Bare Bones Particle Swarm Optimization with Gaussian or Cauchy Jumps

Renato A. Krohling, and Eduardo Mendel

Abstract— **Bare Bones Particle Swarm Optimization (BBPSO) is a powerful algorithm, which has shown potential to solving multimodal optimization problems. Unfortunately, BBPSO may also get stuck into local optima when optimizing functions with many local optima in high dimensional search space. In previous attempts an approach was developed which consists of a *jump strategy* combined with PSO in order to escape from local optima and promising results have been obtained. In this paper, we combine BBPSO with a jump strategy when no fitness improvement is observed. The jump strategy is implemented based on the Gaussian or the Cauchy probability distribution. The algorithm was tested on a suite of well-known benchmark multimodal functions and the results were compared with those obtained by the standard BBPSO algorithm and with BBPSO with re-initialization. Simulation results show that the BBPSO with the jump strategy performs well in all functions investigated. We also notice that the improved performance is due to a successful number of Gaussian or Cauchy jumps.**

Keywords – Bare Bones Particle Swarm, Gaussian and Cauchy probability distribution, nonlinear optimization, jumps

I. Introduction

PARTICLE swarm Optimization (PSO) is an algorithm developed by Kennedy and Eberhart in 1995 [1] with growing research interest. Clerc and Kennedy [2] introduced a PSO with a *constriction factor*, in which a range for the parameters of the algorithm is determined for stability of the swarm. Most PSO algorithms use uniform probability distribution to generate random numbers. However, approaches using Gaussian and Cauchy probability distributions have also been proposed in order to generate random numbers to update the velocity equation of PSO [3-11]. This section reviews those variations used in this study, from which concepts have been borrowed to develop the new algorithm.

Kennedy [3] developed an almost parameter free PSO, named Bare Bones PSO, for short, BBPSO, which consists of sampling new particle positions selected from a Gaussian distribution with the mean given by the average of the global and local best position and the standard deviation given by the absolute difference between the global best and the personal best position. Kennedy's Bare Bones PSO [3] was not the only method proposed but it is the simplest and very effective. This is the reason we investigate it further.

In [5] a strategy was developed to escape from local minima using jumps with promising results. In this paper, we combine the BBPSO [3] with the jump strategy borrowed from [5]. The new version of the algorithm was tested on a suite of well-known benchmark multimodal functions and the results were compared with those obtained by the standard BBPSO algorithm, and BBPSO with re-initialization. The simulation results demonstrate that the jump strategy can improve the performance of BBPSO to escape from local minima. The rest of the paper is organized as follows: In section 2, the canonical PSO (with constriction) and the BBPSO algorithms are described. In section 3, the BBPSO combined with Gaussian and Cauchy jump strategy is presented. Section 4 provides simulation results for some multimodal benchmark test functions followed by conclusions in section 5.

II. Canonical and Bare Bones Particle Swarm Optimization

The particle swarm optimization algorithm consists of, at each time step, changing the velocity of each particle moving toward its *pbest* and *gbest* locations (global version of PSO). Acceleration is weighted by random terms, with separate random numbers being generated for acceleration toward *pbest* and *gbest* locations, respectively. The PSO algorithm consists basically in updating the velocities and positions of the particle respectively as follows [2]:

$$\boldsymbol{v}_i(t+1) = \lambda \cdot [\boldsymbol{v}_i(t) + c_1 r_1 \cdot (\boldsymbol{p}_{best_i} - \boldsymbol{x}_i(t)) + c_2 r_2 \cdot (\boldsymbol{g}_{best} - \boldsymbol{x}_i(t))]. \quad (1)$$

$$\boldsymbol{x}_i(t+1) = \boldsymbol{x}_i(t) + \boldsymbol{v}_i(t). \quad (2)$$

with $\lambda = \dfrac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}$ and $\varphi = c_1 + c_2,\ \varphi > 4$

where:

Manuscript received November 7, 2008. This work was supported in part by FAPES/MCT/CNPq under Grant 37286374/2007.

R. A. Krohling and E. Mendel are with the Departmento de Informática, PPGI, Universidade Federal do Espírito Santo, CEP 29060-270, Vitória, ES, Brazil. Phone: +55-27-3335-2189; fax: +55-27-3335-2850; (e-mail: krohling.renato@gmail.com and mendel.eduardo@gmail.com).

978-1-4244-2959-2/09/$25.00 ©

- $\boldsymbol{x}_i = \left[x_{i1}, x_{i2}, .., x_{in}\right]^T$ is the position of the *i*-th particle in the *n*-dimensional search space.
- $\boldsymbol{v}_i = \left[v_{i1}, v_{i2}, .., v_{in}\right]^T$ is the velocity of the *i*-th particle.
- $\boldsymbol{p}best_i$ is the best previous *i*-th particle position.
- $\boldsymbol{g}_{best}$ is the index of the best particle among all particles.
- λ is the constriction factor.
- c_1 and c_2 are positive constants.
- $r_1, r_2 \sim U(0,1)$ are random numbers generated using the uniform probability distribution in the range [0;1].

Usually, when the constriction factor is used, φ is set to 4.1 $(c_1 = c_2 = 2.05)$, and the constriction factor λ is 0.729, but there are other possible choices for the constriction coefficients. A detailed analysis regarding the derivation of the constriction factor can be found in [2]. In this paper, we consider minimization problems unless stated otherwise. The Bare Bones PSO [3] is described in Algorithm 1.

The BBPSO eliminates the velocity term and the Gaussian distribution is used to sampling the search space based on the global best (*gbest*) and the personal best (*pbest*). So (1) and (2) are replaced by the following equation:

$$x_{i,j} = N(\mu_{i,j}, \sigma_{i,j}^2) \tag{3}$$

where *N* designates a Gaussian distribution with mean value $\mu_{i,j} = (g_{best\,j} + p_{best_{i,j}})/2$ and standard deviation $\sigma_{i,j} = \left|g_{best\,j} - p_{best_{i,j}}\right|$ for each variable $j = 1,...,n$ of the particle *i*.

III. Bare Bones Particle Swarm with Gaussian and Cauchy Jumps

The goal of this approach is to allow particles of the swarm escape from local minima when they are prematurely attracted to a local attractor. 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 are sampled according to (3). In the second case, by monitoring the fitness of a particle, if there is no fitness improvement, this indicates stagnation. Therefore, particles in stagnation should move (sample the search space) according to another dynamic regime. In such cases, the introduction of a jump strategy* allows new points to be sampled, which may help to escape from local attractors. This can be done by introducing a *stagnation interval*, which monitors the fitness value for each particle for a pre-specified number of iterations. If there is no fitness improvement for that particle, then the *stagnation interval* is increased by one in each iteration until this value achieves a pre-specified maximum number of iterations, named *maximum_stagnation_interval,* 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 [12-14]. The new position of the particles is now changed according to

$$\boldsymbol{x}_i(t+1) = \boldsymbol{p}_{best_i} \cdot \left(1 + \eta N(0,1)\right) \tag{4}$$

$$\boldsymbol{x}_i(t+1) = \boldsymbol{p}_{best_i} \cdot \left(1 + \eta C(0,1)\right) \tag{5}$$

where η denotes a scaling parameter. In (4), $N(0,1)$ stands for a random number generated according to a Gaussian distribution. In (5), $C(0,1)$ stands for a random number generated according to a Cauchy probability distribution with scaling parameter *t*=1 centred at the origin as described by

$$f(z) = \frac{1}{\pi}\frac{t}{t^2 + z^2}, \qquad -\infty < z < \infty \tag{6}$$

In both cases, the random numbers are generated anew for each variable $j = 1,...,n$ of the particle *i*. The BBPSO algorithm combined with a strategy to escape from local minima using Gaussian or Cauchy jumps is described in Algorithm 2.

Algorithm 1: BBPSO

Input parameters: Swarm size *P*, dimension *n*

// random initialization of a population of particles with
// positions $\boldsymbol{x}_i$ using uniform probability distribution, where
// $\underline{x}$ and $\overline{x}$ are the lower and upper bound respectively

FOR each particle *i*
  $\boldsymbol{x}_i = \underline{\boldsymbol{x}} + \left(\overline{\boldsymbol{x}} - \underline{\boldsymbol{x}}\right) \cdot U(0,1)$
  $\boldsymbol{p}_{best_i} = \boldsymbol{x}_i$
END FOR
$\boldsymbol{g}_{best} = \arg\min_{i=1,...,P}\{f(\boldsymbol{x}_i)\}$ // global best particle
DO
  FOR each particle *i*
    Update the position $\boldsymbol{x}_i$ according to (3)
    IF $f(\boldsymbol{x}_i) < f(\boldsymbol{p}_{best_i})$ THEN // update personal best
      $\boldsymbol{p}_{best_i} = \boldsymbol{x}_i$
    IF $f(\boldsymbol{x}_i) < f(\boldsymbol{g}_{best})$ THEN // update the global best
      $\boldsymbol{g}_{best} = \boldsymbol{p}_{best\,i}$
  END FOR
WHILE termination condition not met.
Output: $\boldsymbol{g}_{best}$

Algorithm 1: BBPSO

* In the context of PSO, the term jump is more suitable than mutation because particles move (fly) in the search space, not evolve.

Algorithm 2: BBPSO with jumps

Input parameters: Swarm size P, the scaling factor η and the *maximum_stagnation_interval*

// random initialization of a population of particles with
// positions $\boldsymbol{x}_i$ using uniform probability distribution,
// where $\underline{x}$ and $\overline{x}$ are the lower and upper bound, respectively

FOR each particle i

$\boldsymbol{x}_i = \underline{\boldsymbol{x}} + (\overline{\boldsymbol{x}} - \underline{\boldsymbol{x}}) \cdot U(0,1)$

$\boldsymbol{p}_{best_i} = \boldsymbol{x}_i$

END FOR

$\boldsymbol{g}_{best} = \arg \min_{i=1}^{P} \{ f(\boldsymbol{x}_i) \}$ // global best particle

DO

FOR each particle i

IF *stagnation_interval*[i] $\leq$ *maximum_stagnation_interval*

THEN Update the position $\boldsymbol{x}_i$ according to (3).

ELSE

Update the position $\boldsymbol{x}_i$ according to (4) or (5).

stagnation_interval[i]=0 // reset after a jump

END IF

IF $f(\boldsymbol{x}_i) < f(\boldsymbol{p}_{best_i})$ THEN // update personal best

$\boldsymbol{p}_{best_i} = \boldsymbol{x}_i$

ELSE // increase the *stagnation interval* by one

stagnation_interval[i]++ // no improvement

END IF

IF $f(\boldsymbol{x}_i) < f(\boldsymbol{g}_{best})$ THEN // update the global best

$\boldsymbol{g}_{best} = \boldsymbol{p}_{best_i}$

FOR each variable j of particle i // limit position

IF $x_{i,j} > \overline{x}$ THEN $x_{i,j} = p_{best_{i,j}}$

IF $x_{i,j} < \underline{x}$ THEN $x_{i,j} = p_{best_{i,j}}$

END FOR

END FOR

WHILE termination condition not met.

Output: $\boldsymbol{g}_{best}$

Algorithm 2. BBPSO with jumps

When the particles' positions in each dimension go beyond the lower or upper bound of the search space, instead of clamping those positions as standard in PSO, we set those positions to $p_{best_{i,j}}$ as can be seen in Algorithm 2. Experimental tests have been carried out and indicate that this small change in the algorithm influences very positively the convergence of BBPSO with jumps.

The scaling factor η used in the jump strategy investigated for BBPSO was held constant. BBPSO+GJ and BBPSO+CJ stands for BBPSO with Gaussian and Cauchy jump, respectively. For comparison purpose, we also carried out tests with a re-initialization strategy, similar to the approach described, but in this case the particle is randomly initialized according to $\boldsymbol{x}_i = \underline{\boldsymbol{x}} + (\overline{\boldsymbol{x}} - \underline{\boldsymbol{x}}) \cdot U(0,1)$. In the following, results are presented to show the suitability of the proposed approach.

IV. Simulation Results

A. Benchmark functions

In this work, we focus our attention on a suite of well-known benchmark functions with many local minima given in table I taken from [15]. This was exactly the motivation for introducing the jump strategy to escape from local minima. The benchmarks functions in our table I are numbered from f_1 to f_6 which correspond to the functions f_4 to f_9 respectively in table I [15, pp. 4].

B. Experimental settings

In all experiments the number of particles (population size) was set to 50 [16]. We used a non-uniformly (asymmetrically) initialization range as given in table I in order to make the optimization process more difficult. Each experiment is run 50 times and is terminated only when the maximum number of iterations (1500) has been reached.

For all algorithms initial simulations experiments have been carried out to determine the scaling factor η for each function tested and they are listed in table II. A rule of thumb to set up this scaling parameter may be found in [5]. The parameter *maximum_stagnation_interval*, which monitors the fitness (that is, if there is no fitness improvement) was set to 5, but other appropriate values can also be used. Of course, other criteria for monitoring fitness improvement or stagnation may also be used.

C. Results and discussion

The results obtained are summarized in table III. In cases where a value was $< 10^{-8}$ it was rounded to 0.0. Convergence graphs for these functions are shown in Appendix I. For functions $f_1(\boldsymbol{x})$ and $f_2(\boldsymbol{x})$ the algorithms BBPSO+GJ and BBPSO+CJ outperforms the BBPSO and BBPSO with re-initialization as can be seen in Fig. 1 and Fig. 2, respectively in terms of fast convergence as well as in terms of statistical results (see mean results in table III). The percentage (%) of successful jumps is calculated by the numbers of jumps that leads to points with better fitness value divided by the number of total jumps.

For functions $f_3(\boldsymbol{x})$ and $f_4(\boldsymbol{x})$ BBPSO as well as the BBPSO with jump and re-initialization achieves the optimal solution, but BBPSO with re-initialization presents faster convergence as can be seen in Fig. 3, and Fig. 4, respectively. The statistical results for these functions as given in table III show that BBPSO with Cauchy jump presents no difference in terms of mean values as compared to BBPSO with re-initialization.

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, I FOR THE ASYMMETRIC INITIALIZATION OF THE RANGE AND $f_{\min}$ IS THE MINIMUM VALUE OF THE FUNCTION [15].

Test Functions	n	S	I	$f_{\min}$
$f_1(\boldsymbol{x}) = -\sum_{i=1}^{n} x_i \sin(\sqrt{x_i})$	30	$(-500, 500)^n$	$(-500, 250)^n$	-12569.5
$f_2(\boldsymbol{x}) = \sum_{i=1}^{n} \left\{ x_i^2 - 10\cos(2\pi x_i) + 10 \right\}$	30	$(-5.12, 5.12)^n$	$(2.56, 5.12)^n$	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	$(-32, 32)^n$	$(16, 32)^n$	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	$(-600, 600)^n$	$(300, 600)^n$	0
$f_5(\boldsymbol{x}) = \frac{\pi}{n}\left\{ 10\sin^2(\pi y_1) + \sum_{i=1}^{n-1} (y_i - 1)^2 \left\{ 1 + 10\sin^2(\pi y_{i+1}) \right\} + (y_n - 1)^2 \right\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4),\ y_i = 1 + (1/4)\cdot(x_i + 1)$ $u(x, a, k, m) = \begin{cases} k\cdot(x-a)^m, & x > a \\ 0, & -a \le x \le a \\ k\cdot(-x-a)^m, & x < -a \end{cases}$	30	$(-50, 50)^n$	$(25, 50)^n$	0
$f_6(\boldsymbol{x}) = 0.1\left\{ \sin^2(3\pi x_1) + \sum_{i=1}^{n-1} (x_i - 1)^2 \left\{ 1 + \sin^2(3\pi x_{i+1}) \right\} + (x_n - 1)^2 \left\{ 1 + \sin^2(2\pi x_n) \right\} \right\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$	30	$(-50, 50)^n$	$(25, 50)^n$	0

TABLE II

VALUES USED IN THE EXPERIMENTS

Function	Scaling parameter η
$f_1(\boldsymbol{x})$	20
$f_2(\boldsymbol{x})$	1.1
$f_3(\boldsymbol{x})$	1.1
$f_4(\boldsymbol{x})$	1.1
$f_5(\boldsymbol{x})$	1.1
$f_6(\boldsymbol{x})$	0.1

TABLE III

SIMULATION RESULTS FOR THE BENCHMARK FUNCTIONS USING STANDARD BBPSO, BBPSO+GJ (WITH GAUSSIAN JUMP), BBPSO+CJ (WITH CAUCHY JUMP), AND BBPSO+R (WITH RE-INITIALIZATION). THE RESULTS ARE AVERAGED OVER 50 RUNS.

Function	Method	Best	Median	Mean	Standard Deviation	Worst	Successful jumps (%) or Re-initializations (%)
$f_1(\mathbf{x})$	BBPSO	-11029.8	-10200.7	-10179	316.649	-9529.56	n/a
	BBPSO+GJ	**-12569.5**	-12569.5	-12472.2	153.021	-11856.4	3.16
	BBPSO+CJ	**-12569.5**	-12451	-12426.7	136.627	-11977.3	2.06
	BBPSO+R	-10792.9	-10200.7	-10166.3	315.348	-9490.07	0.0028
$f_2(\mathbf{x})$	BBPSO	26.863	42.783	48.613	17.8403	113.425	n/a
	BBPSO+GJ	**0.0**	0.0	1.1689	4.006	23.879	1.36
	BBPSO+CJ	**0.0**	0.0	0.0	0.0	0.0	4.89
	BBPSO+R	9.949	17.411	17.889	4.703	30.843	0.41
$f_3(\mathbf{x})$	BBPSO	0.0	0.0	2.376	6.031	19.346	n/a
	BBPSO+GJ	0.0	0.0	0.0	0.0	0.0	5.33
	BBPSO+CJ	0.0	0.0	0.0	0.0	0.0	17.27
	BBPSO+R	0.0	0.0	0.0	0.0	0.0	0.73
$f_4(\mathbf{x})$	BBPSO	0.0	0.00985	0.0149	0.0172	0.0808	n/a
	BBPSO+GJ	0.0	0.0	0.0	0.00634	0.0369	2.39
	BBPSO+CJ	0.0	0.0	0.0	0.0	0.0	8.71
	BBPSO+R	0.0	0.0	0.0	0.0	0.0	0.54
$f_5(\mathbf{x})$	BBPSO	0.0	0.0	0.0601	0.129	0.622	n/a
	BBPSO+GJ	0.0	0.0	0.0352	0.0614	0.310	0.18
	BBPSO+CJ	0.0	0.0	0.0103	0.0314	0.103	0.69
	BBPSO+R	0.0	0.0	0.0	0.0	0.0	0.51
$f_6(\mathbf{x})$	BBPSO	0.0	0.0	0.0	0.00543	0.0109	n/a
	BBPSO+GJ	0.0	0.0	0.0	0.00750	0.0439	7.36
	BBPSO+CJ	0.0	0.0	0.0	0.00935	0.0439	9.72
	BBPSO+R	0.0	0.0	0.0	0.00801	0.0439	0.65

For functions $f_5(\mathbf{x})$ and $f_6(\mathbf{x})$ the optimal solution was found by all versions, but BBPSO with re-initialization converges quicker as illustrated in Fig. 5, and Fig. 6, respectively.

In summary, it can observed that BBPSO with Gaussian or Cauchy jump present very good performance, which can be seen in the successful percentage of jumps according to table III, that is, jumps that lead to better points in the search space, and therefore to better fitness value for all the 6 multimodal function investigated. The BBPSO with re-initialization shows good results especially in terms of faster convergence, but does not converged in all test functions as was the case for the functions $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$. In addition, we observe that BBPSO+R is able to reach the optimal selection only in the case where BBPSO also achieves the optimal solution.

We have also implemented BBPSO with Gaussian and Cauchy jumps with a growing scaling factor η but the results obtained have shown that increasing the scaling factor when in stagnation regime present almost the same results as the strategy with a constant scaling factor. Therefore, we do not report these results here. It is known that self-adaptive mutation may lead to premature convergence [14]. In such cases mutation using probability distributions like the Levi one [15], which provides larger jumps has demonstrated to be beneficial. In our study, the

approach is quite different from [15], because our BBPSO with jumps works based on two regimes. The jump strategy is used only when the particles exhibit no fitness improvement or get trapped in local minima. We have also implemented the jump strategy with Levi mutation but no significant results have been obtained yet and are not reported here.

V. Conclusions

In this paper, an approach combining Bare Bones Particle Swarm with a jump strategy to escape from local minima has been investigated. It consists of BBPSO with Gaussian Jump (BBPSO+GJ), and BBPSO with Cauchy jump (BBPSO+CJ). The jump strategy introduced increases the chances to escape from local minima, therefore improving the effectiveness of the algorithm. The BBPSO algorithms added with the jump strategy were studied on a suite of well-known benchmark functions with many local minima and compared with the standard BBPSO, and also with a BBPSO with re-initialization (BBPSO+R). The results showed that the BBPSO+GJ and BBPSO+CJ algorithms provided very good results for the multimodal benchmark functions tested. It turns out that the jump strategy is beneficial to escape from local minima. We also notice that the improved performance is due to a successful number of jumps, which can be generated using Gaussian or Cauchy probability distributions. However, other kinds of jump strategy are under investigation and will be reported in the future.

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] 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.

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

[4] 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.

[5] R. A. Krohling, "Gaussian Particle swarm with jumps" in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1080-1085, 2006.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] R. A. Krohling, L. dos S Coelho. "Co-evolutionary particle swarm using Gaussian distribution to solving constraint optimization problems," IEEE Trans. on Systems, Man and Cybernetics, part B Cybernetics, vol. 36, pp.1407-1416, 2006.

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

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

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

[15] 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.

[16] D. Bratton, J. Kennedy, "Defining a standard for particle swarm optimization," in *Proceedings of the IEEE Swarm Intelligence Symposium, (SIS07)*, pp. 120-127, 2007.

Appendix I

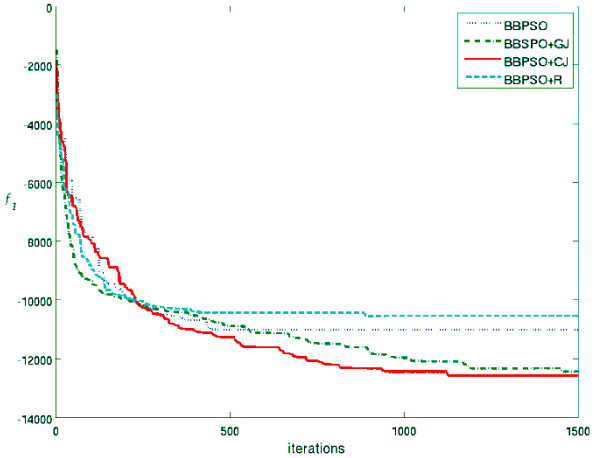

Fig. 1. Convergence for function $f_1(\boldsymbol{x})$

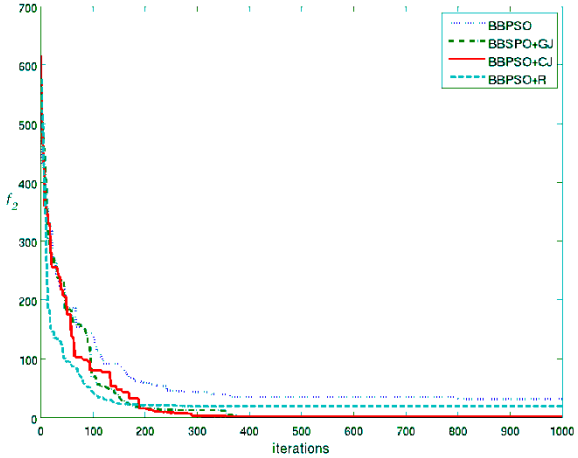

Fig. 2. Convergence for function $f_2(\boldsymbol{x})$

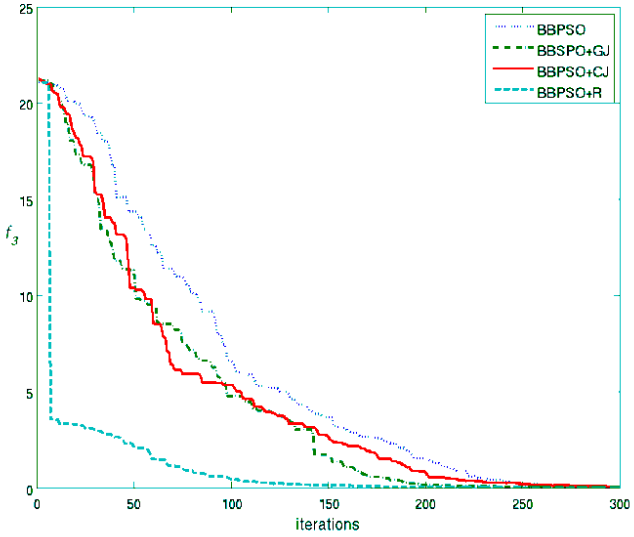

Fig. 3. Convergence for function f_3 ***(x)***

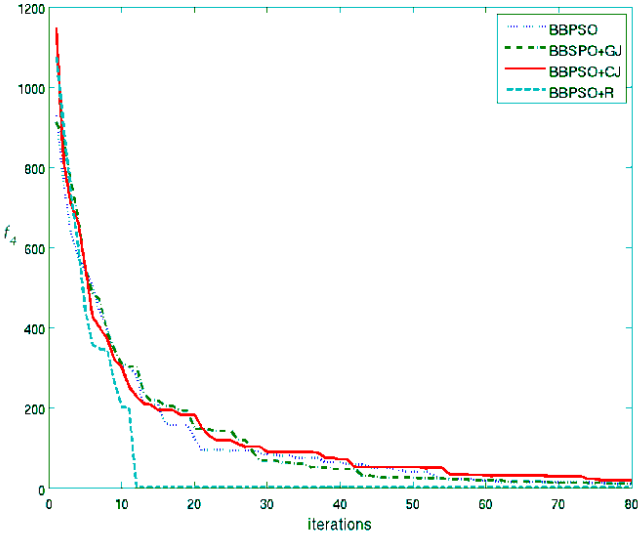

Fig. 4. Convergence for function f_4 ***(x)***

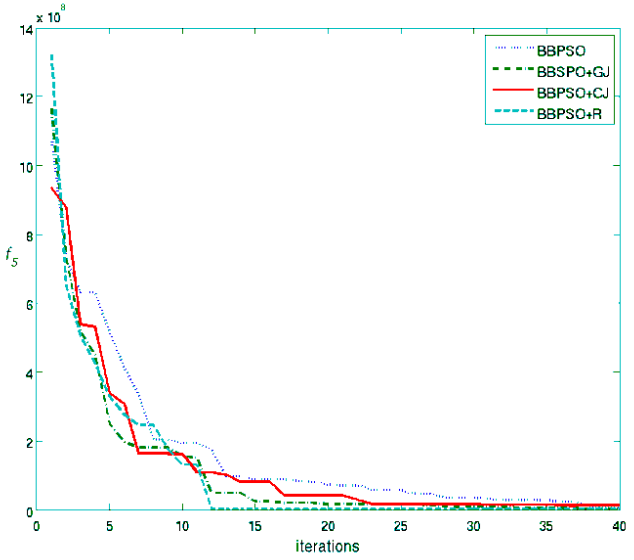

Fig. 5. Convergence for function f_5 ***(x)***

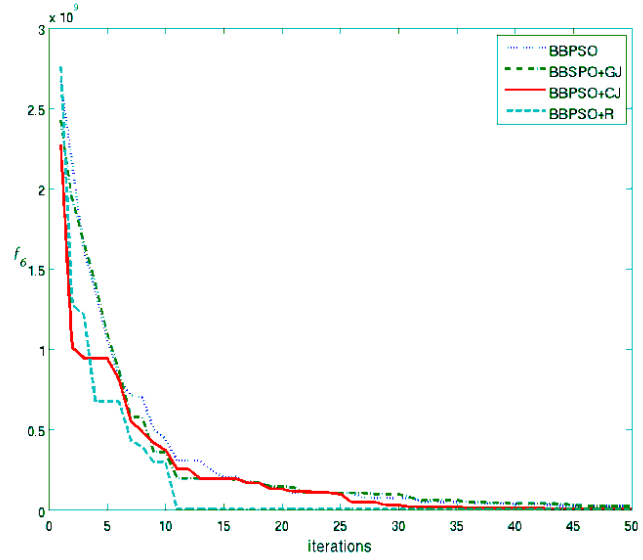

Fig. 6. Convergence for function f_6 ***(x)***