Second Order Differential Evolution for Constrained Optimization

Xinchao Zhao[1](✉), Jia Liu[1], Junling Hao[2], Jiaqi Chen[1], and Xingquan Zuo[3]

[1] School of Science, Beijing University of Posts and Telecommunications, Beijing 100876, China
xcbupt@126.com

[2] School of Statistics, University of International Business and Economics, Beijing 10029, China

[3] School of Computers, Beijing University of Posts and Telecommunications, Beijing 100876, China

Abstract. In this paper, second order differential evolution (SODE) algorithm is considered to solve the constrained optimization problems. After offspring are generated by the second order differential evolution, the ε constrained method is chosen for selection in this paper. In order to show that second order differential vector is better than differential vector in solving constrained optimization problems, differential evolution (DE) with the ε constrained method is used for performance comparison. The experiments on 12 test functions from IEEE CEC 2006 demonstrate that second order differential evolution shows better or at least competitive performance against DE when dealing with constrained optimization problems.

Keywords: Constrained optimization · Evolutionary algorithm · Differential evolution · SODE

1 Introduction

Constrained optimization problems (COPs) are mathematical programming problems frequently encountered in the disciplines of science and engineering application. Evolutionary algorithm is usually used to deal with constrained optimization problems due to its excellent performance, but it is essentially an unconstrained optimization evolutionary algorithm, which must be combined with constraint handing technique to solve the constrained optimization problem. Evolutionary algorithm and an appropriate constraint handing technique are combined to form a complete constrained evolutionary optimization algorithm. Among all the evolutionary algorithms, differential evolution (DE) [1] is one of the most important problem solvers.

Differential evolution was introduced by Price and Storn in 1997 [1], and has numerous attractive advantages. First of all, its structure is simple. In addition, it includes few control parameters. More importantly, its search ability, such as, higher search efficiency, higher robustness and lower computational complexity, has been demonstrated in many real-world applications [2, 3].

© Springer Nature Switzerland AG 2019

Due to the above advantages, DE has been frequently applied to solve COPs. Many DE variants optimization have been tailored to tackle COPs [4]. However, few current studies investigate second order differential evolution (SODE) [5] for constrained optimization. To illustrate that second order differential vector is better than differential vector in solving constrained optimization problems, the ε constrained second order differential evolution (εSODE) is proposed in this paper.

The rest of the paper is organized as follows. In Sect. 2 some preliminary knowledge are presented. The proposed εSODE is shown in Sect. 3. The experimental and analytic results are presented in Sect. 4, and the last Section concludes the paper.

2 Preliminary Knowledge

2.1 Constrained Optimization Problems (COPs)

Without loss of generality, a COP can be described as follows:

$$\begin{array}{ll} \text{minimize} & f(\vec{x}), \vec{x} = (x_1, \ldots, x_D) \in S \\ \text{subject to:} & \mathrm{g}_j(\vec{x}) \leq 0, j = 1, \ldots, l \\ & \mathrm{h}_j(\vec{x}) = 0, j = l+1, \ldots, m \\ & l_i \leq x_i \leq u_i, i = 1, \ldots, D \end{array} \tag{1}$$

where $\vec{x} = (x_1, \ldots, x_D)$ is an D dimensional vector, $f(\vec{x})$ is the objective function, $\mathrm{g}_j(\vec{x}) \leq 0$ and $\mathrm{h}_j(\vec{x}) = 0$ are l inequality constraints and $m - l$ equality constraints, respectively. l_i and u_i are the lower and upper bounds of the i-th decision variable x_i, respectively.

The decision space S is an D-dimensional rectangular space in R^n, in which every point satisfies the upper and lower bound constraints.

The feasible region Ω is defined by the l inequality constraints $\mathrm{g}_j(\vec{x})$ and the $(m - l)$ equality constraints $\mathrm{h}_j(\vec{x})$. Any point $\vec{x} \in \Omega$ is called a feasible solution; otherwise, $\vec{x}$ is an infeasible solution. The aim of solving COPs is to locate the optimum in the feasible region.

Usually, the degree of constraint violation of individual $\vec{x}$ on the j-th constraint is calculated as follows:

$$G_j(\vec{x}) = \begin{cases} \max(0, g_j(\vec{x})), & 1 \leq j \leq l \\ \max(0, \left|h_j(\vec{x})\right| - \delta), & l+1 \leq j \leq m \end{cases} \tag{2}$$

$$G(\vec{x}) = \sum_{j=1}^{m} G_j(\vec{x}) \tag{3}$$

where $\mathrm{G}_j(\vec{x})$ is the degree of constraint violation on the j-th constraint. δ is a positive tolerance value.

2.2 ε Constrained Method

The ε constrained method was proposed by Takahama and Sakai [6, 7]. The core idea of this method is to divide the individual-based constraint violation degree into different regions by artificially setting the ε value, and in different regions, the feasible solution and the infeasible solution adopt different evaluation methods respectively. The details on how to deal with the constraints, especially in constraint evolutionary optimization, can be found in the references [10, 11].

When comparing two individuals, say $\vec{x}_{\mathrm{i}}^{k}$ and $\vec{x}_{j}^{k}$, $\vec{x}_{\mathrm{i}}^{k}$ is better than $\vec{x}_{j}^{k}$ if and only if the following conditions are satisfied:

$$\begin{cases} f(\vec{x}_i) < f(\vec{x}_j), & if \quad G(\vec{x}_i) \le \varepsilon \wedge G(\vec{x}_j) \le \varepsilon \\ f(\vec{x}_i) < f(\vec{x}_j), & if \quad G(\vec{x}_i) = G(\vec{x}_j) \\ G(\vec{x}_i) < G(\vec{x}_j), & \quad otherwise \end{cases} \tag{4}$$

$$\varepsilon(\mathrm{k}) = \begin{cases} \varepsilon(0)(1 - \frac{k}{T_c})^{cp}, & 0 < k < T_c \\ 0, & k \ge T_c \end{cases} \tag{5}$$

$$cp = -\frac{\log \varepsilon(0) + \alpha}{\log\left(1 - \frac{k}{T_c}\right)} \tag{6}$$

where ε in Eq. (4) is controlled by Eqs. (5) and (6) and k is the current generation. $\varepsilon(0)$ is the maximum degree of constraint violation of the initial population. T_c is the maximum generation number. According to [8], α is set to 6.

2.3 Classical Differential Evolution

DE consists of four stages, i.e., initialization, mutation, crossover, and selection.

In the initialization stage, *NP* individuals are usually randomly generated from the decision space.

In the mutation operation stage, DE creates a mutant vector $\vec{v}_i$ for each sample $\vec{x}_i$. The two extensively used mutation operators (called DE/rand/1 and DE/best/1) are introduced as follows.

DE/rand/1:

$$\vec{v}_i = \vec{x}_{r_1} + F(\vec{x}_{r_2} - \vec{x}_{r_3}), \quad i = 1, 2, \ldots, NP \tag{7}$$

DE/best/1:

$$\vec{v}_i = \vec{x}_{best} + F(\vec{x}_{r_1} - \vec{x}_{r_2}), \quad i = 1, 2, \ldots, NP \tag{8}$$

where r_1, r_2, r_3 are three random and mutually different integers chosen from [1, NP], and F is a scaling factor and is used to control the amplification of the differential vector.

In the crossover operation stage, the crossover is applied to the parent individual $\vec{x}_i^k$ and its mutant vector $\vec{v}_i^k$. Then a trial vector $\vec{u}_i^k$ is produced.

$$u_{i,\mathrm{j}}^k = \begin{cases} v_{i,\mathrm{j}}^k, & rand \leq CR \quad or \quad j = j_{rand} \\ x_{i,\mathrm{j}}^k, & otherwise \end{cases} \tag{9}$$

where *rand* is a uniformly distributed random number between 0 and 1. j_{rand} is a random integer in [*1, NP*]. *CR* is the crossover probability.

In the selection operation stage, a comparison is conducted between parent individual $\vec{x}_i^k$ and trial vector $\vec{u}_i^k$ to select the better one among them.

$$x_i^{k+1} = \begin{cases} u_{\mathrm{i}}^k, & f(u_{\mathrm{i}}^k) \leq f(x_{\mathrm{i}}^k) \\ x_{\mathrm{i}}^k, & otherwise \end{cases} \tag{10}$$

2.4 Second Order Differential Evolution

SODE is also composed of four stages, i.e., initialization, mutation, crossover, and selection. Except for the mutation stage, the other stages are the same as those of DE.

In the mutation operation stage, the usually and widely used mutation operations, DE/rand/1, is adopted as analytic model strategies in this paper. In order to efficiently utilize the direction information and the search status of the current population, the second order difference vector mechanism, which is based on the classical mutation strategies, is indicated as in Eqs. (11)–(15).

$$d^k = x_{r1}^k - x_{r2}^k \tag{11}$$

$$d_1^k = x_{r3}^k - x_{r4}^k \tag{12}$$

$$d_1^k = x_{best}^k - x_{r4}^k \tag{13}$$

$$d_2^k = x_{r5}^k - x_{r_6}^k \tag{14}$$

$$d_r^k = d^k + \lambda(d_1^k - d_2^k) \tag{15}$$

where r1, r2, r3, r4, r5, r6 are different random integers in [*1, NP*]. λ is set as 0.1, which is discussed in reference [5]. x_{best}^k is the best vector in generation *k*. d_1^k in Eqs. (12) and (13) sets the same variable to different values, which is combined with Eq. (15) to produce different algorithms.

$(d_1^k - d_2^k)$ in Eq. (15) is the second order difference vector. The mutant vector v_i^k is generated as follows:

$$v_i^k = x_{r_7}^k + F \cdot d_r^k \tag{16}$$

where *F* is a scaling parameter and is set as 0.5. r7 is a random integer in [*1, NP*].

In this paper, two composing patterns of d_r^k will be used. The first form of d_r^k consists of Eqs. (12) and (14). The second form of d_r^k consists of Eqs. (13) and (14). The first form and the second form based SODE, are denoted as SODErand and SODEbest, respectively.

3 The εConstrained Second Order Differential Evolution

Evolutionary algorithm is a general optimization framework. In order to solve the constraint optimization problems, it must be combined with the appropriate constraint handing technique. Evolutionary algorithm and the constraint handing technique are combined to form a complete constrained evolutionary optimization algorithm. Therefore, while retaining the idea of SODE, this paper adds an ε constrained method to select descendants, which is denoted as εSODErand and εSODEbest, respectively.

3.1 ε SODErand

In this paper, εSODErand uses SODErand mentioned in Sect. 2 to combine with ε constrained method. Its details is given in Algorithm 1.

Algorithm 1: εSODErand

```
   Input: NP, maxFES, F, CR
1  $k$=1; // the generation number
2  Create a random initial population $x_k^i \; \forall i, i = 1, \ldots, NP$;
3  $FES$=$NP$; // FES is the number of fitness evaluations;
4  Get the ε value of the ε constrained method according to Eq.(5);
5  for $i = 1 : NP$
6      Use SODErand to generate a trial vector $u_k^i$;
7      Apply the ε constrained method to compare $x_k^i$ and $u_k^i$;
8      Use the better one for the next iteration;
9      $FES$=$FES$+1;
10 end for
11 $k$=$k$+1;
12 Stopping Criterion: If $FES \geq maxFES$, then stop and output the best solution, else go to Step 4.
```

3.2 A Subsection Sample

The difference between εSODEbest and εSODErand is the mutation operation stage. The former uses SODEbest mentioned in Sect. 2 to combine with ε constrained method. Its details is given in Algorithm 2.

Algorithm 2: εSODEbest

```
   Input: NP, maxFES, F, CR
1  k=1; // the generation number
2  Create a random initial population $x_k^i$ $\forall i, i = 1, \ldots, NP$;
3  FES=NP; // FES is the number of fitness evaluations;
4  Get the ε value of the ε constrained method according to Eq.(5);
5  for $i = 1 : NP$
6      Use SODEbest to generate a trial vector $u_k^i$;
7      Apply the ε constrained method to compare $x_k^i$ and $u_k^i$;
8      Use the better one for the next iteration;
9      FES=FES+1;
10 end for
11 k=k+1;
12 Stopping Criterion: If $FES \geq maxFES$, then stop and output the best solution, else go to Step 4.
```

4 Experimental Results and Algorithmic Analysis

4.1 CEC2006 Benchmark Functions

In order to check the performance of the proposed algorithms εSODEbest and εSODErand, 12 functions are selected from IEEE CEC2006 [9] as the preliminary test suite, which is described in Table 1.

Table 1. CEC2006 benchmark functions.

Prob.	D	Type of function	LI	NI	LE	NE	Active
g_{01}	20	Nonlinear	9	0	0	0	1
g_{02}	10	Polynomial	0	0	0	1	1
g_{03}	10	Quadratic	3	5	0	0	6
g_{04}	7	Polynomial	0	4	0	0	2
g_{05}	8	Linear	3	3	0	0	6
g_{06}	2	Quadratic	0	0	1	1	1
g_{07}	5	Nonlinear	0	0	3	3	3
g_{08}	10	Nonlinear	0	0	0	0	3
g_{09}	5	Nonlinear	4	34	0	0	4
g_{10}	6	Nonlinear	0	0	4	4	4
g_{11}	15	Nonlinear	0	5	0	0	0
g_{12}	7	Linear	*0*	1	5	5	6

Where D is the number of decision variables, LI is the number of linear inequality constraints, NI the number of nonlinear inequality constraints, LE is the number of linear equality constraints and NE is the number of nonlinear equality constraints, *active* is the number of active constraints at $\vec{x}$.

4.2 Experimental Settings

In order to show the performance of two proposed algorithms, εDE is chosen to compare with them. In this paper, εDE means to change SODErand in step6 of Algorithm 1 to DE. In addition to the special instructions, the parameters are set as follows.

- Independent running number: RUN = 25.
- Population size: NP = 50.
- Maximum number of function evaluations: maxFES = 240000.

Both parameters, F and CR are initialized to 0.5. Parameter λ is set as 0.1.

It is noteworthy that the feasible rate, i.e., if the algorithm cannot consistently provide feasible solutions in all 25 runs, the running percentage of finding at least one feasible solution is recorded. So, based on the feasible rate, the experimental results are divided into two parts. One part is that the feasible rate of all three algorithms is 100%, and the other part is that there are some algorithms not 100% feasible. The former is called part 1, and the latter part is called part 2.

4.3 Experimental Comparison for Part 1

In the 12 functions, the solutions of 6 functions, include g_{01}, g_{02}, g_{03}, g_{07}, g_{09}, g_{11}, are consistent feasible over all 25 runs. All the final experimental results of the 6 functions over 25 runs, are statistically listed in Table 2, which includes the statistical items of the minimum final result (min), the median final result (median), the average final result (mean) and the standard deviation (std) in multiple runs.

Table 2. Comparison of the results based on CEC2006 functions.

Prob.	*Items*	*εSODErand*	*εSODEbest*	*εDE*
g_{01}	min	**−8.0361E−01**	−8.0360E−01	−8.0360E−01
	mean	−8.0213E−01	**−8.0315E−01**	−8.0112E−01
	median	**−8.0360E−01**	−8.0359E−01	−8.0359E−01
	std	5.0795E−03	**2.1990E−03**	6.9590E−03
g_{02}	min	**−1.0003E+00**	**−1.0003E+00**	**−1.0003E+00**
	mean	−9.9920E−01	**−9.9972E−01**	−9.9936E−01
	median	−9.9948E−01	**−9.9989E−01**	−9.9960E−01
	std	1.1884E−03	**5.4964E−04**	1.4034E−03
g_{03}	min	**5.1265E+03**	**5.1265E+03**	**5.1265E+03**
	mean	5.1273E+03	**5.1265E+03**	5.1266E+03
	median	**5.1265E+03**	**5.1265E+03**	**5.1265E+03**
	std	3.6150E+00	**5.7052E−02**	5.6944E−01
g_{07}	min	5.3942E−02	**5.1196E−02**	5.3942E−02
	mean	5.4130E−02	**5.3850E−02**	5.3956E−02
	median	**5.3942E−02**	**5.3942E−02**	**5.3942E−02**
	std	9.4131E−04	5.5928E−04	**5.8672E−05**

(*continued*)

Table 2. (*continued*)

Prob.	*Items*	*εSODErand*	*εSODEbest*	*εDE*
g_{09}	min	**−1.9052E+00**	**−1.9052E+00**	**−1.9052E+00**
	mean	**−1.9052E+00**	**−1.9052E+00**	−1.6578E+00
	median	**−1.9052E+00**	**−1.9052E+00**	−1.4307E+00
	std	**4.5325E−16**	9.0649E−16	2.4127E−01
g_{11}	min	3.2972E+01	**3.2910E+01**	3.2985E+01
	mean	3.3166E+01	**3.3123E+01**	3.3216E+01
	median	3.3148E+01	**3.3115E+01**	3.3194E+01
	std	1.2913E−01	**8.6868E−02**	1.4633E−01

Observed from Table 2 two proposed algorithms shows even better results in terms of reliability and accuracy when comparing with εDE. Comparatively speaking, εSODErand performs a little worse than that of εSODEbest. The fact of εSODErand and εSODEbest being better than εDE indicates that SODE has more information utilizing ability for solving constrained optimization problems than DE.

4.4 Boxplot Performance Comparison

To compare the performance of algorithms better, the boxplot analysis is taken for perusal. It can easily show the empirical distribution of all the final data in multiple runs pictorially. In order to better analyze the results, the functions are divided into two parts: Figs. 1 and 2.

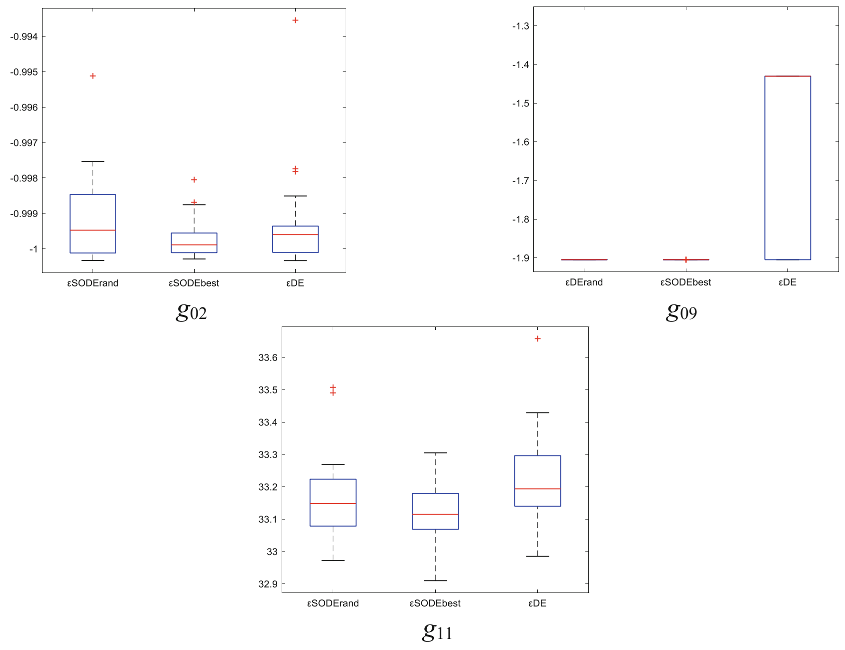

Fig. 1. Performance comparison on g_{02}, g_{09}, g_{11}.

Boxplots are shown in Fig. 1, which shows that both medians and interquartile range of εSODEbest are comparatively lower. The median of εSODErand is lower than that of εDE except for g02 function.

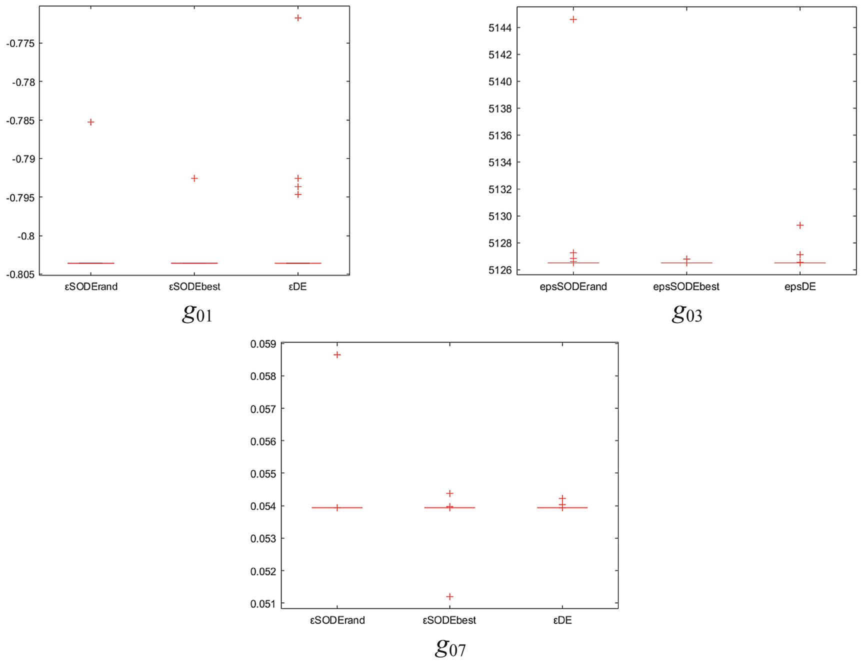

Fig. 2. Performance comparison on g_{01}, g_{03}, g_{07}.

The performance comparison of g01, g03 and g07 are shown in Fig. 2. Observed from Fig. 2, the results of the three algorithms are relatively stable except for some outliers.

4.5 Experimental Comparison for Part 2

In the 12 functions, the solutions of 6 functions, include g_{04}, g_{05}, g_{06}, g_{08}, g_{10}, g_{12}, are inconsistent feasible. To get a more accurate solution, the 6 functions will be run 50 times to get results. The feasible rate, i.e., percentage of runs where at least one feasible solution is found, is recorded if an algorithm fails to consistently provide feasible solutions over all 50 runs. All the final experimental results of the 6 functions are statistically listed in Table 3.

Table 3. Comparison of the results based On CEC2006 functions.

Prob.	*Items*	*SODErand*	*SODEbest*	*DE*
g_{04}	min	90%	**6.8063E+02**	72%
	mean		**6.8063E+02**	
	median		**6.8063E+02**	
	std		**1.7807E−05**	
g_{05}	min	7.0571E+03	**7.0564E+03**	92%
	mean	7.0682E+03	**7.0646E+03**	
	median	7.0650E+03	**7.0645E+03**	
	std	1.4941E+01	**6.9106E+00**	
g_{06}		8%	8%	0%
g_{08}		4%	0%	0%
g_{10}		4%	4%	0%
g_{12}		96%	98%	90%

Observed from Table 3, two proposed algorithms shows even better results in terms of feasible rate when comparing with εDE. Especially the functions g_{06}, g_{08}, g_{10}, the solutions obtained by εDE are not feasible, but two proposed algorithms significantly improve this phenomenon. This shows that SODE is more suitable for solving constrained optimization problems than DE.

5 Conclusions

A simple modification to SODE is proposed to solve COPs in this paper. The idea is that after producing offspring by the second order differential evolution, the ε constrained method is chosen for selection. εSODErand and εSODEbest on the basis of SODE are proposed. To test the effect of the proposed strategies, they are verified on CEC2006 Benchmark Functions. Experimental results show that second order difference vector has a certain role in dealing with constraint optimization problems. This idea can be hybridized with any DE variants, even for all the swarm intelligence and evolutionary computing methods. So, how to even better utilize the second order difference vector deserves further research.

Acknowledgments. This research is supported by National Natural Science Foundation of China (71772060, 61873040, 61375066). We will express our awfully thanks to the Swarm Intelligence Research Team of BeiYou University.

References

1. Storn, R., Price, K.: Differential evolution: a simple and efficient heuristic for global optimization over continuous spaces. J. Global Optim. **11**(4), 341–359 (1997)
2. Harno, H.G., Petersen, I.R.: Synthesis of linear coherent quantum control systems using a differential evolution algorithm. IEEE Trans. Autom. Control **60**(3), 799–805 (2015)

3. Chiu, W.-Y.: Pareto optimal controller designs in differential games. In: 2014 CACS International Automatic Control Conference (CACS), pp. 179–184. IEEE (2014)
4. Wei, W., Wang, J., Tao, M.: Constrained differential evolution with multiobjective sorting mutation operators for constrained optimization. Appl. Soft Comput. **33**, 207–222 (2015)
5. Zhao, X., Xu, G., Liu, D., Zuo, X.: Second order differential evolution algorithm. CAAI Trans. Intell. Technol. **2**, 96–116 (2017)
6. Takahama, T., Sakai, S.: Constrained optimization by the ε constrained differential evolution with an archive and gradient-based mutation. In: IEEE Congress on Evolutionary Computation, pp. 1–9. IEEE (2010)
7. Takahama, T., Sakai, S.: Efficient constrained optimization by the constrained rank based differential evolution. In: 2012 IEEE Congress on Evolutionary Computation, pp. 1–8. IEEE (2012)
8. Wang, B.C., Li, H.X., Li, J.P., Wang, Y.: Composite differential evolution for constrained evolutionary optimization. IEEE Trans. Syst. Man Cybernet. Syst. **99**, 1–14 (2018)
9. Liang, J., et al.: Problem definitions and evaluation criteria for the CEC 2006 special session on constrained real-parameter optimization. J. Appl. Mech. **41**(8), 8–31 (2006)
10. Li, Z.Y., Huang, T., Chen, S.M., Li, R.F.: Overview of constrained optimization evolutionary algorithms. J. Softw. **28**(6), 1529–1546 (2017)
11. Wang, Y., Cai, Z.X., Zhou, Y.R., Xiao, C.X.: Constrained optimization evolutionary algorithms. J. Softw. **20**(1), 11–29 (2009)