EE6227: Programming Assignment 1

Algorithm Description and Implement

i. Particle Swarm Optimization (PSO)

PSO algorithm designs particles represent a candidate solution in a D-dimensional search space to an optimization problem. Each particle stochastically flies through the multidimensional search space to find an optimal solution according to its own experience. A particle is composed of three vectors: the X-vector records the current position; the V-vector records the gradient direction; the P-vector records the position of the best solution found by this particle. It also contains two specific values: X-values records the fitness of the X-vector; P-fitness records the fitness value of P-vector; which is the best solution found by this particle so far.

In each generation, each particle's V-vector will be added to X-vector to update the position. The fitness value and other vectors will also be updated to find the optimal solution. After searching the whole space, particles may find an optimal solution to this problem. The particles are updated as bellow:

$$\mathrm{V}_i \leftarrow w * V_i^d + c1 * rand1 * \left(pbest_i^d - X_i^d\right) + c2 * rand2 * \left(gbest^d - X_i^d\right)$$

$$\mathrm{X}_I \leftarrow X_i^d + V_i^d$$

Where V_i denotes the velocity of the i-th particle, X_i denotes the location of the i-th particle, $pbest_i$ denotes the best fitness value found by the i-th particle, $gbest$ denotes the best solution found by the population.

The algorithm is implemented as below:

1. Initialization

   Randomly initialize the value of the X-vector and V-vector. Calculate the initial fitness value. The pbest records the optimal value of the individual particle and gbest records the optimal value of the group.

---------- **Initialization** ----------

```
mv=0.5*(VRmax-VRmin); %constraint
VRmin=repmat(VRmin,ps,1);
VRmax=repmat(VRmax,ps,1);
Vmin=repmat(-mv,ps,1);
Vmax=-Vmin;
pos=VRmin+(VRmax-VRmin).*rand(ps,D);% initialize the position of each
particles
vel=Vmin+2.*Vmax.*rand(ps,D);%initialize the velocity of each particles
for i=1:ps;
e(i,1)=feval(fhd,pos(i,:),varargin{:});%calculate origin position fitness value
end
fitcount=ps;
pbest=pos; %initialize the pbest position
```

```
pbestval=e; %initialize pbest's fitness value
[gbestval,gbestid]=min(pbestval); %%initialize the gbest's fitness value
gbest=pbest(gbestid,:);%initialize the gbest position
gbestrep=repmat(gbest,ps,1);%each particle store a copy of gbest
```

---------- **Initialization** ----------

2. Update

   Particles fly through the search space and update the pbest and their position recursively. Modify the V-vector based on pbest and gbest as well.

---------- **Update** ----------

```
for i=2:me % maximum update generation
    for k=1:ps % for each particles
    aa(k,:)=cc(1).*rand(1,D).*(pbest(k,:)-
pos(k,:))+cc(2).*rand(1,D).*(gbestrep(k,:)-pos(k,:));
    vel(k,:)=iwt(i).*vel(k,:)+aa(k,:);%update the velocity (PSO with the Inertia
Factor)
    vel(k,:)=(vel(k,:)>mv).*mv+(vel(k,:)<=mv).*vel(k,:);
    vel(k,:)=(vel(k,:)<(-mv)).*(-mv)+(vel(k,:)>=(-mv)).*vel(k,:); %in case V-
vector overflow
    pos(k,:)=pos(k,:)+vel(k,:); %update the position
    pos(k,:)=((pos(k,:)>=VRmin(1,:))&(pos(k,:)<=VRmax(1,:))).*pos(k,:)...
        +(pos(k,:)<VRmin(1,:)).*(VRmin(1,:)+0.25.*(VRmax(1,:)-
VRmin(1,:)).*rand(1,D))+(pos(k,:)>VRmax(1,:)).*(VRmax(1,:)-
0.25.*(VRmax(1,:)-VRmin(1,:)).*rand(1,D)); % in case X-vector overflow
    e(k,1)=feval(fhd,pos(k,:),varargin{:});%update the fitness value
    fitcount=fitcount+1;
    tmp=(pbestval(k)<e(k));
    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 and pbest's
value
    if pbestval(k)<gbestval
    gbest=pbest(k,:);
    gbestval=pbestval(k);
    gbestrep=repmat(gbest,ps,1);%update the gbest
    end
    end
    end
```

---------- **Update** ----------

3. Termination

   When the termination condition is reached (generation excesses the maximum generation or fitcount excesses the maximum fitcount), the search will be finished. The gbest found by particles will be the final solution.

ii. Comprehensive learning PSO(CLPSO)

In the original PSO, particles simultaneously learn from their pbest and gbest, which makes the algorithm converge fast. However, the particles will be dragged to the gbest and be trapped in it if the gbest is a local optimal. CLPSO tries to fix this problem. The Algorithm is similar to PSO, but it changes the learning strategy. In every generation, a particle not only uses its own pbest and gbest, but also uses all particles' pbest. The update strategy is shown as below:

$$V_i^d \leftarrow w * V_i^d + c * \text{rand}_i^d\left(pbest_{f_i(d)}^d - X_i^d\right)$$

Where $pbest_{f_i(d)}^d$ denotes the d-th dimention of $f_i(d)$-th particle's pbest.

Each dimension of a particle will learn from different virtual pbest within certain generation., The algorithm is implement as below:

1. Generate Pc

   Generate a probability Pc. When each particle starts learning, it will generate a random number for each dimension, if the number is larger than Pc, this dimension of particle will learn from its own pbest. Otherwise, it will learn from a selected pbest from other particles.

```
---------------------------------------------- Generate Pc ----------------------------------------------
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)));%generate Pc
---------------------------------------------- Generate Pc ----------------------------------------------
```

2. Generate f_pbest

   Randomly choose 2 particles and compare their pbest value. Use the better one to construct the f_pbest. During certain generations, the particles will learn from this f_best to update its vector.

```
---------------------------------------------- Generate f_pbest ----------------------------------------------
ai=zeros(ps,D);
f_pbest=1:ps;
f_pbest=repmat(f_pbest',1,D);%stores info about each dimension choose from which particle
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%if all learn from itself, randomly choose 1 dimension to learn from other particle
f_pbest(k,:)=bi.*fi+(1-bi).*f_pbest(k,:); %generate f_pbest
end
stop_num=0;%when this num excess a certain number, reconstruct a f_pbest
i=1;
---------------------------------------------- Generate f_pbest ----------------------------------------------
```

3. Update

Update procedure is similar to PSO. But CLPSO change the learning strategy. While choosing a pbest dimension from other particles, tournament selection is used.

----------Update ----------

```
while i<=me&fitcount<=Max_FES
    i=i+1;
    for k=1:ps%update each particle
    if stay_num(k)>=5%if excess 5, reconstruct a f_pbest
    stay_num(k)=0;
    ai(k,:)=zeros(1,D);
    f_pbest(k,:)=k.*ones(1,D);
    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
    for dimcnt=1:D
        pbest_f(k,dimcnt)=pbest(f_pbest(k,dimcnt),dimcnt);
    end
    aa(k,:)=cc(1).*(1-ai(k,:)).*rand(1,D).*(pbest_f(k,:)-
pos(k,:))+cc(2).*ai(k,:).*rand(1,D).*(gbestrep(k,:)-pos(k,:));
    vel(k,:)=iwt(i).*vel(k,:)+aa(k,:); %update v-vector
    vel(k,:)=(vel(k,:)>mv).*mv+(vel(k,:)<=mv).*vel(k,:);
    vel(k,:)=(vel(k,:)<(-mv)).*(-mv)+(vel(k,:)>=(-mv)).*vel(k,:);
    pos(k,:)=pos(k,:)+vel(k,:); %update x-vector

    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
            end
        end
```

----------Update ----------

4. Termination
   Termination condition is similar to PSO.

Test Functions

In order to test the performance of these two algorithm, I choose 10 typical benchmark functions from **CLPSO numerical optimization problems (bound constrained problems).**

1) Sphere function

$$f_1(x) = \sum_{i=1}^{D} x_i^2$$

2) Rosenbrock's function

$$f_2(x) = \sum_{i=1}^{D-1} (100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2)$$

3) Ackley's function

$$f_3(x) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(-\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right) + 20 + e$$

4) Griewanks's function

$$f_4(x) = \sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} \cos\left(\frac{x_i^2}{\sqrt{i}}\right) + 1$$

5) Rastrigin's function

$$f_5(x) = \sum_{i=1}^{D} (x_i^2 - 10\cos(2\pi x_i) + 10)$$

6) Noncontinuous Rastrigin's function

$$f_6(x) = \sum_{i=1}^{D} (y_i^2 - 10\cos(2\pi y_i) + 10)$$

$$y_i = \begin{cases} x_i, & |x_i| < \frac{1}{2} \\ \frac{round(2x_i)}{2}, & |x_i| \geq \frac{1}{2} \end{cases} \quad i = 1,2,3 \cdots D$$

7) Schwefel's function

$$f_7(x) = 418.9829 * D - \sum_{i=1}^{D} x_i \sin(|x_i|^{\frac{1}{2}})$$

8) Rotated Ackley's function

$$f_8(x) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} y_i^2}\right) - \exp\left(-\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi y_i)\right) + 20 + e$$

$$y = M * x$$

9) Rotated Griewanks's function

$$f_9(x) = \sum_{i=1}^{D}\frac{y_i^2}{4000} - \prod_{i=1}^{D}\cos\left(\frac{y_i^2}{\sqrt{i}}\right) + 1 \quad y = M * x$$

10) Rotated Weierstrass function

$$f_{10}(x) = \sum_{i=1}^{D}\left(\sum_{k=0}^{kmax}\left(a^k\cos\left(2\pi b^k(y_i + 0.5)\right)\right)\right) - D\sum_{k=0}^{kmax}\left(a^k\cos(2\pi b^k * 0.5)\right)$$

a=0.5,b=3,kmax=20,y=M*x

The global optimal, search ranges and initialization ranges of these functions are shown in table 1.

Table 1

Function no	x	f(x)	Search range	Initialization range
f1	[0 0 ⋯ 0]	0	[-100 100]	[-100 50]
f2	[1 1 ⋯ 1]	0	[-2.048 2.048]	[-2.048 2.048]
f3	[0 0 ⋯ 0]	0	[-32.768 32.768]	[-32.768 16]
f4	[0 0 ⋯ 0]	0	[-600 600]	[-600 200]
f5	[0 0 ⋯ 0]	0	[-5.12 5.12]	[-5.12 2]
f6	[0 0 ⋯ 0]	0	[-5.12 5.12]	[-5.12 2]
f7	[420.96 420.96 ⋯ 420.96]	0	[-500 500]	[-500 500]
f8	[0 0 ⋯ 0]	0	[-32.768 32.768]	[-32.768 16]
f9	[0 0 ⋯ 0]	0	[-600 600]	[-600 200]
f10	[0 0 ⋯ 0]	0	[-0.5 0.5]	[-0.5 0.2]

Parameters Description and Tuning

In PSO algorithm, there are some parameters related to the performance.

- Learning factor--c1, c2, w
- Number of particles

W is the inertial weight, which is the employed to control the impact of the previous V-vector on the present V-vector. A larger w will facilitate global search while a small one tends to facilitate local area.

I first apply a static inertial weight to the update strategy, and count the generation numbers it need for convergence. The result is shown in Fig 1:

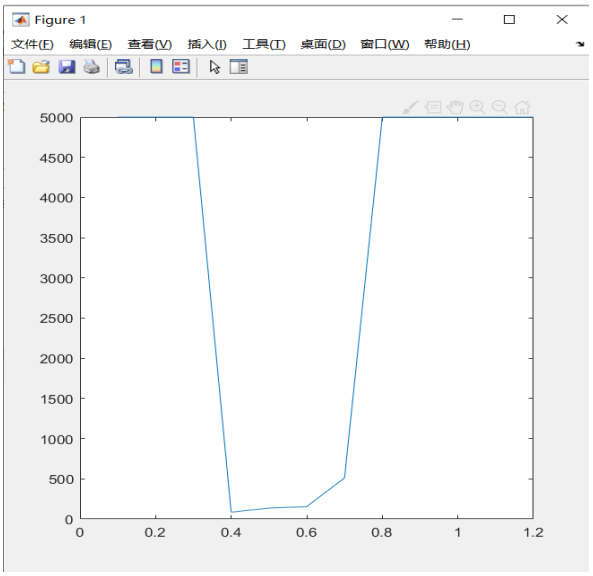


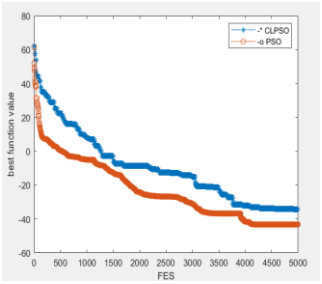
Fig. 1

We can find that w between 0.4 and 0.8 can get a good result. The algorithm can converge quickly to an optimal result. However, as the generation goes up, the w should become smaller to search in a local area. So we separately use 2 static w and a linear changed w to find the solution to 5 functions, each solution run 5 times. The result is shown below:

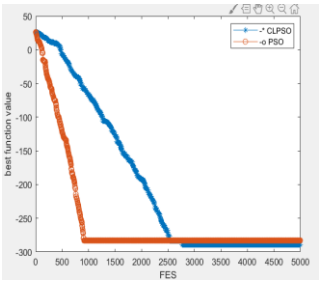
The max generation is 5000, particle number is 15, variance dimension is 10

A. linear inertial weight

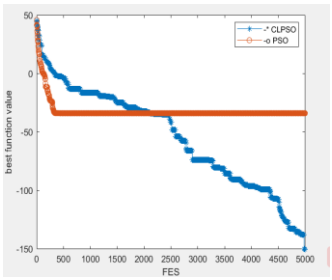
we set $w = 0.8 - \text{generation} * (0.4/\max\_generation)$; the convergence plot of 5 functions is illustrated as fig.2. The result is illustrated as table 2.



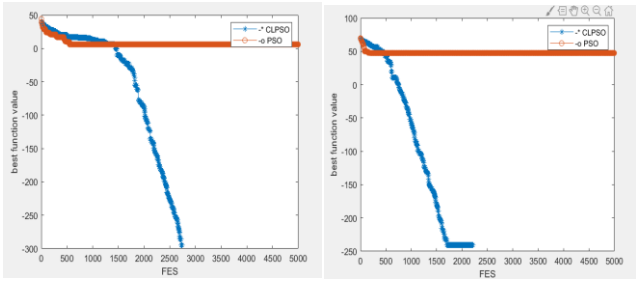
f1



f2



f3



f4 f5

Fig2 w=linear

Red line represents PSO, Blue line represents CLPSO.

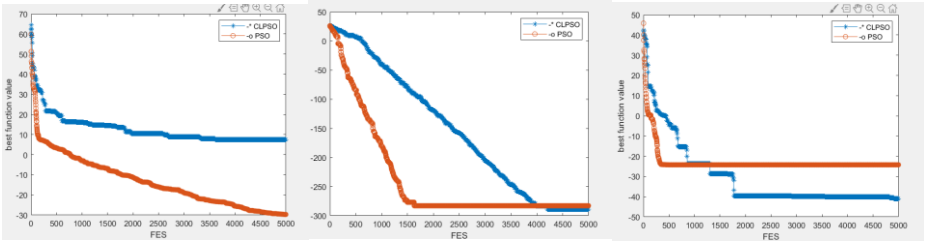
X reprensents generations, y represents fitness value in log scale.

Table 2

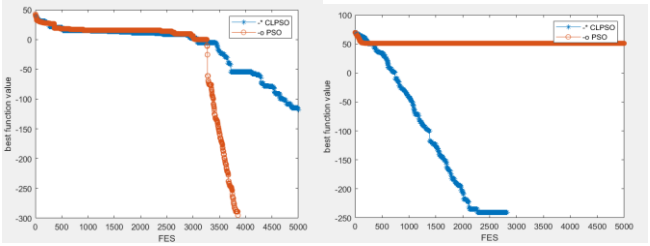
Function no	Mean value PSO	Mean value CLPSO
f1	0.935988996118964	1.94892856354181
f2	0.231029700541974	3.55271367880050e-15
f3	0.0487175837274205	0.00247078565713874
f4	6.40000000000000	0
f5	375.055270798541	23.6876669228875

B. inertial weight w=0.8

we set w=0.8, the convergence plot of 5 functions is illustrated as fig.3. The result is illustrated as table 3.



f1 f2 f3



f4 f5

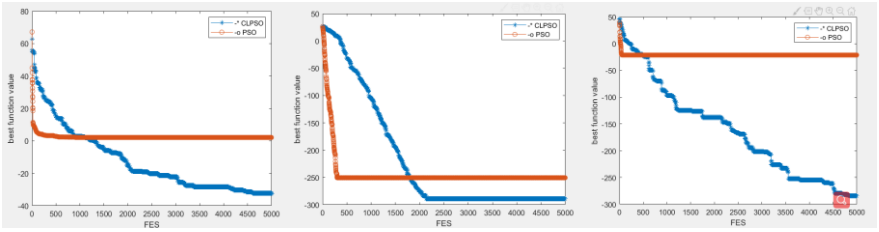
Fig3 w=0.8

Table 3

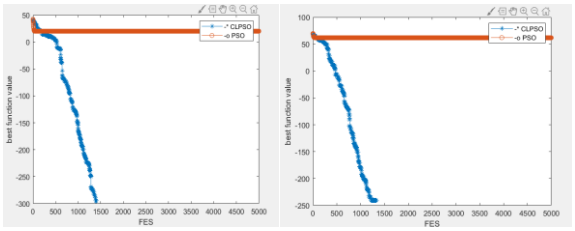
Function no	Mean value PSO	Mean value CLPSO
f1	2.13711646175518	3.98305953439694
f2	7.10542735760100e-15	3.552713678800501e-15
f3	0.0856024903868296	0.0682831974468577
f4	1.20000000000000	0.696907348684435
f5	639.567006917965	23.6876669228875

C. inertial weight w=0.4

we set w=0.4, the convergence plot of 5 functions is illustrated as fig.4. The result is illustrated as table 4.



f1 f2 f3



f4 f5

Fig4 w=0.4

Table 4

Function no	Mean value PSO	Mean value CLPSO
f1	4.73953897535729	0.569873940536255
f2	2.06188560728937	3.552713678800 50e-15
f3	0.222828284448161	0.600000000000000
f4	18	0.696907348684435
f5	1385.89021455741	142.126001537325

From the result, we easily find that the linear changed w (from 0.8 to 0.4) performs better than a static w. when w=0.4, the particle converge more quickly but the result is worse than w=0.8. which could also prove that a small w focus more on local search while a large w focus more on global search. When the particle began to search, a large w can help it to fly far from present position so that it can find a better solution quickly. When the generation goes up, particle should focus on a small area to find the global optima. Therefore, a linear changed w is better in this algorithm. The test result also proves that.

c1 and c2 are learning rates that governing the cognition and social components. When c1=0, c2>0, it is social only mode, particles will learn only from the gbest, it may easily drop into the local optimal solution. When c1>0, c2=0, it is cognition only mode, particles only learn from its pbest, particles don't share information with each other, which makes the algorithm converge slowly. When c1>0, c2>0, it is full mode, algorithm will have better performance in this mode. Experimentally, c1=2, c2=2 could have a good performance.

We also test three modes in 5 function in PSO, each function run 5 times. The result is shown in the table below:

Table 5

Function no	Result(mean) [0,2]	[2,0]	[2,2]	[1.5 1.5]
f1	0.7980	540.073111668600	1.2860	1.4864
f2	0.7913	18.3092319886193	0.0000	0.9629
f3	0.3175	92.4099163679090	0.0689	0.1299
f4	25.8000	65.886119131908840	1.2000	7.0000
f5	722.4886	1.91638121452116e+03	497.4410	414.5363

From the result we can see c1=2, c2=2 has a better performance.

The number of particles determine the ability of searching area. if the number (ps) is too small, the searching space is small, which will lead to premature convergence. If the number is too big, it is compute-intensive which will take too much time. Meanwhile, when the number excess a certain number, the performance will not increase explicitly.
We respectively set ps to 1,5,10,15,50 in PSO to see its performance.
We test in 5 functions, each function run 5 times. The result is shown in the table as below.

Table 6

Function no	Result(mean) ps=1	Ps=5	Ps=10	Ps=15	Ps=50
f1	862.3115180142	4.6083	0.1831	0.0797	0.0695
f2	19.57576290290	2.1137	0.2310	0.0000	0.0000
f3	137.2678771076	0.2875	0.1318	0.0694	0.0630
f4	113.2165524508	21.0000	10.8000	5.8000	3.2000
f5	2600.019409142	694.8448	588.2481	450.0657	379.0027

Therefore, when the particle number excesses 10, the performance in these question will not improve significantly. According to the test result, we use ps=15 in the following parts.

In CLPSO algorithm, what influence the performance is the refreshing gap m. Particles will stop to improve the performance if the generation excess a certain number. If the m is too large, particles could not learn from a good exemplar, and will waste time on poor direction. Therefore, a proper refreshing gap m is very important.

To find out the impact of this parameter, I apply different refreshing gap to 5 problems. The CLPSO is run 5 times in each problem.
The result is shown as fig.5:

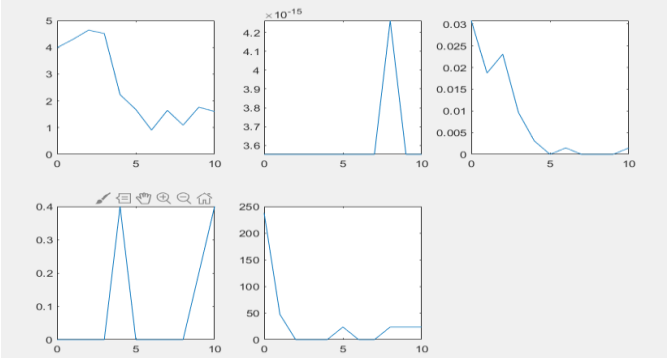


Fig 5

x-axis is the refreshing gap m; y-axis is the optimal solution to the corresponding problem. From the fig.5, we can observe that m effect the final result(gbest). when m=5 or 6, we could have a better performance.

Final Result

According to the important parameters we found in the former sections, I decided to set the value of these parameter as the table shown below:

Table7

w	0.8–0.4(linear decreasing)
C1 C2	[2 , 2]
Particle number	15
Refreshing gap m	5
Variance Dimension	15
Max generation	5000

The chosen problems come from “CLPSO numerical optimization problems (bound constrained problems)”.

We run each problem for 30 times. The test result is shown as the table below:

In PSO:

Table 8

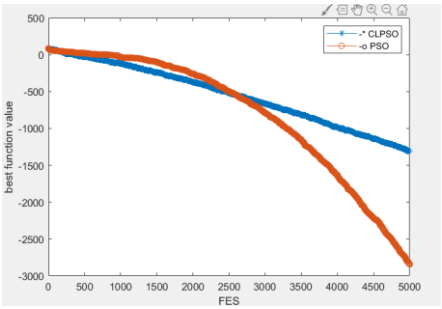
Function no.	mean	median	standard deviation
f1	3.50927836592252e-132	3.96936023405108e-136	9.94481856951314e-132
f2	1.19413686539089	1.04397458451567	0.603989536021889
f3	9.11863177558795e-15	7.10542735760100e-15	3.81081284628988e-15
f4	0.0821446068466369	0.0750105156589254	0.0425328358032912
f5	1.16666666666667	1	1.11674815279973
f6	394.794448714793	414.534171150533	171.304337543837
f7	9.47390314346800e-16	0	2.27261349276905e-15
f8	1.14025213282016	0.976161879391720	0.561500032522331
f9	2.03380134611256	2.14150003001047	0.570111004259672
f10	1.75050426137735e-09	2.73055641271476e-10	2.69964717717597e-09

In CLPSO:

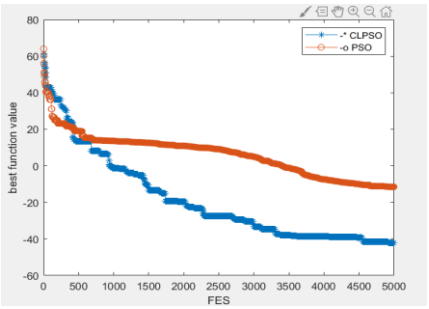
Table 9

Function no.	mean	median	standard deviation
f1	1.25198284831573e-62	2.73236916724016e-63	3.65080768077221e-62
f2	1.23826326149022	0.770171070249482	1.34862770353698
f3	3.55271367880050e-15	3.55271367880050e-15	0
f4	0.00340396467630921	1.07945320915182e-06	0.00537566475994766
f5	0.00340396467630921	0	0.182574185835055
f6	31.5835558971834	0	61.6862922707055
f7	0	0	0
f8	0.755693444233255	0.756657698576275	0.137803674629378
f9	1.49037264918556	1.53042749467589	0.488754467566446
f10	5.93669103010599e-09	1.18520032463031e-09	1.55072677844458e-08

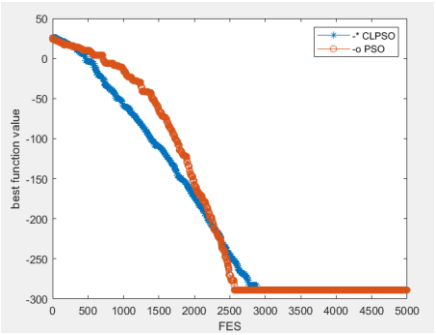
The convergence plot is shown as Fig 6 below:



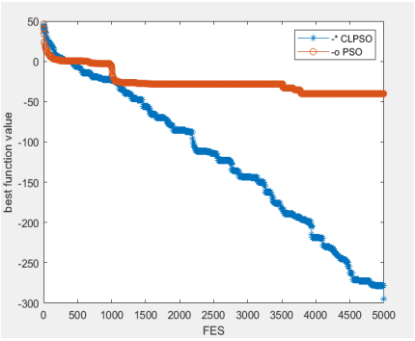
f1



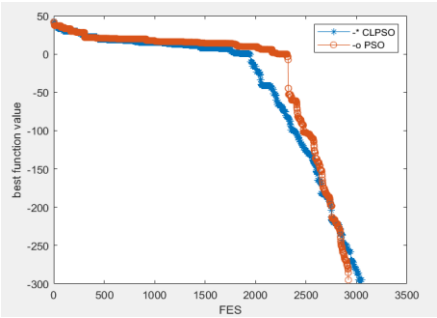
f2



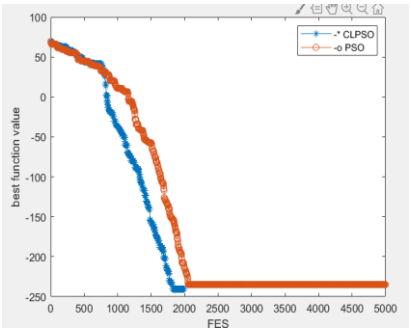
f3



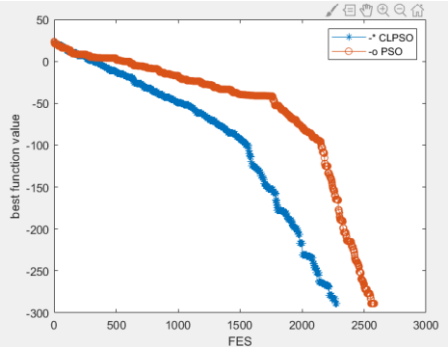
f4



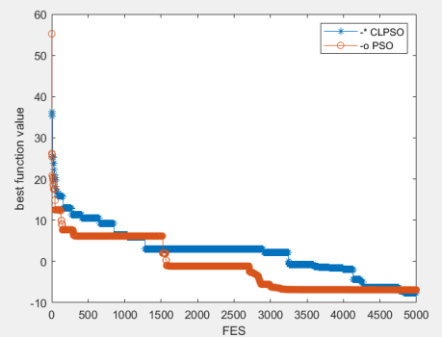
f5



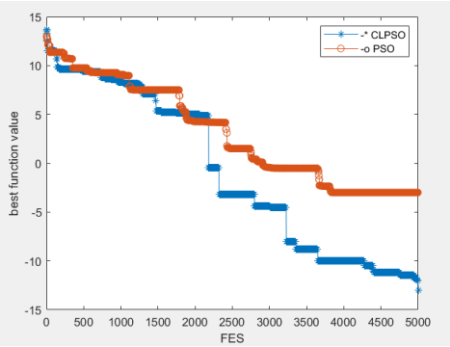
f6



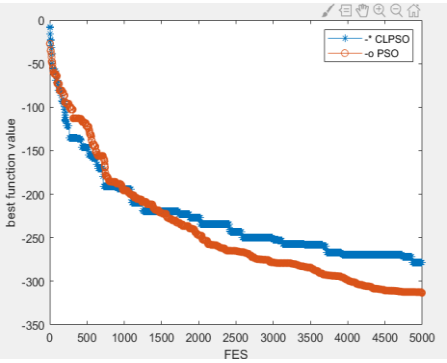
f7



f8



f9



f10

Fig 6

Conclusion

In this EE6227-assignment, we describe and compare CLPSO and PSO algorithm. According to the evaluation result shown above, we could know that different parameter such as w, c1, c2, refreshing gap m, will affect the performance of these two algorithm. After discussion of the better tuning parameter, we compare the performance of these two algorithm in the premise of setting them in the same parameter. In the result shown above, we can find that PSO algorithm could converge quicker than CLPSO in some function (f2 f3 f4 f5 f8), because it has been significantly effect by the local optimal. So the solution it found might not be the global optimal. CLPSO fix this problem, in most cases, CLPSO could find the better result than PSO due to its new learning strategy (f1 f3 f4 f5 f6 f7 f8 f9).

Reference

[1]. J. J. Liang, A. K. Qin, P. N. Suganthan and S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," in *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 3, pp. 281-295, June 2006, doi: 10.1109/TEVC.2005.857610.

[2]. ——, "Parameter selection in particle swarm optimization," in Proc. 7th Conf. Evol. Programming, New York, 1998, pp. 591–600.

[3]. Eberhart, R., & Kennedy, J. (1995). A new optimizer using particle swarm theory. MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science, 39–43. https://doi.org/10.1109/MHS.1995.494215