Journal of Manufacturing Engineering, September 2015, Vol. 10, Issue. 4, pp 223-228

AN EFFICIENT ALGORITHM FOR NUMERICAL FUNCTION OPTIMIZATION: PARTICLE SWARM OPTIMIZATION

***Selva Kumar G and Selvaraj M**

Department of Mechanical Engineering, SSN College of Engineering, Chennai – 603 110, Tamil Nadu, India

ABSTRACT

The Particle swarm optimization (PSO) is one of the evolutionary computation techniques that can be applied to a wide range of real world problems. In this paper, PSO algorithm is numerically illustrated with a one dimensional unconstrained problem. The efficiency and robustness of this algorithm is demonstrated by applying it to the benchmark functions namely Goldstein- Price and De jong functions and the results were compared with those obtained using other optimization algorithms.Matlab code is created and used to solve the benchmark functions.

Keywords: *Particle swarm optimization (PSO), Function optimization, Goldstein- Price and De jong function*

1. Introduction

Particle swarm optimization (PSO) is a population based self-adaptive search optimization technique that was proposed by Eberhart and Kennedy in 1995, where the population is referred to as a swarm [1]. The PSO is based on simulations of social behaviors such as fish in a school, birds in a flock, etc. A swarm in PSO consists of a number of particles. Each particle represents a potential solution of the optimization task. All of the particles iteratively discover a probable solution. Each particle moves to a new position according to the new velocity and the previous positions of the particle. The PSO has ability of fast convergence to local and/or global optimal solutions over a small number of generations. [1, 3]

The advantage of PSO approach over traditional techniques is its robustness and flexibility in solving real-world problems featuring non-differentiability, high dimension, multiple optima and non-linearity. These properties make swarm intelligence a successful design paradigm for algorithms that deal with increasingly complex problems [2].

It has been reported [4, 5, 6] that the PSO technique is superior in comparison with other evolutionary computation techniques such as genetic algorithm (GA), simulated annealing algorithm (SA), Tabu search algorithm (TS), memetic algorithm (MA) and ants colony algorithm (ACO). PSO has found applications in a lot of areas such as constrained optimization problems, Min-max problems, Multi-objective optimization problems and Dynamic tracking. It has also been applied to evolve weights and structure of neural networks, analyze human tremor, register 3D-to-3D biomedical image, play games, control reactive power and voltage, etc. Generally speaking, PSO can be applied to solve most optimization problems and problems that can be converted to optimization problems [7].

2. Particle Swarm Optimization Algorithm

The PSO algorithm is simple in concept, easy to implement and computationally efficient. The original algorithm for implementing PSO is as follows [7]

i. Initialize a population of particles with random positions and velocities on *D* dimensions in the problem space.
ii. For each particle, evaluate the desired optimization fitness function in *D* variables.
iii. Compare particle's fitness evaluation with its *pbest*. If current value is better than *pbest*, then set *pbest* equal to the current value, and *Pi* equals to the current location *Xi* in *D*-dimensional space.
iv. Identify the particle in the neighborhood with the best success so far, and assign its index to the variable *gbest.*
v. Update the velocity and position of the particle according to the following Equations [8]

**Corresponding Author - E- mail: selvakumaratju@gmail.com*

$$V_{i+1} = w\, V_i + c_1 r_1 (pbest_i - X_i) + c_2 r_2 (gbest_i - X_i) \quad (1)$$
$$X_{i+1} = X_i + V_{i+1} \quad (2)$$

vi. Loop to step (ii) until a criterion is met, usually a sufficiently good fitness or a maximum number of iterations.

2.1 Description of the velocity and position update equations

Equation (1) calculates a new velocity (V_{i+1}) for each particle (potential solution) based on its previous velocity, the best location it has achieved (*pbest*) so far, and the global best location (*gbest*), the population has achieved. Equation (2) updates individual particle's position (X_i) in the solution hyperspace. The two random numbers r_1 and r_2 in Eq. (1) are independently generated in the range [0,1]. The acceleration constants c_1 and c_2 in equation (1) represent the weighting of the stochastic acceleration terms that pull each particle towards *pbest* and *gbest* positions. c_1 represents the confidence the particle has in itself (cognitive parameter) and c_2 represents the confidence the particle has in swarm (social parameter). Thus, adjustment of these constants changes the amount of tension in the system. The inertia weight w plays an important role in the PSO convergence behavior since it is employed to control the exploration abilities of the swarm.

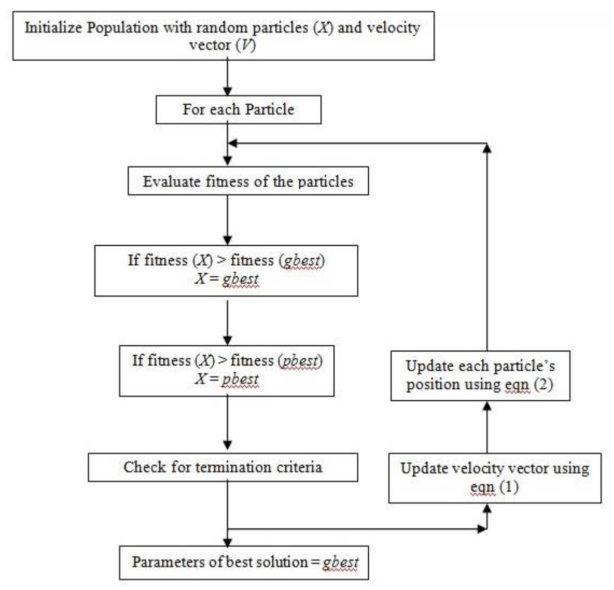
Initialize Population with random particles (X) and velocity vector (V)
For each Particle
Evaluate fitness of the particles
If fitness (X) > fitness (gbest) X = gbest
If fitness (X) > fitness (pbest) X = pbest
Check for termination criteria
Parameters of best solution = gbest
Update each particle's position using eqn (2)
Update velocity vector using eqn (1)

Fig.1 A general flow chart of PSO algorithm [6]

To achieve the dimensional consistency of Equations (1) and (2), the dimension of the term *cr* in equation (1) could be taken as (time)$^{-2}$. This way, the second and the third terms in equation (1) assume the dimension of acceleration. To get the correct dimension of velocity, as required by the left hand side, one needs to multiply them by Δt, the time step, which becomes unity in the present case, denoting changes from iteration *i* to *i + 1*. Similarly, the second term in equation (2) assumes the correct dimension when taken as $V_{i+1}\,\Delta t$. However, the present form results through the implicit assumption that Δt equals 1. [4] In short, the whole concept of PSO can be stated as "A population consisting *N* particles, each particles has *D* variables (dimension) which have their own ranges for each value, velocities and positions are updated every iteration until maximum iteration is reached". Figure 1 describes the scenario mentioned above.

3. PSO Parameter Control

In PSO [1, 5], the following are the parameters that need to be tuned. Here is a list of the parameters and their typical values:

a) The number of particles or swarm size

The optimum swarm size is problem dependent. However the typical range is 10 – 30. In general, 10 particles are large enough to get good results. In some special cases, 100 or 200 particles are employed to get optimum solution.

b) Dimension of particles

Particles dimension is determined by the problem to be optimized.

Example:

1. Minimize $f(x) = 9x^3 - 2x^2 - 6x + 21$

- One Dimensional Problem

2. Max $f(x) = 12x_1^3 + x_2^2 - 32x_1x_2 - 1245$ –

- Two Dimensional Problem

In the above two examples, f(x) is the fitness function and the number of variables associated with the fitness function is known as dimension of the particles.

c) Range of particles

This is also determined by the problem to be optimized.

d) Acceleration coefficients or Learning factors

c_1 and c_2 are usually equal to 2. However, other settings are also used in different papers. But usually c_1 equals to c_2 and ranges from [0, 4]

e) The stop condition

The maximum number of iterations the PSO executes. This stop condition depends on the problem to be optimized.

f) Inertia weight

An important aspect that determines the efficiency and accuracy of an optimization algorithm is the exploration–exploitation trade-off. Exploration is the ability of a search algorithm to explore different regions of the search space in order to locate a good optimum. Exploitation, on the other hand, is the ability to concentrate the search around a promising area in order to refine a candidate solution. A good optimization algorithm optimally balances these contradictory objectives. Within the PSO, these objectives are addressed by the velocity update equation.

The value of *w* is extremely important to ensure convergent behavior, and to optimally trade-off exploration and exploitation. Usually *w* varies from 0.9 to 0.4. To obtain guaranteed convergence for simple PSO [1] the condition shown below should be satisfied.

$$1 > w > \tfrac{1}{2}(c_1 + c_2) - 1 \geq 0 \qquad (3)$$

Generally, *w* is taken as 0.5

4.Computational Implementation of PSO

The implementation of PSO is illustrated with the following one dimensional problem.

Minimize $f = x^2 - 8x; \quad -10 \leq x \leq 10$ (4)

Step 1: Create a population of N particles and calculate its fitness

The number of particles is chosen as three and all three particles initialized randomly within (-10, 10). The initial velocity for all the particles is assumed as zero. Then the fitness is evaluated using equation (4).

Table 1. First iteration

Iteration no.	Particle no.	*X*	*V*	fitness	*pbest*	r1	r2	Updated *V*	new *X*
1	1	**7.0000**	0.0000	-7.0000	7.0000	0.4868	0.3063	0.0000	7.0000
	2	-2.0000	0.0000	20.0000	-2.0000	0.4359	0.5085	9.1530	7.1530
	3	9.0000	0.0000	9.0000	9.0000	0.4468	0.5108	-2.0432	6.9568

gbest is in bold letters, $c_1 = c_2 = 2$ and $w = 0.5$

Step 2: Finding the gbest and pbest

The fitness values of above three particles are compared and the particle whose fitness value is the lowest (since it is minimization problem) is selected. The position of the lowest fitness particle in the solution space is known as *gbest* value. In table 1, particle 1 has the lowest fitness value and its position in the solution space is 7. Therefore *gbest* value for this initial or first iteration is 7(shaded in yellow color).Initially, the *pbest* values for all the particles will be the same as the current *X* values. In case of maximization problem, the position of the particle whose fitness value is highest will be taken as *gbest.*

Step3: Update velocity and position of the particles using equation 1 and 2

The position and velocity of the particles are updated and tabulated in table-1. The detailed computation is illustrated below.

Updated *V* for particle 1 = V_1 =
0.5*0 + {2*0.4868*(7-7)} + {2*0.3063*(7-7)} = 0
Updated *V* for particle2 = V_1 =
0.5*0+{2*0.4359*(-2-(-2)}+{2*0.5085*(7-(-2))}= 9.153
Updated *V* for particle 3 = V_1 =
0.5*0+{2*0.4468*(9-9)} + {2*0.5108*(7-9)} = -2.0432

Table 2. Second and third iteration

Iteration no	Particle. no	*X*	*V*	fitness	*pbest*	r1	r2	Updated *V*	new *X*
2	1	7.0000	0.0000	-7.0000	7.0000	0.8176	0.3786	-0.0327	6.9673
	2	7.1530	9.1530	-6.0586	7.1530	0.7948	0.8116	4.2580	11.4110
	3	**6.9568**	-2.0432	-7.2573	6.9568	0.6443	0.5238	-1.0216	5.9352
3	1	6.9673	-0.0327	-7.1952	6.9673	0.3507	0.5502	-1.1521	5.8152
	2	11.4110	4.2580	38.9233	7.1530	0.9390	0.6225	-12.685	-1.2739
	3	**5.9352**	-1.0216	-12.255	5.9352	0.8759	0.5870	-0.5108	5.4244

gbest is *in bold letters, $c_1 = c_2 = 2$* and w = 0.5

New X for Particle 1 = X_1 = add column X and updated V column in table.1= 7 + 0 = 7
New X for Particle 2 = X_1 = -2 + 9.153 = 7.1530
New X for Particle 3 = X_1 = 9 + (-2.0432) = 6.9568
Similar to the first iteration, iteration 2 and 3 are computed and presented in table 2

Finding *pbest* for iteration number: 3, for particle-1:

The fitness values of particle 1 during iteration 1, 2 & 3 are compared and the lowest fitness value is selected. The position (X value) which corresponds to the lowest fitness value is *pbest* value for particle 1. Similar manner *pbest* values were computed for particle 2 and 3 and presented in table-3.

Table 3. *pbest* values of particles

Iteration no	Particle1		Particle2		Particle3	
	fitness	X	fitness	X	fitness	X
1	-7	7	20	-2	9	9
2	-7	7	-6.0586	**7.1530**	-7.2573	6.9568
3	-7.1952	**6.9673**	38.9233	11.4110	-12.255	**5.9352**

Bold letters represents *pbest* value for iteration no:3

Step: 4 if number of iterations reaches maximum iteration (100), then end.

The present *gbest* value will give the optimized value.

5. PSO Simulation Results

The coding for PSO is done with Matlab 7.6 software for minimization of a one dimensional function and a two dimensional function (Goldstein- Price function). It is also applied to a two dimensional maximization function (De jong function) and the simulated results were presented and discussed below.

5.1 One Dimensional function

The matlab coding is created based on PSO concept to minimize f = x^2-8x; $-10 \le x \le 10$.
The parameters considered to solve this function are listed below.
Number of particles or swarm size = 3
The learning factors $c_1 = c_2 = 2$
Inertia weight w = 0.5
Termination criteria: Total number of iterations = 50

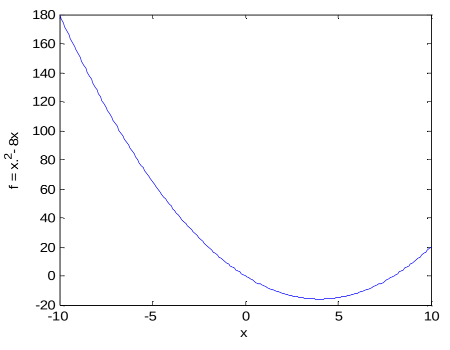

Fig.2. The one dimensional function (equation no:4) to be optimized

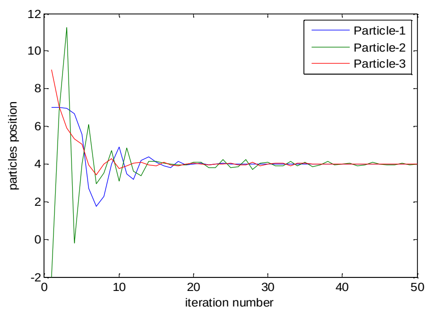

Fig.3. Position of all the particles while solving one dimensional function (equation 4)

Fig.2 illustrates the one dimensional function in the solution space. The positions of all the particles are shown in the fig.3 against its iteration. The *gbest* value and the fitness value corresponding to the *gbest* are shown in fig.4. The particles found the optimum solution (minimum value) at the 8th iteration. That means the convergence of the solution beginning at eighth iteration. The optimal solution and the minimum fitness value are x = 4 and f_{min} = -16 respectively.

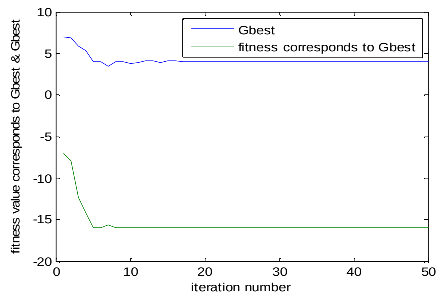

Fig.4 *gbest* and the fitness value corresponds to *gbest* for one dimensional function

5.2 Goldstein- Price function

The function is minimize $F = [1 + (x_1 + x_2 + 1)^2 (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] * [30 + (2x_1 - 3x_2)^2 (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$

In the interval: $[-2, 2]$

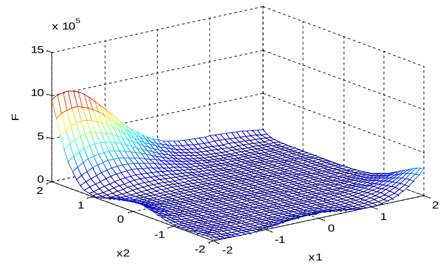

Fig.5 Goldstein- Price function

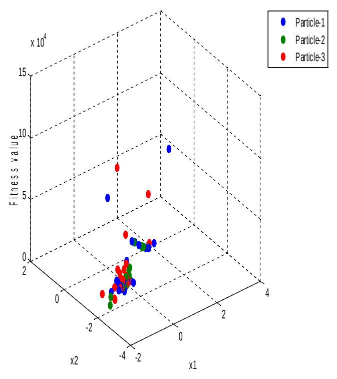

Fig.6. Position of all particles in 3-Dimensional solution space when solving Goldstein- Price function

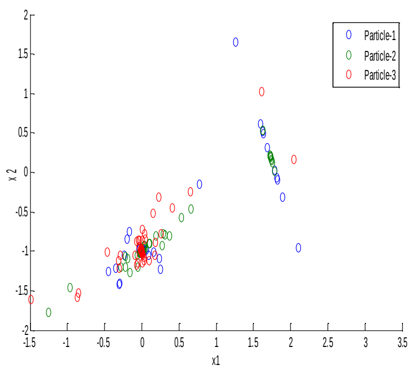

Fig.7. Position of all particles in 2-Dimensional solution space - Goldstein- Price function

The control parameters setting to solve Goldstein- Price function are:
Number of particles or swarm size = 3
The learning factors $c_1 = c_2 = 1.5$
Inertia weight $w = 0.6$
Termination criteria: Total number of iterations = 75

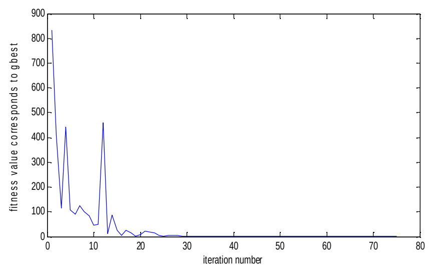

Fig.8. fitness value corresponding to the gbest value for Goldstein- Price function

The Goldstein- Price function is shown in Fig.5. As the iteration proceeds, the positions of the particles are changing and converging to the optimal solution (position). The changes of positions of the particle are illustrated in Fig.6 and Fig.7. The variation of the fitness value corresponding to the *gbest* is shown in Fig.8. From this figure it is known that the particles found optimum position at 29th iteration onwards. During simulation, it is noted that all three particles found the optimum position (solution) at 74th iteration. The optimum solution for this function found by PSO simulation is $X = (0, -0.99999)$ and the minimum value of the function is $F_{min} = 3$.

5.3 De jong function

The De jong function is max $F = 3905.93 - 100(x_1^2 - x_2^2) - (1 - x_1)^2$; In the interval: $[-2.048, 2.048]$

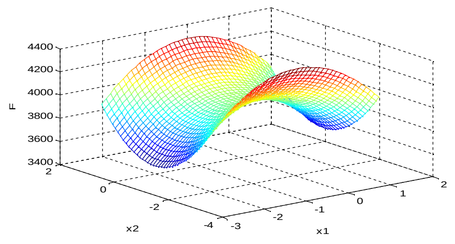

Fig.9. De jong function

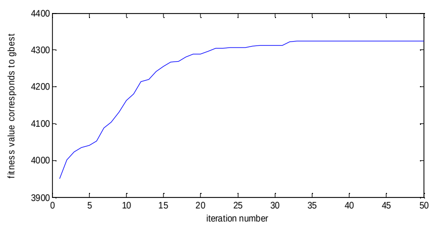

Fig.10. Fitness value corresponding to the gbest value for De jong function

The PSO parameters selected to solve De jong function are listed below:
Number of particles or swarm size = 3
The learning factors $c_1 = c_2 = 1.5$
Inertia weight $w = 0.6$
Termination criteria: Total number of iterations = 50

The De jong function is illustrated in Fig.9. The variation of the fitness value corresponding to the *gbest* with respect to the iteration number is shown in Fig.10. From this figure, it is understood that the 33 iteration onwards the *gbest* particles found the optimum position (solution). The global optimum for De jong function is computed as X= (0.02958, -2.04744); F_{max} = 4324.104.

5.4 Results Comparison

Table 4. Comparison of results

Function	**GA**	**ANTS**	**Bees Algorithm**	**Global optimum by GA, ANTS & Bees**	**PSO**	**Global optimum by PSO**
	mean no. of evaluations	**mean no. of evaluations**	**mean no. of evaluations**		**no. of iterations**	
Goldstein price	5662	5330	999	X(0,-1) $F_{min}=3$	(29)75	x = (0, -0.99999) $F_{min} = 3$
De jong	10160	6000	868	X(1,1) $F_{max} = 3905.93$	(33)50	x = (0.02958, -2.04744) $F_{max} = 4324.104$.

The values within the bracket indicate that the iteration number at which PSO starts giving the optimum solution. The results obtained from the Genetic, Ants and Bees algorithm [10] are compared with PSO and the same is presented in table-4. The PSO gives global optimum solution for Goldstein price function with less number of iterations compared to GA, Ants and Bees algorithm. The second test function was De Jong's, for which the PSO Algorithm found better global optimum solution as compared to other algorithms with minimum number of iterations. Hence, it is concluded that the PSO performs better than other algorithms as tabulated in table-4.

6. Conclusion

This paper has presented the computational implementation of PSO with a numerical illustration. Simulation results on Goldstein price and De jong functions show that the proposed algorithm has remarkable robustness, producing a 100% success rate. The algorithm converged to the maximum or minimum without becoming trapped at local optima. The PSO algorithm generally outperformed other techniques that were compared with it in terms of speed of optimization and accuracy of the results obtained. Future research will focus on the study of the variations in PSO algorithm such as PSO algorithm with multiple swarms, Hybrid algorithms, Multi-start methods etc,.

Reference

1. *Andries P Engelbrecht (2007), "Computational Intelligence- An introduction", Second edition, John wiley & Sons, Ltd, 285-358.*
2. *Christian Blum and Daniel Merkle (Eds.) (2008), "Swarm Intelligence- Introduction and Applications", Springer-Verlag Berlin Heidelberg, 43-86.*
3. *Sheng-Ta Hsieh Tsung-Ying Sun Chan-Cheng Liu Shang-Jeng Tsai (2009), "Efficient Population Utilization Strategy for Particle Swarm Optimizer", IEEE Transactions on systems, man, and cybernetics—Part B: cybernetics, Vol. 39, No. 2, 444-456.*
4. *Lee T S Ting T O Lin Y J Than Htay (2007), "A particle swarm approach for grinding process optimization analysis", Int J Adv Manuf Technol, Vol. 33, 1128–1135. DOI 10.1007/s00170-006-0538-y.*
5. *Saravanan R Siva Sankar R Asokan P Vijayakumar K Prabhaharan G (2005), "Optimization of cutting conditions during continuous finished profile machining using non-traditional techniques", Int J Adv Manuf Technol, Vol. 26, 30–40. DOI 10.1007/s00170-003-1938-x.*
6. *Asokan P Baskar N Babu K Prabhaharan G Saravanan R (2005), "Optimization of Surface Grinding Operations Using Particle Swarm Optimization Technique", ASME Journal of Manufacturing Science and Engineering, Vol. 127 / 885. DOI: 10.1115/1.2037085.*
7. *Yuhui Shi (2004), "Particle Swarm Optimization", IEEE Neural Networks Society, 8-13.*
8. *Venkata Rao R Pawar P J (2010), "Parameter optimization of a multi-pass milling process using non-traditional optimization algorithms", Applied Soft Computing, Vol. 10, 445–456.*
9. *Singiresu S Rao (2009), "Engineering Optimization: Theory and Practice", Fourth Edition, John Wiley & Sons, Inc., pp 708-714.*
10. *Pham D T Ghanbarzadeh A Koç E Otri S Rahim S Zaidi M (2006), "The Bees Algorithm – A Novel Tool for Complex Optimisation Problems", IPROMS Cardiff University, England, 454–459.*