PARTICLE SWARM OPTIMISATION UPDATION

21CSB0B07_B BHARADWAJ

CSE SEC B BTECH II

FACULTY : DR.SUJIT DAS

NIT WARANGAL

PROBLEM STATEMENT :

FIND THE MAXIMUM OF THE FUNCTION

f(x)= -x^2+10*x+20   where x lies between -10 and 10 (both inclusive)

SOLUTIONS :

TWO APPROACHES ONE IS BY USING THE ORIGINAL PSO ALGORITHM AND THE OTHER BY USING MODIFIED PSO ALGORITHM

SOURCE CODE FOR ORIGINAL PSO ALGORITHM

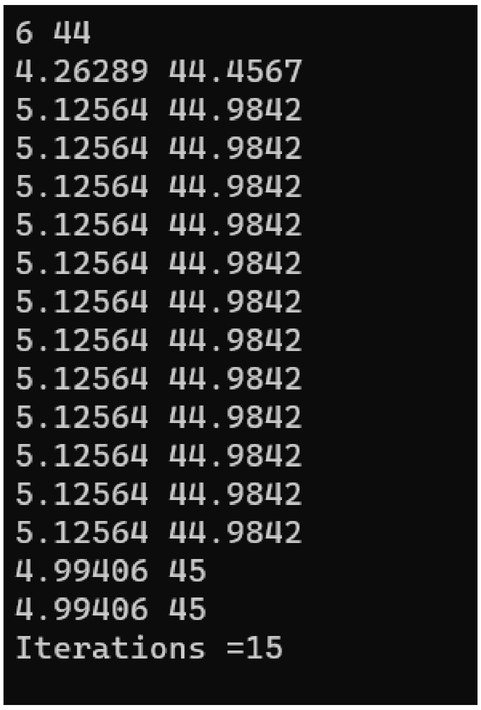
LANGUAGE : C++

```
#include<bits/stdc++.h>

using namespace std;

double pos[10]={0},vel[10]={0},pbest[10]={0};

double Gbest=0;

int c1=1,c2=1;

void init()

{

for(int i=0;i<10;i++) pos[i]=rand()%20-10;//Random initialisation

}

double fitness_value(double x)

{

return (-(x*x)+10*x+20);
```

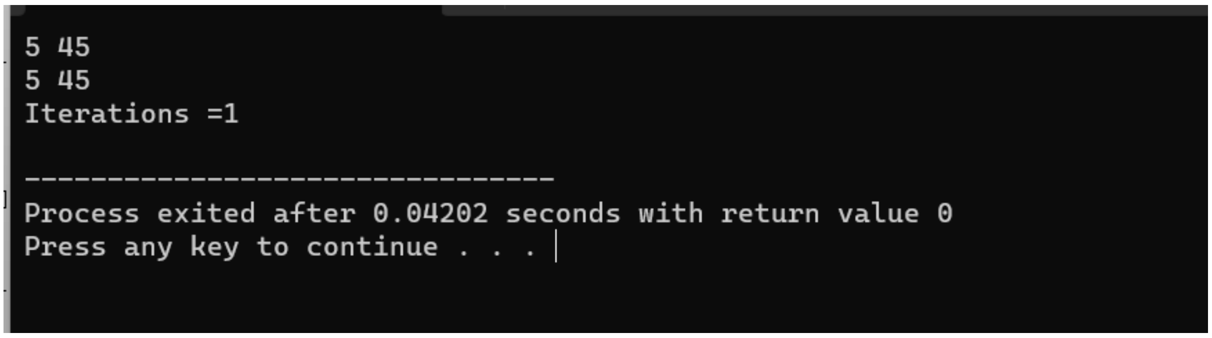
```
}
int main()
{
srand(time(0));
init();
int itr=0;
while(abs(Gbest-5)>0.05)
{
        //Pbest Calculations
for(int j=0;j<10;j++)
{
if(pbest[j]==0) pbest[j]=pos[j];
else if(fitness_value(pbest[j])<fitness_value(pos[j])) pbest[j]=pos[j];
}
//Gbest Calculations
for(int j=0;j<10;j++)
{
if(fitness_value(Gbest)<fitness_value(pbest[j])) Gbest=pbest[j];
}
//Updating velocities and positions
double r1=(double)rand()/RAND_MAX,r2=(double)rand()/RAND_MAX;
for(int j=0;j<10;j++)
{
vel[j]=vel[j]+c1*r1*(pbest[j]-pos[j])+c2*r2*(Gbest-pos[j]);
pos[j]=pos[j]+vel[j];
}
cout<<Gbest<<" "<<fitness_value(Gbest)<<"\n";
itr++;
}
cout<<Gbest<<" "<<fitness_value(Gbest)<<"\n";
cout<<"Iterations ="<<itr<<"\n";
```

}

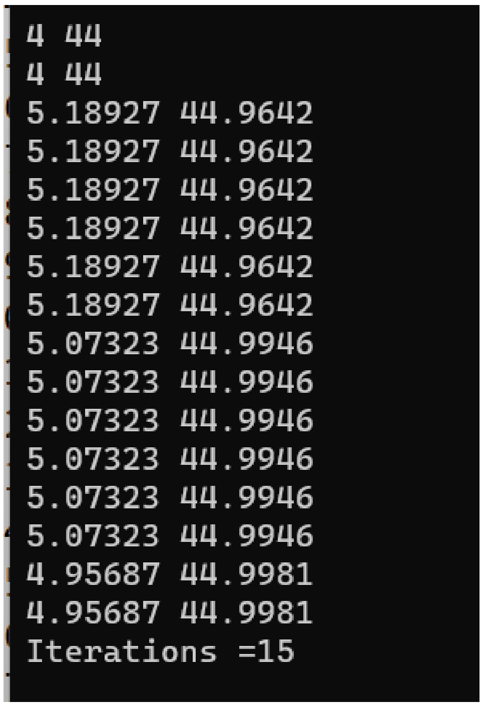
OUTPUTS FOR THE ABOVE CODE(10)



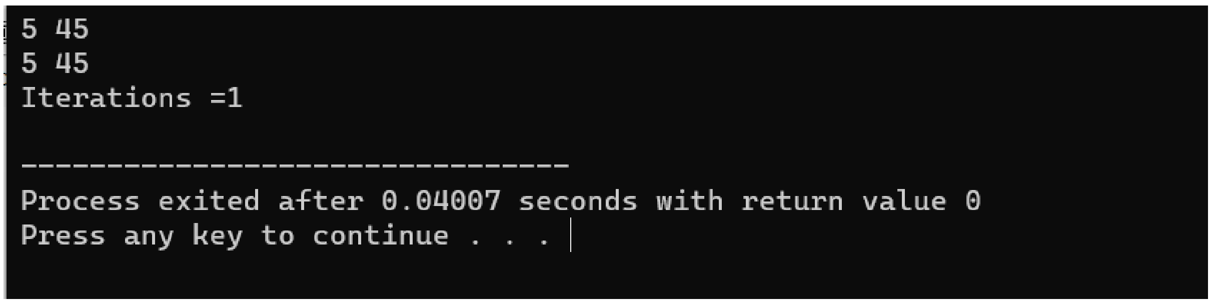
```
6 44
4.26289 44.4567
5.12564 44.9842
5.12564 44.9842
5.12564 44.9842
5.12564 44.9842
5.12564 44.9842
5.12564 44.9842
5.12564 44.9842
5.12564 44.9842
5.12564 44.9842
5.12564 44.9842
5.12564 44.9842
5.12564 44.9842
4.99406 45
4.99406 45
Iterations =15
```



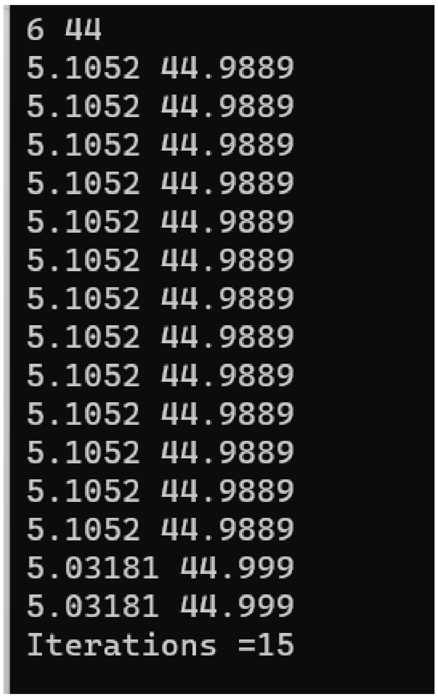
```
5 45
5 45
Iterations =1

--------------------------------
Process exited after 0.04202 seconds with return value 0
Press any key to continue . . .
```



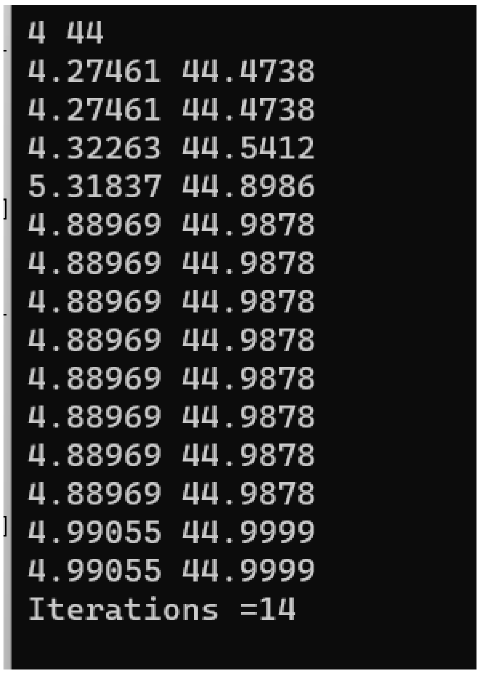
```
4 44
4 44
5.18927 44.9642
5.18927 44.9642
5.18927 44.9642
5.18927 44.9642
5.18927 44.9642
5.18927 44.9642
5.07323 44.9946
5.07323 44.9946
5.07323 44.9946
5.07323 44.9946
5.07323 44.9946
5.07323 44.9946
4.95687 44.9981
4.95687 44.9981
Iterations =15
```

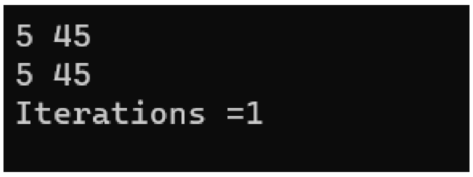


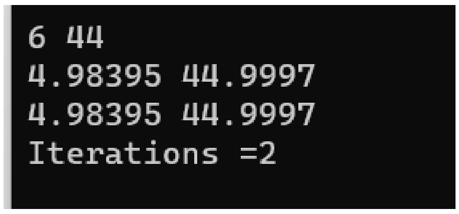
```
5 45
5 45
Iterations =1

--------------------------------
Process exited after 0.04007 seconds with return value 0
Press any key to continue . . .
```

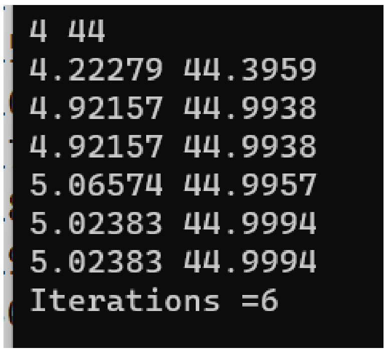


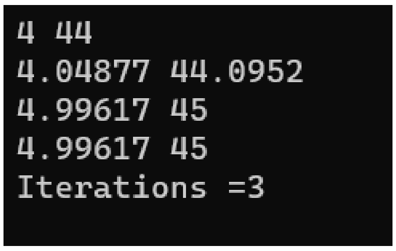
```
6 44
5.1052 44.9889
5.1052 44.9889
5.1052 44.9889
5.1052 44.9889
5.1052 44.9889
5.1052 44.9889
5.1052 44.9889
5.1052 44.9889
5.1052 44.9889
5.1052 44.9889
5.1052 44.9889
5.1052 44.9889
5.1052 44.9889
5.03181 44.999
5.03181 44.999
Iterations =15
```


```
4 44
4.27461 44.4738
4.27461 44.4738
4.32263 44.5412
5.31837 44.8986
4.88969 44.9878
4.88969 44.9878
4.88969 44.9878
4.88969 44.9878
4.88969 44.9878
4.88969 44.9878
4.88969 44.9878
4.88969 44.9878
4.99055 44.9999
4.99055 44.9999
Iterations =14
```


```
5 45
5 45
Iterations =1
```


```
6 44
4.98395 44.9997
4.98395 44.9997
Iterations =2
```


```
4 44
4.22279 44.3959
4.92157 44.9938
4.92157 44.9938
5.06574 44.9957
5.02383 44.9994
5.02383 44.9994
Iterations =6
```



```
4 44
4.04877 44.0952
4.99617 45
4.99617 45
Iterations =3
```

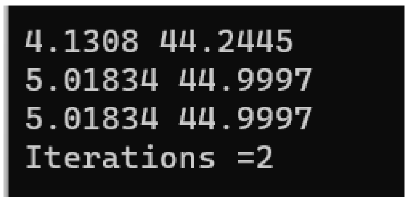
SOURCE CODE FOR UPDATED PSO ALGORITHM

LANGUAGE : C++

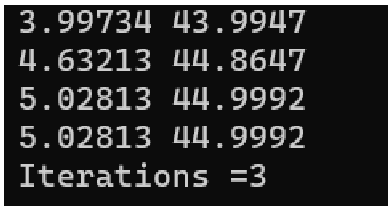
```
    #include<bits/stdc++.h>
    using namespace std;
    double pos[10]={0},vel[10]={0},pbest[10]={0};
    double Gbest=0;
    int c1=1,c2=1;
    int lb=0,ub=10;
    void init()
    {
    for(int i=0;i<10;i++)
    {
            double r=(double)rand()/RAND_MAX;//Modification: New type of initialisation of
            positions to improve the search speed
            pos[i]=i+(ub-lb)*r;
            }
            for(int i=0;i<10;i++)
    {
            double r=(double)rand()/RAND_MAX;//Modification: New type of initialisation of
            velocities to improve the search speed
            vel[i]=r;
            }
    }
    double fitness_value(double x)
```

```
{
return (-(x*x)+10*x+20);
}
int main()
{
srand(time(0));
init();
int itr=0;
while(abs(Gbest-5)>0.05)
{
        //Pbest calculations
for(int j=0;j<10;j++)
{
if(pbest[j]==0) pbest[j]=pos[j];
else if(fitness_value(pbest[j])<fitness_value(pos[j])) pbest[j]=pos[j];
}
//Gbest calculations
for(int j=0;j<10;j++)
{
if(fitness_value(Gbest)<fitness_value(pbest[j])) Gbest=pbest[j];
}
int r=rand()%10;
pbest[r]=Gbest;pos[r]=Gbest;//Modification: Movement of one bird to Gbest to improve the
searching process
//  Updating the velocities and positions
double r1=(double)rand()/RAND_MAX,r2=(double)rand()/RAND_MAX;
for(int j=0;j<10;j++)
{
vel[j]=vel[j]+c1*r1*(pbest[j]-pos[j])+c2*r2*(Gbest-pos[j]);
if(fitness_value(pos[j]+vel[j])>fitness_value(pos[j])) pos[j]=pos[j]+vel[j];//Modification:
Movement only towards the direction of optimum
}
cout<<Gbest<<" "<<fitness_value(Gbest)<<"\n";

itr++;

}

cout<<Gbest<<" "<<fitness_value(Gbest);

cout<<"Iterations ="<<itr<<"\n";

}
```

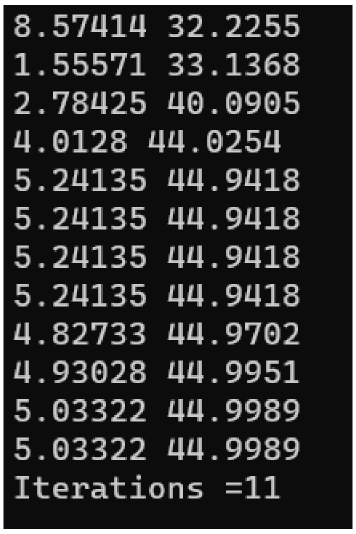
OUTPUTS FOR THE ABOVE CODE(10)



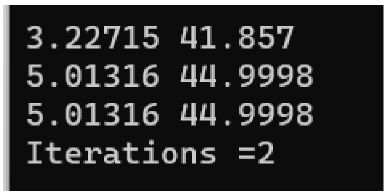
```
4.1308 44.2445
5.01834 44.9997
5.01834 44.9997
Iterations =2
```



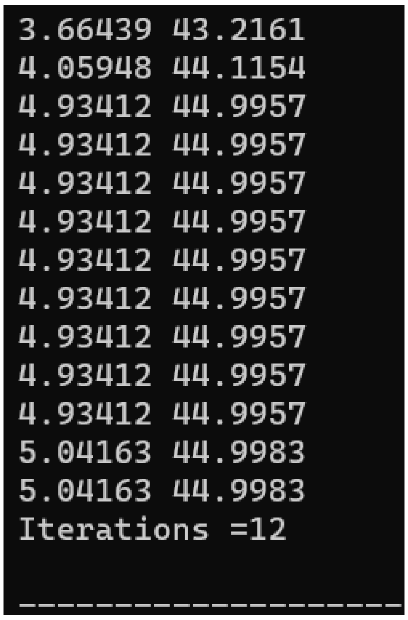
```
3.99734 43.9947
4.63213 44.8647
5.02813 44.9992
5.02813 44.9992
Iterations =3
```



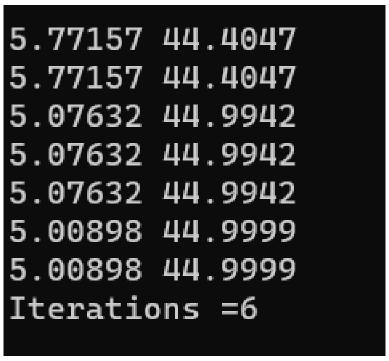
```
8.57414 32.2255
1.55571 33.1368
2.78425 40.0905
4.0128 44.0254
5.24135 44.9418
5.24135 44.9418
5.24135 44.9418
5.24135 44.9418
4.82733 44.9702
4.93028 44.9951
5.03322 44.9989
5.03322 44.9989
Iterations =11
```



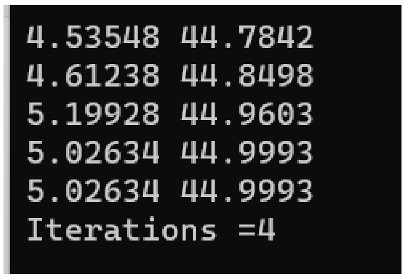
```
3.22715 41.857
5.01316 44.9998
5.01316 44.9998
Iterations =2
```



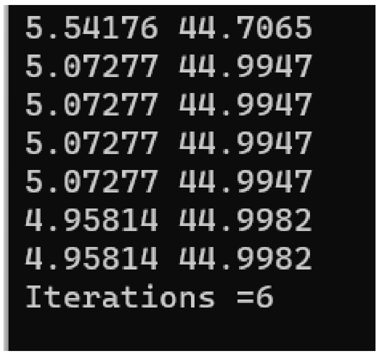
```
3.66439 43.2161
4.05948 44.1154
4.93412 44.9957
4.93412 44.9957
4.93412 44.9957
4.93412 44.9957
4.93412 44.9957
4.93412 44.9957
4.93412 44.9957
4.93412 44.9957
4.93412 44.9957
5.04163 44.9983
5.04163 44.9983
Iterations =12

--------------------
```



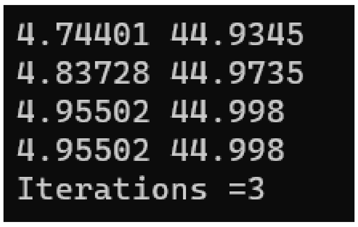
```
5.77157 44.4047
5.77157 44.4047
5.07632 44.9942
5.07632 44.9942
5.07632 44.9942
5.00898 44.9999
5.00898 44.9999
Iterations =6
```



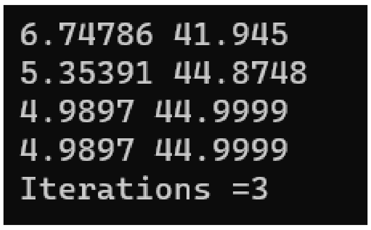
```
4.53548 44.7842
4.61238 44.8498
5.19928 44.9603
5.02634 44.9993
5.02634 44.9993
Iterations =4
```



```
5.54176 44.7065
5.07277 44.9947
5.07277 44.9947
5.07277 44.9947
5.07277 44.9947
4.95814 44.9982
4.95814 44.9982
Iterations =6
```



```
4.74401 44.9345
4.83728 44.9735
4.95502 44.998
4.95502 44.998
Iterations =3
```



```
6.74786 41.945
5.35391 44.8748
4.9897 44.9999
4.9897 44.9999
Iterations =3
```

THE ABOVE IS THE CODE FOR THE MODIFIED PSO ALGORITHM WITH 4 MODIFICATIONS

AIM: HEAURISTICS OR METAHEURISTICS AIMS FOR "GOOD ENOUGH SOLUTION IN FAST ENOUGH MANNER " THESE MODIFICATIONS WILL KEEP THAT "GOOD ENOUGH" PART AS IT IS AND FOCUSES ON "FAST ENOUGH MANNER" .

THIS ALGORITHM TRIES TO IMPROVES THE FASTNESS OF THE SEARCH.

THE MODIFICATIONS ARE:

1) INITIALISATION OF POSITIONS

   INSTEAD OF COMPLETELY RANDOM INITIALISATION ,INITIALISING THE PARTICLES UNIFORMLY BY DIVIDING THE SEARCH SPACE INTO SOME N NUMBER OF BLOCKS .AND THEN IN EACH BLOCK ONE PARTICLE WILL BE THERE.THE POSITION OF THE PARTICLE IN THAT BLOCK IS FOUND RANDOMLY.

   IN THIS WAY MOVEMENT TOWARDS THE OPTIMUM POINT WILL BECOME FASTER BECAUSE SOME OR THE OTHER PARTICLE WILL BE PRESENT IN THAT BLOCK .IN RANDOM INITIALISATION IT MIGHT HAPPEN THAT ALL PARTICLES ARE INITIALISED SO FAR FROM OPTIMUM POINT AND IT TAKES TIME TO REACH THAT POINT.

2)INITIALISATION OF VELOCITIES

INSTEAD OF ZERO INITIAL VELOCITIES ,INITIALISING WITH SOME SMALL INITIAL VELOCITY SO THAT THE MOVEMENT TOWARDS THE OPTIMUM POINT BECOMES FASTER.

3) SUDDEN MOVEMENT OF ONE PARTICLE TOWARDS GBEST

THIS STEP IS AN ADDITION TO THE PSO ALGORITHM .HERE AFTER CALCULATING GBEST ,WE ARE PICKING SOME RANDOM PARTICLE AND MOVING THAT TO GBEST POINT. IN THIS WAY WE ARE TRYING TO MOVE ONE MORE PARTICLE TOWARDS OPTIMUM POINT .HENCE FORTH IF MORE NUMBER OF PARTICLES ARE MOVING TOWARDS OPTIMUM POINT.

THE PROBABILITY OF MOVING TO OR NEAR TO OPTIMUM POINT BECOMES FASTER

4) UPDATING ONLY NO WORSE POSITIONS

IN UPDATING THE POSITION OF A PARTICLE ,HERE IAM UPDATING ONLY IF IT IS MOVING TOWARDS THE DIRECTION OF OPTIMUM POINT.THAT IS IF THE FITNESS VALUE IS BETTER OR NO WORSE THAN THE PRESENT VALUE THEN ONLY WE ARE UPDATING THE POSITION .
BECAUSE IF WE UPDATE TO AN LESS OPTIMUM POINT ,IT TAKES MORE TIME TO COME BACK TO OPTIMUM POINT BECAUSE WE ARE MOVING AWAY FROM THE OPTIMUM POINT.

SO THIS TYPE OF UPDATION ALSO FASTENS OUR SEARCH

COMPARISION BETWEEN THE ORIGINAL PSO ALGORITHM AND PROPOSED PSO ALGORITHM

1)FASTNESS:

HERE FASTNESS IS CALCULATED IN TERMS OF THE NUMBER OF ITERATIONS

THE BEST CASE OF PSO : 1 ITERATIONS

THE AVG CASE OF PSO : 6.3 ITERATIONS

THE WORST CASE OF PSO : 15 ITERATIONS

THE BEST CASE OF PROPOSED PSO : 2 ITERATIONS

THE AVG CASE OF PROPOSED PSO : 5.2 ITERATIONS

THE WORST CASE OF PROPOSED PSO : 12 ITERATIONS

2)BENCH MARK PROBLEM (YET TO BE DONE)

OBSERVATONS :

THE PROPOSED PSO ALGORITHM HAS LESS NUMBER OF ITERATIONS ON AN AVERAGE FOR THIS PROBLEM ,AND SO IS FASTER THAN THE ORIGINAL PSO ALGORITHM FOR THIS PROBLEM