EE6227: Programming Assignment I

Ding Bangjie
G2001686F
ding0130@e.ntu.edu.sg

6 March, 2021

Abstract

In this assignment, I chose CEC 2017 bound-constrained benchmarks problem set, and I chose two algorithms, PSO and CLPSO, to conduct experiments. In the course of the experiment, first, fully understand PSO and CLPSO respectively, and implement it by hand according to the existing code. After implementing the two algorithms with code, select 5 problems from the test functions of CEC2017 to adjust the parameters of the two algorithms respectively. Finally, the optimal parameters are selected for comparison between algorithms. Results will be shown by the mean, standard deviation tables, and convergence plots.

1 Introduction

1.1 PSO

The idea of particle swarm algorithm(PSO) originated from the study of predation behavior of birds flocks and fish schools [1] [2]. It simulates the behavior of bird swarms flying for food. The collective cooperation between birds makes the group achieve the optimal goal. It is an optimization method based on Swarm Intelligence. It does not have the 'Crossover' and 'Mutation' operations of genetic algorithms(GA). It finds the global optimum by following the optimal value currently found. Compared with other modern optimization methods, the obvious feature of PSO is that there are few parameters that need to be adjusted, simple and easy to implement, and fast convergence speed.

PSO algorithm introduces two key speeds and positions for each particle. And through the changes of these two parts in the sequential process to achieve evolution. The specific method is as follows:

$$V_{i,k+1}^{d} \leftarrow V_{i,k}^{d} + c_1 \times rand1_{i,k}^{d} \times (pbest_{i,k}^{d} - X_{i,k}^{d}) + c_2 \times rand1_{i,k}^{d} \times (gbest_{i,k}^{d} - X_{i,k}^{d}) \quad (1)$$

$$X_{i,k+1}^{d} \leftarrow X_{i,k}^{d} + V_{i,k+1}^{d} \quad (2)$$

Where $X_{i,k} = (X_{i,k}^{1}, X_{i,k}^{2}, X_{i,k}^{3}, \cdots, X_{i,k}^{D})$ is the position of the i^{th} particle at the k^{th} iteration. And the D is the number of the parameters to be optimized.$V_{i,k} = (V_{i,k}^{1}, V_{i,k}^{2}, V_{i,k}^{3}, \cdots, V_{i,k}^{D})$

represents velocity of the i^{th} particle at the k^{th} iteration. $pbest_{i,k} = (pbest^1_{i,k}, pbest^2_{i,k}, pbest^3_{i,k}, \cdots, pbest^D_{i,k})$ is the previous position found for the i^{th} particle at k^{th} iteration. Similarly, $gbest = (gbest^1, gbest^2, gbest^3, \cdots, gbest^D)$ is the best position found by the whole population.

The particle velocity update formula 1 consists of three parts: the first part is the 'inertial part', that is, the memory of the particle's previous speed; the second part is the 'self-recognition' part, which can be understood as the distance between the current position of the i^{th} particle and its best position The third part is the 'social experience' part, which represents the information sharing and cooperation between particles, which can be understood as the distance between the current position of the i^{th} particle and the best position of the group.

In 1998, Shi et al. published a paper [3] revised the previous formula 1. Introduced a parameter called inertia weight w into formula 1 as follows:

$$V^d_{i,k+1} \leftarrow w \times V^d_{i,k} + c_1 \times rand1^d_{i,k} \times (pbest^d_{i,k} - X^d_{i,k}) + c_2 \times rand1^d_{i,k} \times (gbest^d_{i,k} - X^d_{i,k}) \quad (3)$$

The larger the w, the stronger global search ability, and the weaker local search ability, with the smaller the value, the weaker global search ability, and the stronger local search ability.

In the paper [3], initially, Shi took w as a constant. But the later experiments had proved that dynamic w can obtain better optimization results than fixed values. Currently, a linearly decreasing weight(LDW) strategy that Shi suggested at the paper is widely used. The specific formula is as follows:

$$w_{iter} = w_{max} - \frac{w_{max} - w_{min}}{iter_{max}} \times iter \quad (4)$$

Where the $iter$ represents the $iter^{th}$ iteration in the process of optimization. And w_{max} stands for the largest w given. Similarly, the w_{min} is the smallest one. In particular, w is only related to the number of iterations, not to the dimensionality. Therefore formula 3 can be written as:

$$V^d_{i,k+1} \leftarrow w_{iter} \times V^d_{i,k} + c_1 \times rand1^d_{i,k} \times (pbest^d_{i,k} - X^d_{i,k}) + c_2 \times rand1^d_{i,k} \times (gbest^d_{i,k} - X^d_{i,k}) \quad (5)$$

In this assignment, I use the update formulas [2, 5] of position and velocity in the PSO algorithm.

1.2 CLPSO

In 2006, J.Liang et al. published a paper [4] proposed a modified PSO algorithm called comprehensive learning particle swarm optimizer(CLPSO). The most significant improvement in this paper is proposed the formula followed:

$$V^d_{i,k+1} \leftarrow w \times V^d_i + c^* \times rand^d_i \times (pbest^d_{f_i(d)} - X^d_i) \quad (6)$$

Where $f_i = [f_i(1), f_i(2), \cdots, f_i(D)]$ defines which particle's $pbest$ the i^{th} particle should follow. And f_i depended by Pc. For each dimension of the i^{th} particle, it will generate a random number. If this random number is larger than Pc_i, the corresponding dimension

will learn from its own *pbest*, otherwise it will learn from another particle's *pbest*. In my viewpoint, the Pc which the paper introduced like the crossover probability we used in genetic algorithms, because the Pc can decide which particle's *pbest* should been chosen. also, the final result after the 'crossover' course, some parts of the each particle's *pbest* can be from its own. According to the paper, the Pc_i's value set by the following expression:

$$Pc_i = 0.05 + 0.45 \times \frac{exp(\frac{10(i-1)}{ps-1}) - 1}{exp(10) - 1} \tag{7}$$

Where ps is the population size. $i$is the i^{th} particle.

According to formula 7, we can get that $Pc \in [0.05, 0.5]$. Figure.1 presents an example of each particle's Pc with population size of 100.

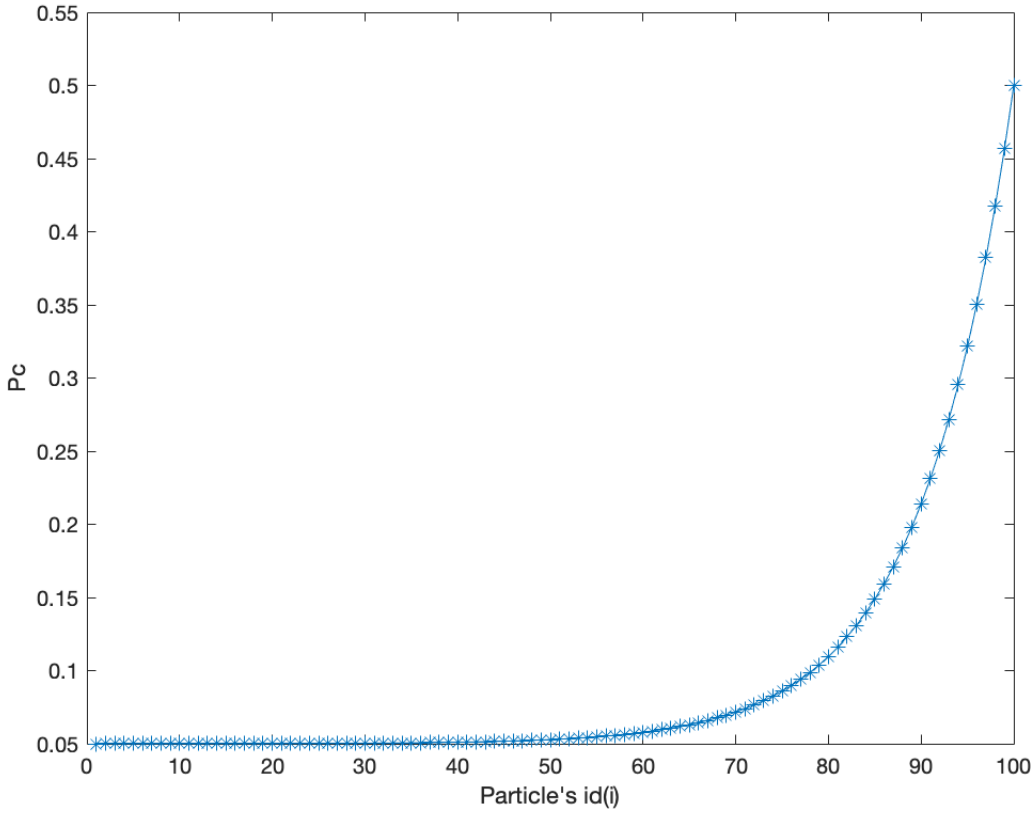

Figure 1: Each particle's Pc with a population size of 100.

Compare the formula 6 with the formula 5, it is obviously that the 'social experience' part is gone. And the 'self-recognition' part is modified.

2 Parameter Tuning

In this section, I use **Grid-search** for algorithmic parameter tuning.using 5 repetitions. And the test functions for tuning come from [5]. All results will be shown by mean and standard deviation.

For tuning, I chose *Shifted and Rotated Zakharov Function, Shifted and Rotated Rosenbrock's Function, Shifted and Rotated Rastrigin's Function, Shifted and Rotated Expanded Scaffer's F6 Function,Shifted and Rotated Lunacek Bi_Rastrigin Function*. Shortly, the chosen functions are $f_1, f_3, f_4, f_5, f_6, f_7$(The function which index is 2 was been deleted). As shown in Table.1

	No.	Functions	$F_i^* = F_i(x^*)$
Unimodal Functions	1	Shifted and Rotated Bent Cigar Function	100
	3	Shifted and Rotated Zakharov Function	300
Simple Multimodal Functions	4	Shifted and Rotated Rosenbrock's Function	400
	5	Shifted and Rotated Rastrigin's Function	500
	6	Shifted and Rotated Expanded Scaffer's F6 Function	600
	7	Shifted and Rotated Lunacek Bi_Rastrigin Function	700
Search Range: $[-100, 100]^D$			

Table 1: Chosen functions for tuning from the CEC'17 Test Functions

2.1 PSO

As described in Section **1**, for PSO, there are 4 parameters that we can tune, which are $c_1, c_2, w_{max}, w_{min}$. In fact, these two components also prevent the exploration and exploitation of the PSO algorithm. The *pbest* pulls the particles to explore the optimal area within the whole searching range, while the *gbest* pulls the particles to move to the existing optimal area.

2.1.1 Learning Factor

When tuning learning factor c_1, c_2, I set the step length of Grid-search as 0.1. And the search range for each parameter is $[1, 2]$. In this way, we will get 10×10 combinations. In order to save space, the whole results table in .mat format can get on my Github homepage[1]. The table only shows the best combination of c_1 and c_2 for each selected test function.

Table.2 and Table.3 show the best performance on standard deviation and mean of each chosen function. From there two tables, we can observe that when $c_1 = \mathbf{1.7}$ and $c_2 = \mathbf{1.3}$ can combine the best learning factor.

[1]https://github.com/bangjieding/course_code.git/EE6227/Assignments/CEC2017-BoundContrained-master/codes/Matlab/tuning_c1_c2_gbestval.mat

line	c_1	c_2	f_3		f_4		f_5	
			mean	std_dev	mean	std_dev	mean	std_dev
1	1	1	300	±0	400.17	±0.0598	508.75	±3.749
22	1.1	2	3.00E+02	±2.842E-14	400.24	±0.023	507.36	±4.371
59	1.5	1.3	3.00E+02	±2.842E-14	401.37	±0.830	503.97	±0
4	1	1.3	3.00E+02	±4.019E-14	400.21	±0.06	505.96	±2.900
92	1.8	1.3	3.00E+02	±4.019E-14	401.94	±0.400	503.97	±0.99
line	c_1	c_2	f_6		f_7			
			mean	std_dev	mean	std_dev		
1	1	1	600	±2.542E-09	715.00	±4.485		
22	1.1	2	600	±0	717.64	±4.679		
59	1.5	1.3	600	±0	715.78	±0.722		
4	1	1.3	600	±0	720.52	±3.184		
92	1.8	1.3	600	±0	714.14	±0.519		

Table 2: The best performance on standard deviation of each chosen function.

line	c_1	c_2	f_3		f_4		f_5	
			mean	std_dev	mean	std_dev	mean	std_dev
1	1	1	300	0	400.17	±0.059	508.75	±3.749
11	1	2	300	±4.92E-14	400.12	±0.084	508.35	±6.07
78	1.7	1	300	±4.92E-14	401.2	±1.072	502.38	±1.13
10	1	1.9	300	±4.92E-14	400.22	±0.134	505.96	±2.814
86	1.7	1.8	300	±4.02E-14	401.45	±0.866	505.57	±1.664
line	c_1	c_2	f_6		f_7			
			mean	std_dev	mean	std_dev		
1	1	1	600	±2.54E-09	715.00	±4.485		
11	1	2	600	±0	718.17	±4.48		
78	1.7	1	600	±1.14E-13	716.15	±2.633		
10	1	1.9	600	±1.14E-13	716.69	±4.06		
86	1.7	1.8	600	±0	712.14	±4.589		

Table 3: The best performance on mean of each chosen function.

2.2 Inertia Weight

As mentioned in section **1**, the inertial weight is introduced by Shi [3], and the larger the w, the stronger global search ability, and the weaker local search ability, with the smaller the value, the weaker global search ability, and the stronger local search ability.

When tuning inertia weight w, I set the step length of one-dimension Grid-search as 0.1. Also, $thesearchrange \in [0.1, 1.0]$. And compute w_i according to the formula 4. Also, the

whole table of tuning w_{max} and w_{min} can be seen on my Github homepage[2].

line	w_{max}	w_{min}	f_3		f_4		f_5	
			mean	std_dev	mean	std_dev	mean	std_dev
7	0.5	0.1	300	±5.68E-14	411.99	±15.353	519.56	±11.6598
16	0.7	0.1	300	±4.02E-14	400.30	±0.224	505.63	±2.07116
31	0.9	0.3	300	±0	401.10	±0.6438	503.31	±1.1488
39	1	0.3	300	±4.02E-14	401.48	±1.0390	505.6	±1.1488
26	0.8	0.5	300	±0	401.62	±0.516	506.3014	±0.5744
8	0.5	0.2	300	±0	406.36	±0.2831	517.245	±3.039
36	0.9	0.8	300	±0	402.88	±0.0568	506.301	±1.51982
43	1	0.7	300	±0	401.80	±0.4233	504.97	±6.96E-14
19	0.7	0.4	300	±0	401.31	±0.7817	507.62	±5.00784
41	1	0.5	300	±4.02E-14	401.87	±0.4354	503.97	±1.98991

line	w_{max}	w_{min}	f_6		f_7	
			mean	std_dev	mean	std_dev
7	0.5	0.1	600.62	±0.866	722.96	±4.8599
16	0.7	0.1	600	±0.0002	719.07	±2.4656
31	0.9	0.3	600	±0	714.5	±2.7196
39	1	0.3	600	±1.14E-13	716.01	±1.6854
26	0.8	0.5	600	±0	710.38	±6.9813
8	0.5	0.2	600.50	±0.8569	725.06	±7.4967
36	0.9	0.8	600	±0	716.08	±2.2588
43	1	0.7	600	±0	716.69	±1.3589
19	0.7	0.4	600	±0	717.65	±1.6689
41	1	0.5	600	±0	714.05	±0.6832

Table 4: The best performance on mean and standard deviation of each chosen function.

From Table. 4, we can get that the best combination of the performance of w_{max} and w_{min} is $w_{max} = \mathbf{0.9}$ and $w_{min} = \mathbf{0.3}$. And according to the formula 4, the expression now comes to as following:

$$w_{iter} = 0.9 - \frac{0.9 - 0.3}{iter_{max}} \times iter \tag{8}$$

Where the $iter_{max}$ is the maximum times of iterations, $iter$ represents the $iter^{th}$ iteration.

[2]https://github.com/bangjieding/course_code.git/EE6227/Assignments/CEC2017-BoundContrained-master/codes/Matlab/tuning_wmax_wmin.mat

References

[1] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in Proc. 6th Int. Symp. Micromachine Human Sci., Nagoya, Japan, 1995, pp. 39–43.

[2] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in Proc. IEEE Int. Conf. Neural Networks, 1995, pp. 1942–1948.

[3] Y. Shi and R. C. Eberhart, "A modified particle swarm optimizer," in Proc. IEEE Congr. Evol. Comput., 1998, pp. 69–73.

[4] J.J. Liang, A.K. Qin, P.N. Suganthan, S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," IEEE Trans. Evol. Comput., 2006, pp. 281–295.

[5] N. H. Awad, M. Z. Ali, J. J. Liang, B. Y. Qu and P. N. Suganthan, "Problem Definitions and Evaluation Criteria for the CEC 2017 Special Session and Competition on Single Objective Bound Constrained Real-Parameter Numerical Optimization," Technical Report, Nanyang Technological University, Singapore, November 2016.