Velocity Restriction-Based Improvised Particle Swarm Optimization Algorithm

H. Mouna, M. S. Mukhil Azhagan, M. N. Radhika, V. Mekaladevi and M. Nirmala Devi

Abstract The Particle Swarm Optimization (PSO) Algorithm attempts on the use of an improved range for inertia weight, social, and cognitive factors utilizing the Pareto principle. The function exhibits better convergence and search efficiency than PSO algorithms that use conventional linearly varying or exponentially varying inertia weights. It also presents a technique to intelligently navigate the search space around the obtained optima and looks for better optima if available and continue converging with the new values using a velocity restriction factor based on the Pareto principle. The improvised algorithm searches the neighborhood of the global optima while maintaining frequent resets in the position of some particles in the form of a mutation based on its escape probability. The results have been compared and tabulated against popular PSO with conventional weights and it has been shown that the introduced PSO performs much better on various benchmark functions.

Keywords Swarm intelligence · Global optimization · Intelligent search
Inertia weight · Velocity restriction · Pareto principle

H. Mouna (✉) · M. S. Mukhil Azhagan · M. N. Radhika · V. Mekaladevi · M. Nirmala Devi
Department of Electronics and Communication Engineering,
Amrita University, Coimbatore, India
e-mail: mouna.harikumar@gmail.com

M. S. Mukhil Azhagan
e-mail: mukhil@outlook.com

M. N. Radhika
e-mail: radhikamnarayan@gmail.com

V. Mekaladevi
e-mail: v_mekaladevi@cb.amrita.edu

M. Nirmala Devi
e-mail: m_nirmala@cb.amrita.edu

K. Saeed et al. (eds.), *Progress in Advanced Computing and Intelligent Engineering*,
Advances in Intelligent Systems and Computing 564,
https://doi.org/10.1007/978-981-10-6875-1_34

1 Introduction

Particle Swarm Optimization (PSO) is a computational algorithm that simulates natural swarm behavior most notably found in certain species of birds, fish, and bees. Kennedy and Eberhart introduced this algorithm in 1995 [1, 2]. The particles in a swarm, analogous to flock of birds or fish in a school, move around the search space in a predefined pattern that is close to the natural search of a bird for its food which is the global optima. The communication between members of a flock is also simulated and each particle in the swarm is aware of its own optima and the best optima among the swarm. Different particles explore in different velocities and from different positions, all of which are randomized to obtain results close to the natural order of swarming [3].

PSO being faster in terms of iterations and processing time has progressed rapidly in recent years and has been used for problems in Artificial intelligence, Material design, and various fields that require a quick optimization [4, 5]. Conventional PSO gets stuck at a local minimum [6]. Research has been focused on trying to accelerate the convergence speed and achieving a better accuracy [7–9]. Reported literature has shown either a linearly decreasing pattern alone or a combination of two different algorithms, popularly known as hybrid algorithms, which are comparatively slow on convergence [10–12]. The improvised algorithm attempted in this paper has a varying inertia weight based on the Pareto principle, thus enabling it to converge to a better value with more precision.

To achieve the above-mentioned superiority over the other algorithms, three techniques have been introduced. The first technique, the *velocity restriction* is based on the Pareto principle (or the 80-20 rule) [16, 17]. The second technique is on a mutation-based term called the *escape probability*, which allows for a way of doing extensive exploration which focuses on finding other local optima easily. The third technique uses an improved *inertia weight* [10], which will progressively converge the search toward the minima over higher iterations. The designer is allowed to set the cognitive and social parameters, so that the user has control over the neighbor of convergence, either toward the pbest or gbest depending on the requirement of the algorithm. At every iteration, there is a condition for mutation that is based on its *escape probability*, which is the number of times it moves out of the boundary. This serves a dual purpose of maintaining the position within the search space of interest and also mutates the position of the particle frequently. This frequency is controlled by the escape probability, which is in turn controlled by the initial velocity that is set by the designer. Combining the three techniques—*mutation*, *velocity restriction*, and the refined *inertia weight* (based on the Pareto principle) the algorithm has been made adaptive and intelligent to work with varied functions. Simulation results have been produced to show better convergence and precision.

2 Particle Swarm Optimization

The algorithm was first proposed by considering a swarm of particles of size N [1, 2]. The optimum position in the search space was found by these particles using their swarm intelligence. The conventional PSO uses five basic principles namely Quality principle, Proximity principle, Stability principle, Diverse response principle, and Adaptability principle [1]. At the start of the PSO, the number of variables D is initialized and the objective *function f* is specified. The required parameters such as population size, swarm size, total number of iterations *itmax*, cognitive factor $c1$, social factor $c2$, and inertia weight w are initialized. The independent variables are given their boundary conditions in which they can search for the best optimum position. The random values of position and velocity are initialized as the pbest values for each particle, and gbest value for the swarm. *pbest* is defined as the personal best of each particle and *gbest* is the global best of swarm. In the given search space, the new position value is found by each of the particles. The position and velocity are calculated using the Eqs. (1) and (2) given below.

$$v_i^D = w * v_{i-1}^D + c1 * rand1_i^d * (pbest_i^d - x_i^d) + c2 * rand2_i^d * (gbest^d - x_i^d) \quad (1)$$

$$x_i^d = x_{i-1}^d + v_i^d \quad (2)$$

Where v_i^D is the velocity of the current iteration for each argument, w is the inertia weight [10]. $rand1_i^d$ and $rand2_i^d$ are two random distributions that range between 0 and 1. x_i^d is the position of each particle that will be updated from its previous value. The new position value is compared and checked with its preceding value. If the new position value is found to be better than the preceding value, the pbest value is updated else the preceding value is retained. The best among the *pbest* of all the particles is taken, if that value is better than the preceding *gbest*, then it is replaced as the *gbest* value, else the older value is retained. Inertia weight significantly affects the accommodation between exploitation and exploration in the PSO process. Different variants of PSO can be obtained by changing parameters such as the cognitive and social factor, different inertia weights, swarm size, network topologies in PSO, etc. [13]. Hybridization and multi-objective are some of the variants. In hybridization, for example, Genetic Algorithm and PSO can be combined; GAs mutation technique can be combined with PSO to prevent PSO from getting stuck at local optima [14].

3 Improvised PSO

The difference between conventional PSO [1, 2] and the improvised PSO is that it has techniques for *mutation*, *velocityrestriction*, and *improvised ranges* for factors that affect PSO. This PSO works just like the conventional PSO except for the fact that it has a restricted search in different range of weights, and velocity restriction based on the Pareto principle. The Pareto principle states that 20% of the cause is responsible

for 80% of the outcome or vice versa, depending on the perspective. Applying this, the *inertia weight*, which is calculated using Eq. (3) is varied from values between 0.8 to 0 to cover 80% of the area which is a higher neighborhood of exploration, to find the rest 20% of the local optima after convergence has started.

$$w_i = wmax - \frac{(wmax - wmin)}{itmax} * it \tag{3}$$

Where w_i is the weight of each iteration, w_max is around 0.8 and w_min is around 0. Values taken in the trials are 0.7 and 0.1, respectively. *itmax* denotes the maximum number of iterations considered and *it* denotes the current iteration. Also, a technique of velocity restriction, calculated using the Eq. (4) that modifies the effect of the preceding velocity on the existing position, is included.

$$V_r = e^{\frac{-it}{k*itmax}} \tag{4}$$

Where Vr is the *velocity restriction* factor. *k* is a constant, whose value is taken as 4 for an optimum range. Equation (4) should be multiplied along with velocity during every iteration to restrict its boundary. It decreases exponentially and the speed can be modified by changing the value of *k* by the user depending on the need. Frequent mutations are performed to help the exploration process, determined by an escape probability. This escape probability is calculated using an algorithm that also prevents the particle to move out of the boundary, thus stabilizing the swarm search. The Pseudo code for the improvised PSO is given in Fig. 1.

PSEUDOCODE

Step a: Initialize the number of dimensions, all the required parameters (the size of population and swarm, total number of iterations *itmax*, cognitive and social factors *and inertia weights)*, and specify the objective function.
Step b: Calculate the *inertia weight* for each iteration using equation (3), within the range 0.8 - 0.0.
Step c: Calculate the *velocity restriction* parameter for each iteration using equation (4).
Step d: Define the boundary conditions where the particles search for optimum position.
Step e: Initialize random values for position and velocity for particles as *pbest* and *gbest*.
Step f: Find the next *gbest* and *pbest* values.
Step g:Find the new position value of the particle
Step h: Find the new velocity of each particle using *velocity restriction* technique using equation (1) and equation (4).
Step i: Each individual position should be updated using equation (2) and the new velocity.
Step j: Restrict each particle to the defined boundary using the mutation technique, utilizing *escape probability*.
Step k: Compare and check the new position value with the preceding value. If the new position value obtained is better than the preceding position value, go to *m*, else go to *n*.
Step l: Update the preceding values of *pbest* and *gbest* with the new best value obtained.
Step m: Go to *Step o*.
Step n: Retain the previous values of *pbest* and *gbest*.
Step o: If i<=*itmax*, i++, go to step *g* else go to *step p*.
Step p: Indicate the optimum value of *pbest* and *gbest*.

Fig. 1 Pseudo code for improvised PSO

4 Simulation Results and Analysis

The improvisation performed on Particle Swarm Optimization (PSO) was tested on benchmark functions and the results acquired are improved comparatively [10–12]. The Code was simulated in Matlab using a PC with core I3 4005u, 1.7GHz, and 4 GB RAM. Benchmark functions used are (a) Sphere function, which is continuous, unimodal, and has D local minima (b) Rastrigin function, which is multimodal and has several local minima (c) Rosenbrock function, also known as valley or banana function, is unimodal (d) Michalewicz function, which is multimodal and has D local minima and are usually referred as valleys and ridges (e) Shubert function, which has many local and global minimas. The results have been obtained for 220 iterations and over 30 independent trials with 100 particles. The search has been done over a search space of $[-5.12, 5.12]$ for (a), (b), (c), and (e) functions and $[-\pi, \pi]$ for (d), as given in [18]. Table 1 shows the 2D plot between mean function value of all particles and number of iterations of Sphere, Rastrigin, Rosenbrock, Michalewicz, and Shubert functions, respectively, including equations and 3D plots.

Table 1 shows plots for the benchmark functions, for (a), (b), (c) the y-axis for the 2D plot is indexed in powers of 10, as the expected values from mathematical calculation [18] are close to zero. For (d), (e) the y-axis is in real values. Table 2 shows the iteration at which the minimum value is obtained for each benchmark function. Combining 2D plots from Tables 1 and 2, various inferences can be made. In Table 1 for the Shubert function, which is multimodal, a large number of spikes can be seen, which denote various particles exploring other parts of the search space for potential global minima until the 220 iterations end, whereas the minima has been reached at around the 26th iteration in Table 2. This demonstrates the ability of the algorithm to explore exhaustively even after getting settled at the minima, due to the mutation factor introduced using *escape probability*. For the sphere function, which has a single minima, the algorithm tries to obtain the best possible value, from Table 3, it can be seen that the algorithm is precise up to 10^{-71} on an average. In such functions, the algorithm is able to choose exploitation over exploration thus leading to much better values as proven in Table 4.

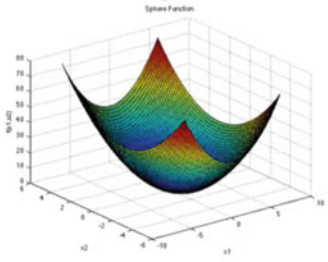
In Table 3, $X1$ denotes the position in the first dimension. $X2$ denotes the position in the second dimension and *fgbest* is the fitness value that is dependent on $X1$ and $X2$ as described by equations in Table 1. Table 3 shows the values of $X1$, $X2$, and *fgbest* using the Eqs. (5), (6), (7), (8), and (9) for various benchmark functions.

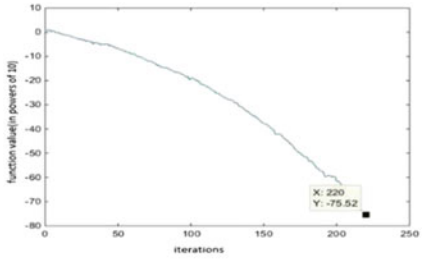
In Table 3, the best-fit column denotes the best value obtained. This value is the global minimum that has been obtained through the algorithm over the trials. The mean and standard deviation denote the algorithms variation from the best-fit value. It has been shown that the algorithm performs with minimal variance for most of the benchmark functions when the fgbest value is concerned. Incase of $X1$ and $X2$, the average and the standard deviation are close to the expected values, except in case of (e), the Shubert function, as the function exhibits the same minimum value at multiple points in the given search space. This discrepancy is expected out of such a function and can be verified mathematically [18].

Table 4 shows the fgbest values using the introduced algorithm, i.e., the fitness value and it is compared with [11, 12, 15]. The proposed algorithm exhibits much

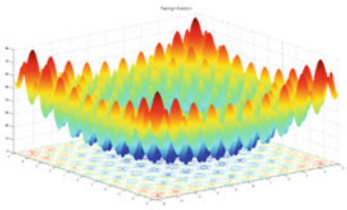
Table 1 Equations and rate of convergence plots for various benchmark functions

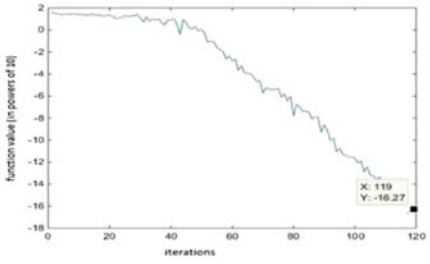
$$SphereFunction : f = \sum_{i=1}^{D} x_i^2 \tag{5}$$



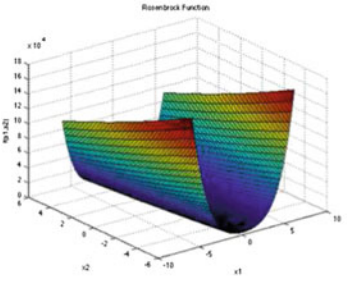


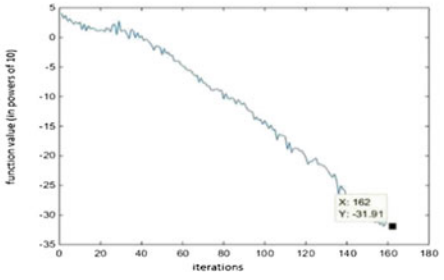
$$RastriginFunction : f = \sum_{i=1}^{D} x_i^2 - 10cos(2\pi i) + 10 \tag{6}$$





$$RosenbrockFunction : f = \sum_{i=1}^{D-1} 100(x_{i+1} - x_i)^2 + (x_i - 1)^2 \tag{7}$$

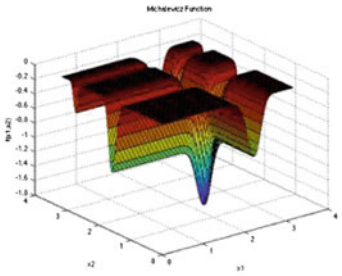


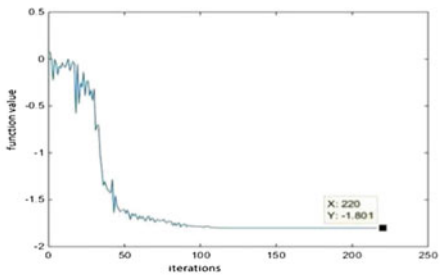


(continued)

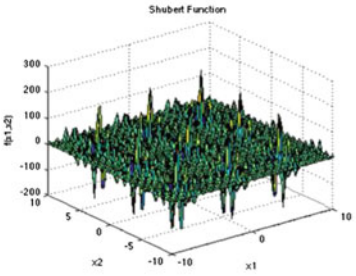
Table 1 (continued)

$$MichalewiczFunction : f = -\sum_{i=1}^{D} sin(x_i) * sin(\frac{ix_i^2}{\pi})^{20} \quad (8)$$





$$ShubertFunction : f = \prod_{i=1}^{D} \sum_{j=1}^{s} jcos((j+1) * x_i + j)) \quad (9)$$



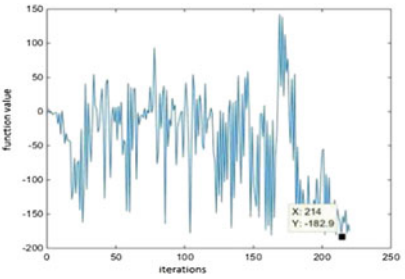


Table 2 Iterations at which benchmark functions converge

Function	Iterations at which the gbest values are obtained			
	Mean	Std. dev.	Fastest	Slowest
Sphere	220	220	220	220
Rastigin	88.0333	6.8956	78	109
Shubert	26.2	2.2652	23	31
Michelawicz	30.0333	3.0904	26	40
Rosenbrock	136.5333	5.3092	128	148

better values in unimodal functions like Sphere function, and almost precise values in multimodal functions like the Rastrigin, Rosenbrock, Michalewicz, etc.

Table 4 contains the comparison of mean and standard deviation on the benchmark functions such as Spherical, Rastrigin, Michaelwicz, and Shubert.

Table 3 Results obtained of the algorithm on various benchmark functions

Benchmark functions		Mean	Std. dev.	Best fit	Worst fit	Expected value [18]
Sphere	X1	–1.45E-37	8.87E-37	6.87E-37	–4.79E-36	0
	X2	–5.60E-37	3.49E-36	2.30E-36	–1.89E-35	0
	Fgbest	1.29E-71	6.54E-71	1.23E-82	3.58E-70	0
Rastrigin	X1	–1.63E-09	2.38E-09	–3.75E-09	3.51E-09	0
	X2	–4.79E-10	2.03E-09	–3.73E-09	2.89E-09	0
	Fgbest	0	0	0	0	0
Shubert	X1	–0.6728	1.5338	–1.4266	4.8581	Several minima
	X2	–0.2959	2.0768	–1.4252	4.8580	Several minima
	Fgbest	–186.725	0.0077	186.7304	–186.705	–186.7309
Michalewicz	X1	2.2029	5.87E-10	2.2029	2.2029	2.20
	X2	1.5708	2.11E-09	1.5708	1.5708	1.57
	Fgbest	–1.8013	6.77E-16	–1.8013	–1.8013	–1.8013
Rosenbrock	X1	1	0	1	1	1
	X2	1	0	1	1	1
	Fgbest	0	0	0	0	0

Table 4 Performance comparison amongst literature and introduced algorithm on benchmark functions

Function name		[11]	[12]	[15]	Obtained results
Mean	Sphere	2.46E-11	1.44E-23	3.80E-27	1.29E-71
	Rastrigin	NA	0.01	0.01	0
	Michalewicz	–1.8769	–1.8947	–1.8966	–1.8013
	Shubert	–186.704	–186.728	–186.717	–186.725
Standard deviation	Sphere	1.35E-10	7.86E-23	2.08E-26	6.54E-71
	Rastrigin	0.1817	0.2524	0.2524	0
	Michalewicz	0.0934	0.0906	0.0868	6.78E-16
	Shubert	0.1418	0.0119	0.0762	0.0077

Rosenbrock function is neglected due to unavailability of information. The best value amongst the compared values is highlighted. On comparison of the mean values with [11] and [15], it is seen that there is 10^{60} increase and a 10^{44} increase, respectively, for sphere function. For Rastrigin function, the expected value of 0 has been obtained. For Michelawicz function, the expected value has been obtained. For Shubert function, [12] exhibits the best value, and the proposed algorithm is only second to it with an error of 0.0016%. In Table 3, it has also been shown that the algorithm produces the expected result for Rosenbrock function.

5 Conclusion

The PSO variant introduced in the paper has three modifications, namely, a new range of linearly varying inertia weight to work with most real-time and natural order functions that follow the exponential rule, a *mutation* technique to control particles that move too fast and increase exploration capability, and a *velocity restriction* factor that converges the search space exponentially over the given range. The algorithm has been proven to work well for both unimodal and multimodal functions. It can especially tackle multimodal functions better due to the inclusion of the Pareto effect in various phases of the algorithm. The algorithm seems to be promising for any number of dimensions with any function and is expected to produce a better solution. Improvement of the order of 10^{40} is seen in spherical function and expected values have been obtained in other benchmark functions.

References

1. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. Proceedings of IEEE International Conference on Neural Networks, Perth, Australia **4**, 1942–1948 (1995)
2. Kennedy, J., Eberhart, R.C., Shi, Y.H.: Swarm Intelligence. Morgan Kaufmann, San Mateo, CA (2001)
3. Eberhart , R.C., Kennedy, J.: A new optimizer using particle swarm theory. In: Proceedings of 6th International Symposium Micromachine Human Science, Nagoya, Japan, pp. 39–43 (1995)
4. Eberhart, R.C., Shi, Y.H.: Particle swarm optimization: developments, applications and resources. In: Proceedings of IEEE Congress on Evolutionary Computation, Seoul, Korea, pp. 81–86 (2001)
5. Ciuprina, G., Ioan, D., Munteanu, I.: Use of intelligent-particle swarm optimization in electromagnetics. IEEE Trans. Magn. **38**(2), 1037–1040 (Mar 2002)
6. Liang, J.J., Qin, A.K., Suganthan, P.N., Baskar, S.: Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Trans. Evol. Comput. **10**(3), 281–295 (Jun 2006)
7. Ho, S.-Y., Lin, H.-S., Liauh, W.-H., Ho, S.J.: OPSO: orthogonal particle swarm optimization and its application to task assignment problems. IEEE Trans. Syst. Man Cybern. A Syst. Hum. **38**(2), 288–298, Mar 2008
8. Liu, B., Wang, L., Jin, Y.H.: An effective PSO-based mimetic algorithm for flow shop scheduling. IEEE Trans. Syst. Man Cybern. B Cybern. **37**(1), 18–27 (Feb 2007)

9. Eberhart, R.C., Shi, Y.: Guest editorial special issue particle swarm optimization. IEEE Trans. Evol. Comput. **8**(3), 201–203 (Jun 2004)
10. Zhan, Z.-H., Zhang, J.: Adaptive particle swarm optimization. In: IEEE Trans. Syst. Man Cybern. B Cybern. **39**(6), Dec 2009
11. Shi, Y., Eberhart, R.C.: A modified particle swarm optimizer. In: Proceedings of IEEE World Congress Computation Intelligence, p. 6973 (1998)
12. Chen, T.-Y., Chi, T.-M.: On the improvements of the particle swarm optimization algorithm. Adv. Eng. Softw. **41**, 229–239 (2010)
13. Bansal, J.C., Singh, P.K., Saraswat, M., Verma, A., Jadon, S.S., Abraham, A.: Inertia weight strategies in particle swarm optimization. Proceedings of IEEE International Conference on Neural Network, Perth, Australia **4**, 1942–1948 (1995)
14. Das, S., Abrahamm, A., Konar, A.: Particle swarm optimization and differential evolution algorithms: technical analysis, applications and hybridization perspectives. Stud. Comput. Intell. (SCI) **116**, 1–38 (2008)
15. Anand, B., Aakash, I., Akshay, Varrun, V., Reddy, M.K., Sathyasai, T., Devi, M.N.: Improvisation of particle swarm optimization algorithm. In: International Conference on Signal Processing and Integrated Networks (SPIN). India (2014)
16. Kiremire, A.R.: The application of pareto principle in software engineering. 19th October (2011)
17. Wikipedia. Pareto principle. http://en.wikipedia.org/wiki/paretoprinciple. Accessed March 2016
18. Virtual library of simulation experiments: test functions and datasets. http://www.sfu.ca/~ssurjano/. Accessed March 2016