–" and "=" show VCS is better than, worse than or similar to CS according to the Wilcoxon signed rank test at $\alpha = 0.05$, respectively

4.4 Scalability study

To analyze the performance of CS with VSF affected by the dimensionality of the problem, we conduct a scalability study on 18 common benchmark functions with $D = 10$, $D = 50$, $D = 100$, and $D = 200$, CEC 2005 and CEC 2013 benchmark functions with $D = 10$ and $D = 50$ since these functions are defined for up to $D = 50$ [21, 22]. All other parameters are unchanged from their values mentioned above except that the population size NP is 30, 50, 50, and 50 on $D = 10$,

$D = 50$, $D = 100$, and $D = 200$, respectively. Tables 4–6 list the average function error obtained by CS and VCS for the 18 common, CEC 2005, and CEC 2013 benchmark functions with $D = 10$ and $D = 50$, respectively. The better results are marked in boldface.

In the case of $D = 10$, observed in Table 4, CS with VSF can achieve solutions with higher accuracy for most functions according to the average function error. For the functions proposed in CEC 2005, seen in Table 5, with the help of the average function error, VSF makes CS obtain the better performance for most functions, but not achieve the best performance for a few. According to Table 6, the conclusion drawn on the functions proposed in CEC2005 can be also come to for the functions introduced in CEC 2013. Nevertheless, for all functions shown in Tables 4–6, in terms of the results of the Wilcoxon signed-rank test, CS with VSF wins, ties, and loses CS without VSF on 39, 27, and 5 functions, respectively.

When $D = 50$, for the 18 common benchmark functions, VCS still outperforms CS for most functions in terms of the average function error listed in Table 4. For the functions proposed in CEC 2005 and CEC 2013, The conclusion reached in the case of $D = 10$ can also be drawn for the case of $D = 50$ according to Table 5 and Table 6. In addition, according to the results of the Wilcoxon signed-rank test given in the three tables, the numbers of VCS wins, ties, and losses over CS are 42, 14, and 15, respectively.

In the case of $D = 100$ and $D = 100$, compared with CS, VCS still achieves solutions with higher accuracy for 15 and 13 out of 18 respective functions according to the average function error listed in Table 4.

Tables 4–6 also show that the performance obtained by CS and VCS are indifferent to the dimensionality of some functions: F_7, F_8, F_{14} of CEC 2005, and F_8, F_{28} of CEC 2013, in terms of the results of the Wilcoxon signed-rank test. In addition, for a few functions, F_{ras}, F_{sch}, and F_{zak}, VCS underperforms to CS significantly in the case of $D = 50$, $D = 100$,

Table 5 Average function error obtained by CS and VCS for CEC 2005 benchmark functions at $D = 10$ and $D = 50$

Fun	$D = 10$			$D = 50$		
	CS		VCS	CS		VCS
F_1	4.38E–26	+	**0.00E+00**	1.95E–16	+	**5.16E–23**
F_2	1.20E–13	+	**2.32E–18**	**2.33E+02**	–	7.94E+02
F_3	**2.31E+02**	–	1.13E+03	**8.22E+06**	–	1.62E+07
F_4	1.13E–05	+	**2.79E–10**	2.63E+04	+	**2.20E+04**
F_5	1.07E–04	+	**4.37E–12**	1.02E+04	+	**6.35E+03**
F_6	1.31E+00	+	**6.57E–02**	6.67E+01	+	**4.91E+01**
F_7	**4.63E–02**	=	4.69E–02	**2.26E–03**	=	4.01E–03
F_8	2.04E+01	=	2.04E+01	2.11E+01	=	2.11E+01
F_9	2.76E+00	+	**4.21E–01**	1.20E+02	+	**1.08E+02**
F_{10}	1.91E+01	+	**1.17E+01**	3.94E+02	+	**2.43E+02**
F_{11}	5.71E+00	+	**4.67E+00**	**5.61E+01**	=	5.66E+01
F_{12}	4.97E+01	=	**2.82E+01**	**1.30E+05**	–	1.89E+05
F_{13}	**9.11E–01**	=	9.17E–01	**1.43E+01**	–	1.87E+01
F_{14}	3.35E+00	=	**3.27E+00**	2.26E+01	=	**2.25E+01**
F_{15}	1.35E+02	+	**7.50E+01**	3.28E+02	=	**3.18E+02**
F_{16}	1.29E+02	+	**1.17E+02**	2.37E+02	+	**1.77E+02**
F_{17}	1.48E+02	+	**1.29E+02**	2.68E+02	=	**2.65E+02**
F_{18}	5.51E+02	+	**4.02E+02**	9.29E+02	+	**9.21E+02**
F_{19}	5.06E+02	+	**3.67E+02**	9.27E+02	+	**9.17E+02**
F_{20}	5.37E+02	+	**4.10E+02**	9.27E+02	+	**9.21E+02**
F_{21}	**4.47E+02**	=	4.73E+02	6.83E+02	+	**5.00E+02**
F_{22}	7.11E+02	+	**6.89E+02**	1.01E+03	+	**9.59E+02**
F_{23}	**5.58E+02**	=	5.72E+02	8.36E+02	+	**6.49E+02**
F_{24}	2.00E+02	=	2.00E+02	3.78E+02	+	**2.00E+02**
F_{25}	3.82E+02	=	3.82E+02	2.24E+02	+	**2.21E+02**
–/=/+	1/9/15			4/6/15		

Note: "+", "–" and "=" show VCS is better than, worse than or similar to CS according to the Wilcoxon signed rank test at $\alpha = 0.05$, respectively

Table 6 Average function error obtained by CS and VCS for CEC 2013 benchmark functions at $D = 10$ and $D = 50$

Fun	$D = 10$			$D = 50$		
	CS		VCS	CS		VCS
F_1	5.08E–26	+	**0.00E+00**	1.27E–16	+	**4.93E–23**
F_2	**4.86E+02**	–	2.21E+03	**1.01E+07**	–	1.73E+07
F_3	8.93E+04	+	**1.71E+04**	**2.25E+08**	=	3.72E+08
F_4	1.65E+02	+	**1.20E+02**	5.87E+04	+	**4.08E+04**
F_5	4.49E–18	+	**0.00E+00**	4.68E–09	+	**3.67E–14**
F_6	**2.82E–02**	=	3.95E–01	4.36E+01	+	**4.34E+01**
F_7	2.09E+01	+	**5.16E+00**	1.21E+02	+	**1.05E+02**
F_8	2.04E+01	=	**2.03E+01**	2.11E+01	=	2.11E+01
F_9	5.27E+00	+	**4.10E+00**	5.89E+01	+	**5.68E+01**
F_{10}	**6.67E–02**	=	6.78E–02	**1.11E–02**	–	3.10E–02
F_{11}	3.66E+00	+	**1.02E+00**	1.30E+02	=	**1.14E+02**
F_{12}	1.87E+01	+	**1.14E+01**	3.68E+02	+	**2.58E+02**
F_{13}	2.20E+01	+	**1.53E+01**	4.09E+02	+	**3.17E+02**
F_{14}	**2.67E+02**	=	2.72E+02	**5.14E+03**	–	7.73E+03
F_{15}	7.65E+02	=	**7.39E+02**	**8.89E+03**	–	1.08E+04
F_{16}	**6.18E–01**	–	8.68E–01	2.75E+00	=	**2.67E+00**
F_{17}	2.08E+01	=	**2.02E+01**	3.15E+02	+	**2.73E+02**
F_{18}	3.11E+01	=	**2.96E+01**	3.99E+02	+	**3.59E+02**
F_{19}	8.97E–01	=	**8.47E–01**	2.31E+01	=	**2.22E+01**
F_{20}	3.03E+00	=	**2.68E+00**	2.30E+01	+	**2.18E+01**
F_{21}	**1.59E+02**	=	2.28E+02	**2.56E+02**	–	2.62E+02
F_{22}	4.46E+02	+	**3.49E+02**	**6.54E+03**	–	8.66E+03
F_{23}	1.06E+03	=	**9.62E+02**	**1.09E+04**	–	1.15E+04
F_{24}	**1.41E+02**	=	1.45E+02	3.57E+02	+	**3.35E+02**
F_{25}	**1.83E+02**	=	1.94E+02	4.22E+02	+	**3.92E+02**
F_{26}	1.24E+02	+	**1.15E+02**	**2.01E+02**	–	2.02E+02
F_{27}	3.76E+02	+	**3.06E+02**	1.93E+03	+	**1.81E+03**
F_{28}	**1.78E+02**	=	2.60E+02	4.00E+02	=	4.00E+02
–/=/+	2/14/12			8/6/14		

Note: "+", "–" and "=" show VCS is better than, worse than or similar to CS according to the Wilcoxon signed rank test at $\alpha = 0.05$, respectively

and $D = 200$, while VCS outperforms CS when $D = 10$. However, compared with CS, VCS can achieve solutions with higher accuracy to most functions with the increase in dimensionality from $D = 10$ to $D = 200$. This suggests that the advantage of VCS over CS is stable when the dimensionality of problems increases.

4.5 Results on integrating VSF into the improved CS versions

To reveal the suitability of VSF to the improved CS versions, we integrate VSF into ICS, CSPSO [24], and DDICS [25], resulting in VICS, VCSPSO, and VDDICS. VICS uses VSF instead of an adaptive varied factor value calculated from the complex equation including four parameters. VCSPSO and VDDICS employ VSF to replace the constant scaling factor value, typically 0.01. Each algorithm is tested on 28 benchmark functions proposed in the CEC 2013 special session on real-parameter optimization [22] at $D = 30$ with 25 independent runs. According to the literature [17], two groups of parameters, employed for ICS and its variants, are listed in Table 7, resulting in ICS-I and ICS-II. Other parameters are unchanged. The results are listed in Table 8.

From Table 8, CS variants with VSF provide significantly better results overall compared with CS methods without VSF. For example, CSPSO with VSF can significantly outperform CSPSO without VSF. DDICS with VSF wins, ties, and loses to DDICS without VSF in 7, 20, and 1 functions, respectively. VICS-II is respectively better than, worse than, and similar to the ICS-II on 11, 7, and 10 out of 28 functions. VICS-I is slightly worse than ICS-I, but shares with it on 23 out of 28 functions. This also reveals that the performance of ICS is influenced by the parameters significantly. Moreover,

Table 7 Parameters of ICS

	α	p_a
I	$\alpha(\max) = 0.5$	$p_a(\max) = 0.5$
	$\alpha(\min) = 0.01$	$p_a(\min) = 0.05$
II	$\alpha(\max) = 0.5$	$p_a(\max) = 1$
	$\alpha(\min) = 0.05$	$p_a(\min) = 0.005$

Table 8 Average function error obtained by different CS algorithms for CEC 2013 benchmark functions at $D = 30$

Fun	ICS-I		VICS-I	ICS-II		VICS-II	CSPSO		VCSPSO	DDICS		VDDICS
F_1	0.00E+00	=	0.00E+00	**2.58E–18**	–	8.96E–17	3.79E–10	+	**0.00E+00**	0.00E+00	=	0.00E+00
F_2	1.27E+07	=	**1.20E+07**	3.79E+07	+	**3.00E+07**	3.20E+06	+	**8.62E+05**	**8.16E+06**	=	8.89E+06
F_3	**5.44E+07**	–	1.28E+08	**1.21E+08**	–	7.06E+08	3.04E+08	+	**1.27E+07**	**4.06E+08**	=	4.54E+08
F_4	**2.19E+04**	=	2.31E+04	**3.73E+04**	=	3.84E+04	1.14E+03	+	**3.50E+02**	9.51E+04	=	**8.94E+04**
F_5	0.00E+00	=	0.00E+00	**4.72E–10**	–	1.39E–09	2.15E–05	+	**1.07E–27**	0.00E+00	=	0.00E+00
F_6	1.48E+01	=	**1.41E+01**	**3.39E+01**	–	3.72E+01	3.61E+01	+	**1.93E+01**	**1.22E+01**	=	1.34E+01
F_7	**6.66E+01**	=	7.38E+01	**6.54E+01**	–	8.58E+01	9.66E+01	+	**7.42E+01**	1.18E+02	+	**1.05E+02**
F_8	2.10E+01	=	2.10E+01	2.10E+01	=	2.10E+01	2.10E+01	=	**2.09E+01**	**2.09E+01**	=	2.10E+01
F_9	2.86E+01	=	**2.85E+01**	3.44E+01	+	**3.11E+01**	2.62E+01	+	**2.21E+01**	**3.10E+01**	=	3.15E+01
F_{10}	**2.40E–02**	=	2.80E–02	**1.60E+00**	–	2.77E+00	4.69E–01	+	**2.68E–02**	**2.88E–01**	–	4.54E–01
F_{11}	3.64E+01	+	**3.19E+01**	**7.48E+01**	=	8.43E+01	1.30E+02	+	**2.10E+01**	0.00E+00	=	0.00E+00
F_{12}	1.09E+02	=	**1.05E+02**	2.00E+02	+	**1.58E+02**	2.08E+02	+	**1.49E+02**	2.95E+02	=	**2.51E+02**
F_{13}	**1.41E+02**	–	1.54E+02	2.07E+02	+	**1.91E+02**	2.41E+02	+	**2.01E+02**	3.48E+02	+	**2.94E+02**
F_{14}	2.90E+03	=	**2.89E+03**	4.71E+03	+	**4.10E+03**	2.14E+03	=	**1.84E+03**	**1.37E+00**	=	1.71E+00
F_{15}	5.05E+03	–	**4.84E+03**	7.02E+03	+	**5.83E+03**	4.37E+03	+	**3.43E+03**	**3.72E+03**	=	3.90E+03
F_{16}	1.91E+00	=	**1.87E+00**	2.55E+00	+	**2.29E+00**	2.37E+00	+	**1.81E+00**	**9.17E–01**	=	9.21E–01
F_{17}	1.14E+02	=	**1.13E+02**	1.89E+02	=	**1.86E+02**	1.72E+02	+	**1.09E+02**	3.04E+01	=	3.04E+01
F_{18}	1.83E+02	=	**1.80E+02**	2.49E+02	=	**2.39E+02**	2.20E+02	+	**1.52E+02**	3.26E+02	=	**2.98E+02**
F_{19}	8.99E+00	=	**8.97E+00**	1.36E+01	+	**1.21E+01**	8.80E+00	+	**5.46E+00**	**3.12E–01**	=	3.55E–01
F_{20}	1.21E+01	=	1.21E+01	1.28E+01	+	**1.26E+01**	1.34E+01	+	**1.20E+01**	1.48E+01	+	**1.43E+01**
F_{21}	2.54E+02	=	**2.29E+02**	**2.94E+02**	=	3.14E+02	**3.05E+02**	=	3.22E+02	**1.69E+02**	=	1.70E+02
F_{22}	3.55E+03	–	**3.09E+03**	4.52E+03	=	**4.28E+03**	2.44E+03	+	**1.05E+03**	2.01E+01	=	**2.00E+01**
F_{23}	5.52E+03	=	**5.40E+03**	7.34E+03	+	**6.31E+03**	4.92E+03	+	**4.17E+03**	5.03E+03	=	**4.90E+03**
F_{24}	**2.65E+02**	–	2.72E+02	**2.61E+02**	–	2.76E+02	2.78E+02	+	**2.62E+02**	2.93E+02	+	**2.86E+02**
F_{25}	**2.92E+02**	=	2.96E+02	**2.96E+02**	=	3.00E+02	3.06E+02	+	**2.84E+02**	3.20E+02	+	**3.13E+02**
F_{26}	2.01E+02	=	2.01E+02	2.02E+02	+	**2.01E+02**	2.19E+02	+	**2.18E+02**	2.01E+02	+	**2.00E+02**
F_{27}	1.05E+03	=	**1.03E+03**	1.11E+03	=	1.11E+03	9.70E+02	+	**8.47E+02**	8.74E+02	=	**7.95E+02**
F_{28}	3.00E+02	=	3.00E+02	3.00E+02	=	3.00E+02	1.22E+03	+	**3.78E+02**	3.35E+02	+	**2.55E+02**
–/=/+	4/23/1			7/10/11			0/3/25			1/20/7		

Note: "+", "–" and "=" show VCS is better than, worse than or similar to CS according to the Wilcoxon signed rank test at $\alpha = 0.05$, respectively

Valian et al. have pointed out that the performance of ICS may deteriorate with an increase in the maximum scaling factor value [17]. In addition, It is worth pointing out that ICS uses a complex equation with four parameters to obtain the varied scaling factor value. In contrast, VICS is simple to get a varied scaling factor without any parameters. In summary, VSF is suitable for the improved versions, and easy to integrate into improving CS in future.

4.6 Explanation

In this section, to give an explanation as to why a random scaling factor is better than a constant one, we divide the range [0, 1] into three sub-ranges [0, 0.33], [0.33, 0.66], and [0.66, 1], named as *low*, *moderate*, and *high*. In addition, we also divide the iteration process of VCS into three stages uniformly, and get VCS-LMH, VCS-HML, VCS-L, and VCS-H according to the F parameter values sampled in the sub-ranges on each stage. In VCS-LMH, the F parameter values are randomly sampled in *low* on the first stage, in *moderate* on the second stage, and in *high* on the third stage. VCS-HML makes the F parameter values be sampled in *high* on the first stage, in *moderate* on the second stage, and in *low* on the third stage. VCS-L and VCS-H get the F parameter values in *low* and *high*, respectively. The four algorithms are tested on the 28 benchmark functions proposed in the CEC 2013 special session on real-parameter optimization [22], and all the parameters are unchanged. The results are listed in Table 9, where the best and the worst results of each benchmark function are marked in boldface and italic, respectively. In addition, Table 9 also shows the results of the Wicoxon's test on VCS with the range [0,1] compared with other four algorithms.

From Table 9, we find that the performance of VCS is sensitive to the scaling factor, and overall in the later stage low range contributes more to the progress of the algorithm. For example, VCS-L and VCS-HML outperform VCS-H and VCS-LMH in terms of the average function error marked in boldface. On the other hand, VCS-HML can obtain better

Table 9 Average function error obtained by five VCS algorithms for CEC 2013 benchmark functions at $D = 30$

Fun	VCS-LMH	VCS-HML	VCS-L	VCS-H	VCS
F_1	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_2	*5.02E+06±1.61E+06*=	5.00E+06±1.35E+06=	**4.03E+06±1.30E+06**–	4.76E+06±1.91E+06=	4.98e+06±1.94e+06
F_3	6.86E+06±7.05E+06+	**2.01E+06±5.71E+06**=	*8.38E+06±9.54E+06*+	3.24E+06±5.82E+06=	3.94e+06±9.79e+06
F_4	1.88E+04±3.20E+03+	**1.57E+04±3.31E+03**=	*1.89E+04±3.47E+03*=	1.81E+04±3.28E+03=	1.67e+04±3.82e+03
F_5	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_6	*1.21E+01±5.03E+00*=	*1.21E+01±5.96E+00*=	**1.12E+01±1.18E+01**=	1.17E+01±2.48E+00=	1.20e+01±6.88e+00
F_7	7.04E+01±1.57E+01=	4.83E+01±1.63E+01–	*7.55E+01±1.46E+01*+	**4.82E+01±1.25E+01**–	6.32e+01±1.50e+01
F_8	**2.09E+01±4.37E–02**=	**2.09E+01±4.36E–02**=	*2.10E+01±4.96E–02*=	*2.10E+01±3.67E–02*=	**2.09e+01±3.69e–02**
F_9	2.92E+01±1.57E+00=	2.86E+01±1.35E+00=	**2.85E+01±1.15E+00**=	*2.97E+01±1.69E+00*+	**2.85e+01±1.47e+00**
F_{10}	*2.51E–02±2.24E–02*=	1.70E–02±1.02E–02=	**1.46E–02±9.91E–03**=	1.82E–02±1.18E–02=	1.52e–02±1.12e–02
F_{11}	*2.79E+01±1.08E+01*+	**1.48E+01±5.27E+00**–	2.31E+01±7.67E+00=	2.10E+01±9.09E+00=	2.03e+01±8.95e+00
F_{12}	*1.23E+02±1.40E+01*+	**7.97E+01±1.84E+01**–	1.10E+02±1.87E+01+	9.79E+01±1.85E+01=	9.76e+01±1.28e+01
F_{13}	1.51E+02±2.19E+01+	**1.23E+02±1.46E+01**–	*1.55E+02±2.11E+01*+	1.33E+02±1.67E+01=	1.34e+02±1.97e+01
F_{14}	*3.00E+03±6.73E+02*+	**2.29E+03±6.70E+02**=	2.58E+03±4.24E+02=	2.78E+03±8.07E+02=	2.56e+03±7.06e+02
F_{15}	5.10E+03±2.99E+02=	5.04E+03±2.52E+02=	**4.72E+03±3.09E+02**–	*5.33E+03±3.34E+02*=	5.00e+03±2.59e+02
F_{16}	1.99E+00±2.60E–01=	2.00E+00±2.34E–01=	**1.84E+00±2.51E–01**=	*2.02E+00±2.92E–01*=	1.94e+00±2.00e–01
F_{17}	*1.14E+02±2.10E+01*=	**9.89E+01±1.33E+01**–	1.10E+02±1.51E+01=	1.11E+02±2.25E+01=	1.13e+02±2.03e+01
F_{18}	*1.91E+02±1.37E+01*+	**1.58E+02±9.35E+00**–	1.66E+02±1.27E+01=	1.78E+02±1.43E+01+	1.65e+02±9.94e+00
F_{19}	*9.01E+00±2.18E+00*=	**6.42E+00±1.52E+00**–	7.37E+00±1.43E+00=	8.85E+00±2.02E+00+	7.85e+00±1.55e+00
F_{20}	*1.22E+01±3.76E–01*=	**1.19E+01±2.75E–01**=	1.22E+01±3.97E–01=	1.20E+01±3.23E–01=	1.20e+01±4.03e–01
F_{21}	2.45E+02±5.24E+01=	*2.67E+02±5.22E+01*=	**2.20E+02±4.08E+01**–	2.59E+02±4.84E+01=	2.55e+02±6.28e+01
F_{22}	*3.24E+03±8.55E+02*=	**2.24E+03±8.09E+02**=	2.87E+03±6.45E+02=	2.86E+03±8.91E+02=	2.78e+03±9.00e+02
F_{23}	5.65E+03±4.26E+02+	**5.37E+03±3.93E+02**=	5.41E+03±5.92E+02=	*5.68E+03±3.55E+02*+	5.42e+03±3.50e+02
F_{24}	*2.65E+02±1.30E+01*=	**2.44E+02±1.32E+01**–	2.62E+02±1.40E+01=	2.45E+02±1.51E+01–	2.59e+02±1.47e+01
F_{25}	*3.02E+02±4.59E+00*+	**2.91E+02±6.10E+00**=	2.97E+02±7.97E+00=	2.96E+02±6.19E+00=	2.94e+02±6.85e+00
F_{26}	2.00E+02±4.72E–02=	2.00E+02±4.30E–02=	2.00E+02±5.02E–02=	2.00E+02±5.39E–02=	2.00e+02±4.89e–02
F_{27}	*1.06E+03±5.44E+01*=	1.03E+03±4.74E+01=	**1.02E+03±1.81E+02**=	1.04E+03±6.06E+01=	1.03e+03±5.91e+01
F_{28}	3.00E+02±0.00E+00=	3.00E+02±3.28E–14=	3.00E+02±5.19E–14=	3.00E+02±8.74E–13=	3.00e+02±6.96e–14
–/=/+	0/19/9	8/20/0	3/21/4	2/21/5	

Note: "+", "–" and "=" show VCS is better than, worse than or similar to other VCSs according to the Wilcoxon signed rank test at $\alpha = 0.05$, respectively

results than those achieved by VCS-L. This shows that for most functions at the beginning the high range contributes more to the progress of the algorithm while in later stages low range contributes more. However, VCS-HML cannot perform better on a few functions like F_6, F_{21}. But VCS with the range [0,1] does not obtain the worst compared with the other four algorithms. Additionally, in terms of the total number of "–/=/+" in the last line of the table, compared with VCS-HML, VCS on the range [0,1] is overall a simple and reasonable choice, but not the optimal one. Nevertheless, VCS-HML also utilizes the random scaling factor on different stages. Therefore, the random scaling factor is more suitable for the progress of the algorithm than a constant one. This is why the CS algorithm with a random scaling factor is overall better than CS with a constant one.

5 Related work

Since Yang and Deb developed CS for optimization problems, various improved CS have been proposed.

Some improvement efforts have been made to solve the traveling salesman problem [26], graph coloring problem [27], permutation flow shop scheduling problems [28], and multi-object optimization problems [29–31].

Some attempts have been made to combine CS with other evolutionary algorithms for optimization problems. Wang et al. [24] proposed a CSPSO algorithm by making use of the advantages of particle swarm optimization (PSO) and CS. Under the framework of the proposed version, the population was optimized by PSO, and then the current best individuals were input into CS. Ghodrati et al. [32] introduced another version of the hybrid CS algorithm with PSO, where velocity update and position update operators were added between Lévy flights random walk and biased/selective random walk. Li et al. [33, 34] implemented OLCS for continuous function optimization problems and parameter estimation problems. Their approach used an orthogonal learning strategy to enhance the exploitation ability of CS. Srivastava et al. [35] presented an effective approach for test data generation using CS and tabu search. Their approach combines the CS strength of converging to the solution in minimal time along with the Tuba mechanism of backtracking from a local optima by Lévy flights. Wang et al. [36] proposed a hybrid CS algorithm with differential evolution, where a mutation and crossover operators are used to replace the Lévy flights random walk to generate new solutions. Layeb et al. [37] presented a quantum inspired CS algorithm for solving combinatorial problems. The proposed algorithm imported quantum computing principles such as quantum-bit representation, measure operation, and quantum mutation into CS to enhance the diversity and the performance of algorithm. Babukartik et al. [38] presented a hybrid algorithm for job scheduling by combining the merits of ACO and CS. Hu et al. [39] and Zheng et al. [40] introduced a cooperative co-evolutionary based CS.

Other works focus on the step-size of random walk. Walton et al. [41] proposed a modified CS algorithm with modifying the step size of Lévy flights by adding information exchange between current solutions and the best solution. Tuba et al. [42] introduced a variant of CS, where the step size was determined by the sorted fitness matrix rather than only the permuted fitness matrix. Mishra [43] introduced cuckoo-host co-evolution, where the step size is calculated from two different populations with different parameters of Lévy flights. Valian et al. [16,17] proposed ICS, an improved CS algorithm with two self-adaptive parameters: step size of Lévy flights random walk and the fraction probability. Wang et al. [25] brought the dimension by dimension improvement strategy to CS where a new step size was generated in BSRW.

6 Conclusion and future work

In this study, we used a simple strategy to set the scaling factor value in LFRW based on a varied scaling factor (VSF). A completely random strategy was utilized to sample a varied scaling factor value at each iteration process for each optimization problem. Extensive experiments were carried on three groups of benchmark functions to test the performances of CS with a varied scaling factor. The results of each group benchmark shows that, for most benchmarks, CS with a varied scaling factor outperform the standard CS, which uses a constant scaling factor. The VSF strategy was also used to test the suitability to the improved CS versions, i.e., ICS, CSPSO, and DDICS. The results suggest that VSF is suitable for the improved versions, and can be combined to easily improve CS in future.

We also explained why VCS overall outperforms CS. This is because at the beginning the high scaling factor contributes to the progress of algorithm more, and in the later stage a low scaling factor contributes more for most functions. Therefore, we plan to investigate a self-adaptive version of VSF using learning methods. Moreover, our scalability study shows the advantage of CS with a varied scaling factor over CS with a constant one is stable as the dimensionality of problem increases. It will be interesting to investigate VCS for large

scale optimization problems. In future, we plan to make further improvements to VCS to make it comparable with other types of algorithms.

Acknowledgements The authors thank the anonymous reviewers for their very helpful and constructive comments and suggestions. This work was supported by the NSFC Joint Fund with Guangdong of China (Key Project U1201258), the Shandong Natural Science Funds for Distinguished Young Scholar (JQ201316) and the Natural Science Foundation of Fujian Province of China (2013J01216 and 2014J01219).

Appendixes

Appendix A

The test suite that we have used for different experiments consists of 18 common benchmark functions [18–20], and 53 functions respectively proposed in CEC 2005 and 2013 special session on real-parameter optimization [21, 22]. The 18 common benchmark functions were as follows.

F_{sph}: Sphere's Function
F_{ros}: Rosenbrock's Function
F_{ack}: Ackley's Function
F_{grw}: Griewank's Function
F_{ras}: Rastrigin's Function
F_{sch}: Generalized Schwefel's Problem 2.26.
F_{sal}: Salomon's Function
F_{pn1}: Generalized Penalized Function 1
F_{pn2}: Generalized Penalized Function 2
F_{pow}: Powell's Function
F_{zak}: Zakharov's Function
F_{lvy}: Levy's Function
F_{sus}: Sum Squares Function
F_{dap}: Dixon and Price Function
F_{scw}: Schwefel's Problem 2.22
F_{sce}: Schwefel's Problem 2.21
F_{ste}: Step's Function
F_{qua}: Quartic Function

References

1. Nocedal J. and Wright S.J. Numerical Optimization. 2nd ed. Springer Press, 2006
2. Holland J.H. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. U Michigan Press, 1975
3. Storn R. and Price K. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization, 1997, 11(4): 341–359
4. Dorigo M, Maniezzo V, and Colorni A. Ant system: optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 1996, 26(1): 29–41
5. Eberhart R. and Kennedy J. A new optimizer using particle swarm theory. In: Proceedings of the 6th International Symposium on Micro Machine and Human Science, 1995, 39–43
6. Kennedy J, Eberhart R. Particle swarm optimization. In: Proceedings of IEEE International Conference on Neural Networks. 1995, 1942–1948
7. Karaboga D. An idea based on honey bee swarm for numerical optimization. Technical Report-tr06. 2005
8. Yang X S. A new metaheuristic bat-inspired algorithm. In: Proceedings of Nature Inspired Cooperative Strategies for Optimization. 2010, 65–74
9. Yang X S. Nature-Inspried Metaheuristic Algorithms. 2nd ed. Luniver Press, 2010
10. Geem Z W, Kim J H, and Loganathan G V. A new heuristic optimization algorithm: harmony search. Simulation, 2001, 76(2): 60–68
11. Simon D. Biogeography-based optimization. IEEE Transactions on Evolutionary Computation, 2008, 12(6): 702–713
12. Yang X S and Deb S. Cuckoo search via lévy flights. In: Proceedings of World Congress on Nature & Biologically Inspired Computing, 2009, 210–214
13. Yang X S and Deb S. Engineering optimisation by cuckoo search. International Journal of Mathematical Modelling and Numerical Optimisation, 2010, 1(4): 330–343
14. Zhan Z H, Zhang J, Li Y, and Shi Y H. Adaptive particle swarm optimization. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 2009, 39(6): 1362–1381
15. Das S, Suganthan P N. Differential evolution: a survey of the state-of-the-art. IEEE Transactions on Evolutionary Computation, 2011, 15(1): 4–31
16. Valian E, Mohanna S, Tavakoli S. Improved cuckoo search algorithm for feedforward neural network training. International Journal of Artificial Intelligence and Applications, 2011, 2(3): 36–43
17. Valian E, Mohanna S, Tavakoli S. Improved cuckoo search algorithm for global optimization. International Journal of Communications and Information Technology, 2011, 1(1): 31–44
18. Yao X, Liu Y, Lin G M. Evolutionary programming made faster. IEEE Transactions on Evolutionary Computation, 1999, 3(2): 82–102
19. Noman N, Iba H. Accelerating differential evolution using an adaptive local search. IEEE Transactions on Evolutionary Computation, 2008, 12(1): 107–125
20. Karaboga D, Akay B. A comparative study of artificial bee colony algorithm. Applied Mathematics and Computation, 2009, 214(1): 108–132
21. 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. KanGAL Report 2005005. 2005
22. Liang J J, Qu B Y, Suganthan P N, Hernández-Díaz A G. Problem Definitions and Evaluation Criteria for the CEC 2013 Special Session on Real-parameter Optimization. Technical Report 201212. 2013
23. Wang Y, Cai Z X, and Zhang Q F. Differential evolution with composite trial vector generation strategies and control parameters. IEEE Transactions on Evolutionary Computation, 2011, 15(1): 55–66
24. Wang F, Luo L G, He X S, Wang Y. Hybrid optimization algorithm of pso and cuckoo search. In: Proceedings of the 2nd International Conference on Artificial Intelligence, Management Science and Electronic

Commerce, 2011, 1172–1175

25. Wang L J, Yin Y L, Zhong Y W. Cuckoo search algorithm with dimension by dimension improvement. Journal of Software, 2013, 24(11): 2687–2698
26. Ouyang X X, Zhou Y Q, Luo Q F, Chen H. A novel discrete cuckoo search algorithm for spherical traveling salesman problem. Applied Mathematics and Information Sciences, 2013, 7(2): 777–784
27. Zhou Y Q, Zheng H Q, Luo Q F, Wu J Z. An improved cuckoo search algorithm for solving planar graph coloring problem. Applied Mathematics and Information Sciences, 2013, 7(2): 785–792
28. Marichelvam M K. An improved hybrid cuckoo search metaheuristics algorithm for permutation flow shop scheduling problems. International Journal of Bio-Inspired Computation, 2012, 4(4): 200–205
29. Yang X S, Deb S. Multiobjective cuckoo search for design optimization. Computers and Operations Research, 2013, 40(6): 1616–1624
30. Chandrasekaran K, Simon S P. Multi-objective scheduling problem: hybrid approach using fuzzy assisted cuckoo search algorithm. Swarm and Evolutionary Computation, 2012, 5: 1–16
31. Marichelvam M K, Prabaharan T, Yang X S. Improved cuckoo search for hybrid flow shop scheduling problems to minimize makespan. Applied Soft Computing, 2014, 19: 93–101
32. Ghodrati A, Lotfi S. A hybrid cs/pso algorithm for global optimization. Lecture Notes in Computer Science, 2012, 89–98
33. Li X T, Yin M H. Parameter estimation for chaotic systems using the cuckoo search algorithm with an orthogonal learning method. Chinese Physics B, 2012, 21(5): 113–118
34. Li X T, Wang J N, Yin M H. Enhancing the performance of cuckoo search algorithm using orthogonal learning method. Neural Computing and Applications, 2014, 24(6): 1233–1247
35. Srivastava P R, Khandelwal R, Khandelwal S, Kumar S, Ranganatha S S. Automated test data generation using cuckoo search and tabu search algorithm. Journal of Intelligent Systems, 2012, 21(2): 195–224
36. Wang G G, Guo L H, Duan H, Liu L, Wang H, Wang B. A hybrid meta-heuristic de/cs algorithm for ucav path planning. Journal of Information and Computational Science, 2012, 5(2012): 4811–4818
37. Layeb A, Boussalia S R. A novel quantum inspired cuckoo search algorithm for bin packing problem. International Journal of Information Technology and Computer Science, 2012, 4(5): 58–67
38. Babukartik R G, Dhavachelvan P. Hybrid algorithm using the advantage of aco and cuckoo search for job scheduling. International Journal of Information Technology Convergence and Services, 2012, 2(4): 25–34
39. Hu X X, Yin Y L. Cooperative co-evolutionary cuckoo search algorithm for continuous function optimization problems. Pattern Recognition and Aritificial Intelligence, 2013, 26(11): 1041–1049
40. Zheng H Q, Zhou Y Q. A cooperative coevolutionary cuckoo search algorithm for optimization problem. Journal of Applied Mathematics, 2013
41. Walton S, Hassan O, Morgan K, Brown M R. Modified cuckoo search: a new gradient free optimisation algorithm. Chaos, Solitons and Fractals, 2011, 44(9): 710–718
42. Tuba M, Subotic M, Stanarevic N. Modified cuckoo search algorithm for unconstrained optimization problems. In: Proceedings of the 5th European Conference on European Computing Conference, 2011, 263–268
43. Mishra S K. Global optimization of some difficult benchmark functions by host-parasite co-evolutionary algorithm. Economics Bulletin, 2013, 33(1): 1–18

Lijin Wang received his BS in 2000 and his MS in 2005 from Fujian Agriculture and Forestry University, China, and his PhD in 2008 from Beijing Forestry University, China. He is currently a post-doctoral fellow with the School of Computer Science and Technology, Shandong University, China. He is also an associate professor with the College of Computer and Information Science, Fujian Agriculture and Forestry University. His research interests include evolutionary algorithms and intelligent information processing.

Yilong Yin received his PhD in 2000 from Jilin University, China. From 2000 to 2002, he worked as a post-doctoral fellow in the Department of Electronics Science and Engineering, Nanjing University, China. He is currently the Director of MLA Group and a Professor of the School of Computer Science and Technology, Shandong University, China. His research interests include machine learning, data mining, and computational medicine.

Yiwen Zhong received his MS in 2002 and his PhD in 2005 from Zhejiang University, China. He is currently a professor with the College of Computer and Information Science, Fujian Agriculture and Forestry University, China. From 2007 to 2008, and from 2011 to 2012, he was a visiting scholar in Indiana University, USA. His research interests include computational intelligence, data visualization, and bioinformatics.