Contents lists available at ScienceDirect

Information Sciences

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

Enhancing particle swarm optimization using generalized opposition-based learning

Hui Wang[a,b,*], Zhijian Wu[a], Shahryar Rahnamayan[c], Yong Liu[d], Mario Ventresca[e]

[a] State Key Lab of Software Engineering, Wuhan University, Wuhan 430072, PR China
[b] School of Information Engineering, Nanchang Institute of Technology, Nanchang 330099, PR China
[c] Faculty of Engineering and Applied Science, University of Ontario Institute of Technology (UOIT), 2000 Simcoe Street North, Oshawa, Canada ON L1H 7K4
[d] University of Aizu, Tsuruga, Ikki-machi, Aizu-Wakamatsu, Fukushima 965-8580, Japan
[e] Centre for Pathogen Evolution, Department of Zoology, University of Cambridge, Downing Street, Cambridge, UK

ARTICLE INFO

Article history:
Available online 31 March 2011

ABSTRACT

Particle swarm optimization (PSO) has been shown to yield good performance for solving various optimization problems. However, it tends to suffer from premature convergence when solving complex problems. This paper presents an enhanced PSO algorithm called GOPSO, which employs generalized opposition-based learning (GOBL) and Cauchy mutation to overcome this problem. GOBL can provide a faster convergence, and the Cauchy mutation with a long tail helps trapped particles escape from local optima. The proposed approach uses a similar scheme as opposition-based differential evolution (ODE) with opposition-based population initialization and generation jumping using GOBL. Experiments are conducted on a comprehensive set of benchmark functions, including rotated multimodal problems and shifted large-scale problems. The results show that GOPSO obtains promising performance on a majority of the test problems.

1. Introduction

Many real world problems can be converted into optimization problems. As their complexity increases, traditional optimization algorithms cannot sufficiently satisfy the problem requirements and more effective algorithms are needed. An unconstrained minimization problem can be formulated as follows:

$$\min\ f(x),$$

where $x = [x_1, x_2, \ldots, x_D]$ and D is the dimension of the problem and f is some evaluation function.

Particle swarm optimization (PSO) is a relatively new optimization technique, which was developed by Kennedy and Eberhart [17]. Although PSO shares many common issues with other population-based algorithms, it differs from them by introducing a velocity vector. Solutions (called particles) interact with each other in the population (called swarm) by learning and sharing experiences with other particles. These phenomena are inspired from the flocking behavior of birds and fish.

PSO has obtained good performance on many optimization problems [23]. However, it may easily become trapped at local minima. In order to enhance the performance of PSO on complex problems, this paper presents a novel PSO algorithm called GOPSO by using GOBL and Cauchy mutation. The GOBL is a generalized opposition-based learning algorithm (OBL) [32] introduced in our previous work [38]. The main idea behind GOBL is to transform solutions in the current search space to a new search space. By simultaneously considering the solutions in the current search space and the transformed search

* Corresponding author.
E-mail addresses: wanghui_cug@yahoo.com.cn (H. Wang), zjwu9551@sina.com (Z. Wu), shahryar.rahnamayan@uoit.ca (S. Rahnamayan), yliu@u-aizu.ac.jp (Y. Liu), muv21@cam.ac.uk (M. Ventresca).

0020-0255/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2011.03.016

space, GOBL can provide a higher chance of finding solutions which are closer to the global optimum. The Cauchy mutation with a long tail helps trapped particles to escape. The GOPSO uses similar procedure of opposition-based differential evolution DE (ODE) when also using opposition-based population initialization and dynamic generation jumping with GOBL [29]. Experimental simulations on 18 well-known benchmark functions, including 6 shifted and large-scale problems, show that GOPSO obtains better performance on the majority of the test problems.

The rest of the paper is organized as follows. In Section 2, the standard PSO algorithm is briefly introduced. Section 3 presents some reviews of related works. Section 4 gives a simple description of OBL. The GOBL and its analysis are given in Section 5. Section 6 explains an implementation of the proposed algorithm, GOPSO. Section 7 presents a comparative study among GOPSO and eight other PSO variants on 12 benchmark problems. In Section 8, GOPSO and its enhanced version are compared on 6 shifted and large scale problems. Finally, the work is concluded in Section 9.

2. Particle swarm optimization

PSO is a population-based search algorithm that starts with an initial population of randomly generated particles [16]. For a search problem in a D-dimensional space, a particle represents a potential solution and each has a velocity and a position. PSO remembers both the best position found by all particles and by each particle during the search process. The velocity v_{ij} and position x_{ij} of the jth dimension of the ith particle are updated according to Eqs. (1) and (2):

$$v_{ij}(t+1) = w \cdot v_{ij}(t) + c_1 \cdot rand1_{ij} \cdot (pbest_{ij}(t) - x_{ij}(t)) + c_2 \cdot rand2_{ij} \cdot (gbest_j(t) - x_{ij}(t)), \tag{1}$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1), \tag{2}$$

where $i = 1,2,\ldots,$ is the particle's index, $X_i = (x_{i1},x_{i2},\ldots,x_{iD})$ is the position of the ith particle; $V_i = (v_{i1}, v_{i2},\ldots, v_{iD})$ represents the velocity of the ith particle; the $pbest_i = (pbest_{i1},pbest_{i2},\ldots,pbest_{iD})$ is the best previous position yielding the best fitness value for the ith particle; and $gbest = (gbest_1,gbest_2,\ldots,gbest_D)$ is the global best particle found by all particles so far. The inertia factor w was proposed by Shi and Eberhart [30], $rand1_{ij}$ and $rand2_{ij}$ are two random numbers generated independently within the range of $[0,1]$, c_1 and c_2 are two learning factors which control the influence of the social and cognitive components, and $t = 1,2,\ldots$ indicates the iteration number.

3. Related works

Since introducing of PSO, it has attracted many researchers to work on improving its performance. In the last decade, many variants of PSO have been proposed. A brief overview of these variants is presented in the following.

Shi and Eberhart [30] introduced a parameter called inertia weight w for the original PSO. The inertia weight is used to balance the global and local search abilities. From the analysis of [30], a linearly decreasing w over the searching process is a good choice. Clerc and Kennedy [7] proposed a constriction factor k in PSO, which can guarantee the convergence and improve the convergence rate. Bergh and Engelbrecht [2] presented a comprehensive study on the parameters of PSO, and provided a formal proof that each particle converges to a stable point.

Kennedy [18] analyzed the effects of neighborhood topology on PSO, and the presented results showed that PSO with a small neighborhood may perform better on complex problems, while PSO with a large neighborhood may perform better on simple problems. Parsopoulos and Vrahaits [24] proposed a unified PSO (UPSO) by combining the global and local versions of PSO. Mendes and Kennedy [22] introduced a fully informed PSO (FIPS) by using a modified velocity updating strategy. Peram et al. [25] presented a variant of PSO called distance-ration-based PSO (FDR-PSO), which employs a new velocity updating method. Bergh and Engelbrecht [1] proposed a cooperative approach to PSO (CPSO-H) for solving multimodal problems. Liang et al. [19] introduced a comprehensive learning PSO (CPSO) to learn other particles' experiences in different dimensions.

Chu et al. [5,6] proposed a parallel PSO by employing a novel communication strategy. Cui et al. [8] presented a fast particle swarm optimization (FPSO). FPSO does not evaluate all new positions owning a fitness and associated reliability value of each particle of the swarm, and only the reliability value is evaluated using the true fitness function if the reliability value is below a threshold. Cai et al. [4] introduced a self-adjusting cognitive selection strategy in PSO, which employs an information index to judge the value for cognitive coefficient of each particle associated with the best location itself. Tripathi et al. [33] proposed a hybrid PSO algorithm with time variant inertia and acceleration coefficients to solve multi-objective optimization problems. Du and Li [10] presented a multi-strategy ensemble PSO algorithm in dynamic environments. Wang and Yang [41] combined a preference order ranking mechanism with PSO for multi-objective optimization.

Wang et al. [37] proposed a hybrid PSO algorithm called opposition-based PSO with Cauchy mutation (OPSO), which employed an opposition operator and a Cauchy mutation operator. In order to solve large scale problems, Hsieh et al. [15] presented an efficient population utilization strategy for PSO (EPUS-PSO), which introduced a population manager and a solution sharing strategy. In EPUS-PSO, the population size is variable. The population manager can increase or decrease particle numbers according to the status of searching process.

Although some opposition-based PSO algorithms [20,37] have been proposed, this paper uses a generalized OBL (GOBL) in PSO. The proposed approach (GOPSO) uses similar schemes of ODE for opposition-based population initialization and generation jumping but this time by utilizing GOBL instead of OBL.

4. Opposition-based learning

Opposition-based Learning (OBL) [32] is a new concept in computational intelligence, and has been proven to be an effective concept to enhance various optimization approaches [21,26–29,35]. When evaluating a solution x to a given problem, simultaneously computing its opposite solution will provide another chance for finding a candidate solution which is closer to the global optimum.

Opposite number [26]: let $x \in [a,b]$ be a real number. The opposite of x is defined by:

$$x^* = a + b - x. \tag{3}$$

Similarly, the definition is generalized to higher dimensions as follows.

Opposite point [26]: let $X = (x_1, x_2, \ldots, x_D)$ be a point in a D-dimensional space, where $x_1, x_2, \ldots, x_D \in R$ and $x_j \in [a_j, b_j]$, $j \in 1,2,\ldots,D$. The opposite point $X^* = (x_1^*, x_2^*, \ldots, x_D^*)$ is defined by:

$$x_j^* = a_j + b_j - x_j. \tag{4}$$

By applying the definition of opposite point, the opposition-based optimization can be defined as follows.

Opposition-based optimization [26]: let $X = (x_1, x_2, \ldots, x_D)$ be a point in a D-dimensional space (i.e., a candidate solution). Assume $f(X)$ is a fitness function which is used to evaluate the candidate's fitness. According to the definition of the opposite point, $X^* = (x_1^*, x_2^*, \ldots, x_D^*)$ is the opposite of $X = (x_1, x_2, \ldots, x_D)$. If $f(X^*)$ is better than $f(X)$, then update X with X^*; otherwise keep the current point X. Hence, the current point and its opposite point are evaluated simultaneously in order to continue with the fitter one.

5. A generalized opposition-based learning (GOBL)

5.1. The concept of GOBL

In our previous work [38], a generalized OBL, called GOBL, is proposed by transforming candidates in current search space to a new search space. By simultaneously evaluating the candidates in the current search space and transformed search space, it can provide more chance of finding candidate solutions closer to the global optimum.

Generalized OBL: let x be a solution in the current search space S, $x \in [a,b]$. The new solution x^* in the transformed space S^* is defined by [38]:

$$x^* = \Delta - x, \tag{5}$$

where Δ is a computable value and $x^* \in [\Delta - b, \Delta - a]$. It is obvious that the differences between the current search space S and the transformed search space S^* are the centers of search spaces. Because the size of search range (indicates the size of intervals) of S and S^* are $b - a$, and the center of current search space moves from $\frac{a+b}{2}$ to $\frac{2\Delta-a-b}{2}$ after using GOBL.

Similarly, the definition of GOBL is generalized to a D-dimensional search space as follows.

$$x_j^* = \Delta - x_j, \tag{6}$$

where $j = 1,2,\ldots,D$.

By applying GOBL, we not only evaluate the current candidate x, but also calculate its transformed candidate x^*. This will provide more chance of finding candidate solutions closer to the global optimum.

However, GOBL could not be suitable for all kinds of optimization problems. For instance, the transformed candidate may jump away from the global optimum when solving multimodal problems. To avoid this case, a new elite selection mechanism based on the population is used after the transformation. The elite selection mechanism is described in Fig. 1. Assume that the current population $P(t)$ has three particles, x_1, x_2 and x_3, where t is the index of generations. According to the concept of GOBL, we get three transformed particles x_1^*, x_2^* and x_3^* in population $GOP(t)$. Then, we select three fittest particles from $P(t)$ and $GOP(t)$ as a new population $P'(t)$. It can be seen from Fig. 1, x_1, x_2^* and x_3^* are three new members in $P'(t)$. In this case, the transformation conducted on x_1 did not provide another chance of finding a candidate solution closer to the global optimum. With the help of the elite selection mechanism, x_1^* is eliminated from the next generation.

Let $\Delta = k(a + b)$, where k is a real number. The GOBL model is defined by:

$$x^* = k(a + b) - x. \tag{7}$$

According to the suggestions of [38], a random value of k obtains good performance. In Eq. (7), k is a random number in [0,1], $x \in [a,b]$ and $x^* \in [k(a+b) - b, k(a+b) - a]$. The center of the transformed search space is at a random position in $\left[-\frac{a+b}{2}, \frac{a+b}{2}\right]$.

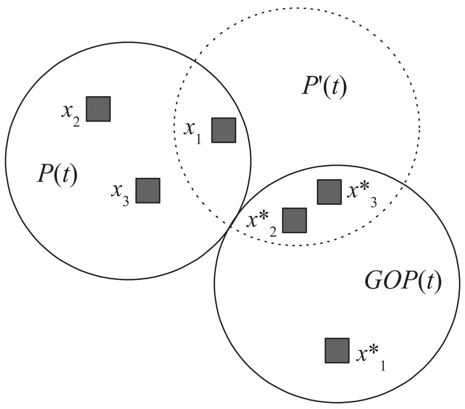


Fig. 1. The elite selection mechanism based on population.

For a given problem, it is possible that the transformed candidate may jump out of the box-constraint $[X_{min},X_{max}]$. When this happens, the GOBL will be invalid, because the transformed candidate is infeasible. To avoid this case, the transformed candidate is assigned to a random value as follows.

$$x^* = rand(a,b), \text{ If } x^* < X_{min} \| x^* > X_{max}, \tag{8}$$

where $rand(a,b)$ is a random number in $[a,b]$, and $[a,b]$ is the interval boundaries of current population.

5.2. GOBL-based optimization

By staying within variables' interval static boundaries, we would possibly jump outside of the already shrunken search space and so the knowledge of the current converged search space would be lost. Hence, we calculate opposite particles by using dynamically updated interval boundaries $[a_j(t),b_j(t)]$ as follows [29].

$$X^*_{i,j} = k[a_j(t) + b_j(t)] - X_{i,j}, \tag{9}$$

$$a_j(t) = \min(X_{i,j}(t)), \quad b_j(t) = \max(X_{i,j}(t)), \tag{10}$$

$$X^*_{i,j} = rand(a_j(t),b_j(t)), \text{ If } X^*_{i,j} < X_{min} \| X^*_{i,j} > X_{max}, \quad i = 1,2,\ldots,ps, \quad j = 1,2,\ldots,D, \quad k = rand(0,1), \tag{11}$$

where $X_{i,j}$ is the jth vector of the ith candidate in the population, $X^*_{i,j}$ is the transformed candidate of $X_{i,j}$, $a_j(t)$ and $b_j(t)$ are the minimum and maximum values of the jth dimension in current population, respectively, $rand(a_j(t),b_j(t))$ is a random number in $[a_j(t),b_j(t)]$, $[X_{min},X_{max}]$ is the box-constraint, ps is the population size, $rand(0,1)$ is a random number in [0,1], and $t = 1,2,\ldots,$ indicates the generation number.

In our recent study [39], we experimentally explained why ODE obtains better solutions with a faster convergence rate than standard DE. In ODE, there are two important steps, generation jumping and elite selection. The first step is beneficial for increasing diversity and exploring more promising regions, while the second one is helpful to speed up convergence. Although these two steps are incompatible, ODE makes a good balance between them towards searching candidate solutions. This explanation is also suitable for GOPSO, because GOPSO uses similar procedure of ODE for generation jumping and elite selection.

Algorithm 1. The GOPSO Algorithm

```
 1 Randomly initialize each particle in swarm $P$;
 2 double $k = rand(0,1)$;
 3 for $i = 1$ to $ps$ do
 4   for $j = 1$ to $D$ do
 5     $GOP_{i,j} = k(a_j + b_j) - P_{i,j}$;
 6   end
 7   Calculate the fitness value of $GOP_i$;
 8   NE++;
 9 end
10 Select $ps$ fittest particles from $\{P,GOP\}$ as an initial population $P$;
11 While $NE \leqslant MAX_{NE}$ do
12   If $rand(0,1) \leqslant p_o$ then
```

```
13      Update the dynamic interval boundaries $[a_j(t), b_j(t)]$ in $P$ according to Eq. (10);
14      $k = rand(0,1)$;
15      for $i = 1$ to $ps$ do
16        for $j = 1$ to $D$ do
17          $GOP_{i,j} = k[a_j(t) + b_j(t)] - P_{i,j}$;
18        end
19        Calculate the fitness value of $GOP_i$;
20        NE++;
21      end
22      Select $ps$ fittest particles from $\{P, GOP\}$ as current population $P$;
23    end
24    else
25      for $i = 1$ to $ps$ do
26        Calculate the velocity of particle $P_i$ according to Eq. (1);
27        Update the position of particle $P_i$ according to Eq. (2);
28        Calculate the fitness value of particle $P_i$;
29        NE++;
30      end
31    end
32    Update $pbest$, $gbest$ in $P$ if needed;
33    Mutate $gbest$ according to Eq. (14);
34    Calculate the fitness value of $gbest'$;
35    $NE$++;
36    if $f(gbest') < f(gbest)$ then
37      $gbest = gbest'$;
38    end
39  end
```

6. Enhancing PSO using GOBL (GOPSO)

The standard PSO was inspired by the social and cognitive behavior of swarms. According to Eq. (1), particles are greatly influenced by its previous best particles and the global best particle. Once the best particle has no change in a local optimum, all the rest particles will quickly converge to the position of the best particle. If the neighbors of the global best particle would be added in each generation, it would extend the search space around the best particle. It is helpful for all particles to move to the better positions. This can be accomplished by conducting a Cauchy mutation [36,37] on the global best particle at every generation. In this paper, we also employ a Cauchy mutation on the global best particle in GOPSO.

The one-dimensional Cauchy density function centered at the origin is defined by

$$f(x) = \frac{1}{\pi}\frac{t}{t^2 + x^2}, \quad -\infty < x < \infty, \tag{12}$$

where $t > 0$ is a scale parameter [11]. The Cauchy distributed function is

$$F_t(x) = \frac{1}{2} + \frac{1}{\pi}\arctan\left(\frac{x}{t}\right). \tag{13}$$

The Cauchy mutation operator is defined by

$$gbest'_j = gbest_j + cauchy(), \tag{14}$$

where $gbest_j$ is the jth value of the global best particle in the swarm, $cauchy()$ is a random number generated by the Cauchy distributed function with the scale parameter of $t = 1$.

The proposed approach (GOPSO) uses a similar procedure to that of ODE for opposition-based population initialization and dynamic opposition with a generalized OBL concept. The framework of GOPSO is shown in Algorithm 1, where P is the current population, GOP is the population after using GOBL, P_i is the ith particle in P, GOP_i is the ith particle in GOP, k is a random number in $[0,1]$, p_o is the probability of GOBL, ps is the population size, D is the dimension size, $[a_j(t), b_j(t)]$ is the interval boundaries of current population, NE is the number of function evaluations and MAX_{NE} is the maximum number of function evaluations.

When background information of problems is unavailable, we always generate an initial population based on a uniform probability. By using GOBL in population initialization, we can obtain fitter starting candidate solutions [29]. Steps 1–10 from Algorithm 1 show the implementation of GOBL-based initialization for GOPSO. Besides the population initialization, we also use GOBL in the whole evolutionary process. If the probability p_o is satisfied, the transformed population GOP is calculated and the ps fittest particles are selected from the union of the current population P and GOP; otherwise the algorithm executes the standard PSO.

7. Experimental studies on GOPSO

7.1. Test problems

A comprehensive set of benchmark problems, including 12 different global optimization problems, was chosen for the following experimental studies. These functions were considered in an earlier study [19] as well. According to their properties, they are divided into three classes: unimodal and simple multimodal problems ($f_1 - f_2$), unrotated multimodal problems ($f_3 - f_8$), and rotated multimodal problems ($f_9 - f_{12}$). All the functions used in this paper are minimization problems. A brief description of these benchmark problems is listed in Table 1, and more details can be found in [19].

7.2. Parameter settings for the involved PSO algorithms

Experiments are conducted to compare nine PSO algorithms including the proposed GOPSO on 12 test problems with $D = 30$. The algorithms are listed below.

- PSO with inertia weight (PSO-w) [30];
- PSO with constriction factor (PSO-cf) [7];
- unified PSO (UPSO) [24];
- fitness-distance-ratio based PSO (FDR-PSO) [25];
- fully informed particle swarm (FIPS) [22];
- cooperative PSO (CPSO-H) [1];
- comprehensive learning PSO (CLPSO) [19];
- opposition-based PSO with Cauchy mutation (OPSO) [37];
- our approach (GOPSO).

The parameter settings for PSO-w, PSO-cf, UPSO, FIPS, FDR-PSO, CPSO-H and CLPSO are described in [19]. In OPSO and GOPSO, $c_1 = c_2 = 1.49618$, $w = 0.72984$ and the maximum velocity V_{max} was set to the half range of the search space for each dimension. The probability of using GOBL in GOPSO and the probability of using OBL in OPSO use the same value $p_o = 0.3$. The above nine PSO algorithms use the same population size ($ps = 40$) and maximum number of evaluations ($\text{MAX}_{\text{NE}} = 200{,}000$). All the experiments were conducted 30 times, and the mean function error value $f(x) - f(x^o)$ ($f(x^o)$ is the global optima of $f(x)$) and standard deviation of the results are recorded.

7.3. Results and discussions

The results of the nine PSO algorithms on the 12 test problems are presented in Table 2, where "Mean" indicates the mean function error value, and "Std Dev" stands for the standard deviation. Results of PSO-w, PSO-cf, UPSO, FDR-PSO, FIPS, CPSO-H and CLPSO are taken from Table 4 in [19]. The best results among the nine algorithms are shown in bold.

As seen, GOPSO surpasses all other algorithms on functions f_1, f_4, f_7, f_{10} and f_{12}, and significantly improves the results on functions f_7, f_{10} and f_{12}. GOPSO achieves the same performance as the CPSO-H on function f_6, and they both are much better than the other PSO algorithms on this problem. UPSO achieves better results than GOPSO on function f_3, while GOPSO outperforms UPSO on the remaining functions. GOPSO surpasses FDR-PSO in all test cases except for function f_2. FIPS performs better than GOPSO on functions f_9 and f_{11}, while GOPSO outperforms FIPS on the rest of the functions. GOPSO achieves better results than CPSO-H on all test functions except for f_5, f_6 and f_8. CLPSO obtains better performance than GOPSO on functions f_3 and f_8, while GOPSO achieves better results than CLPSO on the other functions. GOPSO outperforms OPSO on all test functions except for f_2. According to performance graphs given in Fig. 2, GOPSO shows very fast convergence speed.

Table 1
The 12 test functions used in the experiments, where D is the dimension of the functions, $X \in R^n$ is the definition domain, and $f(x^o)$ is the minimum value of the function.

F	Name	D	X	$f(x^o)$
f_1	Sphere function	30	$[-100,100]$	0
f_2	Rosenbrock's function	30	$[-2.048,2.048]$	0
f_3	Ackley's function	30	$[-32.768,32.768]$	0
f_4	Griewanks's function	30	$[-600,600]$	0
f_5	Weierstrass function	30	$[-0.5,0.5]$	0
f_6	Rastrigin's function	30	$[-5.12,5.12]$	0
f_7	Nocontinuous Rastrigin's function	30	$[-5.12,5.12]$	0
f_8	Schwefel's function	30	$[-500,500]$	0
f_9	Rotated Ackley's function	30	$[-32.768,32.768]$	0
f_{10}	Rotated Griewanks's function	30	$[-600,600]$	0
f_{11}	Rotated Weierstrass function	30	$[-0.5,0.5]$	0
f_{12}	Rotated Rastrigin's function	30	$[-5.12,5.12]$	0

Table 2
Comparison among the nine PSO algorithms on 12 test problems, where “Mean” indicates the mean function error value, and “Std Dev” stands for the standard deviation. The best results among the nine algorithms are shown in bold.

	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
	f_1		f_2		f_3		f_4	
PSO-w	9.78e−30	2.50e−29	2.93e+01	2.51e+01	3.94e−14	1.12e−14	8.13e−03	7.16e−03
PSO-cf	5.88e−100	5.40e−100	1.11e+01	1.81	1.12	8.65e−01	2.06e−02	1.90e−02
UPSO	4.17e−87	3.15e−87	1.51e+01	8.14e−01	1.22e−15	3.16e−15	1.66e−03	3.07e−03
FDR-PSO	4.88e−102	1.53e−101	**5.39**	**1.76**	2.84e−14	4.10e−15	1.01e−02	1.23e−02
FIPS	2.69e−12	6.84e−13	2.45e+01	2.19e−01	4.81e−07	9.17e−08	1.16e−06	1.87e−06
CPSO-H	1.16e−113	2.92e−113	7.08	8.01	4.93e−14	1.10e−14	3.63e−02	3.60e−02
CLPSO	1.46e−14	1.73e−14	2.01e+01	2.98	**0**	**0**	3.14e−10	4.64e−10
OPSO	8.05e−81	1.22e−80	9.78	6.33	1.39	1.33	1.27e−02	1.76e−02
GOPSO	**1.24e−272**	**0**	1.48e+01	9.57e−01	3.43e−15	1.59e−15	**0**	**0**
	f_5		f_6		f_7		f_8	
PSO-w	1.30e−04	3.30e−04	2.90e+01	7.70	2.97e+01	1.39e+01	1.10e+03	2.56e+02
PSO-cf	4.10	2.20	5.62e+01	9.76	2.85e+01	1.14e+01	3.78e+03	6.02e+02
UPSO	9.6	3.78	6.59e+01	1.22e+01	6.34e+01	1.24e+01	4.84e+03	4.76e+02
FDR-PSO	7.49e−03	1.14e−02	2.84e+01	8.71e+01	1.44e+01	6.28e+01	3.61e+03	3.06e+02
FIPS	1.54e−01	1.48e−01	7.30e+01	1.24e+01	6.08e+01	8.35e+01	2.05e+03	9.58e+02
CPSO-H	**7.82e−15**	**8.50e−15**	**0**	**0**	1.00e−01	3.16e−01	1.08e+03	2.59e+02
CLPSO	3.45e−07	1.94e−07	4.85e−10	3.63e−10	4.36e−10	2.44e−10	**1.27e−12**	**8.79e−13**
OPSO	2.21	8.55e−01	5.23e+01	1.51e+01	8.64e+01	9.86e+01	1.45e+03	3.04e+02
GOPSO	1.04e−08	2.19e−08	**0**	**0**	**0**	**0**	1.14e+03	1.87e+02
	f_9		f_{10}		f_{11}		f_{12}	
PSO-w	2.80e−01	5.86e−01	1.64e−01	9.40e−02	6.66e−01	7.12e−01	9.90	3.76
PSO-cf	1.19	1.13	1.38e−01	1.07e−01	2.17	1.30	1.44e+01	6.04
UPSO	1.00	9.27e−01	7.76e−02	6.40e−02	2.61	9.48e−01	1.52e+01	5.25
FDR-PSO	1.40e−01	4.38e−01	1.44e−01	7.84e−02	3.34e−01	3.90e−01	9.25	2.50
FIPS	**2.25e−15**	**1.54e−15**	1.70e−01	1.26e−01	**5.93e−14**	**1.86e−13**	1.20e+01	6.22
CPSO-H	1.36	8.85e−01	1.20e−01	8.07e−02	4.35	1.35	2.67e+01	1.06
CLPSO	3.65e−05	1.57e−04	4.50e−02	3.08e−02	3.72e−10	4.40e−10	5.97	2.88
OPSO	1.89	8.59e−01	4.37e−02	4.92e−02	8.65	1.23	9.02e+01	4.41e+01
GOPSO	9.59e−13	0	**0**	**0**	2.64e−13	2.45e−13	**0**	**0**

Table 3 shows the statistical comparison of the GOPSO algorithm to the other eight PSO algorithms using a two-tailed t-test with 58 degrees of freedom at a 0.05 level of significance. The performance difference is significant if the absolute value of the t-test result is greater than 2. It can be seen from the t-test results that most of them are less than −2. Generally speaking, GOPSO is significantly better than the compared algorithms on the majority of test functions.

To compare the performance of multiple algorithms on this test suite, the average ranking of the Friedman test is conducted by the suggestions considered in [12,13]. Table 4 shows the average ranking of the nine PSO algorithms on functions $f_1 - f_{12}$. These algorithms can be sorted by average ranking into the following order: GOPSO, CLPSO, FDR-PSO, CPSO-H, PSO-w, FIPS, UPSO, PSO-cf, and OPSO. The best average ranking is obtained by the GOPSO algorithm.

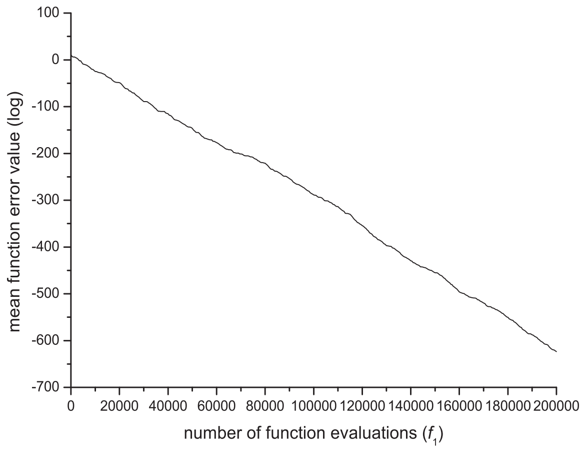
To compare the performance differences between GOPSO and the other eight PSO algorithms, we conduct a Wilcoxon signed-ranks test [9,14]. Table 5 shows the resultant p-values when comparing between GOPSO and the other eight algorithms. The p-values below 0.05 are shown in bold. From the results, it can be seen that GOPSO is significantly better than all algorithms except for CPSO-H. Although GOPSO is not significantly better than CPSO-H, GOPSO performs better than CPSO-H according to the average rankings shown in Table 4.

Results analysis – GOPSO shows better performance than the other PSO variants on a majority of the test problems. GOBL can provide a higher chance of finding better solutions than OBL, for OPSO. The random GOBL used in GOPSO is more flexible than the OBL used for OPSO because random GOBL has a larger search range than OBL, but the same size of search range for each generation.

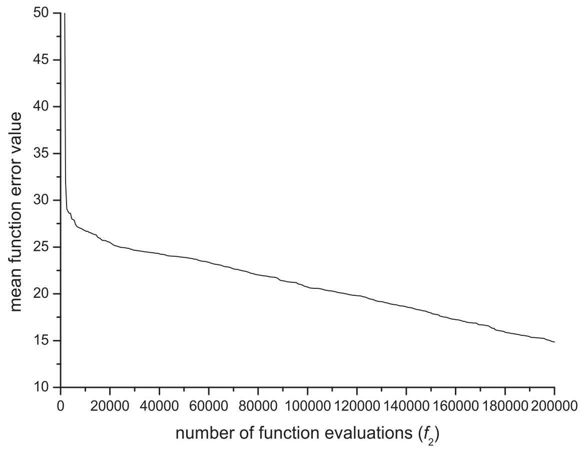
Assume that x is the current candidate solution, x' is the opposite candidate solution of x in OPSO, and x^* is the transformed candidate of x in GOPSO, where $x \in [a,b]$ and $b > a > 0$. The search range of x' satisfies $a \leqslant x' \leqslant b$, and the search range of x^* satisfies $k(a+b) - b \leqslant x^* \leqslant k(a+b) - a$, where $k \in [0,1]$. It is easy to deduce

$$-b \leqslant k(a+b) - b \leqslant a, \tag{15}$$
$$-a \leqslant k(a+b) - a \leqslant b. \tag{16}$$

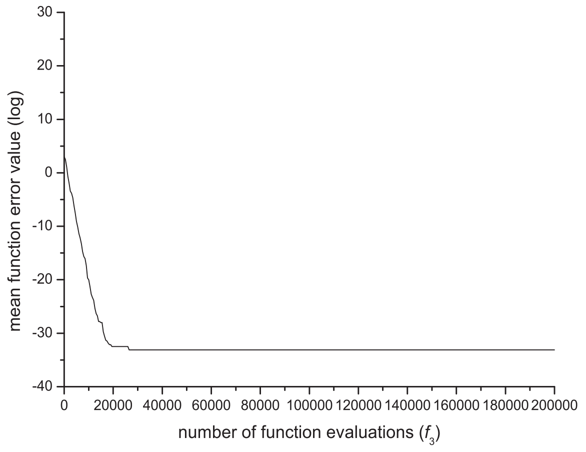
Hence, the search range of x^* satisfies $-b \leqslant x^* \leqslant b$. For the whole search process, random GOBL has a larger search range than OBL, while both of them have the same size of search range $(b-a)$ for every transformation. To give a clear explanation, Fig. 3 presents the comparison of search range between OBL and GOBL schemes.



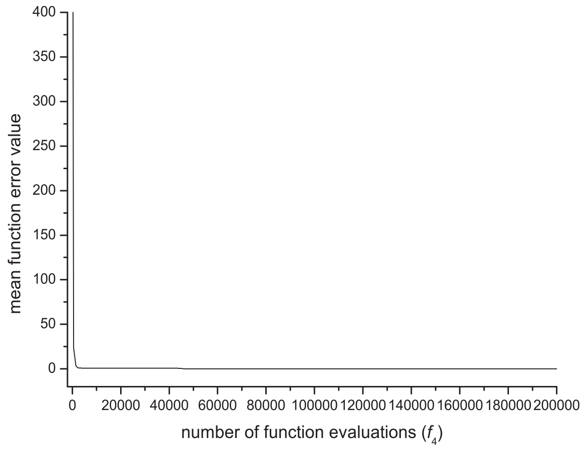
(a) f_1



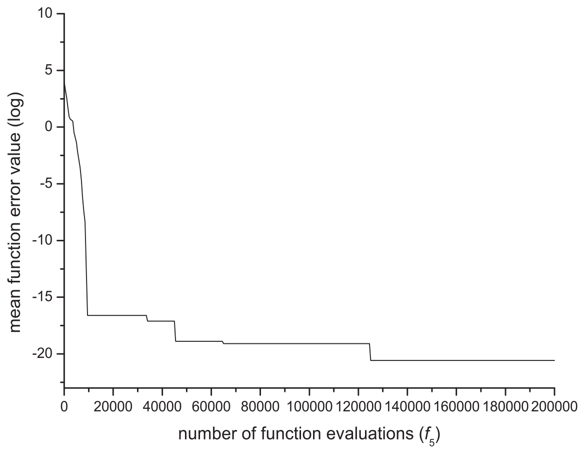
(b) f_2



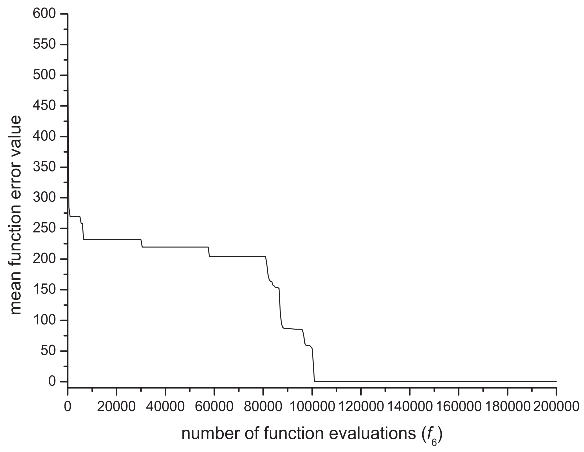
(c) f_3



(d) f_4



(e) f_5



(f) f_6

Fig. 2. The convergence trend of GOPSO on $f_1 - f_6$.

Table 3
The t-test results between GOPSO and the other eight PSO algorithms, where "∼∼" means the results of each run is the same.

Functions	PSO-w	PSO-cf	UPSO	FDR-PSO	FIPS	CPSO-H	CLPSO	OPSO
f_1	−2.14269	−5.96409	−7.2508	−1.74698	−21.5406	−2.17588	−4.70397	−3.61407
f_2	−3.16184	9.89816	−1.30788	25.7271	−54.1174	5.24165	−9.27484	4.2949
f_3	−17.4161	−7.16646	3.42184	−31.1009	−28.73	−22.6051	11.8157	−5.72432
f_4	−6.21925	−5.93847	−2.96163	−4.49756	−3.39764	−5.52287	−3.70657	−3.95232
f_5	−2.15752	−10.2076	−13.9104	−3.59863	−5.69927	2.60105	−9.38718	−14.1575
f_6	−20.6285	−31.5389	−29.586	−1.78592	−32.245	∼∼	−7.31806	−18.9708
f_7	−11.7031	−13.6931	−28.0045	−1.25592	−3.98821	−1.7333	−9.78717	−4.88876
f_8	0.691078	−22.9385	−39.6268	−37.7249	−5.10642	1.02874	33.3906	−4.75733
f_9	−2.6171	−5.76805	−5.90855	−1.75071	3402.82	−8.41698	−1.27337	−12.0512
f_{10}	−9.55601	−7.06409	−6.64114	−10.0602	−7.38991	−8.14457	−8.00244	−4.86493
f_{11}	−5.12336	−9.14275	−15.0797	−4.69075	3.64489	−17.6488	−4.62746	−38.5187
f_{12}	−14.4214	−13.0583	−15.8579	−20.2657	−10.567	−137.964	−11.3538	−11.2029

Table 4
Average rankings of the nine PSO algorithms.

Algorithm	Ranking
GOPSO	7.96
CLPSO	6.83
FDR-PSO	5.67
CPSO-H	5.38
PSO-w	4.50
FIPS	4.08
UPSO	3.75
PSO-cf	3.58
OPSO	3.25

Table 5
Wilcoxon test between GOPSO and other eight PSO variants on functions $f_1 - f_{12}$. The *p*-values below 0.05 are shown in bold.

GOPSO vs.	*p*-values
PSO-w	**0.034**
PSO-cf	**0.012**
UPSO	**0.004**
FDR-PSO	**0.019**
FIPS	**0.005**
CPSO-H	0.374
CLPSO	**0.041**
OPSO	**0.012**

8. Experiments on shifted and large scale problems

8.1. Test problems

In order to further verify the performance of GOPSO, this section presents a comparative study of GOPSO on six shifted and large scale problems, which are considered in CEC 2008 special session and competition on large scale global optimization [31]. According to their properties, there are two unimodal functions ($F_1 - F_2$) and four multimodal functions ($F_3 - F_6$). All functions are shifted by a random point $O = [o_1, o_2, \dots, o_D]$ in D-dimensional search space. The employed shifted points in this paper are also provided by the above mentioned session [31]. In this paper, we mainly focus on large scale problems with $D = 100$. All the functions used in the following experiments are to be minimized. The description of the six benchmark functions and their properties are as follows.

F_1: shifted sphere function

$$F_1(X) = \sum_{i=1}^{D} Z_i^2 + f\_bias_1,$$

where $D = 100$, $X \in [-100, 100]$, $Z = (X - O)$, $X = [x_1, x_2, \dots, x_D]$, $O = [o_1, o_2, \dots, o_D]$, and the global optimum $F_1(X^o) = f\_bias_1 = -450$ at $X^o = O$.

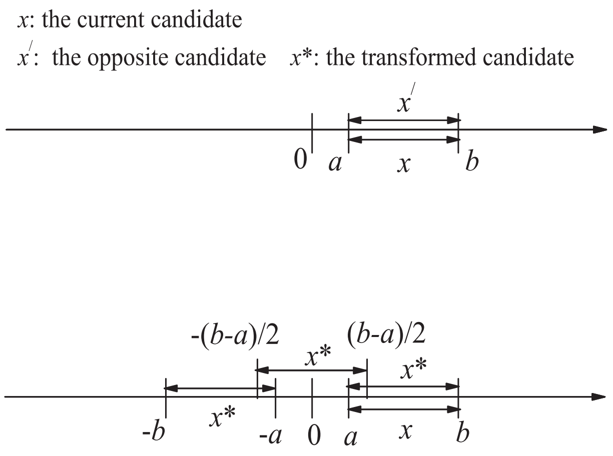


Fig. 3. The comparison of OBL with GOBL coverage intervals.

Problem properties: unimodal, shifted, separable, and scalable.

F_2: Schwefel's problem 2.21

$$F_2(X) = max_i\{|Z_i|, 1 \leqslant i \leqslant n\} + f\_bias_2,$$

where $D = 100$, $X \in [-100,100]$, $Z = (X - O)$, $X = [x_1,x_2,\ldots,x_D]$, $O = [o_1,o_2,\ldots,o_D]$, and the global optimum $F_2(X^o) = f\_bias_2 = -450$ at $X^o = O$.

Problem properties: unimodal, shifted, non-separable, and scalable.

F_3: shifted Rosenbrock's function

$$F_3(X) = \sum_{i=1}^{D}[100\left(Z_{i+1} - Z_i^2\right)^2 + \left(1 - Z_i^2\right)^2] + f\_bias_3,$$

where $D = 100$, $X \in [-100,100]$, $Z = (X - O)$, $X = [x_1,x_2,\ldots,x_D]$, $O = [o_1,o_2,\ldots,o_D]$, and the global optimum $F_3(X^o) = f\_bias_3 = 390$ at $X^o = O$.

Problem properties: multimodal, shifted, non-separable, scalable, and having a very narrow valley from local optimum to global optimum.

F_4: shifted Rastrigins function

$$F_4(X) = \sum_{i=1}^{D}[Z_i^2 - 10\cos 2\pi Z_i + 10] + f\_bias_4,$$

where $D = 100$,$X \in [-5,5]$, $Z = (X - O)$, $X = [x_1,x_2,\ldots,x_D]$, $O = [o_1,o_2,\ldots,o_D]$, and the global optimum $F_4(X^o) = f\_bias_4 = -330$ at $X^o = O$.

Problem properties: multimodal, shifted, separable, scalable, and having a huge number of local optima.

F_5: shifted Griewank's function

$$F_F(X) = \frac{1}{4000}\sum_{i=1}^{D}(Z_i)^2 - \prod_{i=1}^{D}\cos\left(\frac{Z_i}{\sqrt{i}}\right) + 1 + f\_bias_5,$$

where $D = 100$, $X \in [-600,600]$, $Z = (X - O)$, $X = [x_1,x_2,\ldots,x_D]$, $O = [o_1,o_2,\ldots,o_D]$, and the global optimum $F_5(X^o) = f\_bias_5 = -180$ at $X^o = O$.

Problem properties: multimodal, shifted, non-separable, and scalable.

F_6: shifted Ackley's function

$$F_6(X) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D}Z_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi Z_i)\right) + 20 + e + f\_bias_6,$$

where $D = 100$, $X \in [-32,32]$, $Z = (X - O)$, $X = [x_1,x_2,\ldots,x_D]$, $O = [o_1,o_2,\ldots,o_D]$, and the global optimum $F_6(X^o) = f\_bias_6 = -140$ at $X^o = O$.

Problem properties: multimodal, shifted, separable, and scalable.

In the past two decades, different kinds of evolutionary methods have been proposed to solve optimization problems, e.g., EAs, differential evolution (DE), PSO, Ant Colony Optimization (ACO), Estimation of Distribution Algorithm (EDA), etc. Although these approaches have shown good optimization performance when dealing with some lower dimensional problems ($D < 100$), many of them suffers from the "curse of dimensionality", which implies that their performance deteriorates quickly as the dimension increases. The main reason is that the complexity of the problem and search space increase with problem dimensionality. The previous population-based methods may lack the power of searching the optima solution. So, more efficient search strategies are required to explore all the promising regions in a given time budget [31]. Yang et al. [42] combined cooperative co-evolution and self-adaptive neighborhood search DE (SaNSDE) to solve large scale problems. Hsieh et al. [15] presented an efficient population utilization strategy for PSO (EPUS-PSO) to manage the population size. Brest et al. [3] introduced a population size reduction mechanism into self-adaptive DE, where the population size decreases during the evolutionary process. Tseng and Chen [34] presented multiple trajectory search (MTS) by using multiple agents to search the solution space concurrently. Zhao et al. [43] used dynamic multi-swarm PSO with local search (DMS-PSO) for large scale problems. Wang and Li [40] proposed a univariate EDA (LSEDA-gl) by sampling under mixed Gaussian and lévy probability distribution.

8.2. GOPSO with dynamic population size (DP-GOPSO)

To solve large scale problems, we modify the original GOPSO by introducing a dynamic population (DP) mechanism. The new approach is called DP-GOPSO, in which the population size is variable, because the DP method can increase or decrease the number of particles in terms of the search status of the current population. If the population cannot find a better particle to update the global best particle *gbest* in *m* generations, particles maybe have been trapped in a local optimum. Under this circumstance, dependence on the original velocity and position equations will hardly help a particle jump out of the local optima. The proposed method aims to introduce new potential solutions (particles) to escape such situations. In order to avoid unlimited increase in particles, the current population size must have an upper limit, *max_ps*. The implementation of the DP mechanism includes three steps which are described as follows:

(1) If the *gbest* has no change in *m* generations, and if the current population size (*c_ps*) is less than *max_ps*, a new particle will be added into the current population, and then *c_ps* = *c_ps* + 1. The new particle is generated by:

$$x_j = r_1 pbest_{i_1}.x_j + r_2 pbest_{i_2}.x_j + r_3 pbest_{i_3}.x_j, \tag{17}$$

$$v_j = r_1 pbest_{i_1}.v_j + r_2 pbest_{i_2}.v_j + r_3 pbest_{i_3}.v_j, \tag{18}$$

where x and v are the position and velocity vector of the new particle, respectively. $pbest_{i_1}$, $pbest_{i_2}$ and $pbest_{i_3}$ are three different previous best particles, i_1, i_2 and i_3 are different random integers within $[0, c\_ps - 1]$, r_1, r_2 and r_3 are three different random numbers in [0,1], where $r_1 + r_2 + r_3 = 1$. The random numbers i_1, i_2, i_3, r_1, r_2 and r_3 are the same for all $j = 1,2,\ldots,D$, and they are generated anew in each generation.

(2) If the *gbest* has been updated more than once in *m* generations, and if the *c_ps* is larger than *min_ps*, the worst particle in the current population will be deleted, and then *c_ps* = *c_ps* − 1, where *min_ps* is a predefined lower boundary of the population size. If *c_ps*=*min_ps*, the worst particle in the current swarm will be re-initialized as follows.

$$x_{wj} = r_4 x_{wj} + r_5 gbest.x_j + r_6 pbest_w.x_j, \tag{19}$$

$$v_{wj} = r_4 v_{wj} + r_5 gbest.v_j + r_6 pbest_w.v_j, \tag{20}$$

where x_w and v_w are the position and velocity vectors of the worst particle, respectively. *gbest* is the global best particle, $pbest_w$ is the best of the worst particles and r_4, r_5 and r_6 are three different random numbers in [0,1], where $r_4 + r_5 + r_6 = 1$. The random numbers r_4, r_5 and r_6 are the same for all $j = 1,2,\ldots,D$, and they are generated anew in each generation. In this paper, *min_ps* is set to 5.

(3) If the global best particle (*gbest*) has no change in *m* generations, and if *c_ps* = *max_ps*, the worst particle in current population will be re-initialized according to Eqs. (19) and (20).

For the first case in DP, three different *pbest* particles are selected to recombine a new particle. So the minimum population size should be no less than 3 (*min_ps* $\geqslant$ 3). Our empirical studies led to setting *min_ps* to 5. The parameter *m* determines the monitoring frequency on the global best particle. A large *m* will not reflect the changes of *gbest* in time, while a small *m* will cost much computational time to monitor the search status of the current population. When *m* is set to 1, the population size will be adjusted in each generation. In this paper, *m* is set to 3. By the suggestions of [15], the maximum population size *max_ps* is set to 30.

To illustrate the mechanism of the dynamic population size, Fig. 4 presents the changes of population size *c_ps* achieved by DP-GOPSO on the Quadric function with 30 dimensions. The initial population size is set to *min_ps* = 5, and the maximum population size *max_ps* is set to 30. The DP mechanism adjusts the population size according to the current state of the global best particle.

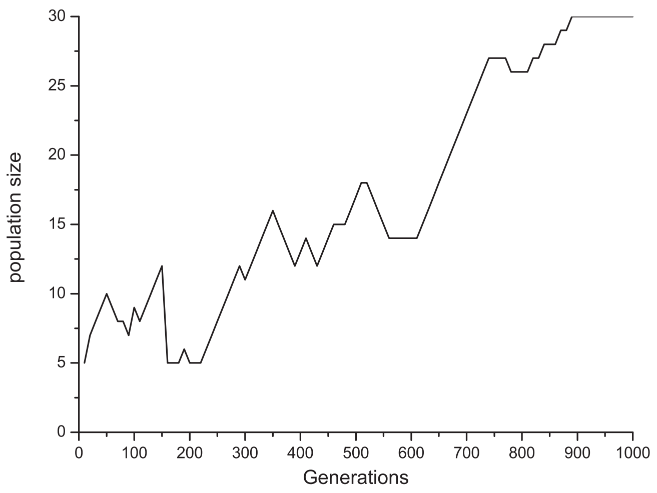


Fig. 4. The changes of population size for DP-GOPSO when solving the Quadric function with 30 dimensions.

Algorithm 2. The Framework of DP-GOPSO

```
1 Initialization (the same as GOPSO);
2 while NE $\leqslant$ $MAX_{NE}$ do
3   Execute GOPSO algorithm (see Algorithm 1);
4   Conduct the dynamic population mechanism (see the 3 steps described in Section 8.2);
5   Update the pbest and gbest in current population;
6 end
```

The framework of DP-GOPSO is shown in Algorithm 2. The DP-GOPSO can be regarded as a simple combination of GOPSO and the DP mechanism. We only introduce a DP mechanism to manage the population size of GOPSO.

8.3. Parameter settings

In these experiments, we compared GOPSO and DP-GOPSO with OPSO [37], dynamic multi-swarm PSO (DMS-PSO) [43] and EPUS-PSO [15]. To have a fair comparison, we used the common parameter settings: DMS-PSO and EPUS-PSO used the original parameter settings in [15,43]. For OPSO, GOPSO and DP-GOPSO, $c_1 = c_2 = 1.49618$, $w = 0.72984$, $ps = 30$ (except for DP-GOPSO), and the maximum velocity V_{max} was set to the half range of the search space on each dimension. The probability of GOBL in GOPSO and the probability of opposition-based generation jumping in OPSO use the same value $p_o = 0.3$. For DP-GOPSO, $m = 3$, $max\_ps = 30$ and $min\_ps = 5$. By the suggestions of [31], we use the same maximum number of evaluations ($MAX_{NE} = 5000 * D = 500{,}000$). All experiments are run 25 times and we record the mean and standard deviation of $f(x) - f(x^o)$.

8.4. Results and discussions

In this section, we have two series of comparisons: (1) comparisons of DP-GOPSO with PSO, OPSO and GOPSO; (2) comparison of DP-GOPSO with DMS-PSO and EPUS-PSO, where DMS-PSO and EPUS-PSO results have been published in CEC 2008 Special Session and Competition on Large Scale Global Optimization [31]. The first comparison aims to check whether opposition and DP mechanism work well on large scale problems. The second comparison focuses on investigating how good DP-GOPSO is within the similar context of PSO variants.

The results of the two comparisons are presented in Tables 6 and 7, respectively. Table 8 shows the statistical comparisons of the DP-GOPSO with the other five PSO algorithms, using a two-tailed t-test with 48 degrees of freedom at a 0.05 level of significance. In all the t-test results the performance difference is significant if the absolute value of the t-test result is greater than 2.009. Tables 9 and 10 present the average ranking of the six PSO algorithms and the p-values of applying a Wilcoxon test among DP-GOPSO and other five PSO algorithms, respectively.

(1) Comparison of DP-GOPSO with PSO, OPSO and GOPSO: Table 6 presents the mean and standard deviation of the 25 runs for PSO, OPSO, GOPSO and DP-GOPSO. It can be seen that DP-GOPSO outperforms PSO, OPSO and GOPSO in all test cases. GOPSO and OPSO performs better than PSO except on function F_3. On this function, GOPSO and OPSO

Table 6
Comparison among PSO, OPSO, GOPSO and DP-GOPSO on six shifted and large scale problems ($D = 100$), where “Mean” indicates the mean function error value, and “Std Dev” stands for the standard deviation. The best results among the comparison are shown in bold.

Functions	PSO		OPSO [37]		GOPSO		DP-GOPSO	
	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
F_1	2.37e+04	4.59e+03	3.64e+03	6.57e+02	3.43e+03	4.82e+02	**8.75e+02**	**9.33e+01**
F_2	5.31e+01	6.83	3.52e+01	4.24	3.37e+01	3.63	**1.53e+01**	**3.17**
F_3	1.59e+02	8.85e+01	4.86e+07	7.21e+06	4.53e+07	8.41e+06	**1.49e+02**	**7.96e+01**
F_4	6.16e+02	1.92e+02	4.72e+02	1.13e+02	4.24e+02	5.09e+01	**3.48e+02**	**5.73e+01**
F_5	1.17e+02	3.95e+01	2.28	1.47e−01	2.15	1.73e−01	**1.18**	**7.28e−01**
F_6	1.76e+01	1.48	1.31e+01	7.48e−01	1.36e+01	4.93e−01	**1.62**	**2.75e−01**

Table 7
Comparison among DMS-PSO, EPUS-PSO and DP-GOPSO on six shifted and large scale problems ($D = 100$), where “Mean” indicates the mean function error value, and “Std Dev” stands for the standard deviation. The best results among the comparison are shown in bold.

Functions	DMS-PSO [43]		EPUS-PSO [15]		DP-GOPSO	
	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
F_1	**0**	**0**	7.47e−01	1.07e−01	8.75e+02	9.33e+01
F_2	**3.65**	**7.30e−01**	1.86e+01	2.26	1.53e+01	3.17
F_3	2.83e+02	9.40e+02	4.99e+03	4.71e+02	**1.49e+02**	**7.96e+01**
F_4	**1.83e+02**	**2.16e+01**	4.71e+02	5.94e+01	3.48e+02	5.73e+01
F_5	**0**	**0**	3.72e−01	5.60e−02	1.18	7.28e-01
F_6	**0**	**0**	2.06	4.40e-01	1.62	2.75e−01

Table 8
The t-test results between DP-GOPSO and the other five PSO algorithms.

Functions	PSO	OPSO	GOPSO	DMS-PSO [43]	EPUS-PSO [15]
F_1	−14.1415	−22.8221	−28.5047	51.3673	51.3234
F_2	−58.7466	−20.5887	−20.9119	19.6158	−4.64275
F_3	−0.47093	−36.9199	−29.5027	−0.778011	−55.5085
F_4	−16.1162	−5.36061	−5.4313	14.7583	−8.16281
F_5	−11.4498	−8.1123	−7.10022	8.87792	6.06121
F_6	−71.6627	−78.899	−116.237	32.2658	−4.6446

Table 9
Average rankings of the six PSO algorithms on large scale problems.

Algorithm	Ranking
DMS-PSO	5.67
DP-GOPSO	4.83
EPUS-PSO	4.00
GOPSO	2.83
OPSO	2.00
PSO	1.67

Table 10
Wilcoxon test between DP-GOPSO and the other five PSO variants on large scale problems. The p-values below 0.05 are shown in bold.

DP-GOPSO vs.	p-values
PSO	**0.028**
OPSO	**0.028**
GOPSO	**0.028**
DMS-PSO	0.173
EPUS-PSO	0.463

perform worse and the search almost stagnates. It seems that opposition hinders the search of finding better solutions in this case. From the t-test results listed in Table 8, the DP-GOPSO is significantly better than OPSO and GOPSO, because the values of the t-test are less than −2.009. DP-GOPSO is significantly better than PSO except for unction

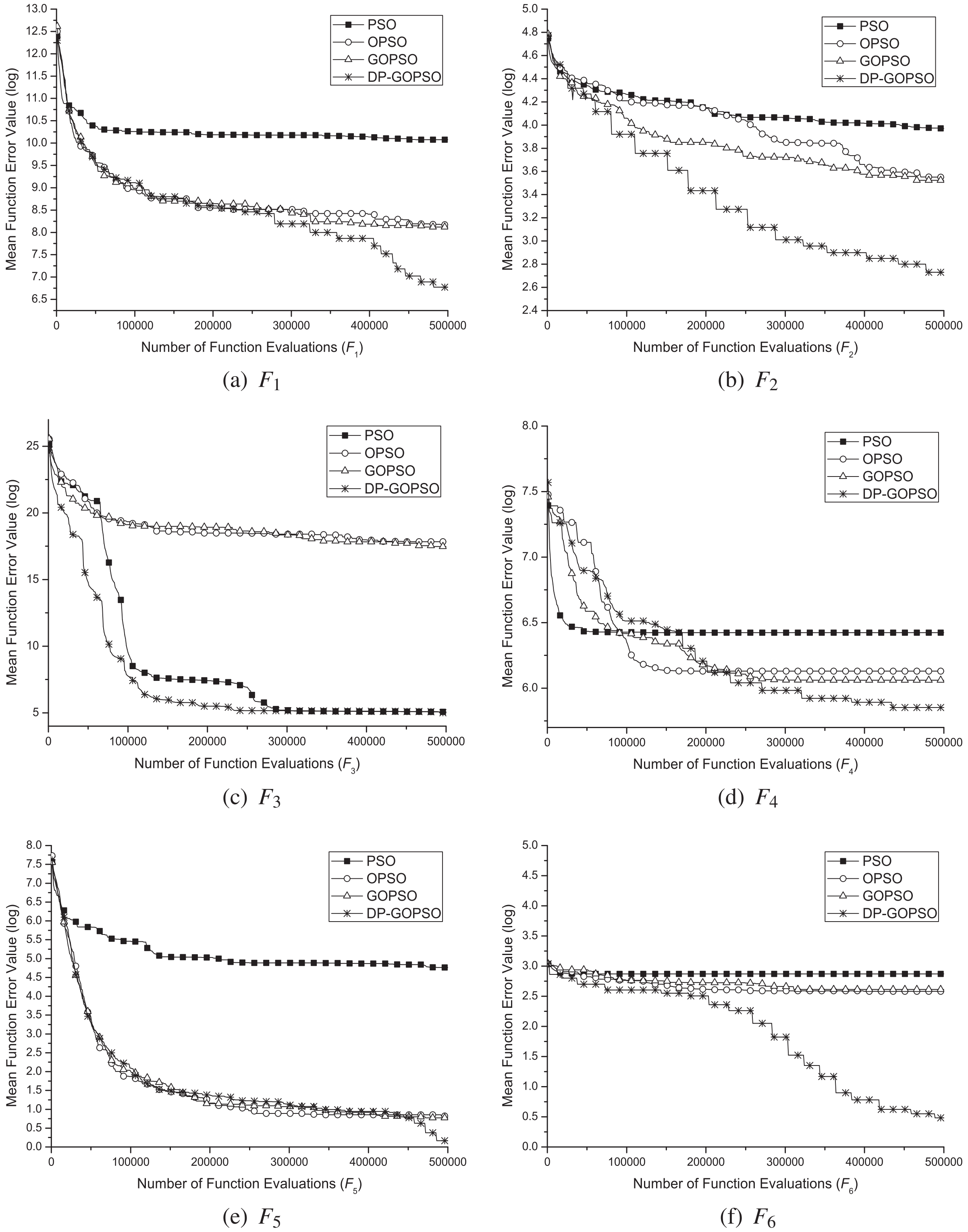


Fig. 5. The convergence trend of PSO, OPSO, GOPSO and DP-GOPSO on shifted and large-scale problems.

F_3. On this function, both DP-GOPSO and PSO almost have the same performance. From the convergence trend of PSO, OPSO, GOPSO and DP-GOPSO as given in Fig. 5, DP-GOPSO converges faster than the other three algorithms on all test functions except for F_5. On this function, OPSO, GOPSO and DP-GOPSO almost have the same convergence rate.

From the comparisons of PSO, OPSO and GOPSO, it can be seen that opposition does not work well for all kinds of problems. For function F_3, both OPSO and GOPSO perform much worse. That is because F_3 is a multimodal, shifted and non-separable function with a narrow valley from local to global optimum. A small change added to a particle will help the particle to escape from a local optimum. Finally, opposition hinders the search for good solutions during the evolution. Maybe opposition is effective when the global optimum is far away from local minima. Under this case, opposition is helpful to generate good solutions, which may force the particle to jump to a better position.

With the help of the dynamic population mechanism, the DP-GOPSO achieves better results than GOPSO and PSO. The DP mechanism manages the population size in the whole searching process. When the current population can continuously find better solutions (the *gbest* has changes), a smaller size population will be enough. When searching with the current population stagnates (the *gbest* has no change), new power (add a new particle or re-initialized the worst particle) will be introduced for the population to push the search forward.

(2) Comparisons of DP-GOPSO with DMS-PSO and EPUS-PSO: Table 7 presents the mean and standard deviation of the 25 runs of DMS-PSO, EPUS-PSO and DP-GOPSO. It can be seen that DMS-PSO outperforms EPUS-PSO and DP-GOPSO except on F_3. DP-GOPSO performs better than EPUS-PSO on F_2, F_3, F_4 and F_6. DMS-PSO shows excellent search abilities on F_1, F_5 and F_6. On these functions, DMS-PSO can find the global optimum, while EPUS-PSO and DP-GOPSO fall into the local minima. From the t-test results listed in Table 8, the DMS-PSO is significantly better than DP-GOPSO except on F_3 because these values of the t-test are greater than 2.009. On function F_3, DP-GOPSO and DMS-PSO almost have the same result. EPUS-PSO is significantly better than DP-GOPSO on F_1 and F_5, while DP-GOPSO is significantly better than EPUS-PSO on the rest 4 functions.

(3) From the results in Table 9, the six algorithms can be sorted by average ranking into the following order: DMS-PSO, DP-GOPSO, EPUS-PSO, GOPSO, OPSO, and PSO. DP-GOPSO obtains the second place, which outperforms the other four PSO algorithms. From the results in Table 10, DP-GOPSO outperforms GOPSO, OPSO, and PSO. Though DP-GOPSO performs better than EPUS-PSO according to the results of average rankings, DP-GOPSO is not better than EPUS-PSO.

Compared with other similar PSO variants (DMS-PSO and EPUS-PSO), DP-GOPSO and EPUS-PSO show a poor performance on shifted and large scale problems. DP-GOPSO only performs better on F_3 than two other algorithms. That can be due to the DP mechanism, but not the opposition. Because GOPSO performs much worse on this function. It suggests that GOPSO, DP-GOPSO and EPUS-PSO are not good choices to solve shifted and large scale problems.

However, our claim is not to introduce the fastest and most robust population-based algorithm, but to develop an acceleration scheme (GOBL) to make them faster. In this direction, PSO has been selected as a case study. That is why we compare the proposed GOPSO algorithm with other variants of PSO.

9. Conclusion

This paper presents an enhanced PSO algorithm called GOPSO by using generalized opposition-based learning and Cauchy mutation. The GOBL sampling scheme could provide more chance of finding better solutions by transforming candidate solutions from current population into a new search space. With the help of a new elite selection mechanism, we can select better particles after this transformation. The Cauchy mutation with a long tail may help trapped particles to jump out of local minima. From the analysis and experiments, we observe that the GOBL and Cauchy mutation enables the GOPSO to achieve better results on unrotated multimodal and rotated multimodal problems when GOPSO is compared with eight other PSO variants. In order to deal with shifted and large scale problems, we proposed an improved GOPSO (DP-GOPSO) by introducing a new dynamic population technique. Experimental results show that DP-GOPSO outperforms PSO, OPSO, GOPSO and EPUS-PSO, while DMS-PSO performs much better than DP-GOPSO.

However, GOPSO is not suitable for all kinds of problems. For instance, GOPSO fails to solve f_2, f_8 and most of the large scale problems. Although DP-GOPSO performs better than EPUS-PSO on 4 functions, DP-GOPSO is not a good choice to solve shifted and large scale problems when it is compared with DMS-PSO.

GOBL is used to accelerate the convergence speed by simultaneously evaluating the current population and the opposite population. The experimental results show that GOBL plays an important role in solving unrotated multimodal and rotated multimodal problems, but performs badly on shifted and large scale problems. Possible future work is to investigate the effectiveness of GOBL on many other different kinds of problems.

Acknowledgments

The authors thank the editor and anonymous reviewers for their detailed and constructive comments that help us to increase the quality of this work. This work was supported by the National Natural Science Foundation of China (No. 61070008).

References

[1] F. van den Bergh, A.P. Engelbrecht, A cooperative approach to particle swarm optimization, IEEE Trans. Evol. Comput. 8 (2004) 225–239.
[2] F. van den Bergh, A.P. Engelbrecht, A study of particle swarm optimization particle trajectories, Inform. Sci. 176 (2006) 937–971.

[3] J. Brest, A. Zamuda, B. Bošković, M.S. Maučec, V. Žumer, High-dimensional real-parameter optimization using self-adaptive differential evolution algorithm with population size reduction, Proc. Congr. Evol. Comput. (2008) 2032–2039.
[4] X. Cai, Z. Cui, J. Zeng, Y. Tan, Particle swarm optimization with self-adjusting cognitive selection strategy, Int. J. Innovat. Comput. Inform. Cont. 4 (2008) 943–952.
[5] J. Chang, S. Chu, J.F. Roddick, J. Pan, A parallel particle swarm optimization algorithm with communication strategies, J. Inform. Sci. Eng. 21 (2005) 809–818.
[6] S. Chu, J.F. Roddick, J. Pan, Communication strategies based particle swarm optimization algorithms, in: Proceedings of International Workshop on Fuzzy System and Innovational Computing, 2004, pp. 425–430.
[7] M. Clerc, J. Kennedy, The particle swarm-explosion, stability, and convergence in a multidimensional complex space, IEEE Trans. Evolut. Comput. 6 (2002) 58–73.
[8] Z. Cui, J. Zeng, G. Sun, A fast particle swarm optimization, Int. J. Innovat. Comput. Inform. Cont. 2 (2006) 1365–1380.
[9] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[10] W. Du, B. Li, Multi-strategy ensemble particle swarm optimization for dynamic optimization, Inform. Sci. 178 (2008) 3096–3109.
[11] W. Feller, An Introduction to Probability Theory and its Applications, second ed., Wiley, New York, 1971.
[12] S. García, A. Fernández, J. Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power, Inform. Sci. 180 (2010) 2044–2064.
[13] S. García, F. Herrera, An extension on Statistical comparisons of classifiers over multiple data sets for all pairwise comparisons, J. Mach. Learn. Res. 9 (2008) 2677–2694.
[14] S. García, D. Molina, M. Lozano, F. Herrera, A study on the use of non-parametric tests for analyzing the evolutionary algorithms behaviour: a case study on the CEC2005 special session on real parameter optimization, J. Heuristics 15 (2009) 617–644.
[15] S. Hsieh, T. Sun, C. Liu, S. Tsai, Solving large scale global optimization using improved particle swarm optimizer, Proc. Congr. Evol. Comput. (2008) 1777–1784.
[16] X. Hu, Y. Shi, R.C. Eberhart, Recent advances in particle swarm, Proc. Congr. Evol. Comput. (2004) 90–97.
[17] J. Kennedy, R.C. Eberhart, Particle swarm optimization, Proc. IEEE Int. Conf. Neural Networks (1995) 1942–1948.
[18] J. Kennedy, Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance, Proc. Congr. Evol. Comput. (1999) 1931–1938.
[19] J.J. Liang, A.K. Qin, P.N. Suganthan, S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, IEEE Trans. Evolut. Comput. 10 (2006) 281–295.
[20] H. Lin, X. He, A novel opposition-based particle swarm optimization for noisy problems, in: Proceedings of International Conference on Natural Computation, 2007, pp. 624–629.
[21] A.R. Malisia, H.R. Tizhoosh, Applying opposition-based ideas to the ant colony system, in: Proceedings of IEEE Swarm Intelligence Symposium, 2007, 182–189.
[22] R. Mendes, J. Kennedy, J. Neves, The fully informed particle swarm: simpler, maybe better, IEEE Trans. Evolut. Comput. 8 (2004) 204–210.
[23] K.E. Parsopoulos, M.N. Vrahatis, Recent approaches to global optimization problems through particle swarm optimization, Nat. Comput. (2002) 235–306.
[24] K.E. Parsopoulos, M.N. Vrahatis, UPSO–A unified particle swarm optimization scheme, Lect. Ser. Comput. Sci. (2004) 868–873.
[25] T. Peram, K. Veeramachaneni, C.K. Mchan, Fitness-distance-ratio based particle swarm optimization, in: Proceedings of IEEE Swarm Intelligence Symposium, 2003, 174–181.
[26] S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, Opposition-based differential evolution algorithms, Proc. Congr. Evol. Comput. (2006) 2010–2017.
[27] S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, Opposition-based differential evolution for optimization of noisy problems, Proc. Congr. Evol. Comput. (2006) 1865–1872.
[28] S. Rahnamayan, H.R. Tizhoosh, M.M. ASalama, Quasi-oppositional differential evolution, Proc. Congr. Evol. Comput. (2007) 2229–2236.
[29] S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, Opposition-based differential evolution, IEEE Trans. Evolut. Comput. 12 (2008) 64–79.
[30] Y. Shi, R.C. Eberhart, A modified particle swarm optimizer, Proc. Congr. Evol. Comput. (1998) 69–73.
[31] K. Tang, X. Yao, P.N. Suganthan C. Macnish, Y. Chen, C. Chen, Z. Yang, Benchmark functions for the CEC'2008 special session and competition on high-dimensional real-parameter optimization. Technical Report, Nature Inspired Computation and Applications Laboratory, USTC, China, 2007. <http://www.nical.ustc.edu.cn/cec08ss.php>.
[32] H.R. Tizhoosh, Opposition-based learning: a new scheme for machine intelligence, in: Proceedings of International Conference on Computational Intelligence for Modeling Control and Automation, 2005, 695–701.
[33] P.K. Tripathi, S. Bandyopadhyay, S.K. Pal, Multi-objective particle swarm optimization with time variant inertia and acceleration coefficients, Inform. Sci. 177 (2007) 5033–5049.
[34] L. Tseng, C. Chen, Multiple trajectory search for large scale global optimization, Proc. Congr. Evol. Comput. (2008) 3057–3064.
[35] M. Ventresca, H.R. Tizhoosh, A diversity maintaining population-based incremental learning algorithm, Inform. Sci. 178 (21) (2008) 4038–4056.
[36] H. Wang, Y. Liu, C. Li, S. Zeng, A hybrid particle swarm algorithm with Cauchy mutation, in: Proceedings of IEEE Swarm Intelligence Symposium, 2007, pp. 356–360.
[37] H. Wang, Y. Liu, S. Zeng, H. Li, C. Li, Opposition-based particle swarm algorithm with Cauchy mutation, Proc. Congr. Evol. Comput. (2007) 4750–4756.
[38] H. Wang, Z. Wu, Y. Liu, J. Wang, D. Jiang, L. Chen, Space transformation search: a new evolutionary technique, Proc World Summit Genetic Evolut. Comput. (2009) 537–544.
[39] H. Wang, Z. Wu, S. Rahnamayan, J. Wang, Diversity analysis of opposition-based differential evolution–an experimental study, in: International Symposium on Intelligence Computation and Applications (ISICA 2010), (LNCS 6382) 2010, pp. 95–102.
[40] Y. Wang, B. Li, A restart univariate estimation of distribution algorithm sampling under mixed Gaussian and Lévy probability distribution, Proc. Congr. Evol. Comput. (2008) 3918–3925.
[41] Y. Wang, Y. Yang, Particle swarm optimization with preference order ranking for multi-objective optimization, Inform. Sci. 179 (2009) 1944–1959.
[42] Z. Yang, K. Tang, X. Yao, Large scale evolutionary using cooperative coevolution, Inform. Sci. 178 (2008) 2985–2999.
[43] S. Zhao, J. Liang, P.N. Suganthan, M.F. Tasgetiren, Dynamic multi-swarm particle swarm optimizer with local search for large scale global optimization, Proc. Congr. Evol. Comput. (2008) 3846–3853.