Gaussian Particle Swarm with Jumps

Renato A. Krohling
Lehrstuhl Elektrische Steuerung und Regelung, ESR
Fakultät für Elektrotechnik und Informationstechnik
Universität Dortmund
D-44221 Dortmund, Germany
E-mail: renato.krohling@uni-dortmund.de

Abstract- Gaussian Particle Swarm Optimization (GPSO) algorithm has shown promising results for solving multimodal optimization problems in low dimensional search space. But similar to Evolutionary Algorithms (EAs), GPSO may also get stuck in local minima when optimizing functions with many local minima like the Rastrigin or riewank functions in high dimensional search space. In this paper, an approach which consists of a GPSO with *jumps* to escape from local minima is presented. The jump strategy is implemented as a mutation operator based on the Gaussian and Cauchy probability distribution. The new algorithm was tested on a suite of well-known benchmark functions with many local optima and the results were compared with those obtained by the standard PSO algorithm, and PSO with constriction factor. Simulation results show that the GPSO with Gaussian and Cauchy jump outperforms the standard one and presents a very competitive performance compared to PSO with constriction factor and also self-adaptive Evolutionary Programming.

1 Introduction

Particle swarm Optimization (PSO) [1] is an effective optimization method with growing research interest. Most PSO algorithms use uniform probability distribution to generate random numbers. However, new approaches using Gaussian and Cauchy probability distributions to generate random numbers to updating the velocity equation of PSO have been proposed [2-9]. Clerc and Kennedy [11] have introduced a constriction factor, which may be necessary to ensure convergence of the PSO. New results about the convergence of PSO analyzing the trajectory of a deterministic particle in one dimension via dynamic systems theory were presented [12]. However, a deterministic approach might not be straightforward generalized for the analysis of trajectory of stochastic particles in multi-dimensional search space.

In this paper, a novel approach combining Gaussian Swarm recently proposed in [13] and one strategy to escape from local minima is developed. The new algorithm which differs from the previous [2-9], named Gaussian PSO with jumps was tested on a suite of well-known benchmark multimodal functions and the results were compared with those obtained by the standard PSO algorithm, PSO with constriction factor, and Evolutionary Programming. The simulation results demonstrate that the jump strategy can significantly improve the performance of PSO to escape from local minima.

The rest of the paper is organized as follows: In section 2, the standard PSO, the PSO with constriction factor and the Gaussian PSO are described. In section 3, the GPSO combined with jump is presented. Section 4 provides simulation results for some multimodal benchmark optimization problems followed by conclusions in section 5.

2 Particle Swarm Optimization

2.1 Standard Particle Swarm Optimization

Particle Swarm Optimization (PSO) [1] is a population-based algorithm. PSO is initialized with a population of candidate solutions. Each candidate solution in PSO, called *particle*, has associated a randomized velocity. Each particle moves through the search space and keeps track of its coordinates in the search space, which is associated with the best solution (fitness) it has achieved so far, *pbest*. Another "*best*" value tracked by the *global* version of the particle swarm optimizer is the overall best value, *gbest,* and its location, obtained so far by any particle in the population. The algorithm proceeds updating first all the velocities and then the positions of the particle as follows:

$$\boldsymbol{v}_i = w\boldsymbol{v}_i + c_1 rand(\boldsymbol{p}_i - \boldsymbol{x}_i) + c_2 Rand(\boldsymbol{p}_g - \boldsymbol{x}_i) \quad (1)$$

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \boldsymbol{v}_i \quad (2)$$

The vector $\boldsymbol{x}_i = [x_{i1}, x_{i2}, ..., x_{in}]^T$ stands for the position of the i-th particle, $\boldsymbol{v}_i = [v_{i1}, v_{i2}, ..., v_{in}]^T$ stands for the velocity of the i-th particle and $\boldsymbol{p}_i = [p_{i1}, p_{i2}, ..., p_{in}]^T$ represents the best previous position (the position giving the best fitness value) of the i-th particle. The index g represents the index of the best particle among all the

0-7803-9363-5/05/$20.00 ©2005 IEEE.

particles in the swarm. The variable w is the inertia weight, c_1 and c_2 are positive constants; *rand* and *Rand* are random numbers in the range [0,1] generated according to a uniform probability distribution. In standard PSO, Particles' velocities along each dimension are clamped to a maximum velocity *Vmax*. If the sum of accelerations causes the velocity on that dimension to exceed *Vmax*, which is a parameter specified by the user, then the velocity on that dimension is limited to *Vmax*.

The inertia weight w represents the degree of the momentum of the particles. The second part is the "cognition" part, which represents the independent behaviour of the particle itself. The third part is the "social" part, which represents the collaboration among the particles. The constants c_1 and c_2 represent the weighting of the "cognition" and "social" parts that pull each particle toward *pbest* and *gbest* positions. Unless stated otherwise, it is assumed minimization problems in this paper. The standard PSO is described in Fig 1.

2.2 Particle Swarm Optimization with a Constriction Factor

A *constriction factor* K was introduced by Clerc and Kennedy [11] and the velocity updating (1) has been substituted by

$$\boldsymbol{v}_i = K\left[\boldsymbol{v}_i + c_1 rand(\boldsymbol{p}_i - \boldsymbol{x}_i) + c_2 Rand(\boldsymbol{p}_g - \boldsymbol{x}_i)\right] \quad (3)$$

$$K = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|} \quad (4)$$

where $\varphi = c_1 + c_2$, $\varphi > 4$ and K is a function of c_1 and c_2. A detailed discussion of the constriction factor is beyond the scope of this paper. For more details, the reader is referred to [11]. Usually, when the constriction factor is used, φ is set to 4.1 ($c_1 = c_2 = 2.05$), and the constriction factor K is 0.729. Each of the two stochastic coefficients of the $(\boldsymbol{p}_i - \boldsymbol{x}_i)$ terms is calculated by multiplying 0.729*2.05=1.49445 (times a random number between 0 and 1) with expected value equal 0.5, provides a mean value of 0.729.

The results presented by Trelea in [12] based on the theory of dynamic systems for analysis of a particle trajectory have been carried out with a different parameter set (w=0.6, and $c_1 = c_2 = 1.7$). The expected value for each of the two stochastic coefficients is calculated by multiplying 1.7*0.5=0.85, since the expected value of a uniform random number between 0 and 1 is 0.5. The results compared to that proposed in [11] showed a slightly superior performance.

Algorithm 1: Standard PSO

Input parameters: Swarm size, $w, c_1, c_2, V_{\max}$.
FOR each particle i
// random initialization of a population of particles with positions $\boldsymbol{x}_i$ using uniform probability distribution
$\boldsymbol{x}_i = \underline{x}_i + (\overline{x}_i - \underline{x}_i) U_i(0,1)$
$\boldsymbol{p}_i = \boldsymbol{x}_i$
$f(\boldsymbol{x}_i)$ // Compute the fitness
$\boldsymbol{p}_g := \arg\min\{f(\boldsymbol{x}_i)\}$ // global best particle
END FOR
DO
FOR each particle i
Update the position $\boldsymbol{x}_i$ according to (1) and (2).
Compute $f(\boldsymbol{x}_i)$
IF $f(\boldsymbol{x}_i) < f(\boldsymbol{p}_i)$ THEN // update the best
$\boldsymbol{p}_i = \boldsymbol{x}_i$
IF $f(\boldsymbol{x}_i) < f(\boldsymbol{p}_g)$ THEN // update the global best
$\boldsymbol{p}_g = \boldsymbol{p}_i$
END FOR
WHILE termination condition not met.

Figure 1: Standard PSO

2.3 Gaussian Particle Swarm Optimization

Observing that the expected values for the stochastic coefficients of the terms $(\boldsymbol{p}_i - \boldsymbol{x}_i)$ and $(\boldsymbol{p}_g - \boldsymbol{x}_i)$ are 0.729 [11] and 0.85 [12] then a probability distribution that generates random number with expected values between the limits [0.729 0.85] is presented. The goal consists of generating the stochastic coefficients for the terms according to the Gaussian probability distribution.
This can be accomplished by defining the velocity equation as

$$\boldsymbol{v}_i = |rand|(\boldsymbol{p}_i - \boldsymbol{x}_i) + |Rand|(\boldsymbol{p}_g - \boldsymbol{x}_i) \quad (5)$$

where $|randn|$ and $|Randn|$ are positive random numbers generated according to the absolute value of the Gaussian probability distribution, i.e., $abs[N(0,1)]$. $N(0,1)$ denotes a normally distributed random number with zero mean and unit standard deviation.This approach with Gaussian motion of the particle recently proposed in [13] is quite different from those proposed in previous works [2-9], [14].

Algorithm 2: Gaussian Particle Swarm optimization with Jumps

Input parameters: Swarm size, scale parameter η and the maximum_ failure_ counter.
FOR each particle i
// random initialization of a population of particles with positions $\boldsymbol{x}_i$ using uniform probability distribution
$\boldsymbol{x}_i = \underline{x}_i + (\overline{x}_i - \underline{x}_i) U_i(0,1)$
$\boldsymbol{p}_i = \boldsymbol{x}_i$
$f(\boldsymbol{x}_i)$ // Compute the fitness
$\boldsymbol{p}_g := \arg\min\{f(\boldsymbol{x}_i)\}$ // global best particle
END FOR
DO
FOR each particle i
IF failure_counter [i] <= maximum_ failure_ counter THEN
Update the velocity $\boldsymbol{v}_i$ according to (5).
Update the position $\boldsymbol{x}_i$ according to (2).
ELSE
Update the position $\boldsymbol{x}_i$ according to (6) or (7).
Compute $f(\boldsymbol{x}_i)$
IF $f(\boldsymbol{x}_i) < f(\boldsymbol{p}_i)$ THEN // update the best
$\boldsymbol{p}_i = \boldsymbol{x}_i$
Compute $f(\boldsymbol{x}_i)$
failure_counter [i] = 0 // reset the failure counter
ELSE
failure_counter [i]++
// increase the failure counter by one
END IF // no improvement of the fitness
IF $f(\boldsymbol{x}_i) < f(\boldsymbol{p}_g)$ THEN // update the global best
$\boldsymbol{p}_g = \boldsymbol{p}_i$
END FOR
WHILE termination condition not met.

Figure 2: GPSO with Jumps

3 Gaussian Particle Swarm with Jumps

The goal of this approach is to allow that particles of the swarm escape from local minima when they are prematurely attracted to local attractors. The motion of the particles is governed by two dynamical regimes. In the first case, if there is improvement of the fitness from iteration to iteration, then the particles move according to the velocity updating according to (5). In the second case, by monitoring the fitness of a particle, if there is no improvement of the fitness, then it is a sign that for this particle both terms of (5) are (almost) zero since $\boldsymbol{x}_i = \boldsymbol{p}_i = \boldsymbol{p}_g$ resulting in no motion at all in the search space. In such a case, it is necessary to introduce a jump to a new point in the search space, which may help to escape from local attractors. This can be done by introducing a *failure counter*, which monitors the improvement of the fitness value for each particle for a pre-specified number of iterations. If there is no improvement of the fitness, then the *failure counter* is increased by one in each iteration until this value achieves a pre-specified value named *maximum_failure_ counter,* when then the particle should jump to a new point. This can be carried out by introducing a mutation operator to change the position of the particle. We used the two most common mutation operators in EAs, i.e., Gaussian and Cauchy mutation [10]. The new position of the particles is now changed according to

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \eta N_j(0,1) \tag{6}$$

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \eta C_j(0,1) \tag{7}$$

In (6), η denotes a scale parameter, and $N(0,1)$ stands for a random number generated according to a Gaussian distribution. In (7), $C(0,1)$ stands for a random number generated according to a Cauchy probability distribution with scale parameter t=1 centred at the origin as described by

$$f(x) = \frac{1}{\pi} \frac{t}{t^2 + x^2}, \qquad -\infty < x < \infty$$

In both cases the random numbers are generated anew for each variable $j = 1,...,N$ of the particle i. Gaussian Particle Swarm combined with a strategy to escape from local minima using Gaussian or Cauchy jumps is described in Fig. 2. In the following, results will be presented for the proposed approach.

4 Simulation Results

4.1 Benchmark Functions

In an early study [13], we observed that GPSO outperformed the standard PSO and may outperform PSO with constriction factor on functions with no local optima (just one global), and functions with a few local optima but have difficulties on functions with many local minima. In order to overcome the difficulties of GPSO on such kind of problems we concentrate on this paper on a suite of well-known benchmark functions with many local minima given in Table I taken from [16]. This was exactly the motivation for introducing the jump strategy. The benchmarks functions in our Table I are numbered f_1 to f_6 and correspond to the functions f_4 to f_9 respectively in [16, pp. 4, Table I].

4.2 Experimental Settings

For the standard PSO, $c_1 = c_2 = 2$, and the inertia weight is linearly decreased over the course of each run, starting from 0.9 and ending at 0.4. For PSO with constriction factor, named KPSO, the parameters was set as $c_1 = c_2 = 2.05$, and $K = 0.729$. For the Gaussian PSO with Gaussian jump, shortly, GPSO+GJ and with Cauchy jump, shortly GPSO+CJ the *maximum_failure_ counter* was set to 5. In all experiments the number of particles (population size) was set to 100. Each experiment is run 50 times and is terminated only when the maximum number of generations (1500) has been reached.

The results obtained using GPSO+GJ and GPSO+CJ are compared with the results taken from [16, pp. 5, TABLE II] using Evolutionary Programming with the following parameters [16]: the same population size as PSO (100), tournament size q=10, initial standard deviation $\sigma = 3.0$, and the same maximum number of generations (1500). For the GPSO+GJ and GPSO+CJ algorithm initial simulations experiments have been carried out to determine the scale factor η_i for each function investigated and they are listed in Table II. As a rule of thumb, η may lie between $0.01*(\bar{x}_i - \underline{x}_i)$ and $0.1*(\bar{x}_i - \underline{x}_i)$. The choice of the parameter *maximum_failure_counter*, which monitors the fitness of the search (improvement or no) was not critical. In the experiments we set it to 5 but other appropriate values can also be used. Of course, other criteria for monitoring improvement or stagnation of the fitness may also be used.

Table II: η values

Function	η_i
$f_1(\boldsymbol{x})$	10
$f_2(\boldsymbol{x})$	0.01
$f_3(\boldsymbol{x})$	0.1
$f_4(\boldsymbol{x})$	10
$f_5(\boldsymbol{x})$	0.1
$f_6(\boldsymbol{x})$	0.1

Table I: Benchmark functions with many local minima used in this study, where N stands for the dimension, S for the lower and upper bounds of the search space and $f_{\min}$ is the minimum value of the function [10], [15]-[16]

Test Functions	N	S	$f_{\min}$
$f_1(\boldsymbol{x}) = \sum_{i=1}^{N} x_i \sin(\sqrt{x_i})$	30	$(-500, 500)^N$	-12569.6
$f_2(\boldsymbol{x}) = \sum_{i=1}^{N} \left\{ x_i^2 - 10\cos(2\pi x_i) + 10 \right\}$	30	$(-5.12, 5.12)^N$	0
$f_3(\boldsymbol{x}) = -20\exp\left\{ -0.2\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2} \right\} - \exp\left\{ \frac{1}{N}\sum_{i=1}^{N} \cos(2\pi x_i) \right\} + 20 + e$	30	$(-32, 32)^N$	0
$f_4(\boldsymbol{x}) = \frac{1}{4000}\sum_{i=1}^{N} x_i^2 - \prod_{i=1}^{N} \cos(\frac{x_i}{\sqrt{i}}) + 1$	30	$(-600, 600)^N$	0
$f_5(\boldsymbol{x}) = \frac{\pi}{N}\left\{ 10\sin^2(\pi y_i) + \frac{1}{N}\sum_{i=1}^{N-1} (y_i - 1)^2 \left\{ 1 + 10\sin^2(\pi y_{i+1}) \right\} + (y_n - 1)^2 \right\}$ $+ \frac{1}{N}\sum_{i=1}^{N} u(x_i, 10, 100, 4),\ y_i = 1 + 1/4(x_i + 1)$	30	$(-50, 50)^N$	0
$f_6(\boldsymbol{x}) = 0.1\left\{ \sin^2(3\pi x_i) + \frac{1}{N}\sum_{i=1}^{N-1} (x_i - 1)^2 \left\{ 1 + \sin^2(3\pi x_{i+1}) \right\} + (x_N - 1)^2 \cdot \left\{ 1 + \sin^2(3\pi x_{N+1}) \right\} \right\} + \sum_{i=1}^{N} u(x_i, 5, 100, 4)$	30	$(-50, 50)^N$	0

Table III: Results of the optimization for the benchmark functions using standard PSO, PSO with constriction factor (KPSO), Gaussian PSO with Gaussian Jump (GPSO+GJ) and Gaussian PSO with Cauchy Jump (GPSO+CJ). The results are averaged over 50 runs. "Mean Best" indicates the mean best function values found in the last generation and "Std. Dev." stands for standard deviation. The t-test values listed are EP-GPSO+CJ

	PSO standard	KPSO constriction factor	GPSO	GPSO+GJ Gaussian jump	GPSO+CJ Cauchy jump	Gaussian EP taken from [16]	T-test[a]
Function	Mean Best (Std. Dev.)	Mean Best (Std. Dev.)	Mean Best (Std. Dev.)	Mean Best (Std. Dev.)	Mean Best (Std. Dev.)	Mean Best (Std. Dev.)	
$f_1(x)$	-7678.3 (353.219)	-9472.6 (786.25)	-8140.4 (681.13)	-10745 (361.591)	**-10843 (295.304)**	-7976.6767 (510.368)	34.02
$f_2(x)$	167.898 (13.941)	64.370 (30.074)	71.229 (17.598)	27.343 (8.288)	**12.770 (4.864)**	63.4156 (15.660)	21.62
$f_3(x)$	6.804 (0.297)	0.555 (2.783)	4.107 (1.858)	-0.0059 (0.0035)	**-0.0078 (5.03e-4)**	8.882 (3.259)	19.1
$f_4(x)$	6.641 (0.656)	**0.0174 (0.018)**	0.687 (0.885)	0.0263 (0.030)	0.055 (0.046)	0.0582 (0.0638)	0.28
$f_5(x)$	8.583 (1.580)	0.122 (0.168)	3.728 (3.387)	**4.60e-6 (2.33e-5)**	0.0528 (0.1930)	0.6208 (1.064)	3.68
$f_6(x)$	6.690 (0.825)	0.029 (0.049)	3.123 (2.783)	**1.0e-7 (4.629e-7)**	6.85e-7 (2.28e-6)	0.0948 (0.475)	1.40

[a] The t value of 49 degrees of freedom is significant at a level a 0.05 level of significance by a two-tailed t-test.

4.3 Results and Discussion

The results obtained are summarized in Table III and also the results from [16, pp. 5, Table II] for comparisons purpose. Firstly, it can be seen in Table III that GPSO outperforms the standard PSO but not PSO with constriction. Is it an indication that GPSO may also needs the momentum term?.

It can be observed that for the multimodal function optimization problems investigated, the Gaussian PSO with Gaussian jump and with Cauchy jump show considerably better performance than the standard PSO in all cases, and also outperforms PSO with constriction factor except for the function $f_4(x)$. For this function the result obtained by PSO with constriction is slightly superior.

The PSO algorithms (standard, with constriction, and Gaussian) may stagnate, and the solution no longer improves anymore. If the Gaussian PSO algorithm gets stuck in local minima the jump strategy may help to escape. This is especially important for functions with many local minima as investigated here. The results obtained using GPSO with Gaussian and Cauchy jumps outperformed also Evolutionary Programming with Gaussian mutation [16]. It is known that self-adaptive mutation may lead to premature convergence [17]. In such cases mutation using probability distribution like the Lévi one, which provides larger jumps has demonstrated to be beneficial [16]. In our study, the approach is quite different from [16], because our GPSO with jumps works based on two regimes. The jump strategy is used only when GPSO gets trapped in local minima.

5 Conclusions

In this paper, an approach combining Gaussian Particle Swarm with a strategy to escape from local minima, which consists of Gaussian or Cauchy jumps named GPSO+GJ and GPSO+CJ respectively has been proposed. GPSO+GJ and GPSO+CJ algorithms have shown very promising results. The jump strategy introduced increases the chances to escape from local minima, therefore improving the effectiveness of the algorithm. The GPSO+GJ and GPSO+CJ algorithms were studied on a suite of well-known benchmark functions with many local minima and compared with the standard PSO, and also with a PSO with constriction factor. The results showed that the GPSO+GJ and GPSO+CJ algorithms outperform the standard PSO for all functions studied and also the PSO with constriction factor for most of the problems investigated, except for function $f_4(x)$.

We have also compared our results with those provided by using standard self-adaptive Evolutionary Programming with Gaussian probability distribution [16]. It is clear that the jump strategy is very beneficial to escape from local minima over algorithm with self-adaptive step length as has been seen in the comparison. For all problems investigated GPSO with jumps outperformed self-adaptive EP.

The algorithm proposed is very simple and more effective than the earlier versions of PSO, therefore easier to be applied to real-world problems. Work in progress is investigating the use of other probability distributions and also a comparison with PSO with mutations (Gaussian

and Cauchy) added to the updating velocity term, and PSO with re-initialization.

Acknowledgments

R. A. Krohling thanks Dr. F. Hoffmann (Lehrstuhl ESR, University Dortmund) for supporting this research work.

References

[1] J. Kennedy, and R. Eberhart, "Particle swarm optimization," in *Proceedings of the IEEE International Conference on Neural Networks* (Perth, Australia), IEEE Service Center, Piscataway, NJ, IV: pp. 1941-1948, 1995.

[2] L. S. Coelho, and R. A. Krohling, "Predictive controller tuning using modified particle swarm optimisation based on Cauchy and Gaussian distributions (in Portuguese)," in *Proceedings of the VI Brazilian Conference on Neural Networks, Sao Paulo, Brazil, June* 2003.

[3] L. S. Coelho, and R. A. Krohling, "Predictive controller tuning using modified particle swarm optimisation based on Cauchy and Gaussian distributions," in *Proceedings of the 8th On-line World Conference on Soft Computing in Industrial Applications,* WSC8, Dortmund, September, 2003.

[4] J. Kennedy, "Bare bones particle swarms," in *Proceedings of the IEEE Swarm Intelligence Symposium*, pp. 80-87, 2003.

[5] B. R. Secrest, and G.B. Lamont, "Visualizing particle swarm optimization - Gaussian particle swarm optimization," in *Proceedings of the IEEE Swarm Intelligence Symposium*, Indianapolis, IN, pp. 198-204, 2003.

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

[7] V. Miranda and N. Fonseca, "EPSO - Best-of-two-worlds meta-heuristic applied to power system problems," in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1080-1085, 2002.

[8] C. Wei, Z. He, Y. Zheng and W. Pi, "Swarm directions embedded in fast evolutionary programming," in *Proceedings of the IEEE Congress on Evolutionary Computation,* pp.1278-1283, 2002.

[9] A. Stacey, M. Jancic, and I. Grundy, "Particle swarm optimization with mutation," in *Proceedings of the IEEE Congress on Evolutionary Computation*, Canberra, Australia USA, pp. 1425-1430, 2003.

[10] X. Yao, and Y. Liu, "Fast evolutionary programming", in *Proceedings of 5th Annual Conference on Evolutionary Programming*, San Diego, CA, pp. 451-460, 1996.

[11] M. Clerc, and J. Kennedy, "The particle swarm - explosion, stability, and convergence in a multidimensional complex space," *IEEE Trans. on Evolutionary Computation*, vol. 6, no.1, pp. 58-73, 2002.

[12] C. Trelea, "The particle swarm optimization algorithm: convergence analysis and parameter selection," *Information Processing Letters*, vol. 85, no. 6, pp. 317-325, 2003.

[13] R. A. Krohling, "Gaussian swarm: a novel particle swarm optimization algorithm," in *Proceedings of the IEEE Conference on Cybernetics and Intelligent Systems (CIS)*, Singapore, pp. 372-376, 2004.

[14] R. A. Krohling, F. Hoffmann, and L. S. Coelho, "Co-evolutionary particle swarm optimization for min-max problems using Gaussian distribution," in *Proceedings of the Congress on Evolutionary Computation*, Portland, Oregon USA, pp. 959-964, 2004.

[15] X. Yao, Y. Liu, G. Lin , "Evolutionary programming made faster," *IEEE Trans. on Evolutionary Computation*, vol. 3, no.2, pp. 82-102, 1999.

[16] X. Yao, Y. Liu, G. Lin , "Evolutionary programming using mutations based on Levi probability distribution," *IEEE Trans. on Evolutionary Computation*, vol. 8, no.1, pp. 1-24, 2004.

[17] G. Rudolph, "Self adaptive mutations may lead to premature convergence," *IEEE Trans. on Evolutionary Computation*, vol. 5, no. 4, pp. 410-414, 2001.