Laplace Mutated Particle Swarm Optimization (LMPSO)

Muhammad Imran[1], Rathiah Hashim[1] and Noor Eliza Abd Khalid[2]

[1]University Tun Hussein Onn Malaysia86400 Parit Raja, Batu Pahat Johor Malaysia
[2]Universiti Teknologi MARA, Malysia
[1]malikimran110@gmail.com, [1]radhiah@uthm.edu.my, [2]elaiza@tmsk.uitm.edu.my

Abstract: Particle Swarm Optimization (PSO) algorithm has shown good performance in many optimization problems. However, it can be stuck into local minima. To prevent the problem of early convergence into a local minimum, various researchers have proposed some variants of PSO. In this research different variants of PSO are reviewed that have been proposed by different researchers for function optimization problem and one new variant of PSO is proposed using Laplace distribution named as LMPSO. The performance of LMPSO is compared with existing variants of PSO proposed for function optimization. The analysis in this research shows the effect of different mutation operator on Particle Swarm Optimization (PSO). To validate the LMPSO, experiments are performed on 22 benchmark functions. The result shows that the LMPSO achieved better performance as compared to previous PSO varients.
[Imran M, Hashim R, Khalid NEA. **Laplace Mutated Particle Swarm Optimization (LMPSO).** *Life Sci J* 2014;11(10):292-299] (ISSN:1097-8135). http://www.lifesciencesite.com. 42

Keywords: *PSO, Mutation, Laplace, function optimization, PSO variants*

1. Introduction

PSO is a population based optimization method proposed by Kennedy and Eberhart [1] . The algorithm simulates the behaviour of bird flock flying together in multi dimensional space in search of some optimum place, adjusting their movements and distances for better search [1]. PSO is very similar to evolutionary computation such as Genetic algorithm (GA). The swarms are randomly initialized and then search for an optimum solution by updating generations [1].

PSO is a combination of two approaches, one is cognition model that is based on self expression and the other is a social model, which incorporates the expressions of neighbours. The algorithm mimics a particle flying in the search space and moving towards the global optimum solution. A particle in PSO can be defined as $P_i \in [a,b]$ where i=1, 2, 3…. D and a, b ∈ R, D is for dimensions and R is for real numbers [2]. All the particles are initialized with random positions and with random velocities [1], then particles move towards the new position based on their own experience and with neighbourhood experience. Each particle in PSO maintains two important positions called p_{best} and g_{best} where p_{best} is the particle's own best position and g_{best} is the global best position among all the particles. The velocity and position of each particle are updated by equation (1) and (2).

$$V_i(t+1) = V_i(t) + c_1 * r_1 * (p_{best} - n_i(b)) + c_2 * r_2 * (g_{best} - x_i(t)) \quad \ldots\ldots (1)$$

$$X_i(t+1) = x_i(t) + v_i(t+1) \quad \ldots\ldots (2)$$

where x_i is the position, v_i is the velocity and P_{best} is the personal best position and g_{best} is the global best position for PSO. In this equation r_1 and r_2 are two random numbers ranges from (0,1) and c_1 and c_2 are learning factors specifically the cognition and cognition component influential respectively.

2. PSO Variants

J. Kennedy and R. Eberhart proposed PSO in 1995. Despite the successful implementation of PSO for the purpose of optimization one of the problem with PSO was to stuck in local minima, to fix this dilemma number of variants of PSO variants have been proposed by researchers with respect to different parameters and operators. Following section discuss in detail about the variants of PSO with respect to variant type.

2.1. Initialization

Initialization of population plays an important role in the evolutionary and swarm based algorithms. In case of inappropriate initialization, the algorithm may search in unwanted areas and may be unable to search for the optimal solution.

Nguyen et al [3] inspect the some randomized low discrepancy sequence to initialize the swarm to increase the performance of PSO. They used three low discrepancy sequence Halton, Faur and Sobol. Halton sequence is actually the extension of van der Corput. Ven Der Corput sequence is one dimensional in order to cover search space in N dimensions and Halton is defined as one of the extension of Vender Corput sequence.

Pant et al [4] explore the effect of initializing swarm with the vender Corput sequence which is a low discrepancy sequence to solve the global optimization problem in large dimension search space. They named the proposed algorithm as VC-PSO. The author claim that PSO performance is very well for problems having low dimensions but as the dimensions evolve the performance deteriorates, this problem become more severe in case of multimodal functions. They used the linear decreasing inertia weight from .9 to .4 with c_1=c_2=2.0.

Jabeen et al [2] proposed opposition based initialization which calculates opposition of randomly initialized population and selects better particles among random and opposition of initial population. This population is provided as an input for traditional PSO algorithm.

Chang et al [5] proposed an enhanced version of opposition based PSO called quasi-oppositional comprehensive learning PSO (QCLPSO). Instead of calculating traditional opposite of a point, the proposed modification calculates Qausi opposite particle, which is generated from the interval between median and opposite position of the particle.

2.2. Constriction Coefficient

For balancing the exploration-exploitation trade off, Clerc proposed a new approach [6] to improve the performance of PSO, which uses a new parameter 'χ' called the constriction factor. The velocity update scheme proposed by Clerc can be expressed for the d^{th} dimension of i^{th} particle as:

$$v_{ij}(t+1) = \chi\left[v_{ij}(t+1) + \varphi_1\left(v_{ij}(t) - x_{ij}(t)\right) + \varphi_2(\hat{y}_{ij}(t) - x_{ij}(t))\right] \ldots\ldots\ldots\ldots\ldots\ldots (3)$$

Where $\chi = 2 \div \left|4 - \varphi - \sqrt{\varphi 2 - 4\varphi}\right|$

With $\varphi = C_1 + C_2, \varphi_1 = c_1 r_1, \varphi_1 = c_2 r_2$

2.3. Inertia Weight

The inertia weight is a scaling factor associated with the velocity during the previous time step resulting in a new velocity update equation as eq (2.1), introduced by Shi [7]. Inertia Weight is used to control the exploration and exploitation abilities of the swarm. Large value of inertia weight promotes exploration while small value promotes local exploitation [7]. Some researchers [[8], [9], [10]] used fixed inertia weight and some [[11], [12], [13]] used decreasing inertia weight.

In linearly decreasing inertia weight, large value (0.9) linearly decreased to small value (0.4)

The exponent decreasing inertia weight is introduced by Li [14]. In order to beat the early convergence Li [14] proposed a variant of PSO with exponent decreasing inertia weight and stochastic mutation as shown in equation (4).

$$w = (w_{ini} - w_{end} - d_1)\exp\left(\frac{1}{1+\frac{d_2 t}{t_{max}}}\right) \ldots\ldots\ldots\ldots (4)$$

where t_{max} denotes the max iteration, t denotes the t^{th} iteration, w_{ini} denotes the original inertia weight, w_{end} denotes the inertia weight value when the algorithm process run the max iterations, d_1 and d_2 is a factor to control w between $w_{ini} and\ w_{end}$.

In non linear decreasing inertia weight, a large value decreases to small value but decrease non-linearly. In non linear decreasing of inertia weight, search space can be explored in shorter time but take larger time to exploit the search space Chongpeng et al [15]. The inertia weight was calculated by equation (5)

$$w_0 = w_{end} + (w_{end} - w_{start}) \times \left(1 - \left(\frac{t}{t_{max}}\right)^{k_1}\right)^{k_2} \quad (5)$$

where k_1, k_2 are two natural numbers, w_{start} is the initial inertia weight, w_{end} is the final value of weighting coefficient, t_{max} is the maximum number of iteration and t is current iteration. Value of k_1>1 and K_2=1.

Using fuzzy sets and rules, the inertia weight is adjusted dynamically [12].

Zhang et al [16] set the inertia weight as uniformly random number between 0 and 1. Author claimed that this inertia weigh scheme is more capable to escape from local minima and can overcome two problems of linearly decreasing inertia weight

1. Can overcome the problem of linearly inertia weight dependency on maximum iteration.
2. Another is avoiding the lack of local search ability at early process of PSO and global search ability at the end of PSO process.

Wei et al [17] Proposed dynamic PSO with dimension mutation. First they introduce dynamic inertia weight which is changed dynamically based on speed and accumulation factors then they present a dimension mutation operator to overcome the premature convergence. They calculate inertia weight as following.

$$\omega = \omega_0 - h\omega_h - s\omega_s$$

where ω_0=1, $\omega_h \in (0.4,0.6)$, $\omega_s \in (0.1,0.2)$

Pant et al [18] proposed a new version of PSO that uses Gaussian inertia weight. Responsible factor for the uniqueness of the modified algorithm was

- Development of new inertia weight using Gaussian distribution
- Use of different distribution instead of uniform distribution for the generation of the initial swarm.

Inertia weight equation is given by

$$w = abs(rand)/2 \ldots\ldots\ldots\ldots\ldots\ldots\ldots (6)$$

where rand is the random number having gaussion distribution..

Shu-Kai [19] proposed a variant of PSO using an adaptive dynamic weight scheme. They proposed a novel nonlinear function amendable inertia weight adaptation with an active method for inertia weight to improve the performance of PSO algorithms. The nonlinear function is given by

$$\omega = d^{r}_{\omega_{initial}}$$

where d is the decrease rate from 1.0 to 0.1 and r is dynamic adaption rule depending on the following rule. For minimization case it follows.

Xuedan Liu et al [20] proposed the PSO with dynamic inertia weight using mutation, reinitialized the swarm when it get stagnate. Author used the linearly decreasing inertia weight with following formula

$$w(t) = .9 - \left(\frac{t}{max_t}\right) \times 0.5 \ldots\ldots\ldots (7)$$

where max_t is maximum number of iteration then they used the wheel structure.

2.4. Mutation Operators

In PSO, Mutation operator is used to change the position of particle from previous position to new positions. The basic role of mutation operator in PSO is quick convergence. It prevents the PSO from stagnation at any local minima. Some researchers used different mutation operators to overcome the premature convergence of the PSO. Through mutation operators authors mutate the $g_{best,}$ after mutating the $g_{best,}$ they compared the fitness performance of mutated g_{best} & original one and selects the best one for rest of the PSO process. Some of the mutation operators are Cauchy Mutation [13], [21], Adaptive Mutation [22], Power Mutation [23] and Gaussian Mutation [24].

To prevent PSO trapping from local optima, Li et [25] introduced Cauchy mutation in PSO. They named their algorithm as CPSO. They updated the particles position by Cauchy mutation as

$V' = V + \exp(\delta)$

$X' = X + V'(\delta)$

where δ is a Cauchy random number

Wang [13] proposed a new Cauchy mutation operator for PSO. This operator is applied to perform local search around the global best particle. The motivation for using such a mutation operator is to increase the probability of escaping from a local optimum. The inertia weight used is 0.72984 and c1=c2 =1.49618. This mutation operator is used to increase the probability of escaping from a local optimum. The cauchy mutation operator is:

$$gbest_i(i) = gbest_i(i) + W(i) * N(x_{min}, x_{max} \ldots (8)$$

where N is a Cauchy distributed function with scale parameter t=1, $N(x_{min}, x_{max})$ is a random number with in (x_{min}, x_{max}) of defined domain of test function and

$$W(i) = \left(\sum_{j=1}^{popsize} V[j][i]\right)/popsize$$

where $V[j][i]$ is the i^{th} velocity vector of j^{th} particle in the population, popsize is the size of population

Wang [21] proposed opposition based initialization in PSO coupled with Cauchy mutation operator. Cauchy mutation operator is used on the global.

Pant et al [22] proposed two variants of PSO: AMPSO1 and AMPSO2 for global optimization problem. The author used adaptive mutation for both techniques. The main goal of author was to improve the diversity of PSO without compromising on the solution quality. In AMPSO1 personal best position of the swarm is mutated while in AMOPSO2 global best particle of the swarm is mutated. Following formula is used to mutate the particle.

$$g_{best} = g_{best} + \sigma' * Betarand() \ldots\ldots\ldots\ldots\ldots (9)$$

where $\sigma' = \sigma * \exp(\tau N(0,1) + \tau' N_j(0,1))$, N(0,1) denotes a normally distributed function with mean zero and standard deviation one, $N_j(0,1)$ that a different random number is generated for each value of j, τ and τ' are set as $\frac{1}{\sqrt{2n}}$ and $\frac{1}{\sqrt{2\sqrt{n}}}$ respectively and value of σ is originally set as 3. Betarand() is a random number generated by beta distribution with parameter less than 1.

Pant et al [26] explored the Sobol mutation operator in PSO to enhance the performance of basic PSO algorithm, which uses Qausi random Sobol sequence as they claimed that random probability distribution cannot cover the search domain as Qausi random can cover. They named their operator Systematic mutation (SM).

Different mutation operator performs well for different type of problem. Li et al [27] proposed an adaptive mutation with three mutation operators to escape the particle from local optima. They apply Cauchy mutation, Gaussian mutation and levy mutation on the position and velocity

Gao [28] presented new adaptive mutation operator by fitness variance and space position aggregation degree to give a new PSO with adaptive mutation. They claimed that algorithm can be stuck into local convergence when fitness variance of particles is near to zero.

Wu et al [23] proposed a new variant of PSO called power mutation PSO (PMPSO), which employs a power mutation operator. The core plan of PMPSO is to apply power mutation on the fittest particle of current swarm. The purpose of power mutation is to help the particles to jump out from the local optima.

Pant et al [29] introduced the new mutation operator for improving the Quantum Particle Swarm

Optimization (QPSO) algorithm. The mutation operator uses the Qausi random Sobol sequence and called as a Sobol mutation operator.

Silva et al [30] proposed predator pray optimization technique used for function optimization. PSO has been applied for constrained non linear optimization problem [31]. Brits et al [32] proposed another variant of PSO which intended to locate multiple best possible solutions in multimodal problems by using sub swarms and the convergent sub swarms algorithm. By considering the particles previous best position and mistakes, Yang et al [33] proposed a new variant of PSO. To share the information of particles Zhi-Feng [34] proposed a PSO with cross over operator. Omran et al [8] used an opposition based learning to improve the performance of PSO.

Tang et al [35] proposed an enhance opposition based PSO which is called as EOPSO. According to the authors, opposite point is closer to global optima as compare to current point. This provides more chances to get close to global optima.

3. Proposed LMPSO

To avoid the PSO issues related to early convergence and stuck in local minima, this study has modified PSO to improve its performance. The modification is done through Laplace distribution and this modification is called as Laplace Mutated PSO (LMPSO). Laplace distribution is continuous and double exponential distribution. It is the distribution of differences between two independent variables with identical exponential distributions. A random variable has a Laplace (μ, b) distribution if its probability density function is

$$F(x|\mu,b) = \frac{1}{2b} exp\left(-|\frac{x-\mu}{b}|\right) \quad \ldots\ldots\ldots (10)$$

where μ is a location parameter and b>0 is a scale parameter

In LMPSO, global best particle is mutated by Laplace mutation to escape PSO from local minima. In LMPSO g_{best} is mutated by (11).

$$g_{best} = \begin{cases} g_{best} + t, & g_{best}\ is + ve \\ g_{best} - t, & g_{best}\ is - ve \end{cases} \quad \ldots\ldots\ldots (11)$$

where $t = \frac{x^l}{x^u} * \mathcal{L}$

where x^l, x^u are the boundaries of the current search space and $\mathcal{L}$ is the Laplace random number generated by Laplace distribution.

3.1. Experiments

To validate the performance of the proposed LMPSO, we run the proposed LMPSO and other 4 variants of PSO on the 22 benchmark functions. The parameter setting of each PSO variant used for comparison purpose are shown in Table 1. All the functions used for the experimental purpose are given in the appendix at the end of this paper.

The results of the all 22 functions are shown in the table 3.

We used the same experimental setting as shown in Table 2 for all the variants of PSO. All variants including proposed LMPSO run in same session for one function. We try our best that there should be no biasness to get the results. The results are given in table 3.

Table 1: Comparison of parameters

PSO Variants	Use Population Min	Use Population Max	Initialization Randomly
CPSO	No	No	Yes
AMPSO	No	No	Yes
PMPSO	Yes	Yes	Yes
NMPSO	No	No	Yes
RMPSO	No	No	Yes
LMPSO	Yes	Yes	Yes

Table 2: Experimental Settings

Parameter	Value
Search Space	[100,-100]
Dimensions	10
	20
	30
Iterations	1000
	1500
	2000
Population size	30
Number of PSO Runs	30

4. Results

4.1. Analysis

Detail Results of all the Benchmark functions are given in table 3. The results of function f_1 show that LMPSO has better performance as compared to other techniques.

In function f_2, when 10 dimensions with 1000 iterations are used then PMPSO performance is best, whereas by increasing the dimensions and iterations, LMPSO performance can be boost up as compared to other techniques.

f_3 results shows that LMPSO has better performance than other techniques.

By Result of f_4, we can infer that all techniques have equal performance in all three cases.

The results of function f_5 shows that PMPSO has better performance in case of 10 dimensions and 1000 iterations, while AMPSO perform better in case of 20 dimensions and 1500 iterations and CPSO perform better in case of 30 dimensions and 2000 iterations.

Table 3: Results of LMPSO and four previous variants of PSO where, bold one is the best results among all.

Function	Dim	Iterations	PSO	CPSO	AMPSO	PMPSO	LMPSO
			Avg Fitness	Avg Fitness	Avg Fitness	Avg Fitness	Avg Fitness
f_1	10	1000	5.35E-56	2.96E-65	1.12E-56	2.53E-96	**2.84E-97**
	20	1500	2.58E-16	4.79E-25	2.09E-16	4.66E-81	**5.72E-87**
	30	2000	3.56E-07	4.30E-14	2.13E-09	2.86E-65	**9.19E-78**
f_2	10	1000	1.71E-66	2.30E-71	1.73E-67	**2.21E-76**	1.62E-48
	20	1500	1.47E-17	3.64E-27	4.53E-18	1.57E-54	**2.77E-66**
	30	2000	4.04E-07	1.25E-13	3.04E-09	6.12E-11	**2.50E-37**
f_3	10	1000	4.97E+00	5.42E+00	5.72E+00	3.93E+00	**1.89E+00**
	20	1500	4.25E+01	3.67E+01	3.76E+01	2.24E+01	**4.03E+00**
	30	2000	9.76E+01	9.55E+01	9.86E+01	3.89E+01	**5.12E+00**
f_4	10	1000	**1.00E+00**	**1.00E+00**	**1.00E+00**	**1.00E+00**	**1.00E+00**
	20	1500	**1.00E+00**	**1.00E+00**	**1.00E+00**	**1.00E+00**	**1.00E+00**
	30	2000	**1.00E+00**	**1.00E+00**	**1.00E+00**	**1.00E+00**	**1.00E+00**
f_5	10	1000	2.57E+00	3.54E+00	1.88E+00	**2.50E-03**	7.06E+00
	20	1500	1.45E+01	1.27E+01	**1.38E+01**	4.62E+01	1.88E+01
	30	2000	2.91E+01	**2.14E+01**	2.62E+01	6.95E+01	2.88E+01
f_6	10	50	-7.50E+05	-3.27E+11	-5.46E+05	1.16E+06	**-9.08E+05**
	20	80	**-5.58E+07**	-1.71E+15	-8.94E+07	6.15E+07	-1.21E+08
	30	100	-2.41E+09	-2.77E+18	-1.82E+09	**4.48E+07**	-6.18E+08
f_7	10	50	**2.00E+01**	**2.00E+01**	**2.00E+01**	**2.00E+01**	**2.00E+01**
	20	80	2.00E+01	2.00E+01	2.00E+01	2.00E+01	**1.99E+01**
	30	100	2.00E+01	2.00E+01	2.00E+01	2.00E+01	**1.90E+01**
f_8	2	100	**3.98E-01**	**3.98E-01**	**3.98E-01**	**3.98E-01**	**3.98E-01**
f_9	2	100	**-1.00E+00**	**-1.00E+0**	**-1.00E+0**	**1.00E+00**	**-1.00E+0**
f_{10}	10	1000	1.04E-11	1.72E-17	1.32E-11	1.76E-16	**2.06E-17**
	20	1500	5.36E-01	1.68E-02	3.59E-01	9.70E-01	**4.64E-06**
	30	2000	1.03E+01	3.61E+00	8.96E+00	1.62E+01	**4.07E-04**
f_{11}	10	1000	**-3.14E-01**	**-3.14E-01**	**-3.14E-01**	**-3.14E-01**	**-3.14E-1**
	20	1500	-1.57E-01	-1.57E-01	-1.57E-01	**-6.38E-02**	-1.57E-1
	30	2000	**-7.39E-02**	-8.92E-02	**-1.60E-02**	2.63E-01	4.51E-02
f_{12}	10	100	-1.33E+06	-1.24E+69	-1.71E+05	-2.38E+01	**-2.29E+01**
	20	400	-3.97E+16	1.96E+28	-5.02E+16	**-3.32E+01**	-7.35E+01
	30	800	-1.66E+17	2.94E+70	2.94E+11	**-6.91E+01**	-8.57E+01
f_{13}	2	50	2.09E-26	4.46E-26	3.41E-26	7.39E-21	7.26E-19
f_{14}	2	50	**-1.03E+0**	-1.03E+00	**-1.03E+0**	**-1.03E+00**	**-1.03E+0**
f_{15}	10	50	2.14E+02	1.75E+02	1.63E+02	1.57E+02	**1.91E+02**
	20	80	5.94E+02	7.44E+02	7.98E+02	6.41E+02	**7.76E+02**
	30	100	1.19E+03	1.23E+03	1.26E+03	1.13E+03	**1.21E+3**
f_{16}	2	100	**3.00 E+00**	**3.00 E+00**	**3.00 E+00**	**3.00 E+00**	**3.00 E+00**
f_{17}	2	1000	2.94E-13	2.06E-13	**3.03E-14**	8.08E-13	3.25E-12
f_{18}	2	100	**-1.03E+00**	**-1.03E+00**	**-1.03E+00**	**-1.03E+00**	**-1.03E+0**
f_{19}	10	50	5.39E+00	5.67E+00	8.34E+00	3.76E-01	**1.26E-01**
	20	80	1.15E+02	1.01E+02	9.21E+01	2.10E-01	**3.31E-02**
	30	100	1.08E+03	9.78E+02	1.03E+03	1.11E+00	**7.49E-02**
f_{20}	10	1000	3.17E-17	2.01E-22	1.69E-17	6.43E-44	**3.69E-46**
	20	1500	3.80E+02	1.43E+02	2.19E+02	2.49E+00	**2.06E+0**
	30	2000	2.71E+03	1.48E+03	1.71E+03	2.56E+02	**6.15E+2**
F_{21}	10	1000	9.58E-08	5.87E-08	1.83E-08	3.69E-07	**2.83E-15**
	20	1500	2.23E-01	8.63E-01	3.73E-01	4.49E-02	**1.27E-14**
	30	2000	4.90E+00	7.57E+00	8.18E+00	1.94E+00	**2.48E-15**
F_{22}	2	100	1.54E-07	1.45E-07	1.59E-08	4.52E-08	**3.67E-09**
	2	400	2.87E-12	6.66E-11	5.75E-13	4.56E-13	**2.66E-16**
	2	800	7.54E-15	4.32E-14	2.89E-16	3.05E-13	**0**

In case of function f_6, traditional PSO remains good then other techniques when dimensions are 10 and iterations are 50. But if we increase dimensions to 20, 30 and iterations to 80,100 iterations respectfully, PMPSO's performance is better. LMPSO's performance was on the second number.

The performance of LMPSO is better than other techniques for function f_7 in all three cases.

The result of function f_8 remains same for all variants.

The performance of all techniques remains same for function f_9.

Results of f_{10} show that the performance of NMPSO and RMPSO is outstanding, while traditional PSO remain second best in case of 10 dimension and 1000 iterations while in other two

cases LMPSO remain second best. For f_{11} all techniques almost have same result. The Performance of all the techniques is almost same for function f_{12}, f_{13} and f_{14} .

From the results of function f_{15}, it can be seen that performance of LMPSO remains best in all three cases while PMPSO was the second best technique during this function.

Again function f_{16}' and f_{18} results shows that the performance of all techniques have same result.
The performance of AMPSO is best for function f_{18}.
The performance of LMPSO is best for the f_{19}- f_{22}.

5. Conclusion

From the results that are given in the table 3, we have seen that in 71% cases LMPSO perform batter then other techniques. Power mutation gave good performance in some cases. While the performance of CPSO and AMPSO was just good in one or two cases. The performance of each technique remains same for function *f4, f13, f14, f15, f16, f17.*

The performance of LMPSO remains good because it uses some statistics of search space while mutating the g_{best}. The result of power mutation is good from CPSO and AMPSO because they don't use the statistics of search space while mutating the g_{best}. In simple 2 dimensions functions, the performance of all techniques remains same.

Acknowledgements
The researchers would like to thank University Tun Hussein Onn Malaysia (UTHM) for supporting this project under Project Vote No 1315.

Corresponding Author:
Mr. Muhammad Imran
Faculty of Computer Science and Information Technology, University Tun Hussein Onn Malaysia, 86400 Parit Raja, Batu Pahat, Johor, Malaysia
E-mail: malikimran110@gmail.com

References

[1] J. Kennedy and R. Eberhart, "“Particle Swarm Optimization," in *Proceedings of IEEE International Conference on Neural Networks*, Perth, WA , Australia, 1995, pp. 1942-1948.

[2] H. Jabeen, Z. Jalil and A.R Baig, "Opposition Based Initialization in Particle Swarm Optimization," in *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, New York, NY, USA, 2009, pp. 2047- 2052.

[3] N. Q. Uy, N. X. Hoai, R. McKay and P. M. Tuan., "Initializing PSO with randomized low-discrepancy sequences: the comparative results," in *Proceedings of the IEEE Congress on Evolutionary Computation,* Singapore, 2007, pp. 1985-1992.

[4] M. Pant, R. Thangaraj, C. Grosan, and A. Abraham, "Improved Particle Swarm Optimization with Low-Discrepancy Sequences," in *IEEE Cong. on Evolutionary*, Hong Kong , 2008, pp. 3011-3018.

[5] C. Zhang et al, "A Novel Swarm Model With Quasi-Oppositional Particle," in *International Forum on Information Technology and Applications,* Chengdu, 2009, pp. 325 – 330.

[6] Clerc M and J.Kennedy, "The Particle Swarm – Explosion, Stability, and Convergence in a Multidimensional Complex Space," in *IEEE transactions on evolutionary computation,* vol. 6, 2002, pp. 58–73.

[7] Y.Shi and R. Eberhart., "A modified particle swarm optimizer," in *In Proceedings of the*, 1998, pp. 69–73.

[8] M. G. H. Omran and S.al-Sharhan., "Using Opposition-based Learning to improve the Performance of Particle Swarm Optimization.”," in *IEEE Swarm Intelligence Symposium*, St. Louis, MO , 2008, pp. 1 – 6.

[9] X. Liu et al, "Particle Swarm Optimization with Dynamic Inertia Weight and Mutation," in *Third International Conference on Genetic and Evolutionary Computing*, Guilin, China, 2009, pp. 620-623.

[10] C. Zhang et al, "Novel Swarm Model With Quasi-Oppositional Particle," in *International Forum on Information Technology and Applications*, Chengdu, 2009, pp. 325 – 330.

[11] M. Clerc, "The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation*, Washington, DC , USA , 1999, pp. 1951-1957.

[12] Y. Shi and R. C. Eberhart, "Fuzzy Adaptive particle Swarm Optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation*, Seoul , South Korea, 2001, pp. 101-106.

[13] H, Wang et al., "A Hybrid Particle Swarm Algorithm with Cauchy Mutation," in *Proceeding of IEEE Swarm Intelligence Symposium*, Honolulu, HI, 2007, pp. 356 - 360.

[14] H-R LI and Y-L Gao., "Particle swarm optimization algorithm with exponent decreasing inertia weight and stochastic mutation," in *Second International Conference*

on Information and Computing Science, Manchester , 2009, pp. 66-69.

[15] Huang Chongpeng, Zhang Yuling, Jiang Dingguo and Xu Baoguo, "On Some Non-linear Decreasing Inertia Weight Strategies in Particle Swarm Optimization*," in *Proceedings of the 26th Chinese Control Conference*, Zhangjiajie, Hunan, China, 2007, pp. 570-753.

[16] L. Zhang, H. Yu, and S. Hu, "A new approach to improve particle swarm optimization," in *Proceedings of the 2003 international conference on Genetic and evolutionary computation*, 2003, pp. 134-139.

[17] A.P. Engelbrecht, *Fundamentals of Computational Swarm intelligence*. England: John Wily and Sons Ltd, 2005.

[18] M. Pant and T. Thangaraj, V.P. Singh, "Particle Swarm Optimization Using Gaussian Inertia Weight," in *International Conference on Computational Intelligence and Multimedia Applications*, Sivakasi, Tamil Nadu , 2007, pp. 97-102.

[19] S. Kai, S. Fan and J. M. Chang, "A Modified Particle Swarm Optimizer Using an Adaptive Dynamic Weight Scheme," in *Proceedings of the 1st international conference on Digital human modeling*, 2007 , pp. 56-65.

[20] X. Liu et al, "Particle Swarm Optimization with Dynamic Inertia Weight and Mutation," in *Third International Conference on Genetic and Evolutionary Computing*, Guilin, 2009, pp. 620-623.

[21] H. Wang et al, "Opposition-based Particle Swarm Algorithm with Cauchy Mutation ," in *IEEE Congress on Evolutionary Computation*, Singapore , 2007, pp. 4750 – 4756.

[22] M. Pant, R. Thangaraj, and A. Abraham , "Particle Swarm Optimization Using Adaptive Mutation," in *19th International Conference on Database and Expert Systems*, Washington, DC, USA, 2008, pp. 519-523.

[23] X. Wu and M. Zhong, "Particle Swarm Optimization Based on Power Mutation," in *ISECS International Colloquium on Computing, Communication, Control, and Management*, Sanya, 2009, pp. 464 – 467.

[24] Higashi, N. Iba, H, "Particle swarm optimization with Gaussian mutation," in *Proceedings of the IEEE Swarm Intelligence Symposium*, 2003 , pp. 72 – 79.

[25] C. Li et al, "A Fast Particle Swarm Optimization Algorithm with Cauchy Mutation and Natural Selection Strategy,"," in *ISICA'07 Proceedings of the 2nd international conference on Advances in computation and intelligence*, 2007, pp. 334-343.

[26] M. Pant, R. Thangaraj1, V.P Singhand and A. Abraham, "Particle Swarm Optimization Using Sobol Mutation," in *First International Conference on Emerging Trends in Engineering and Technology*, Nagpur, Maharashtra , 2008, pp. 367-372.

[27] C. Li, S. Yang, and I. A. Korejo, "An adaptive mutation operator for particle swarm optimization," in *Proceeding of the 2008 UK Workshop on Computational Intelligence* , 2008, pp. 165-170.

[28] Y. Gao and Y. Duan , "A New Particle Swarm Optimization Algorithm with Adaptive Mutation Operator," in *Second International Conference on Information and Computing Science*, Manchester , 2009 , pp. 58-61.

[29] M. Pant, R.Thangaraj and V. P. Singh, "Sobol Mutated Quantum Particle Swarm Optimization," *International Journal of Recent Trends in Engineering*, vol. 1, no. 1, pp. 95-99, May 2009.

[30] A. Silva, A. Neves and E.Costa, "Chasing The Swarm: A Predator Pray Approach to Function Optimization," in *Proceedings of MENDEL 8th International Conference on Soft Computing*, 2002.

[31] X. Hu and R. Eberhart, "Solving Constraint non-linear optimization problems with Partilcle Swarm Optimization," in *6th World Multiconference on Systemics, Cybernetics and Informatics*, 2002.

[32] R. Brits.A.P. Engelbrecht and F. v. Den Bergh, "A Niching Particle Swarm Optimizer," in *Proceedings of the Conference on Simulated Evolution and Learning*, 2002.

[33] C. Yang and D. Simon, "A New Particle Swarm Optimization Technique," in *Proceedings of the 18th International Conference on Systems Engineering*, 2005, pp. 164-169.

[34] Z. F Hao,Z. G Wang and H. Huang, "A Particle Swarm Optimization Algorithm with Crossover Operator," in *Proceedings of the Sixth International Conference on Machine Learning andCybernetics*, Hong Kong , 2007, pp. 1036 – 1040.

1. Appendix

a. Test Functions

1. $f_1(x) = \sum_{i=0}^{n} x_i^2$
2. $f_2(x) = \sum_{i=0}^{n} i * x_i^2$
3. $f_3(x) = \sum_{i=0}^{n} [x_i^2 - 10cos(2\pi x_i) + 10]$
4. $f_4(x) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$
5. $f_5(x) = \sum_{i=1}^{n} [100(x_{i+1} - x_i^2)^2 + (1 - x_i^2)^2]$
6. $f_6(x) = \sum_{i=1}^{n} -x_i * \sin(-1\sqrt{|x_i|})$
7. $f_7(x) = -20 \exp\left(-0.2\sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n} \sum_{i=1}^{n} cos\, 2\pi x_i\right) + 20 + e$
8. $f_8(x) = (x_2 - \frac{5.1}{4\pi^2} x_1^2 + \frac{5}{\pi} x_2 - 6)^2 + 10\left(1 - \frac{1}{8\pi}\right) cos x_1 + 10$
9. $f_9(x) = -\cos(x_1) - \cos(x_2) \exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2)$
10. $f_{10}(x) = \max|x_i|, \quad 0 \le i \le n$
11. $f_{11}(x) = \frac{\pi}{n}\{10sin^2(\pi y_i) + \sum_{i=1}^{n}(y_i - 1)^2 [1 + 10sin^2(\pi y_{i+1})] + (y_n - 1)^2\} + \sum_{i=1}^{n} u(x_i, 10,100,4)$,
    a. $y_i = 1 + \frac{1}{4}(x_i + 1)$
    b. $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & , x_i > a_i \\ 0, & -a < x_i < a, \\ k(-x_i - a)^m & , \end{cases}$
12. $f_{12}(x) = .1\{sin^2(3\pi x_i) + \sum_{i=1}^{n-1}(x_i - 1)^2[1 + sin^2(3\pi x_{i+1})] + (x_n - 1)[1 + sin^2(2\pi x_n)]\} + \sum_{i=1}^{n} u(x_i, 5,100,4)$
13. $f_{13}(x) = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6}\right]^{-1}$
14. $f_{14}(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 x_1 x_2 - 4x_2^2 + 4x_2^4$
15. $f_{15}(x) = \sum_{i=1}^{n}|x_i| + \prod_{i=1}^{n}|x_i|$
16. $f_{16}(x) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 16x_1x_2 + 3x_2^2)]X[30 + (2x - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$
17. $f_{17}(x) = -\frac{1+\cos(12\sqrt{x_1^2+x_2^2})}{\frac{1}{2}(x_1^2+x_2^2)+2}$
18. $f_{18}(x) = (4 - 2.1x_1^2 + \frac{x_1^4}{3})x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2$
19. $f_{19}(x) = \sum_{i=1}^{n}|x_i + .5|^2$
20. $f_{20} = \sum_{i=1}^{n} x_i^2 + (\sum_{i=1}^{n} 0.5ix_i)^2 + (\sum_{i=1}^{n} 0.5ix_i)^4$
21. $f_{21} = \sum_{i=1}^{n}|x_i \sin(x_i) + 0.1x_i|$
22. $f_{22} = 0.5 + \frac{sin^2\sqrt{(x_1^2+x_2^2)} - 0.5}{1 + 0.01(x_1^2 + x_2^2)^2}$