Assignment: Evaluation on Particles Swarm Optimizer for Multimodal Functions

Wang Maosen

School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore
Email: wang1496@e.ntu.edu.sg

Abstract: Particle swarm algorithm is one of a current research hotspot in the field of optimization algorithms. In this assignment, five algorithms including particle swarm optimizer (PSO), comprehensive learning particle swarm optimize (CLPSO), unified particle swarm optimization (UPSO), cooperative particle swarm optimizer (CPSO), and fitness-distance-ratio based PSO (FDR-PSO) are evaluated. All the algorithms are described in the first part, among them, PSO and CLPSO are described in conjunction with Matlab code. Parameters are tuned in part 3. The method is to tune parameter in a large range at first, then tune parameter in a small range. Combining the average value, standard deviation, and convergence plots, the final parameter can be selected. In part 4, each algorithm run 30 times on ten chosen test functions. To evaluate their performance, t-test and convergence plots are used in this section. Then, experimental results are discussed, and the conclusion is presented in section 5. (Assignment Codes and experimental data available from https://github.com/WWangMaosen/My_Assignment_Code)

Keywords: PSO, CLPSO, UPSO, CPSO, FDR-PSO, Parameter Tuning, T-Test.

---

1. Algorithm Descriptions

1.1. PSO Description

PSO is a random swarm intelligent optimization method with high efficiency, which has been widely used in a wide range of scientific research and engineering practice [1]. It imitates the process of birds foraging, all birds are randomly distributed anywhere in a specific area with different flying speeds, and eventually all birds fly to places where have foods.

In PSO algorithm, all particles are randomly distributed in the feasible solution space of the problem and move at a certain speed. The Matlab code to implement PSO is shown in Fig. 1, where *VRmax* and *VRmin* represent the solution space. For example, when we are finding the minimal value of a function f(x), *VRmax* and *VRmin* represent the upper and lower boundaries of the x value in each dimension, respectively. The position of particles is randomly generated in code line 19, and the velocity is also randomly generated in line 24 according to *VRmax* and *VRmin*. Then, particles need to calculate its fitness value according to the position as shown in line 21. Since the position and velocity of the particles have just been randomly generated, and no iteration has occurred, so the present fitness value is used to initialize *pbest* (*pbest* represents particle's the best solution found by so far) and *gbest* (*gbest* represents the global best solution) fitness value as shown in line 26 and line 28. In line 27, the algorithm is finding the best fitness value and its position is recorded in *gbest* vector.

Fig. 2 represents the code of PSO iteration process. After initialization, every particle updates its position and velocity according to formula (1) and formula (2) and it is realized from code line 33 to line 37. The original PSO velocity updating method is shown in formula (3). Y. Shi et.al introduce a new parameter into the original PSO formula, inertia factor (w), which determines the influence of the particle's current speed on the direction of movement [2]. The larger the inertia factor is, it is helpful to jump out of the local extreme value. The smaller the inertia factor, the more conducive to the convergence of the algorithm and local search. Usually, the inertia factor will decrease as the number of iterations increases. cc(1) and cc(2) are regarded as learning rate or acceleration constants. The function of these two coefficients is to make particles have social and cognition ability.

$$V_{id} \leftarrow w * V_{id} + \phi_1 * \mathrm{rnd}() * (P_{id} - X_{id}) + \phi_2 * \mathrm{rnd}() * (P_{gd} - X_{id}) \quad (1)$$

$$X_{id} \leftarrow X_{id} + V_{id} \quad (2)$$

$$V_{id} \leftarrow V_{id} + \phi_1 * \mathrm{rnd}() * (P_{id} - X_{id}) + \phi_2 * \mathrm{rnd}() * (P_{gd} - X_{id}) \quad (3)$$

```
14 mv=0.5*(VRmax-VRmin);
15 VRmin=repmat(VRmin,ps,1);
16 VRmax=repmat(VRmax,ps,1);
17 Vmin=repmat(-mv,ps,1);
18 Vmax=-Vmin;
19 pos=VRmin+(VRmax-VRmin).*rand(ps,D);
20
21 e=feval(fhd,pos',varargin{:});
22
23 fitcount=ps;
24 vel=Vmin+2.*Vmax.*rand(ps,D);%initialize the velocity of the particles
25 pbest=pos;
26 pbestval=e; %initialize the pbest and the pbest's fitness value
27 [gbestval,gbestid]=min(pbestval);
28 gbest=pbest(gbestid,:);%initialize the gbest and the gbest's fitness value
29 gbestrep=repmat(gbest,ps,1);
```

Figure 1. Initialization process.

```
31 for i=2:me
32
33     aa=cc(1).*rand(ps,D).*(pbest-pos)+cc(2).*rand(ps,D).*(gbestrep-pos);
34     vel=iwt(i).*vel+aa;
35     vel=(vel>Vmax).*Vmax+(vel<=Vmax).*vel;
36     vel=(vel<Vmin).*Vmin+(vel>=Vmin).*vel;
37     pos=pos+vel;
38     pos=((pos>=VRmin)&(pos<=VRmax)).*pos...
39         +(pos<VRmin).*(VRmin+0.25.*(VRmax-VRmin).*rand(ps,D))+(pos>VRmax).*...
40         (VRmax-0.25.*(VRmax-VRmin).*rand(ps,D));
41     e=feval(fhd,pos',varargin{:});
42     fitcount=fitcount+ps;
43     tmp=(pbestval<e);
44     temp=repmat(tmp',1,D);
45     pbest=temp.*pbest+(1-temp).*pos;
46     pbestval=tmp.*pbestval+(1-tmp).*e;%update the pbest
47     [gbestval,tmp]=min(pbestval);
48     gbest=pbest(tmp,:);
49     gbestrep=repmat(gbest,ps,1);%update the gbest
50 end
```

Figure 2. Iteration Process.

Code line 35 and line 36 is to make sure that particles' velocity is within the specified range. The upper boundary value is taken when the position exceeding the maximum range, and the lower boundary value is taken when the position lower than the minimum range. Code line 38 is to make sure that particles' position is within the specified range. When it is outside the boundary, then take the close boundary value, and then move inward randomly. After that, every particle calculates its new fitness value and updates pbest and gbest vector from code line 41 to code line 49.

1.2. CLPSO Description

PSO and its variants have a main deficiency, that is premature convergence. In these algorithms, each particle closely follows pbest and gbest, even if gbest has deviated from the global optimum. This will cause the particles to fall into a local optimum. In 2006, J. Liang and Suganthan et. al proposed CLPSO algorithm, which effectively improves the shortcomings of the premature convergence of the PSO algorithm [3]. The innovations of CLPSO are as follows:

- Particles learn from other particle's pbest in the same dimension by random or tournament selection, instead of only learning from their own historical information and global social information.
- Combine two exemplars that guide the direction of particle's flight in each generation into one exemplar. Learning strategy is shown in formula (4) and (5).

$$v_i^d \leftarrow w \times v_i^d + c \times rand_i^d \times (pbest_{f_i(d)}^d - x_i^d) \quad (4)$$

$$x_i^d \leftarrow x_i^d + v_i^d \quad (5)$$

In order to explain the operation process of CLPSO in detail, this part describes the CLPSO algorithm in conjunction with Matlab code. Fig. 3. Presents the initialization process. Line 9 generates a crossover probability matrix increasing from 0 to 0.5. Then, it defines the boundary of swarms' velocity and position according to the input parameter. Line 24 calculate the fitness of the current position by calling the testing file. After that, current pbest and gbest vector is initialized.

Fig. 4 shows the process of generating comprehensive learning factor. Firstly, we randomly generate two matrix fi1 and fi2. The parameters in the two matrixs represent the number of particles in the corresponding dimension. Then, comparing the fitness of two particles in the same dimension, selecting the smaller fitness value, and storing the particle number into fi in the same dimension.

f_pbest is a 100*10 matrix containing particle number. For each particle, some dimension will be replaced by fi matrix according to the crossover probability. *f_pbest* is the learning factor which will be used later. Fig. 5 shows the position generating process. *pbest_f* contains the positions of particles in each dimension after crossover, and it representative $pbest^{d}_{f_i(d)}$ in formula (4).

```
cc=[1 1];   %acceleration constants
t=0:1/(ps-1):1;t=5.*t;
Pc=0.0+(0.5-0.0).*(exp(t)-exp(t(1)))./(exp(t(ps))-exp(t(1)));
m=0.*ones(ps,1);
iwt=0.9-(1:me)*(0.7/me);
cc=[1.49445 1.49445];
if length(VRmin)==1
    VRmin=repmat(VRmin,1,D);
    VRmax=repmat(VRmax,1,D);
end
mv=0.2*(VRmax-VRmin);
VRmin=repmat(VRmin,ps,1);
VRmax=repmat(VRmax,ps,1);
Vmin=repmat(-mv,ps,1);
Vmax=-Vmin;
pos=VRmin+(VRmax-VRmin).*rand(ps,D);
for i=1:ps;
e(i,1)=feval(fhd,pos(i,:),varargin{:});
end

fitcount=ps;
vel=Vmin+2.*Vmax.*rand(ps,D);%initialize the velocity of the particles
pbest=pos;
pbestval=e; %initialize the pbest and the pbest's fitness value
[gbestval,gbestid]=min(pbestval);
gbest=pbest(gbestid,:);%initialize the gbest and the gbest's fitness value
gbestrep=repmat(gbest,ps,1);
stay_num=zeros(ps,1);
```

Figure 3. Initialization process.

```
ai=zeros(ps,D);
f_pbest=1:ps;f_pbest=repmat(f_pbest',1,D);
for k=1:ps
ar=randperm(D);
ai(k,ar(1:m(k)))=1;
fi1=ceil(ps*rand(1,D));
fi2=ceil(ps*rand(1,D));
fi=(pbestval(fi1)<pbestval(fi2))'.*fi1+(pbestval(fi1)>=pbestval(fi2))'.*fi2;
bi=ceil(rand(1,D)-1+Pc(k));
if bi==zeros(1,D),rc=randperm(D);bi(rc(1))=1;end
f_pbest(k,:)=bi.*fi+(1-bi).*f_pbest(k,:);
end
```

Figure 4. Comprehensive Learning Factor.

```
for dimcnt=1:D
    pbest_f(k,dimcnt)=pbest(f_pbest(k,dimcnt),dimcnt);
end
```

Figure 5. Position Generating.

After preparing $pbest^{d}_{f_i(d)}$ in formula (4), CLPSO algorithm starts to update the speed as shown in Figure 5. In fact, ai and m are zero matrix. So, in line 75, the second term in can be ignored. One minus zero matrix is one. Therefore, the shape of code line 75 and formula are exactly same. It can be seen from the code that cc(2) has no effect on the velocity, since CLPSO does not use global best matrix. After calculating new velocity, all values whose speed exceeds the upper or lower boundaries will be replaced by the closer boundary values as shown in line 78 and line 79. The new position can be calculated by adding velocity to the former position.

```
75   aa(k,:)=cc(1).*(1-ai(k,:)).*rand(1,D).*(pbest_f(k,:)-pos(k,:))...
76           +cc(2).*ai(k,:).*rand(1,D).*(gbestrep(k,:)-pos(k,:));
77   vel(k,:)=iwt(i).*vel(k,:)+aa(k,:);
78   vel(k,:)=(vel(k,:)>mv).*mv+(vel(k,:)<=mv).*vel(k,:);
79   vel(k,:)=(vel(k,:)<(-mv)).*(-mv)+(vel(k,:)>=(-mv)).*vel(k,:);
80   pos(k,:)=pos(k,:)+vel(k,:);
```

Figure 6. Updating Velocity.

After gaining new position, particle will calculate its new fitness value and update the *pbest* vector. In Matlab code, the *gbest* vector is also be updated. In fact, these codes are redundant, because CLPSO does not require global best information.

```
if (sum(pos(k,:)>VRmax(k,:))+sum(pos(k,:)<VRmin(k,:)))==0;
e(k,1)=feval(fhd,pos(k,:),varargin{:});
fitcount=fitcount+1;
tmp=(pbestval(k)<=e(k));
if tmp==1
    stay_num(k)=stay_num(k)+1;
end
temp=repmat(tmp,1,D);
pbest(k,:)=temp.*pbest(k,:)+(1-temp).*pos(k,:);
pbestval(k)=tmp.*pbestval(k)+(1-tmp).*e(k);%update the pbest
if pbestval(k)<gbestval
gbest=pbest(k,:);
gbestval=pbestval(k);
gbestrep=repmat(gbest,ps,1);%update the gbest
end
```

Figure 7. Updating pbest vector

1.3. Other PSO Variants

FDR-PSO was proposed by Thanmaya Peram et al [4]. The velocity updating method is shown in formula (6), and the position updating method is the same as PSO. FDR-PSO has one more item in the speed updating formula than PSO. In this algorithm, particle not only move to the global optimal particle, but also move to nearby particle, which has a better fitness value. In order to find the best neighbor particle, FDR-PSO design a computing method, which should be maximized, as shown in formula (7). This algorithm alleviates the premature convergence problem in PSO algorithm especially when optimizing complex functions.

$$v_{\mathrm{id}}^{t+1} \leftarrow w \times v_{\mathrm{id}}^{t} + \psi_1 \times (p_{id} - x_{id}) + \psi_2 \times (p_{gd} - x_{id}) + \psi_3 \times (p_{nd} - x_{id}) \tag{6}$$

$$\frac{Fitness(P_j) - Fitness(X_i)}{\left| P_{jd} - X_{id} \right|} \tag{7}$$

Frans van den Bergh et al. proposed the CPSO in 2004 [5]. The idea of the method is to divide the particle swarm into many small swarms, just like many families. Each sub-particle swarm works collaboratively to optimize different components of the problem to achieve better optimization results. UPSO is proposed in 2004 by K.E. Parsopoulos and M.N. Vrahatis [6]. This algorithm combines two basic PSO variants' exploitation and exploration ability to make it perform global search and converge faster [7]. The main scheme of UPSO is adding a parameter to control the influence of velocity in each direction as shown in formula (8). In this formula, u is regarded as unification factor, determining the proportion of global and local components.

$$U_{\mathrm{i}}^{(k+1)} = uG_i^{(k+1)} + (1-u)L_i^{(k+1)} \tag{8}$$

Where $u \in [0,1]$, $G_i^{(k+1)}$ and $L_i^{(k+1)}$ represent velocity update for global and local variant respectively. In this experiment, u is set at 0.1.

2. Test Functions

To evaluate the algorithms above, ten problems are chosen from CEC 2017 bound constrained benchmarks, as shown in table 1. To evaluate function comprehensively, four basic functions, two hybrid functions, and four composition functions are selected. When I adjust the parameters, I use the four composition functions and the first basic function, which can make sure that the parameter can be well applied to more functions.

Table 1. Test Functions.

Function name and number(case) in "cec17_func.cpp" file	Function Formula
1) Shifted and Rotated Bent Cigar	$f_1(x) = x_1^2 + 10^6 \sum_{i=2}^{D} x_i^2$ $F_1(x) = f_1(M(x - o_1)) + F_1{*}$
4) Shifted and Rotated Rosenbrock's Function	$f_4(x) = \sum_{i=1}^{D-1} (100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2)$ $F_4(x) = f_4(M(\frac{2.048(x - o_4)}{100}) + 100) + F_4{*}$
7) Lunacek bi-Rastrigin Function	$f_7(x) = \min(\sum_{i=1}^{D} (x_i - u_0)^2, dD + s\sum_{i=1}^{D} (x_i - u_1)^2) + 10(D - \sum_{i=1}^{D} \cos(2\pi z_i))$ $F_7(\mathrm{x}) = f_7(M(\frac{600(x - o_7)}{100})) + F_7{*}$
10) Shifted and Rotated Schwefel's Function	$f_{10}(x) = 418.9829 \times D - \sum_{i=1}^{D} g(z_i)$ $z_i = x_i + 4.209687462275036e + 002$ $g(z_i) = \begin{cases} z_i \sin(\lvert z_i \rvert^{1/2}), if\ \lvert z_i \rvert \le 500 \\ (500 - \mathrm{mod}(z_i, 500)) \sin(\sqrt{\lvert 500 - \mathrm{mod}(\lvert z_i \rvert, 500) \rvert}) - \frac{(z_i - 500)^2}{10000D}, if z_i > 500 \\ (\mathrm{mod}(z_i, 500) - 500) \sin(\sqrt{\lvert \mathrm{mod}(\lvert z_i \rvert, 500) - 500 \rvert}) - \frac{(z_i + 500)^2}{10000D}, if z_i < -500 \end{cases}$ $F_{10}(\mathrm{x}) = f_{10}(M(\frac{1000(x - o_{10})}{100})) + F_{10}{*}$
15) Hybrid Function 5	$N = 4$ $p = [0.2, 0.2, 0.3, 0.3]$ g_1: Bent Cigar Function f_1 g_2: HGBat Function f_{18} g_3: Rastrigin's Function f_5 g_4: Rosenbrock's Function f_4
18) Hybrid Function 8	$N = 5$ $p = [0.2, 0.2, 0.2, 0.2, 0.2]$ g_1: High Conditioned Elliptic Function f_1 g_2: Ackley's Function f_{13} g_3: Rastrigin's Function f_5 g_4: HGBat Function f_{18} g_5: Discus Function f_{12}
26) Composition Function 6	N=5, σ=[10,20,20,30,40], λ=[1e-26,10,1e-6,10,5e-4], $bias$=[0,100,200,300,400] g_1: Expanded Scaffer's F6 Function F_6' g_2: Modified Schwefel's Function F_{10}' g_3: Griewank's Function F_{15}' g_4: Rosenbrock's Function F_4' g_5: Rastrigin's Function F_5'

27) Composition Function 7	N=6, σ=[10,20,30,40,50,60], λ=[10,10,2.5,1e-26,1e-6,5e-4], $bias$=[0,100,200,300,400,500] g_1: HGBat Function F_{18}' g_2: Rastrigin's Function F_5' g_3: Modified Schwefel's Function F_{10}' g_4: Bent Cigar Function F_{11}' g_4: High Conditioned Elliptic Function F_{11}' g_5: Expanded Scaffer's F6 Function F_6'
28) Composition Function 8	N=6, σ=[10,20,30,40,50,60], λ=[10,10,1e-6,1,1,5e-4], $bias$=[0,100,200,300,400,500] g_1: Ackley's Function F_{13}' g_2: Griewank's Function F_{15}' g_3: Discus Function F_{12}' g_4: Rosenbrock's Function F_4' g_4: HappyCat Function F_{17}' g_5: Expanded Scaffer's F6 Function F_6'
29) Composition Function 9	N=3, σ=[10,30,50], λ=[1,1,1], $bias$=[0,100,200] g1: Hybrid Function 5 F_5' g2: Hybrid Function 8 F_8' g3: Hybrid Function 9 F_9'

3. Tuning Experiments

From algorithm description, PSO and CLPSO has three and two parameters separately, that we can tune to improve the accuracy of the algorithm, including one inertia factor and two learning factors in PSO, one inertia factor and one learning factor in CLPSO. The basic idea is changing parameters and running algorithm on five test functions for ten times, so the objective value of five functions under different parameters can be obtained. Then, Plotting the results in the same graph, analyzing the fluctuation, standard deviation and convergence time to select the most suitable parameter.

3.1. PSO Tuning

3.1.1. Inertia Factor (*iwt*)

According to reference [2], a decrease from 1.4 to 0.5 can have a good result in their experiments. Usually, this parameter should run in a large value at the early stage to help the algorithm jump out local extreme value and run in a small value in the last generations to converge. Therefore, I design six parameters for inertia factor. They are decreasing from 1.4 to 0.5, 1.2 to 0.5, 1.0 to 0.4, 0.9 to 0.4, 0.8 to 0.3, 0.7 to 0.2. The results are presented in Fig. 6, and the standard deviation is calculated in Table 2. From the data in the table and figure, we can see that when the inertia factor is decreasing from 0.9 to 0.4, its standard deviation is the smallest among the three test functions. In test function 28, all the parameters have large fluctuation, and in test function 29, the stability of *iwt* decreasing from 0.9 to 0.4 ranks second, and the stability of the first parameter in function 29 is not good in function 26 and function 28. From the function calculation result, this parameter can get the most minimum value in this parameter set, so 0.9~0.4 is regarded as an optional parameter. Then, convergence plot is drawn as shown in Fig.8 (f), it can be seen from the picture that the smaller the inertia factor, the faster the convergence. When it is decreasing from 0.9 to 0.4, not only it converges faster, but also converges into a smaller value. So this parameter is regarded as my final value.

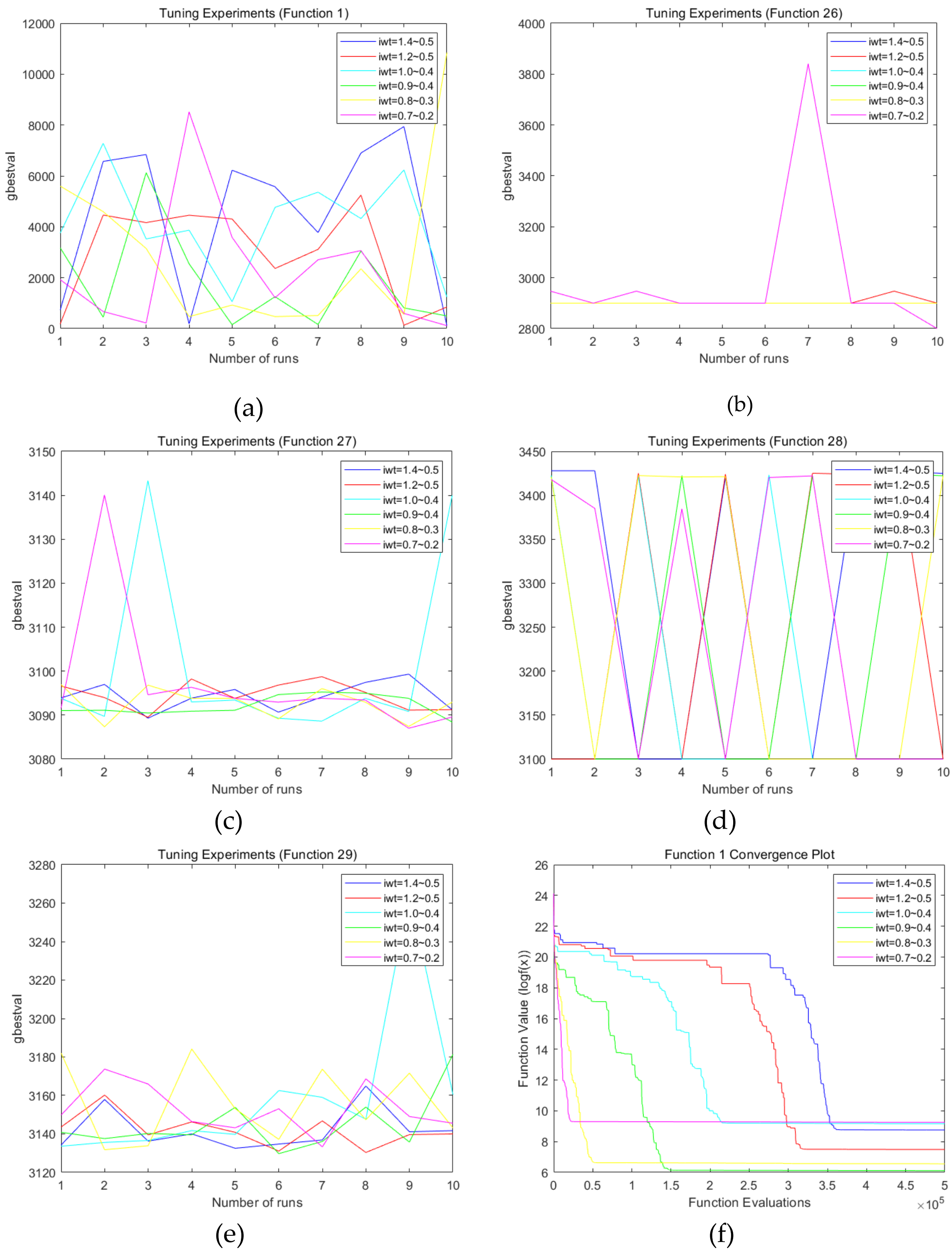


Figure 8. PSO Inertia Factor Tuning Experiments, (a) Test Function 1, (b) Test Function 26, (c) Test Function 27, (d) Test Function 28, (e) Test Function 29, (f)Convergence Plot.

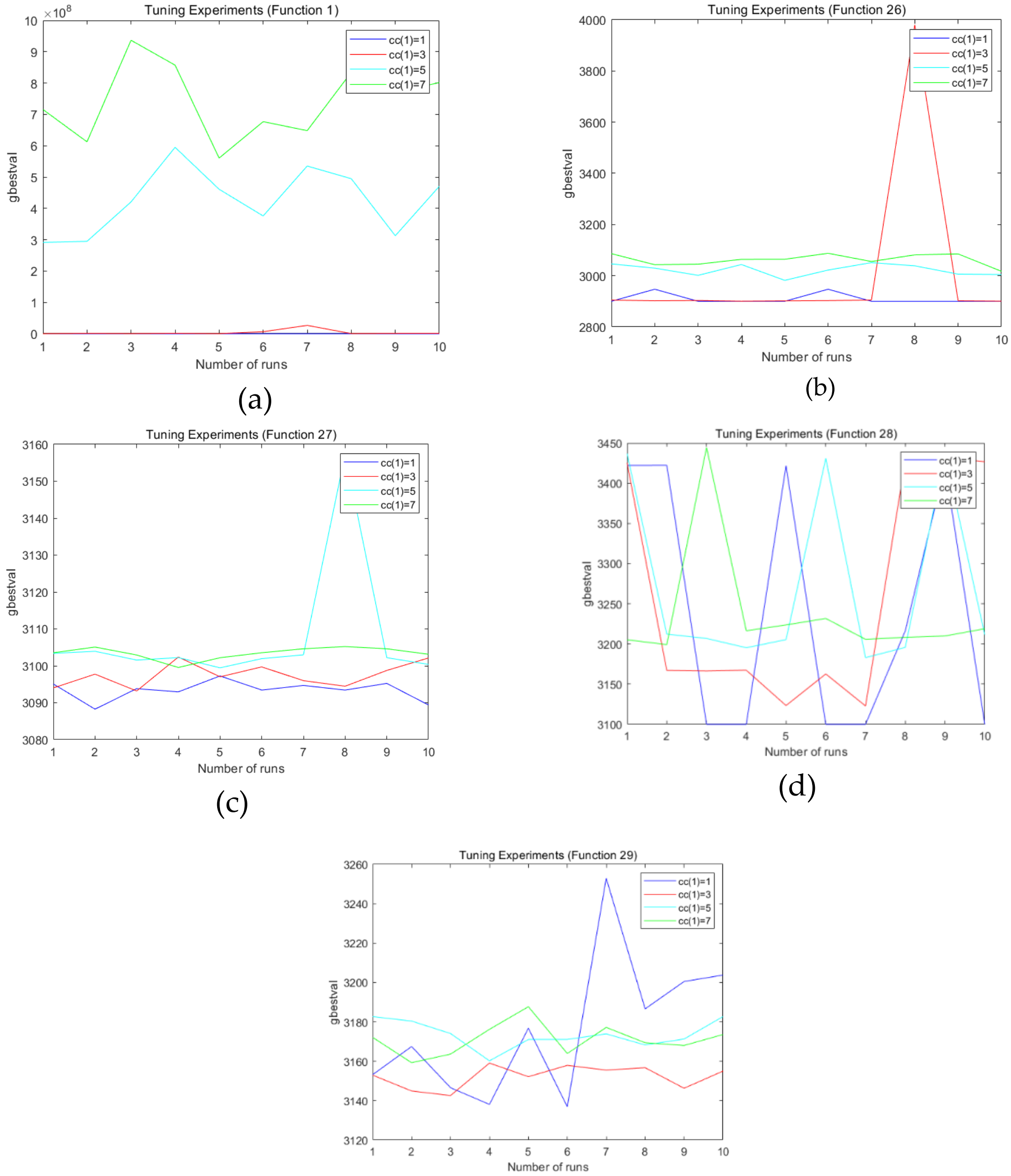
Table 2. PSO Inertia Factor Tuning Experiments (standard deviation).

Inertia Factor	Function 1	Function 26	Function 27	Function28	Function29
1.4~0.5	3057.3	0	3.2	167.9	10.8
1.2~0.5	1930	14.9	3.12	171.3	8.5
1.0~0.4	1962.8	0	21.2	155.8	39.1

0.9~0.4	1911.8	0	2.3	169.9	14.9
0.8~0.3	3342.8	0	3.6	169.5	20.3
0.7~0.2	2516.7	300.4	15.3	161.8	12.7

3.1.2. First Learning Rate (cc1)

The method to tune learning rate is tuning parameter in a large range, and gain the results running in different function. Then, the results of different parameter would be obvious, so it is easier to choose the preferred range to tune the parameter. After that, the most suitable parameter can be found in the small range according to the results. The tuning experiment with large range is presented in Fig. 9 and table 3. It's obvious that when cc1 is set from 1 to 3, the average value is smaller than 5 and 7, so the next step is to tune parameter in range (1,3).



(e)

Figure 9. PSO First Learning Rate Wide Range Tuning Experiments, (a) Test Function 1, (b) Test Function 26, (c) Test Function 27, (d) Test Function 28, (e) Test Function 29.

Table 3. PSO First Learning Rate Wide Range Tuning Experiments (standard deviation).

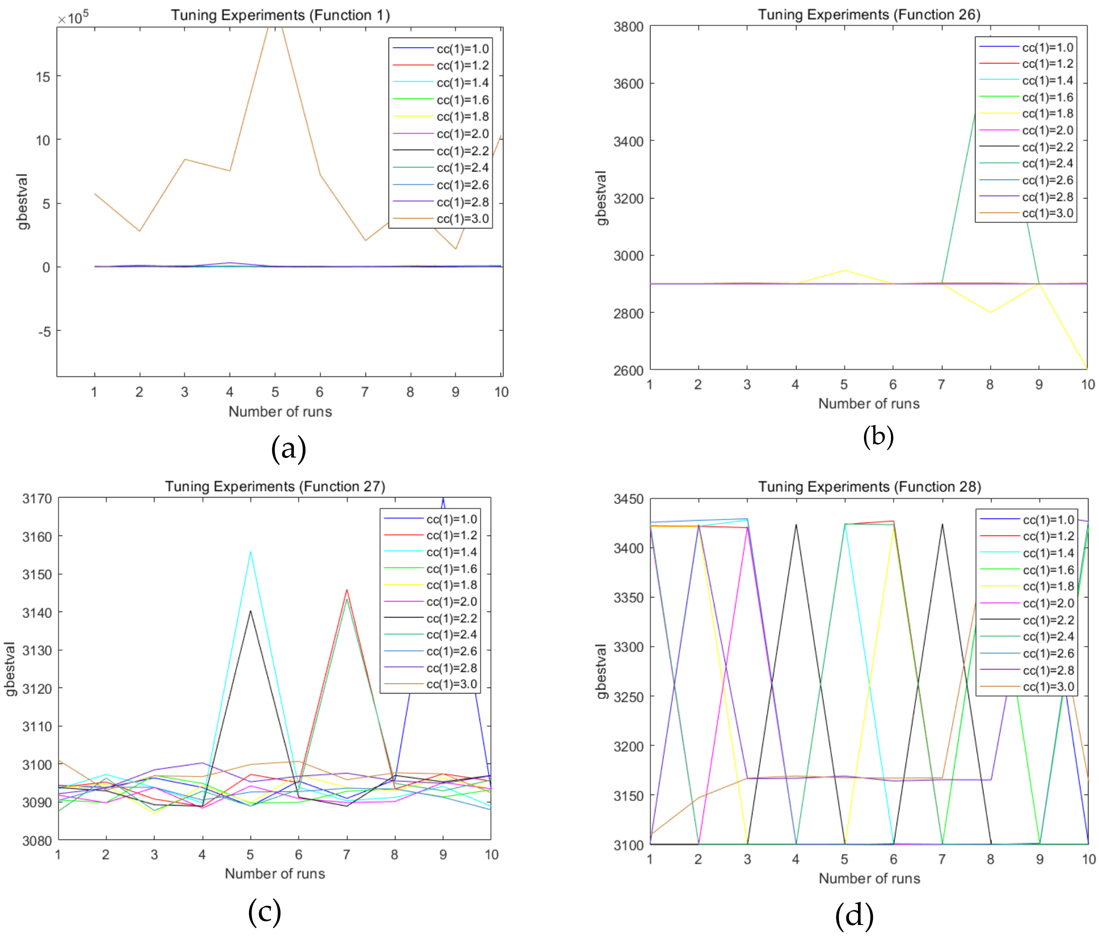
Learning rate	Function 1	Function 26	Function 27	Function28	Function29
cc(1)=1	1342.4	19.8	2.7	160.2	36.2
cc(1)=3	8207847.2	340.7	3.2	143.7	5.78
cc(1)=5	104934006.3	23.1	17.5	113	7
cc(1)=7	118975846.5	23.2	1.7	74.7	8.2

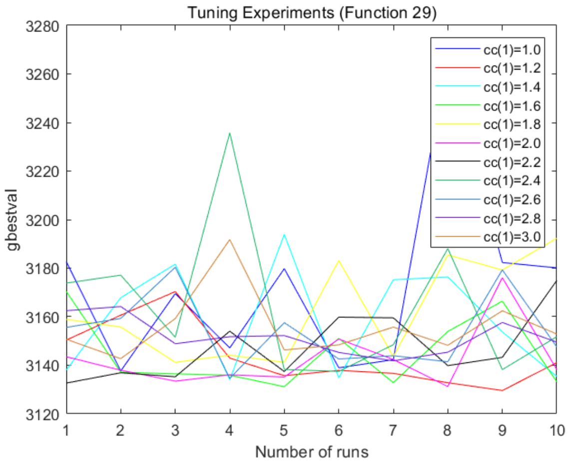
The first learning rate tuning experiments' result with small range is shown in Fig 10. and Table 4. Obviously, the standard deviation in function 1 is very great when the learning rate close to 3, so the learning rate of 3 will not be considered in subsequent selection. For the remaining 10 parameters, standard deviation percentage sum (SDPS) is calculated in the last column in the table. The SDPS calculating formula is shown below.

$$p_i = \sum_{f=1}^{5} \frac{v_{if}}{\sum_{j=1}^{10} v_{jf}} \tag{9}$$

Where p_i means the SDPS, v_{if} means the standard deviation value of the i-th row and j-th column.

From the table 4, we can see that when cc(1) is 1.6 and 2.0, the value of SDPS is the smallest. When cc(1) is equal 1.6, it has a better performance in Function 1, while when cc(1) is equal to 2.0, it has a better performance in Function 27, 28, and 29. In addition, the mean values of these two curves are approximately equal, so I choose 2.0 as my final value of the first learning rate.





(e)

Figure 10. PSO First Learning Rate Small Range Tuning Experiments, (a) Test Function 1, (b) Test Function 26, (c) Test Function 27, (d) Test Function 28, (e) Test Function 29.

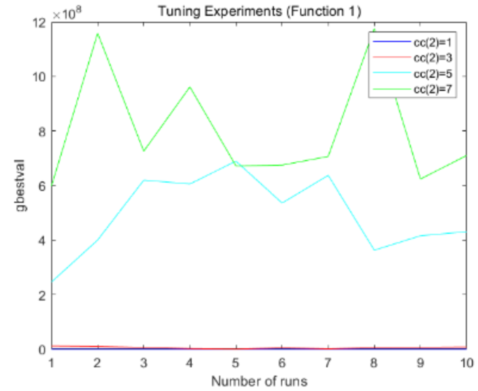
Table 4. PSO First Learning Rate Small Range Tuning Experiments (standard deviation).

Learning rate	Function 1	Function 26	Function 27	Function28	Function29	SDPS
cc(1)=1.0	3113.9	0	24.2	156.6	38.2	62.26%
cc(1)=1.2	3454.8	0	16.6	166.7	13.0	43.37%
cc(1)=1.4	2030.4	0	20.2	167.2	22.6	47.84%
cc(1)=1.6	2007.7	0	2.4	155.3	14.6	25.83%
cc(1)=1.8	4114.7	100	3.0	154.9	20.5	62.18%
cc(1)=2.0	3029.0	0	2.3	135.4	13.1	26.55%
cc(1)=2.2	2554.2	0	15.4	136.6	13.9	38.18%
cc(1)=2.4	2566.5	274.5	16.5	156.5	30.9	122.81%
cc(1)=2.6	2718.8	0	2.0	169.0	15.6	28.90%
cc(1)=2.8	9572.2	0.05	2.4	132.5	7.4	42.08%
sum	35162.2	374.55	105	1530.7	189.8	500%
cc(1)=3.0	562586.5	1	2.9	115.7	14.0	-

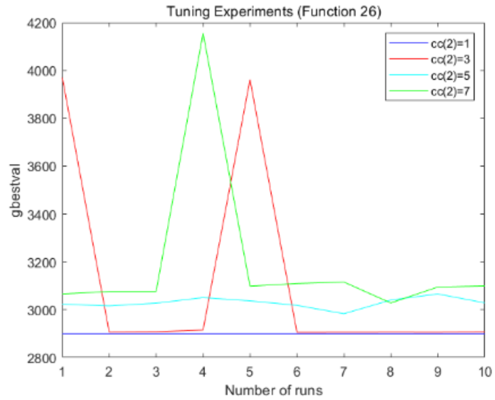
3.1.3. Second Learning Rate (cc2)

For second learning rate, the same method is used to tune this parameter. First adjust the parameters in a large range, get a good small range, and adjust the parameters in the small range. When tuning parameters in the small range, first delete a column of parameters with large standard deviations, and then use SDPS to select the final parameters.

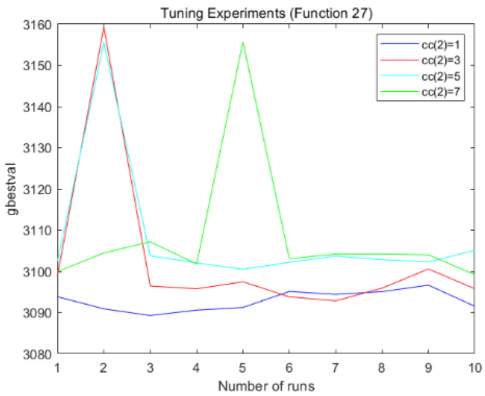
The results of tunning experiments are shown in Fig.11 and Table 5. It can be seen from the results that the range from 1 to 3 is the most preferred range, since these two curves are below the other two.



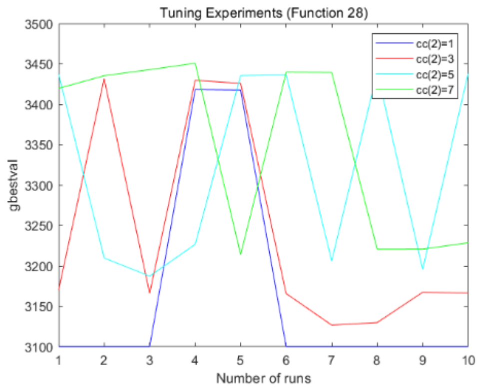
(a)



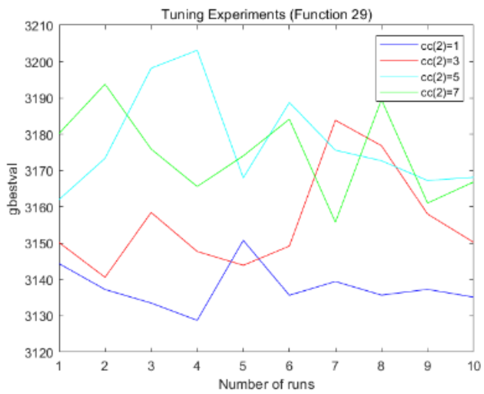
(b)



(c)



(d)



(e)

Figure 11. PSO Second Learning Rate Small Range Tuning Experiments, (a) Test Function 1, (b) Test Function 26, (c) Test Function 27, (d) Test Function 28, (e) Test Function 29.

Table 5. PSO Second Learning Rate Wide Range Tuning Experiments (standard deviation).

Learning rate	Function 1	Function 26	Function 27	Function28	Function29
cc(1)=1	2752.4	0	2.4	134	6.1
cc(1)=3	3598517.9	446.5	20	132.8	14.1
cc(1)=5	143826004.4	22.2	16.7	122.8	14
cc(1)=7	216376084.5	339.4	16.8	112.4	12.4

The second learning rate tuning results is shown in Fig. 10 and Table 6. When cc(2) is equal to 2.8 and 3.0, the mean value and fluctuation of the curve are both very large, so we delete these two data in the future calculation. Then, the SDPS is calculated according to formula (8). Obviously, when cc(2) is equal to 1.6, its SDPS value is smaller than other SDPS values, which means less fluctuation. Therefore, I choose 1.6 as my final parameter in the subsequent experiments.

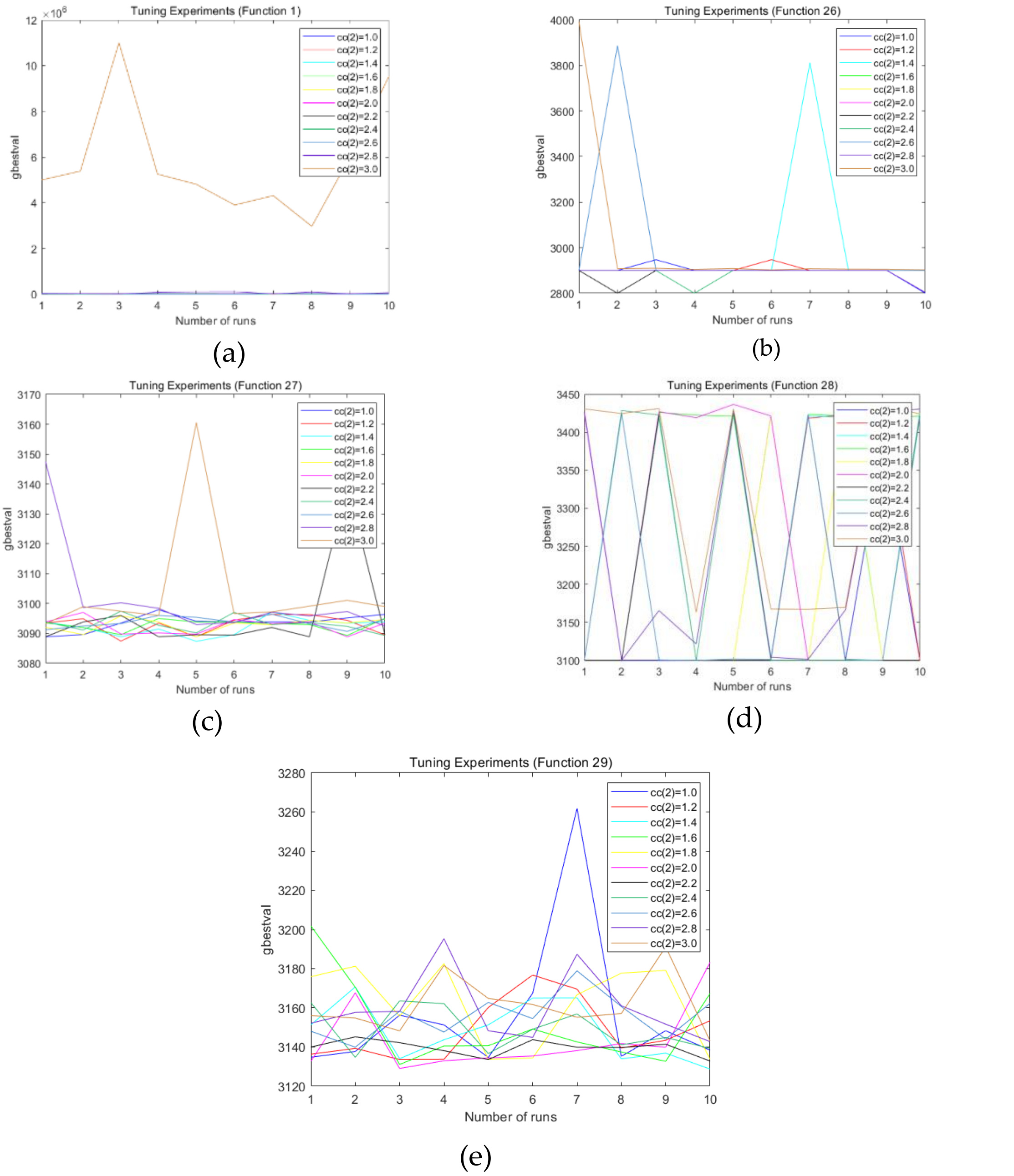


Figure 12. PSO Second Learning Rate Small Range Tuning Experiments, (a) Test Function 1, (b) Test Function 26, (c) Test Function 27, (d) Test Function 28, (e) Test Function 29.

Table 6. PSO Second Learning Rate Small Range Tuning Experiments (standard deviation).

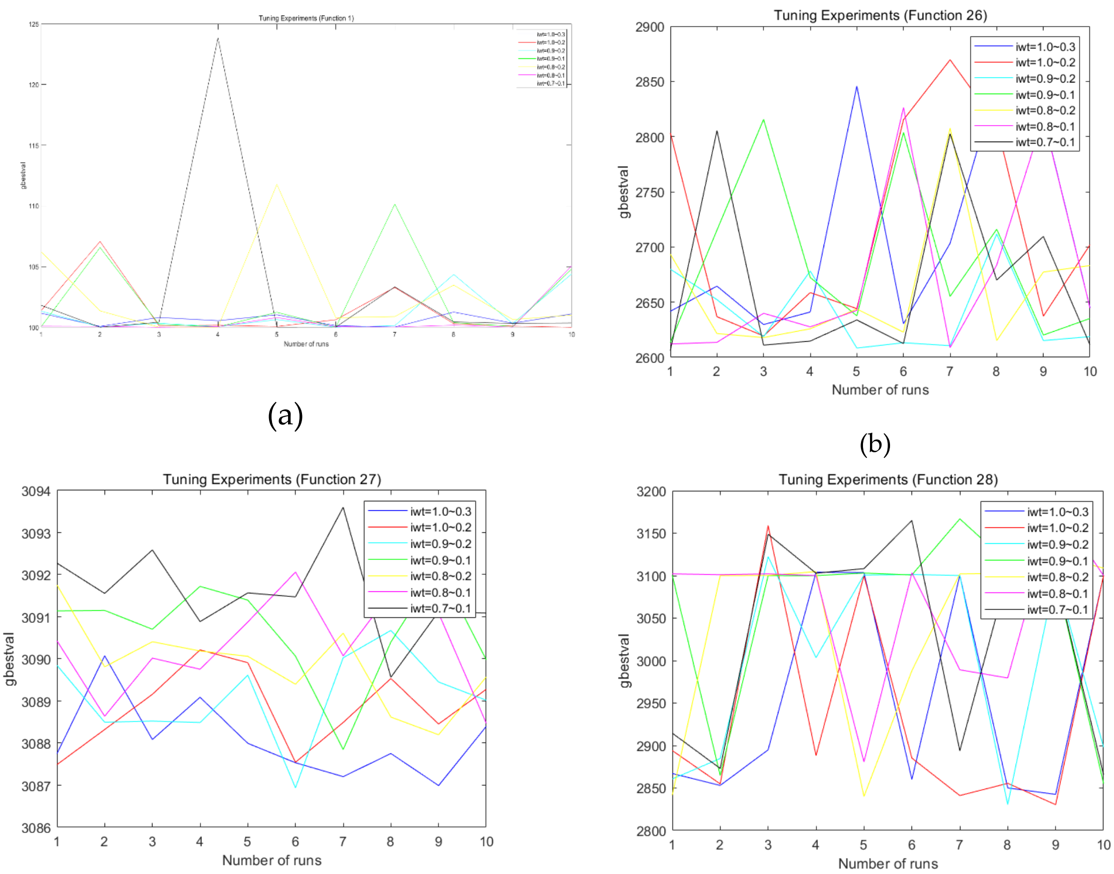
Learning rate	Function 1	Function 26	Function 27	Function28	Function29	SDPS
cc(1)=1.0	2858.98	36.4	2.7	128	38.6	66.67%
cc(1)=1.2	2293.46	14.9	3.2	165.3	15.6	44.28%
cc(1)=1.4	1480.14	288.5	2.9	165.8	15	75.59%
cc(1)=1.6	1082.50	0	2	155.6	22.2	36.29%
cc(1)=1.8	1398.78	31.6	2.1	166.6	21	42.43%
cc(1)=2.0	2834.49	0	2.7	167.8	17.6	45.10%
cc(1)=2.2	1856.03	31.6	15.8	136.3	4	69.93%

cc(1)=2.4	3112.09	31.6	2.6	168.1	11.4	46.53%
cc(1)=2.6	2707.34	312.1	1.9	155.6	11.4	79.20%
sum	19623.81	746.70	35.90	1409.10	156.80	500%
cc(1)=2.8	39771.44	30.8	16.4	157.5	17.6	-
cc(1)=3.0	2518859.93	343.6	20	135.5	14.7	-

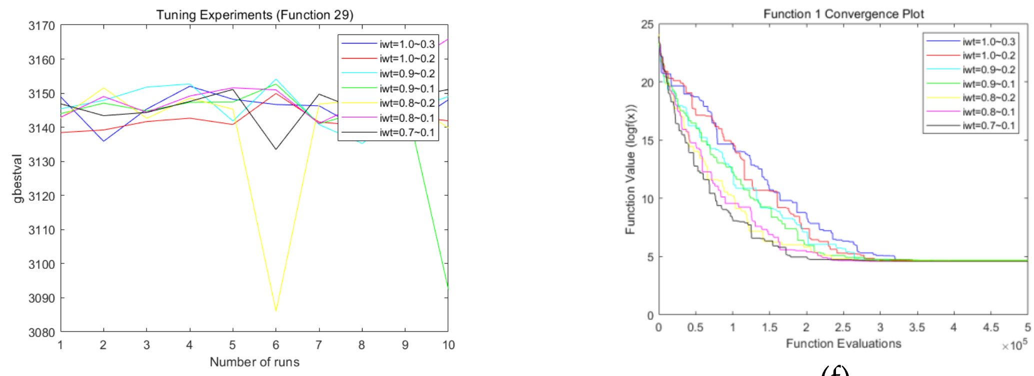
3.2. CLPSO Tuning

3.2.1. Inertia Factor

The method to tune inertia factor in CLPSO is the same as tuning inertia factor in PSO, and the function of the parameter is also same. Large inertia weight contributes to global search, and small inertia factor is more appropriate for local search [3]. In this assignment, inertia factor is set in different range to run test function, and the suitable value can be found by combining five test functions' testing results. The testing results are shown below. In addition, one of the convergence plots is also shown in Fig. 13. Since the function evaluation times are set relatively large, all parameters can be converged before the end of the operation, so the influence of the parameters on convergence is not considered when adjusting the parameters, and more attention is paid to the results after tuning.



(a) (b) (c) (d)



(e) (f)

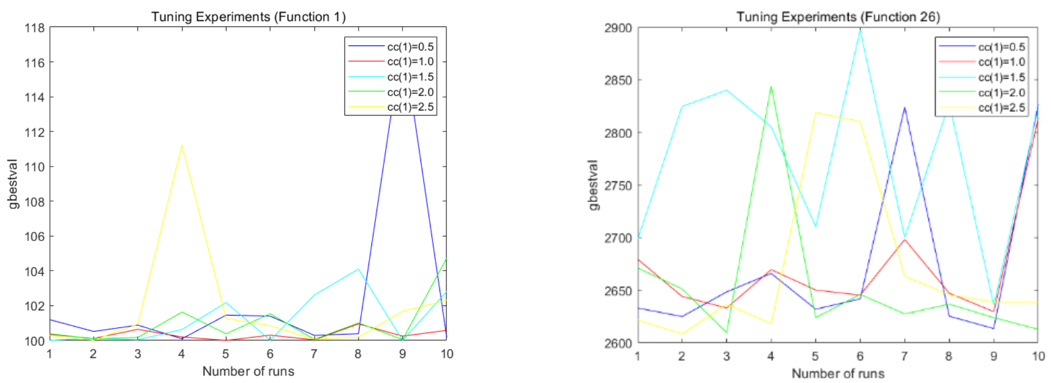
Figure 13. CLPSO Inertia Factor Tuning Experiments, (a) Test Function 1, (b) Test Function 26, (c) Test Function 27, (d) Test Function 28, (e) Test Function 29 (f) Convergence Plots.

Table 7. CLPSO Inertia Factor Tuning Experiments (standard deviation).

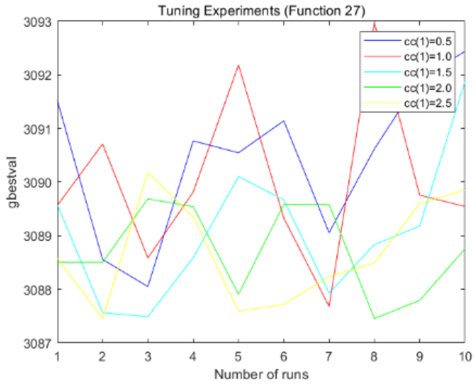
Inertia Factor	Function 1	Function 26	Function 27	Function28	Function29	SDPS
1.0~0.3	0.4831	92.824	0.9173	125.1219	5.1027	56.58%
1.0~0.2	0.261	94.5202	0.9338	126.1979	3.1539	53.01%
0.9~0.2	1.7395	37.1787	1.0559	119.0128	5.9483	54.99%
0.9~0.1	3.5851	73.0203	1.2138	107.8369	17.3254	90.44%
0.8~0.2	3.7368	59.4799	1.0119	111.9936	19.2602	89.47%
0.8~0.1	1.5769	82.1171	1.1668	87.5661	7.3793	62.63%
0.7~0.1	7.3771	78.9387	1.0846	123.5395	5.2017	92.88%
sum	18.7595	518.0789	7.3841	801.2687	63.3715	500.00%

3.2.2. Leaning Rate

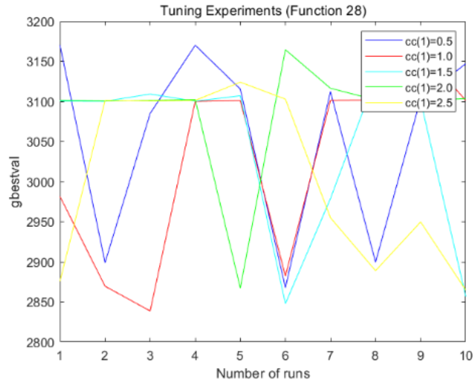
The CLPSO learning rate (acceleration constants) tuning method is the same as the PSO learning rate tuning method. Tune the parameters in a larger range in advance, and then select a better interval to refine the parameter adjustment range. Observe the fluctuation range and average value by plotting the test curves, and select the optimal parameters in conjunction with the calculation of SDPS value. The results are presented below.



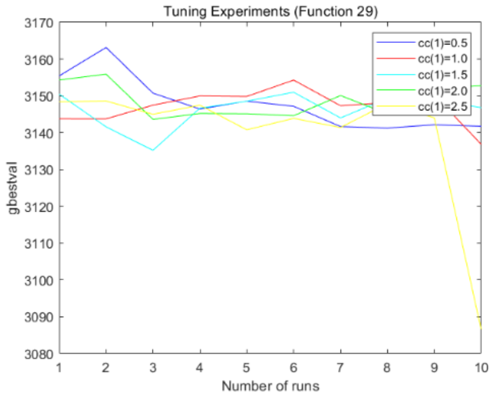
(a) (b)


Tuning Experiments (Function 27)
gbestval
Number of runs
cc(1)=0.5
cc(1)=1.0
cc(1)=1.5
cc(1)=2.0
cc(1)=2.5

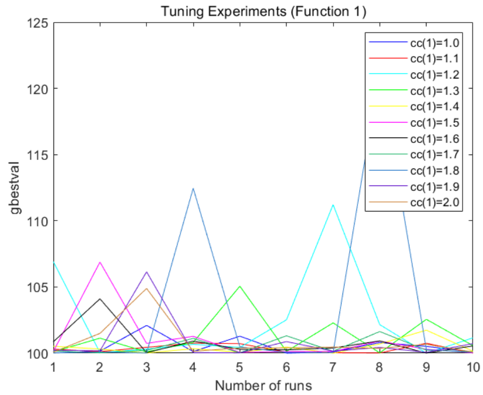
(c)


Tuning Experiments (Function 28)
gbestval
Number of runs
cc(1)=0.5
cc(1)=1.0
cc(1)=1.5
cc(1)=2.0
cc(1)=2.5

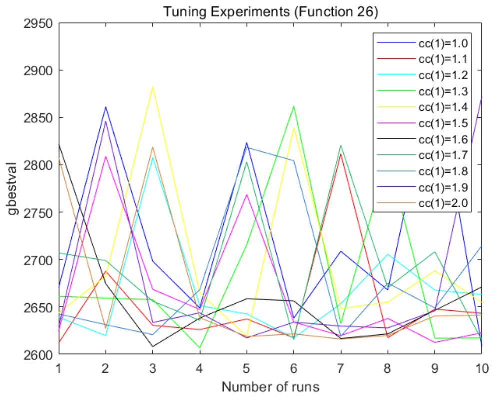
(d)


Tuning Experiments (Function 29)
gbestval
Number of runs
cc(1)=0.5
cc(1)=1.0
cc(1)=1.5
cc(1)=2.0
cc(1)=2.5

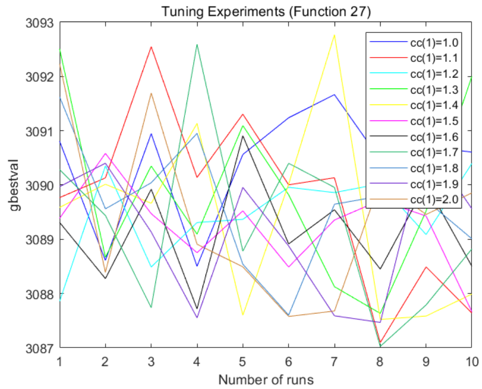
(e)


Tuning Experiments (Function 1)
gbestval
Number of runs
cc(1)=1.0
cc(1)=1.1
cc(1)=1.2
cc(1)=1.3
cc(1)=1.4
cc(1)=1.5
cc(1)=1.6
cc(1)=1.7
cc(1)=1.8
cc(1)=1.9
cc(1)=2.0

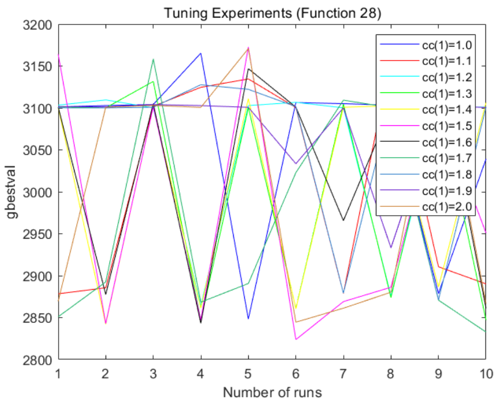
(f)


Tuning Experiments (Function 26)
gbestval
Number of runs
cc(1)=1.0
cc(1)=1.1
cc(1)=1.2
cc(1)=1.3
cc(1)=1.4
cc(1)=1.5
cc(1)=1.6
cc(1)=1.7
cc(1)=1.8
cc(1)=1.9
cc(1)=2.0

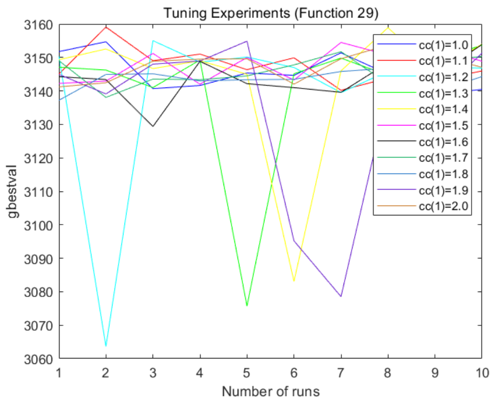
(g)


Tuning Experiments (Function 27)
gbestval
Number of runs
cc(1)=1.0
cc(1)=1.1
cc(1)=1.2
cc(1)=1.3
cc(1)=1.4
cc(1)=1.5
cc(1)=1.6
cc(1)=1.7
cc(1)=1.8
cc(1)=1.9
cc(1)=2.0

(h)


Tuning Experiments (Function 28)
gbestval
Number of runs
cc(1)=1.0
cc(1)=1.1
cc(1)=1.2
cc(1)=1.3
cc(1)=1.4
cc(1)=1.5
cc(1)=1.6
cc(1)=1.7
cc(1)=1.8
cc(1)=1.9
cc(1)=2.0

(i)


Tuning Experiments (Function 29)
gbestval
Number of runs
cc(1)=1.0
cc(1)=1.1
cc(1)=1.2
cc(1)=1.3
cc(1)=1.4
cc(1)=1.5
cc(1)=1.6
cc(1)=1.7
cc(1)=1.8
cc(1)=1.9
cc(1)=2.0

(j)

Figure 14. CLPSO Learning Rate Tuning Experiments, (a) Test Function 1, (b) Test Function 26, (c) Test Function 27, (d) Test Function 28, (e) Test Function 29, (f) Test Function 1, (g) Test Function 26, (h) Test Function 27, (i) Test Function 28, (j) Test Function 29.

Table 8. CLPSO Learning Rate Tuning Experiments (standard deviation).

Learning rate	Function 1	Function 26	Function 27	Function28	Function29	SDPS
cc(1)=0.5	5.3044	81.3589	1.4592	119.58	7.1052	130.22%
cc(1)=1.0	0.2989	54.4394	1.5791	120.0478	4.7789	77.16%
cc(1)=1.5	1.5273	83.4654	1.3295	108.5014	4.8432	89.36%
cc(1)=2.0	1.4382	69.11	0.8382	79.307	4.6284	70.78%
cc(1)=2.5	3.3668	77.7557	0.9865	108.9722	18.7798	132.48%
sum	11.9356	366.1294	6.1925	536.4084	40.1355	500.00%

Learning rate	Function 1	Function 26	Function 27	Function28	Function29	SDPS
cc(1)=1.0	0.7	101.61	1.04	105.69	5.42	35.19%
cc(1)=1.1	0.34	59.74	1.62	125.54	5.27	34.07%
cc(1)=1.2	3.68	55.55	0.82	98.42	26.91	58.01%
cc(1)=1.3	1.61	86.61	1.61	130.15	22.9	57.03%
cc(1)=1.4	0.51	88.67	1.74	125.01	21.51	51.57%
cc(1)=1.5	2.09	68.03	0.78	142.6	4.79	38.28%
cc(1)=1.6	1.11	60.8	0.96	124.8	6.54	34.05%
cc(1)=1.7	0.61	71.57	1.63	124.32	4.28	35.96%
cc(1)=1.8	7.39	72.82	1.14	98.49	2.91	61.29%
cc(1)=1.9	1.87	95.98	1.23	55.24	25.67	52.90%
cc(1)=2.0	1.46	78.71	1.57	141.42	4.2	41.65%
sum	21.37	840.09	14.14	1271.68	130.4	500.00%

4. Experimental Results

4.1. 30 Runs Results

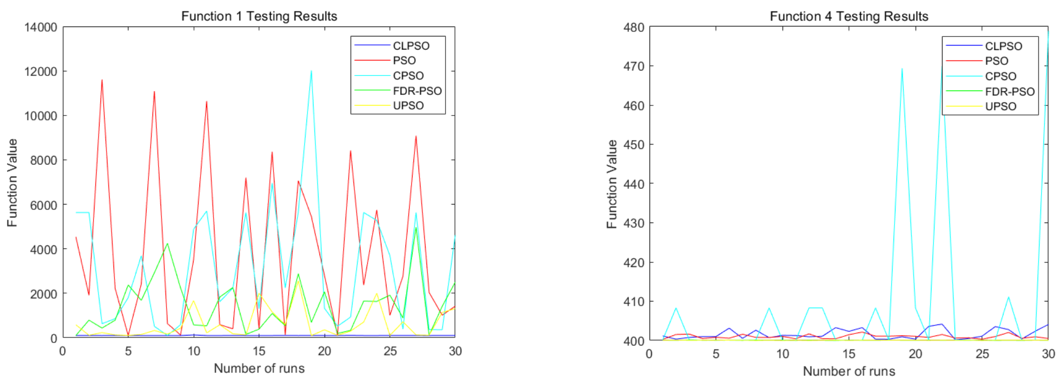
Table 9 presents the mean, median, max, min, variance and standard deviation of 30 runs of the five algorithms on ten test function selected in section 2. The experiment results of each run are shown in Fig. 15. The dimension of each problem is 10, and the maximum fitness evaluations is set at 500,000. The convergence picture is plotted in Fig. 16 to present convergence characteristics in the median run including five algorithms for each test function. Each algorithm's min value is selected six times to form a 30*1 benchmark matrix. T-test method is used to test results between benchmark matrix and each algorithm. h value presented in Table 9 is the result of T-test. If h equals to 0, it means that there is no statistically difference between them. On the contrary (h=1), there exist differences between benchmark matrix and the algorithm.

Table 9. Result For 10-D Problems

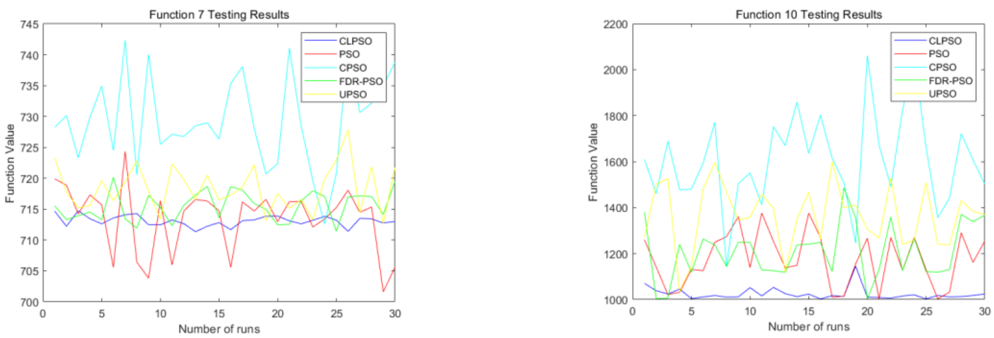
Function No.	PSO	CLPSO	FDR-PSO	UPSO	CPSO
mean – 1	3840.551	101.847	1475.195	614.878	3214.894
median – 1	2396.214	100.113	1255.371	315.564	2239.938
max – 1	11607.475	130.075	4973.531	2561.552	12019.000
min – 1	105.141	100.001	101.737	101.493	102.361
variance -- 1	13592036.557	30.706	1486527.387	458043.776	7881542.312
std. – 1	3686.738	5.541	1219.232	676.789	2807.408
Ttest – 1	5.45E-06	0.779	1.01E-06	0.00027	1.30E-06

h – 1	1	0	1	1	1
mean – 4	400.985	401.642	400.053	400.055	409.370
median – 4	400.874	401.080	400.048	400.063	400.123
max – 4	402.146	404.175	400.135	400.101	478.935
min – 4	400.267	400.171	400.022	400.000	400.007
variance – 4	0.263	1.667	0.001	0.001	477.243
std. – 4	0.513	1.291	0.024	0.026	21.846
Ttest – 4	1.42E-10	3.35E-07	0.054	0.068	0.027
h – 4	1	1	0	0	1
mean – 7	713.509	713.097	715.547	718.404	729.408
median – 7	714.996	713.169	715.585	717.769	728.528
max – 7	724.339	714.748	720.152	727.869	742.338
min – 7	701.624	711.342	711.440	712.672	712.591
variance – 7	28.591	0.730	5.767	13.517	54.886
std. – 7	5.347	0.854	2.401	3.677	7.409
Ttest – 7	0.0059	0.00038	1.11E-07	2.82E-11	2.03E-16
h – 7	1	1	1	1	1
mean – 10	1176.314	1025.013	1209.662	1365.473	1605.768
median – 10	1151.941	1016.667	1237.415	1377.165	1608.872
max – 10	1376.738	1145.380	1485.910	1600.224	2061.009
min – 10	1003.477	1002.941	1003.540	1035.114	1145.590
variance – 10	13416.268	773.582	14980.614	19376.863	41487.153
std. – 10	115.829	27.813	122.395	139.201	203.684
Ttest – 10	5.94E-07	0.26	1.84E-08	1.82E-14	3.76E-16
h – 10	1	0	1	1	1
mean – 15	1502.655	1505.020	1506.953	1530.359	2923.888
median – 15	1502.112	1504.344	1505.397	1525.075	1831.211
max – 15	1506.289	1512.157	1518.560	1588.800	32433.069
min – 15	1500.372	1500.794	1501.093	1502.998	1545.877
variance – 15	2.995	8.631	20.560	524.665	31175364.669
std. – 15	1.731	2.938	4.534	22.906	5583.490
Ttest – 15	0.030	0.131	0.345	0.00040	0.176
h – 15	1	0	0	1	0
mean – 18	3114.795	1923.724	3696.267	3121.614	10239.559
median – 18	2221.216	1915.444	2570.266	3030.167	6570.199
max – 18	8195.978	2172.400	9286.719	5164.576	35262.289
min – 18	1824.233	1807.106	1824.858	1936.625	2825.480
variance – 18	2550822.263	8689.258	5439721.097	860071.811	64688400.085
std. – 18	1597.129	93.216	2332.321	927.401	8042.910
Ttest – 18	0.0012	0.120	0.0006	8.18E-07	5.06E-06
h – 18	1	0	1	1	1
mean – 26	2959.473	2708.862	2863.333	2845.748	2988.711
median – 26	2900.000	2667.675	2900.000	2900.000	2985.894
max – 26	3824.271	2900.000	2900.000	2947.160	4039.840
min – 26	2900.000	2609.832	2600.000	2600.000	2600.000
variance – 26	51296.775	7205.398	8609.195	7332.924	50143.201

std. – 26	226.488	84.885	92.786	85.632	223.927
Ttest – 26	1.02E-07	0.088	1.70E-09	1.07E-08	9.61E-09
h – 26	1	0	1	1	1
mean – 27	3094.716	3088.588	3097.676	3079.464	3100.609
median – 27	3093.726	3088.655	3093.819	3074.973	3097.632
max – 27	3141.118	3090.639	3172.361	3200.002	3157.220
min – 27	3088.861	3087.046	3089.006	3071.023	3089.297
variance – 27	83.681	0.950	315.424	526.400	211.144
std. – 27	9.148	0.975	17.760	22.943	14.531
Ttest – 27	2.96E-05	0.012	0.00087	0.212	4.47E-06
h – 27	1	1	1	0	1
mean – 28	3207.342	3003.188	3206.847	3272.500	3271.526
median – 28	3100.000	3100.159	3100.000	3272.500	3218.549
max – 28	3426.795	3145.580	3411.822	3272.500	3412.053
min – 28	3100.000	2844.790	3100.000	3272.500	3100.000
variance – 28	23841.725	14221.942	20458.252	0.000	14541.869
std. – 28	154.408	119.256	143.032	0.000	120.590
Ttest – 28	0.0018	0.0197	0.0013	3.34E-08	6.589E-07
h – 28	1	1	1	1	1
mean – 29	3147.971	3144.540	3161.753	3153.318	3215.669
median – 29	3142.443	3145.571	3159.584	3154.257	3203.007
max – 29	3183.497	3149.787	3190.715	3178.836	3321.909
min – 29	3129.409	3132.853	3137.727	3132.870	3157.644
variance – 29	259.800	15.704	234.185	98.310	2193.771
std. – 29	16.118	3.963	15.303	9.915	46.838
Ttest – 29	0.0068	0.0028	5.10E-09	2.60E-07	4.25E-10
h – 29	1	1	1	1	1



(a) (b)



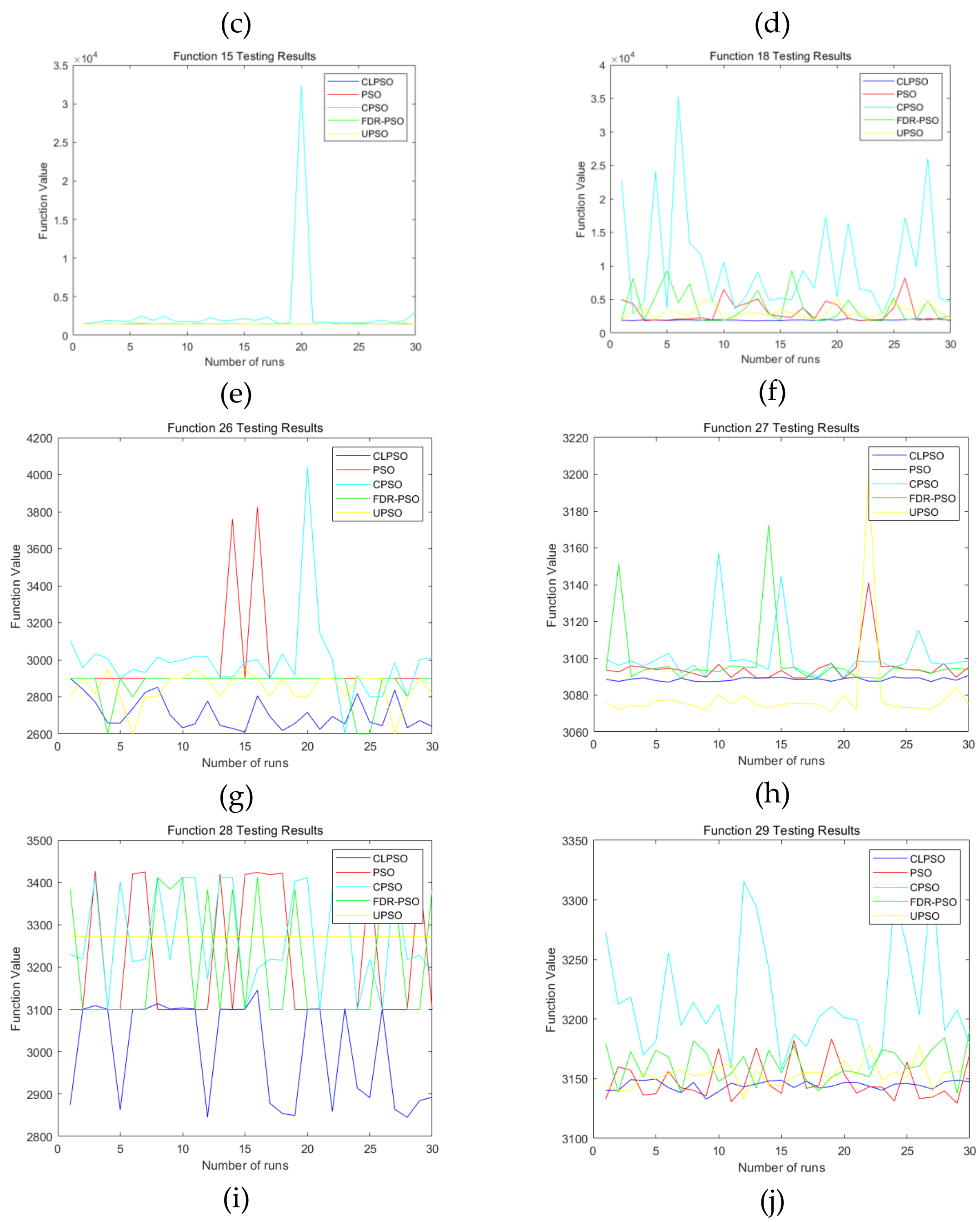
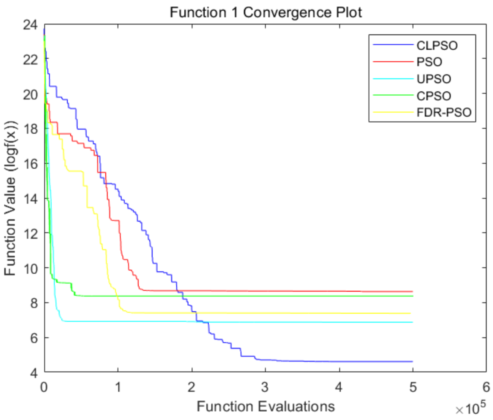
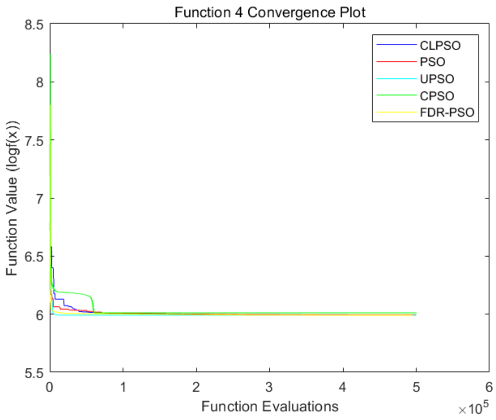


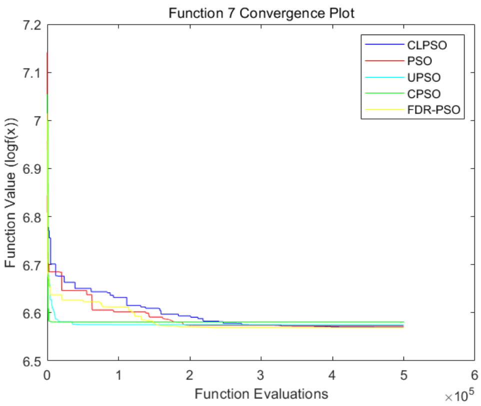
Figure 15. Experiment Results in Each Run, (a) Function 1, (b)Function 4, (c)Function 7, (d) Function15, (e) Function 18, (f) Function26, (g) Function27, (i) Function 28, (j) Function 29.


Function 1 Convergence Plot
CLPSO
PSO
UPSO
CPSO
FDR-PSO
Function Value (logf(x))
Function Evaluations
×10^5

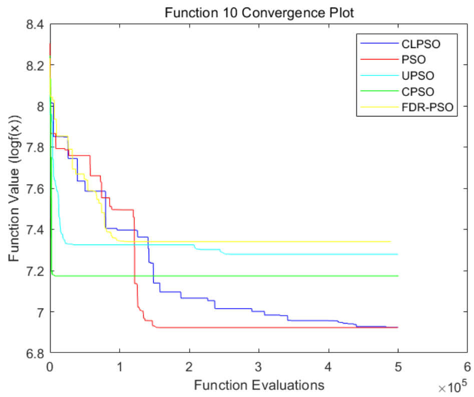
(a)


Function 4 Convergence Plot
CLPSO
PSO
UPSO
CPSO
FDR-PSO
Function Value (logf(x))
Function Evaluations
×10^5

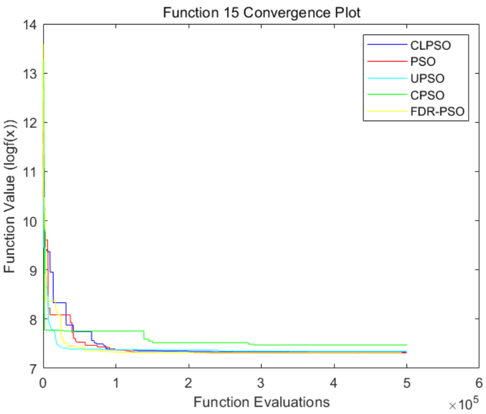
(b)


Function 7 Convergence Plot
CLPSO
PSO
UPSO
CPSO
FDR-PSO
Function Value (logf(x))
Function Evaluations
×10^5

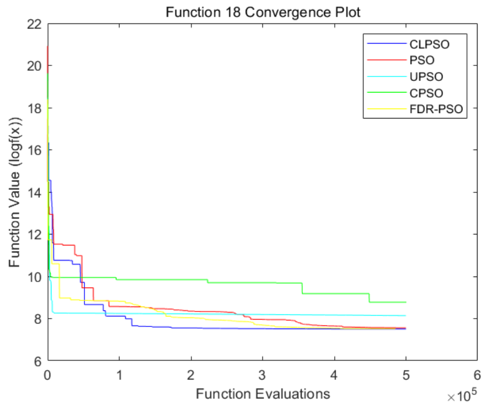
(c)


Function 10 Convergence Plot
CLPSO
PSO
UPSO
CPSO
FDR-PSO
Function Value (logf(x))
Function Evaluations
×10^5

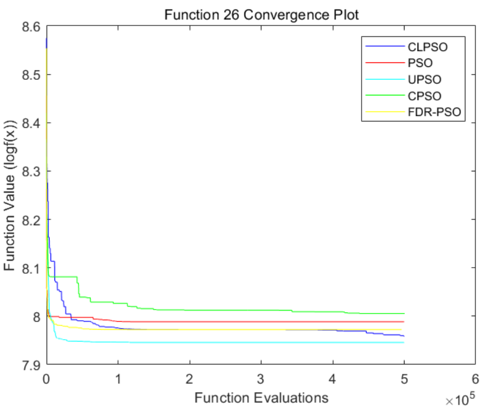
(d)


Function 15 Convergence Plot
CLPSO
PSO
UPSO
CPSO
FDR-PSO
Function Value (logf(x))
Function Evaluations
×10^5

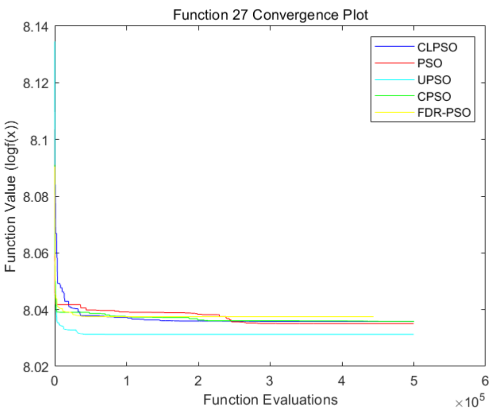
(e)


Function 18 Convergence Plot
CLPSO
PSO
UPSO
CPSO
FDR-PSO
Function Value (logf(x))
Function Evaluations
×10^5

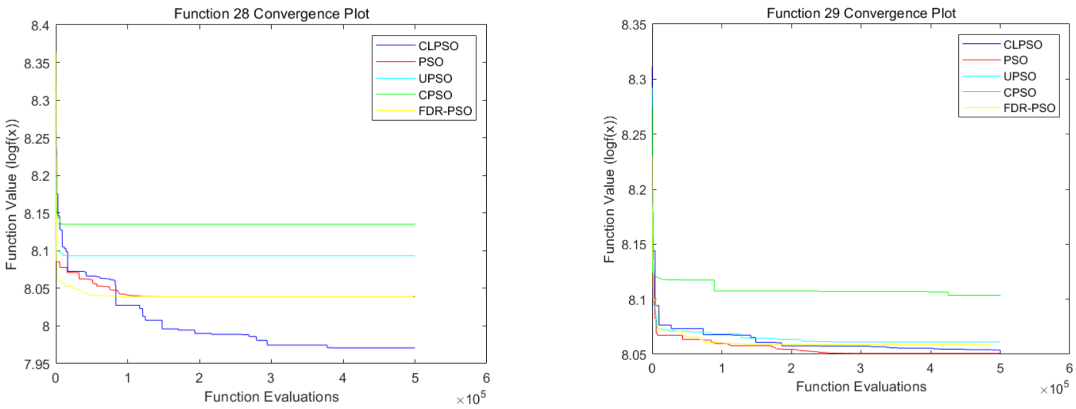
(f)


Function 26 Convergence Plot
CLPSO
PSO
UPSO
CPSO
FDR-PSO
Function Value (logf(x))
Function Evaluations
×10^5

(g)


Function 27 Convergence Plot
CLPSO
PSO
UPSO
CPSO
FDR-PSO
Function Value (logf(x))
Function Evaluations
×10^5

(h)



(i) (j)

Figure 16. Convergence Plots, (a) Function 1, (b)Function 4, (c)Function 7, (d) Function15, (e) Function 18, (f) Function26, (g) Function27, (i) Function 28, (j) Function 29.

4.2. Discussions

It can be seen from the table that after t-test, the number of times that h equals to 0 in PSO, CLPSO, FDR-PSO, UPSO and CPSO algorithm are 0, 5, 2, 2, 1 respectively. Therefore, compared with other four algorithms, CLPSO can get more good results in the selected test functions. In test function 1, 10, 18, 26, CLPSO perform better than other four algorithms. From the convergence plots above, the convergence speed of CLPSO can be controlled by tuning inertia factor , but CLPSO still converges more slowly than the other four algorithms on average. This is because CLPSO has more potential search space than other algorithms, and at the same time, its ability to avoid local optimal solutions is better. CPSO and UPSO can converge faster than other algorithms, and UPSO performs well in some multi-modal functions such as function 4 and 27. From the picture of each run, CPSO has the largest fluctuations especially in the rotated problems.

5. Conclusion

In this assignment, PSO and its four variants algorithm are evaluated. In section 1, PSO and CLPSO algorithms are described in detail in conjunction with Matlab code. In section 2, ten test functions are selected from CEC2017 benchmark. The characteristics of each selected test function are not the same, so that the algorithm can be evaluated more comprehensively. In section 3, the parameter experiment is presented in this part. In section 4, each algorithm is run 30 times on the 10 test functions, so I gain 1500 experimental data. The results are presented by figure, so that the fluctuation can be easily observed. The results are statistically tested by t-test, and the running process is observed by plotting a convergence graph. Then, the characteristics of each algorithm can be discovered.

References

1. Eberhart, Russell, and James Kennedy. "A new optimizer using particle swarm theory." MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science. Ieee, 1995.
2. Shi, Yuhui, and Russell Eberhart. "A modified particle swarm optimizer." 1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360). IEEE, 1998.
3. Liang, Jing J., et al. "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions." IEEE transactions on evolutionary computation 10.3 (2006): 281-295.
4. Peram, Thanmaya, Kalyan Veeramachaneni, and Chilukuri K. Mohan. "Fitness-distance-ratio based particle swarm optimization." Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No. 03EX706). IEEE, 2003.
5. Van den Bergh, Frans, and Andries Petrus Engelbrecht. "A cooperative approach to particle swarm optimization." IEEE transactions on evolutionary computation 8.3 (2004): 225-239.

6. Parsopoulos, Konstantinos E. "UPSO: A unified particle swarm optimization scheme." Lecture series on computer and computational science 1 (2004): 868-873.
7. Chiong, Raymond, ed. Nature-inspired algorithms for optimisation. Vol. 193. Springer, 2009.