Particle Swarm Optimization Using Adaptive Mutation

Millie Pant[1], Radha Thangaraj[1] and Ajith Abraham[2]
[1]*Department. of Paper Technology, IIT Roorkee, India*
[2]*Center of Excellence for Quantifiable Quality of Service,*
Norwegian University of Science and Technology, Norway
millifpt@iitr.ernet.in, t.radha@ieee.org, ajith.abraham@ieee.org

Abstract

Two new variants of Particle Swarm Optimization (PSO) called AMPSO1 and AMPSO2 are proposed for global optimization problems. Both the algorithms use adaptive mutation using Beta distribution. AMPSO1 mutates the personal best position of the swarm and AMPSO2, mutates the global best swarm position. The performance of proposed algorithms is evaluated on twelve unconstrained test problems and three real life constrained problems taken from the field of Electrical Engineering. The numerical results show the competence of the proposed algorithms with respect some other contemporary techniques.

1. Introduction

PSO [1] is comparatively a newer addition to a class of stochastic algorithms that has been successfully used to solve complex test and real life problems. However the point of criticism is that they have not demonstrated themselves as efficient as promised. For example, in case of PSO although the rate of convergence is good due to fast information flow among the solution vectors, its diversity decreases very quickly in the successive iterations resulting in a suboptimal solution [2]. At the same time, in comparison to PSO, Genetic Algorithms (GA) and Evolutionary Programming (EP) have slower convergence rate. In order to improve the diversity of PSO, without compromising with the solution quality we propose two new variants of PSO using Self adaptive Mutation Technique (used in Evolutionary Programming (EP)) [3].

In the modified versions AMPSO1 and AMPSO2, instead of mutating the entire population of potential solutions (as in EP), we mutated only the personal and global best particles of the population. The remaining of the paper is organized as follows. Section 2, describes the PSO algorithm, Section 3 gives the proposed algorithm. Section 4 gives the experimental settings and numerical results. The paper finally concludes with Section 5.

2. Particle Swarm Optimization

For a D-dimensional search space the position of the ith particle is represented as $X_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$. Each particle maintains a memory of its previous best position $P_{besti} = (p_{i1}, p_{i2} \ldots p_{iD})$. The best one among all the particles in the population is represented as $P_{gbest} = (p_{g1}, p_{g2} \ldots p_{gD})$. The velocity of each particle is represented as $V_i = (v_{i1}, v_{i2}, \ldots v_{iD})$. In each iteration, the P vector of the particle with best fitness in the local neighborhood, designated g, and the P vector of the current particle are combined to adjust the velocity along each dimension and a new position of the particle is determined using that velocity. Equations of velocity vector and position vector given by:

$$v_{id} = wv_{id} + c_1 r_1 (p_{id} - x_{id}) + c_2 r_2 (p_{gd} - x_{id}) \qquad (1)$$

$$x_{id} = x_{id} + v_{id} \qquad (2)$$

First part of equation (1) represents the inertia of the previous velocity, second part is the cognition part, third part represents the cooperation among particles and is called social component. Acceleration constants c_1, c_2 and inertia weight w are the predefined by the user.r_1, r_2 are the uniformly generated random numbers between 0 and 1.

3. Adaptive Mutation Based PSO

The proposed versions AMPSO1 and AMPSO2differ from each other in the sense that in AMPSO1, personal best Pbest position of the swarm particles is mutated and in AMPSO2, the global best (gbest) position of the particle is mutated. The particles are mutated at the end of each iteration according to the following rule:

$$x_{ij} = x_{ij} + \sigma'_{ij} * Betarand_j() \qquad (3)$$

where, $\sigma'_{ij} = \sigma_{ij} * \exp(\tau\ N(0,1) + \tau'\ N_j(0,1))$, N(0, 1) denotes a normally distributed random number with mean zero and standard deviation one. N_j (0, 1) indicates that a different random number is generated for each value of j. τ and τ' are set as $1/\sqrt{2n}$ and $1/\sqrt{2\sqrt{n}}$ respectively [4]. *Betarand$_j$* () is a random number generated by beta distribution with parameters less than 1.

The computational steps of proposed algorithm are given below.

1. Initialize the swarm with uniformly distributed random numbers. Each particle is taken as a pair

of real-valued vectors, (x_i, σ_i). The x_i's give the i^th particle of the swarm and σ_i's the associated strategy parameters.

2. Update velocity and position vector using equation (1) and (2).
3. Calculate the fitness value, fitness_i.
4. If (fitness_i < Pbest_i) then set Pbest_i = fitness_i. If (Pbest_i < gbest) then set gbest = Pbest_i.
5. If (U (0, 1) < 1/d) then

   Apply mutation to *Pbest* using equation (3).

   Evaluate the fitness value, fitness_i.

   If (fitness_i < Pbest_i) then set Pbest_i = fitness_i.

   If (Pbest_i < gbest) then set gbest = Pbest_i.

   Here U (0, 1) is the uniformly distributed random number in the interval (0, 1). d is the dimension.
6. Repeat the loop until stopping criteria is reached.

The algorithmic steps of PSO with *gbest* mutation is the same as the above, just Beta distribution mutating the *gbest* instead of Pbest.

4. Experimental Settings

Experimental settings for BPSO, AMPSO1, AMPSO2 and EP

	BPSO	AMPSO I and II	EP
Population size is taken 30 for all algorithms			
w (linearly decreasing)	(0.9-0.4)	(0.9-0.4)	NA
c1, c2	1.8	1.8	NA
Mutation Probability	NA	Beta Distribution	Gaussian Distribution
Computer settings	All experiments are conducted on a PIV PC using DEV C++		
Maximum iterations for all algorithms taken as 3000			

A total of 30 runs for each experimental setting were conducted and the average fitness of the best solutions throughout the run was recorded.

4.1. Benchmark problems and numerical results

The proposed versions are tested on 12 classical benchmark problems (unconstrained) given in Table 1 [5]. Functions f1, f3, f5, f6, f7, f8 are highly multimodal; f2, f4, f10, f11 are unimodal; 'f2' is unimodal function but when the dimensions are increased the convergence rate becomes slow; f12 is a step function which is discontinuous and has one minimum;f9 is a nosiy function. All the problems are scalable in order to check the efficiency of the proposed algorithms for higher dimensions. Since AMPSO contains the features of both PSO and EP we compared its performance with basic PSO (BPSO), EP and some refined versions of PSO and EP. In Table 2, we show comparison of AMPSO1 and AMPSO2 with BPSO and EP. In Table 3, we compare the performance of proposed algorithms with modified versions of PSO having Gaussian [6] and Cauchy mutation [7]. Table 4, gives the comparison with EP having adaptive Gaussian and Cauchy mutations [8]. Numerical results given in Table 2, show that AMPSO2 gives the best values in terms of average fitness function value for all the test functions except f5, where EP gave a slightly better performance. Here we would like to mention that results given in Table 2 are strictly with respect to the experimental settings given in Section 4.

Table 3, gives only four test problems because we were able to locate only four problems that have been solved by the other mentioned versions of PSO. Here also it is evident from the table that AMPSO2 gave best performance for all the 4 test problems. Table 4, gives the comparison of AMPSO versions with two versions of EP namely FEP and CEP. FEP uses self adaptive Cauchy mutation and CEP is the classical EP using self adaptive Gaussian mutation. The difference in the results of EP given in Table 2 and CEP given in Table 5 is because of different experimental settings (please see [8] for experimental settings of CEP and FEP). Despite different experimental settings, it can be observed from Table 5, that AMPSO2 gave a superior performance in 7 out of 12 problems whereas FEP gave better results in 4 out of 12 test problems. In case of f12, except for CEP, all the algorithms converged to 0. Figures 1 - 4 show the performance of some selected benchmark problems.

4.2. Real life problems and numerical results

Three constrained optimization problems common in the field of electrical engineering are taken: Dynamic Power and Static power scheduling problem [9] and electric network optimization problem [10]. Constraints are handled by Penalty method [4]. For the comparison of real life problems, we have only considered only BPSO and EP besides AMPSO1 and AMPSO2, because as mentioned earlier the other algorithms cited in this paper have not been used to solve the same set of constrained real life problems. From the numerical results no algorithm can be called a clear winner. For the first problem, AMPSO2 gave the best performance followed AMPSO1. In the second problem, AMPSO1 gave the best results; in second place we got AMPSO2. However, in the third problem EP emerged as a winner followed by AMPSO2 with a marginal difference which in turn was followed by AMPSO1 and finally by BPSO.

Table 1. Numerical Benchmark Problems. All problems are of dimension 30

Function	Range	Optimum
$f_1(x)=\sum_{i=1}^{n}(x_i^2-10\cos(2\pi x_i)+10)$	[-5.12,5.12]	0
$f_2(x)=\sum_{i=1}^{n}x_i^2$	[-5.12,5.12]	0
$f_3(x)=\frac{1}{4000}\sum_{i=0}^{n-1}x_i^2-\sum_{i=0}^{n-1}\cos(\frac{x_i}{\sqrt{i+1}})+1$	[-600,600]	0
$f_4(x)=\sum_{i=0}^{n-1}100(x_{i+1}-x_i^2)^2+(x_i-1)^2$	[-30,30]	0
$f_5(x)=-\sum_{i=1}^{n}x_i\sin(\sqrt{\lvert x_i\rvert})$	[-500,500]	-418.9829*n
$f_6(x)=(0.1)\{\sin^2(3\pi x_1)+\sum_{i=1}^{n-1}((x_i-1)^2(1+\sin^2(3\pi x_{i+1})))$ $+(x_n-1)(1+\sin^2(2\pi x_n))\}+\sum_{i=0}^{n-1}u(x_i,5,100,4)$	[-50,50]	-1.1428
$f_7(x)=\frac{\pi}{n}\{10\sin^2(\pi y_1)+\sum_{i=1}^{n-1}(y_i-1)^2[1+10\sin^2(y_{i+1}\pi)]\ +(y_n-1)^2\}+\sum_{i=1}^{n}u(x_i,10,100,4)$	[-50,50]	0
$f_8(x)=20+e-20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2})\ -\exp(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i))$	[-32,32]	0
$f_9(x)=(\sum_{i=0}^{n-1}(i+1)x_i^4)+rand[0,1]$	[-1.28,1.28]	0
$f_{10}(x)=\max\lvert x_i\rvert,\quad 0\le i<n$	[-100,100]	0
$f_{11}(x)=\sum_{i=0}^{n-1}\lvert x_i\rvert+\prod_{i=0}^{n-1}\lvert x_i\rvert$	[-10,10]	0
$f_{12}(x)=\sum_{i=0}^{n-1}\lfloor x_i+1/2\rfloor^2$	[-100,100]	0

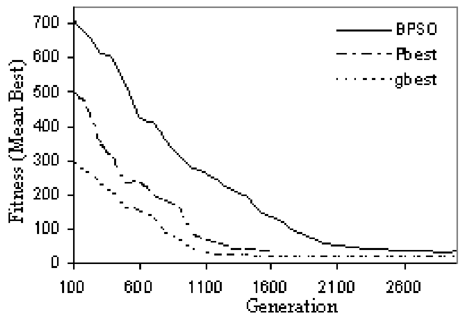

Figure 1. Performance for function f1.

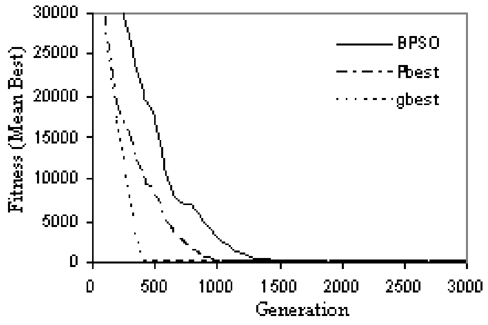

Figure 2. Performance for function f2

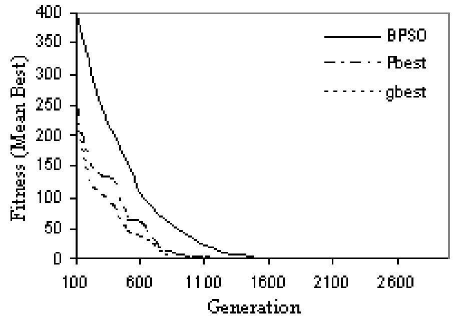

Figure 3. Performance for function f3

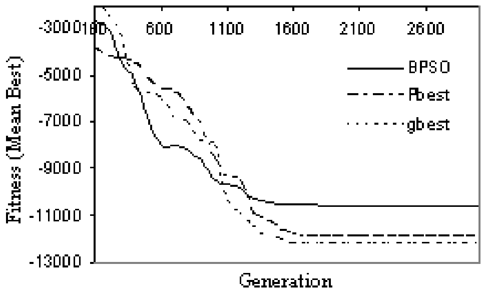

Figure 4. Performance for function f5

Table 2. Numerical results of proposed algorithms in comparison with BPSO and EP

Funct	BPSO	AMPSO1	AMPSO2	EP
f1	37.819 (7.455)	31.838 (6.004)	**18.307** (2.658)	184.097 (20.483)
f2	3.542e-16(4.26e-16)	2.320e-36 (4.573e-36)	**4.343e-40** (7.930e-40)	25.257(5.714)
f3	0.018(0.023)	1.626-19(9.812e-09)	**8.673e-20**(3.160e-20)	1.073(0.025)
f4	81.273(41.218)	29.137(25.014)	**24.171**(16.361)	157.385(154.873)
f5	-10652.33(663.174)	-11937.801 (383.462)	-12214.17(212.29)	**-12297.4**(676.107)
f6	-1.138 (0.005)	-1.139 (0.024)	**-1.144** (0.010)	-0.986(0.185)
f7	0.020(0.052)	**5.505e-13**(1.757e-12)	**5.505e-13**(1.757e-12)	14.76(5.046)
f8	1.02e-08(1.90e-08)	5.61e-17(1.046e-17)	**3.187e-17**(3.913e-13)	10.30(1.053)
f9	0.508(0.2508)	0.4670(0.313)	**0.416**(0.220)	5.387(1.711)
f10	5.357(3.204)	0.155(0.110)	**0.147**(0.131)	9.919(1.301)
f11	2.06e-11(5.853e-12)	4.55e-16 (1.82e-15)	**1.392e-17**(3.438e-17)	32.894(4.478)
f12	0.05(0.217)	**0.000**(0.000)	**0.000** (0.000)	1.533 (2.276)

Table 3. Numerical results of proposed algorithms in comparison with CPSO and GMPSO.

Funct	AMPSO1	AMPSO2	CPSO [7]	GMPSO [6]
f1	31.838(6.004)	**18.307**(2.658)	69.050(16.419)	26.266(3.258)
f2	2.320e-36(4.573e-36)	**4.343e-40**(7.930e-40)	2.36e-26(1.5e-25)	2.65e-15(2.56e-14)
f4	29.137(25.014)	**24.171**(16.361)	29.084(31.460)	27.932(26.115)
f8	5.616e-17(1.046e-17)	**3.187e-17**(3.913e-13)	5.651(1.333)	6.75e-12(3.21e-12)

Table 4. Numerical results of proposed algorithms in comparison with FEP and CEP

Funct	AMPSO1	AMPSO2	FEP[8]	CEP[8]
f1	31.838(6.004)	18.307(2.658)	**4.6e-2**(1.2e-2)	89.0(23.1)
f2	2.32e-36(4.57e-36)	**4.34e-40** (7.93e-40)	5.7e-4(1.3e-4)	2.2e-4(5.9e-4)
f3	1.626e-19(9.812e-09)	**8.673e-20** (3.160e-20)	1.6e-22.(2e-10)	8.6e-20.12
f4	29.137(25.014)	24.171(16.361)	**5.06**(5.87)	6.17(13.61)
f5	-11937.80(383.46)	-12214.17(212.29)	**-12554.5**(52.6)	-7917.1(634.5)
f6	-1.139 (0.024)	**-1.144**(0.010)	1.6e-4(7.3e-5)	1.4(3.7)
f7	**5.505e-13**(1.757e-12)	**5.505e-13**(1.75e-12)	9.2e-6(3.6e10)	1.76(2.4)
f8	5.616e-17(1.046e-17)	**3.187e-17**(3.91e-13)	1.8e-2(2.1e-3)	9.2(2.8)
f9	0.467(0.313)	0.4168(0.22)	**7.6e-3**(2.6e-3)	1.8e-2(6.4e-3)
f10	0.155(0.110)	**0.147**(0.131)	0.3(0.5)	2.0(1.2)
f11	4.55e-16(1.821e-15)	**1.392e-17**(3.438e-17)	8.1e-3(7.7e-4)	2.6e-3(1.7e-4)
f12	**0.000**(0.000)	**0.000**(0.000)	**0.000**(0.000)	577.76(1125.7)

Table 5. Numerical results of Real Life problems

Dynamic Power Scheduling problem				
Algorithm	Best	Average	Worst	Std
BPSO	666.6352	717.552687	802.85215	25.031686
EP	685.444	705.118	736.237	13.7668
AMPSO1	665.3734	698.83783	723.0659	20.174152
AMPSO2	**664.45585**	700.761712	717.09795	16.833555
Static Power Scheduling				
Algorithm	Best	Average	Worst	Std
BPSO	5061.01219	5154.529748	5221.47662	42.762777
EP	5078.47	5266.97	5432.6	55.9326
AMPSO1	**5051.31152**	5135.814972	5195.84009	36.365148
AMPSO2	5055.06909	5149.44626	5203.13869	35.75362
Electric Network Optimization				
Algorithm	Best	Average	Worst	Std
BPSO	8873.01753	9084.32555	9307.9852	141.867249
EP	**8860.69**	8998.56	9115.98	106.282
AMPSO1	8867.42125	9039.478666	9308.63886	130.941717
AMPSO2	8866.28340	9031.81355	9294.83343	125.210766

5. Conclusion

Two new variants viz. AMPSO1 and AMPSO2 with self adaptive mutation probability are proposed. The novelty of the approach is the use of Beta distribution, which to the best of our knowledge, has not been used for self adaptive mutation. The numerical results show that the performance of Beta probability distribution is at par with Gaussian and Cauchy distributions and is in fact better in some of the cases under the given experimental settings. We would like to add that the performance comparison given in Table 3 and Table 4 are not very fair because of different experimental settings. However, we can say that under the experimental settings taken in this paper AMPSO2 gives good results for unconstrained test problems. For constrained real life problems the picture is not very clear and we are doing further investigations. In future we plan to compare different algorithms under similar parameter settings, in order to give a fair chance to every algorithm.

References

[1] R.C. Eberhart, Y. Shi, "Particle Swarm Optimization: developments, Applications and Resources," *IEEE Int. Conference on Evolutionary Computation*, 2001, pg. 81 -86.

[2] H. Liu, A. Abraham and W. Zhang, "A Fuzzy Adaptive Turbulent Particle Swarm Optimization", International Journal of Innovative Computing and Applications, Volume 1, Issue 1, 2007, pp. 39-47.

[3] D. B. Fogel, "Evolutionary Computation: Toward a new Philosophy of Machine Intelligence", IEEE press, 1995.

[4] A.P. Engelbrecht, "*Fundamentals of Computational Swarm Intelligence*," John Wiley & Sons Ltd, 2005.

[5] J. Vesterstrom, R. Thomsen, "A Comparative study of Differential Evolution, Particle Swarm optimization, and Evolutionary Algorithms on Numerical Benchmark Problems," in *Proc. IEEE Congr. Evolutionary Computation*, Portland, OR, Jun. 20 – 23, 2004, pg. 1980 – 1987.

[6] M. Pant, T. Radha and V. P. Singh, "A New Diversity Based Particle Swarm Optimization using Gaussian Mutation", Int. J. of Mathematical Modeling, Simulation and Applications, 1(1), pp. 47-60, 2008.

[7] A. Stacey, M. Jancic, and I. Grundy. Particle Swarm Optimization with Mutation. In *Proceedings of the IEEE Congress on Evolutionary Computation*, (2), pp. 1425 – 1430, 2003.

[8] X. Yao and Y. Liu, "Fast Evolutionary Programming", In Proc. of 5th Ann. Conf. on Evoluationary Programming, pp. 451 – 460, 1996.

[9] M. C. B-Biggs, "A Numerical Comparison between Two Approaches to the Nonlinear Programming Problem", In: L. C. W. Dixon and G. P. Szego, eds., Towards Global Optimization 2, Amsterdam, Holland: North Holland Publishing Company, pp. 293 – 312, 1978.

[10] D. M. Himmelblau, "Applied Non-Linear programming", New York: McGraw-Hill, 1972.