2008 IEEE Swarm Intelligence Symposium
St. Louis MO USA, September 21-23, 2008

Using Opposition-based Learning to improve the Performance of Particle Swarm Optimization

Mahamed G. H. Omran and Salah al-Sharhan

Abstract— **Particle swarm optimization (PSO) is a stochastic, population-based optimization method, which has been applied successfully to a wide range of problems. However, PSO is computationally expensive and suffers from premature convergence. In this paper, opposition-based learning is used to improve the performance of PSO. The performance of the proposed approaches is investigated and compared with PSO when applied to eight benchmark functions. The experiments conducted show that opposition-based learning improves the performance of PSO.**

I. INTRODUCTION

PARTICLE swarm optimization (PSO) [1,3] is a stochastic, population-based optimization method, which has been applied successfully to a wide range of problems as summarized in [4,5]. Similar to other population-based methods, PSO suffers from long computational time due to its stochastic nature. A number of PSO variations have been developed in the past decade to improve the performance of PSO [4,5].

Recently, opposition-based learning (OBL) was proposed by [6] and was successfully applied to several problems [7]. The basic concept of OBL is the consideration of an estimate and its corresponding opposite estimate simultaneously to approximate the current candidate solution. Opposite numbers were used by [7] to enhance the performance of Differential Evolution [8]. In addition, [2,16] used OBL to improve the performance of PSO. However, in both cases, several parameters were added to the PSO that are difficult to tune. Wang et al. [16] used OBL during swarm initialization and in each iteration with a user-specified probability. In addition, Cauchy mutation is applied to the best particle to avoid being trapping in local optima. Similarly, [2] used OBL in the initialization phase and also during each iteration. However, a constriction factor is used to enhance the convergence speed. In this paper, OBL is used to enhance the performance of PSO without adding any extra parameter. Opposite numbers are mainly used to replace the least-fit particle in the swarm.

The remainder of the paper is organized as follows: PSO is summarized in Section II. OBL is briefly reviewed in Section III. The proposed methods are presented in Section IV. Section V presents and discusses the results of the experiments. Finally, Section VI concludes the paper.

II. PARTICLE SWARM OPTIMIZATION

Particle swarm optimization (PSO) is a stochastic, population-based optimization algorithm modeled after the simulation of social behavior of bird flocks. In a PSO system, a swarm of individuals (called *particles*) fly through the search space. Each particle represents a candidate solution to the optimization problem. The position of a particle is influenced by the best position visited by itself (i.e. its own experience) and the position of the best particle in its neighborhood (i.e. the experience of neighboring particles). Particle position, $\boldsymbol{x}_i$, are adjusted using

$$\boldsymbol{x}_i(t+1) = \boldsymbol{x}_i(t) + \boldsymbol{v}_i(t+1) \tag{1}$$

where the velocity component, $\boldsymbol{v}_i$, represents the step size. For the basic PSO,

$$v_{i,j}(t+1) = wv_{i,j}(t) + c_1 r_{1,j}(t)(y_{i,j}(t) - x_{i,j}(t)) + c_2 r_{2,j}(t)(\hat{y}_j(t) - x_{i,j}(t)) \tag{2}$$

Manuscript received May 15, 2008.

M. G. Omran and S. al-Sharhan are with the Department of Computer Science, Gulf University for Science and Technology, Kuwait (phone: +965-886644; e-mail: {omran.m,alsharhans}@gust.edu.kw).

978-1-4244-2705-5/08/$25.00 ©2008 IEEE

where w is the inertia weight [9], c_1 and c_2 are the acceleration coefficients, $r_{1,j}$, $r_{2,j} \sim U(0,1)$, $\boldsymbol{y}_i$ is the personal best position of particle i, and $\hat{\boldsymbol{y}}_i$ is the neighborhood best position of particle i.

The neighborhood best position $\hat{\boldsymbol{y}}_i$, of particle i depends on the neighborhood topology used [10,11]. If a fully-connected topology is used, then $\hat{\boldsymbol{y}}_i$ refers to the best position found by the entire swarm. That is,

$$\begin{aligned}\hat{\boldsymbol{y}}_i(t) &\in \{\boldsymbol{y}_0(t), \boldsymbol{y}_1(t), \ldots, \boldsymbol{y}_s(t)\} \\ &= \min\{f(\boldsymbol{y}_0(t)), f(\boldsymbol{y}_1(t)), \ldots, f(\boldsymbol{y}_s(t))\}\end{aligned} \tag{3}$$

where s is the swarm size.

The resulting algorithm is referred to as the global best (*gbest*) PSO. For the ring topology, the swarm is divided into overlapping neighborhoods of particles. In this case, $\hat{\boldsymbol{y}}_i$ is the best position found by the neighborhood of particle i. The resulting algorithm is referred to as the local best (*lbest*) PSO. The Von Neumann topology defines neighborhoods by organizing particles in a lattice structure. A number of empirical studies have shown that the Von Neumann topology outperforms other neighborhood topologies [11,12]. It is important to note that neighborhoods are determined using particle indices, and are not based on any spatial information.

III. Opposition-based learning

Opposition-based learning (OBL) was first introduced by [6] and was successfully applied to several problems [7]. Opposite numbers are defined as follows:

Let $x \in [a,b]$, then the opposite number x' is defined as

$$x' = a + b - x$$

The above definition can be extended to higher dimensions as follows:

Let $P(x_1, x_2, \cdots, x_n)$ be an n-dimensional vector, where $x_i \in [a_i,b_i]$ and i=1, 2, …, n. The opposite vector of P is defined by $P'(x_1', x_2', \cdots, x_n')$ where

$$x_i' = a_i + b_i - x_i$$

IV. Opposition-based PSO

In this paper, OBL is used to enhance the performance of PSO without adding any extra parameter. The *gbest* PSO is used as the parent algorithm and the proposed opposition-based concepts are embedded into it. Three variants are proposed as follows:

A. OPSO:

This variant uses the initialization approach proposed by [7]. Hence, the initialization step of PSO is modified as follows:

1) Initialize the swarm randomly, i.e.,

$$x_{i,j} = LB_j + r_j \times (UB_j - LB_j)$$

where $r_j \sim U(0,1)$, $x_{i,j} \in [LB_j, UB_j]$, i =1,2,…,s, j=1,2,…,N_d and N_d is the dimension of the problem.

2) Calculate opposite swarm (swarm of *anti-particles*) by

$$ox_{i,j} = LB_j + UB_j - x_{i,j}$$

3) Select the s fittest particles from $\{\boldsymbol{X} \cup \boldsymbol{OX}\}$ as the initial swarm.

B. Improved OPSO (iOPSO):

An improved version of OPSO is proposed such that in each iteration the particle with the lowest fitness, $\boldsymbol{x}_b$, is replaced by its opposite (the *anti-particle*). The velocity and personal experience of the anti-particle are reset. The global best solution is also updated. A pseudo-code for *i*OPSO is shown in Alg. 1.

The rationale behind this approach is the basic idea of opposition-based learning: if we begin with a random guess, which is very far away from the existing solution, let say in worst case it is in the *opposite location*, then we should look in the *opposite direction*. In our case, the guess that is “very far away from the existing solution” is the particle with the lowest fitness.

C. Improved PSO (iPSO):

This variant is the same as the *i*OPSO, the only difference is that *i*PSO uses random initialization (as in the classical PSO) rather than using the initialization method of OPSO.

The main difference between *i*OPSO and *i*PSO on one side and the approaches proposed by [2,16] on the other side, is that we did not introduce any extra parameter to the original PSO. In addition, *i*OPSO and *i*PSO use only OBL to enhance the performance of PSO while [2,16] use OBL combined with other techniques (e.g. Cauchy mutation).

V. Experimental Results

This section compares the performance of the proposed methods with that of the *gbest* PSO algorithm discussed in Section II. For all the algorithms, w = 0.72 and $c_1 = c_2$ = 1.49. These values have been shown to provide very good results [13]-[15]. In addition s = 50. All functions were implemented in 30 dimensions except for the two-dimensional Camel-Back function.

The following functions have been used to compare the performance of the different approaches. These benchmark functions provide a balance of unimodal and multimodal functions.

For each of these functions, the goal is to find the global minimizer, formally defined as

Given f: $\Re^{N_d} \rightarrow \Re$

find $\boldsymbol{x}^* \in \Re^{N_d}$ such that $f(\boldsymbol{x}^*) \le f(\boldsymbol{x}), \forall \boldsymbol{x} \in \Re^{N_d}$

The following functions were used:

A. *Sphere* function, defined as

$$f(\boldsymbol{x}) = \sum_{i=1}^{N_d} x_i^2$$

where $\boldsymbol{x}^* = \mathbf{0}$ and $f(\boldsymbol{x}^*) = 0$ for $-100 \le x_i \le 100$.

B. *Step* function, defined as

$$f(\boldsymbol{x}) = \sum_{i=1}^{N_d} \left(\lfloor x_i + 0.5 \rfloor \right)^2$$

where $\boldsymbol{x}^* = \mathbf{0}$ and $f(\boldsymbol{x}^*) = 0$ for $-100 \le x_i \le 100$.

C. *Rosenbrock* function, defined as

$$f(x) = \sum_{i=1}^{N_d - 1} \left(100\left(x_i - x_{i-1}^2\right)^2 + \left(x_{i-1} - 1\right)^2 \right)$$

where $\boldsymbol{x}^* = (1,1,\ldots,1)$ and $f(\boldsymbol{x}^*) = 0$ for $-2 \le x_i \le 2$.

D. *Rastrigin* function, defined as

$$f(\boldsymbol{x}) = \sum_{i=1}^{N_d} \left(x_i^2 - 10\cos(2\pi x_i) + 10 \right)$$

where $\boldsymbol{x}^* = \mathbf{0}$ and $f(\boldsymbol{x}^*) = 0$ for $-5.12 \le x_i \le 5.12$.

E. *Ackley*'s function, defined as

$$f(x) = -20\exp\left(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{N_d} x_i^2} \right)$$

$$-\exp\left(\frac{1}{30}\sum_{i=1}^{N_d} \cos(2?x_i) \right) + 20 + e$$

where $\boldsymbol{x}^* = \mathbf{0}$ and $f(\boldsymbol{x}^*) = 0$ for $-32 \le x_i \le 32$.

F. *Griewank* function, defined as

$$f(\boldsymbol{x}) = \frac{1}{4000}\sum_{i=1}^{N_d} x_i^2 - \prod_{i=1}^{N_d} \cos\left(\frac{x_i}{\sqrt{i}} \right) + 1$$

where $\boldsymbol{x}^* = \mathbf{0}$ and $f(\boldsymbol{x}^*) = 0$ for $-600 \le x_i \le 600$.

G. *Salomon* function, defined as

$$f(x) = -\cos\left(2\pi \sum_{i=1}^{N_d} x_i^2 \right) + 0.1\sqrt{\sum_{i=1}^{N_d} x_i^2} + 1$$

where $\boldsymbol{x}^* = \mathbf{0}$ and $f(\boldsymbol{x}^*) = 0$ for $-100 \le x_i \le 100$.

H. *Six-Hump Camel-Back* function, defined as

$$f(\boldsymbol{x}) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$$

where $\boldsymbol{x}^* = (-0.08983, 0.7126), (-0.08983, 0.7126)$ and $f(\boldsymbol{x}^*) = -1.0316285$ for $-5 \le x_i \le 5$.

Sphere and Rosenbrock are unimodal, while the Step function is a discontinuous unimodal function. Rastrigin, Ackley, Griewank and Salomon are difficult multimodal functions where the number of local optima increases exponentially with the problem dimension. The Camel-Back function is a low-dimensional function with only a few local optima.

The results reported in this section are averages and standard deviations over 30 simulations. In order to have a fair comparison, each simulation was allowed to run for 50,000 evaluations of the objective function.

Table I summarizes the results obtained by applying the different approaches to the benchmark functions. The statistically significant best solutions have been shown in bold (using the *z*-test with $\alpha = 0.05$). In general, the results show that all the opposite-based variants performed better than (or equal to) *gbest* PSO. In particular, the results show that both *i*OPSO and *i*PSO outperformed the other methods. Table I shows also that there is no significant difference between *i*OPSO and *i*PSO which suggests that using the simple idea of replacing the worst particle is the main reason for improving the performance of *i*OPSO compared to OPSO. The number of function evaluations (FEs) required to reach an error value less than 10^{-6} (provided that the maximum limit is 50,000 FEs) was recorded in the 30 runs and the mean and standard deviation of FEs were calculated and shown in Table II. FEs can be used to compare the convergence speed of the different methods. A smaller FE means higher convergence speed. On the other hand, having FEs equal to 50,000 indicates that the approach cannot converge to the global optima. Table II shows that *i*OPSO and *i*PSO generally reached good solutions faster than the other approaches. Thus, the results show that using opposite numbers to replace the worst particle improve the convergence speed of both PSO and OPSO. Figure 1 illustrates results for selected functions. The figure shows that both *i*OPSO and *i*PSO generally reached good solutions faster than the other approaches. In general, the PSO was the slowest approach. Hence, we can conclude that opposition-based learning improved the performance of PSO without requiring any extra parameter.

VI. Conclusions

Opposition-based learning was used in this paper to improve the performance of PSO. Three opposition-based PSO variants were proposed (namely, OPSO, *i*OPSO and *i*PSO). OPSO employ opposition-based swarm initialization. OPSO is similar to the work of [2,16]. The main contribution of this paper is *i*OPSO and *i*PSO. The *i*OPSO algorithm uses opposition-based swarm initialization and it also replaces the least-fit particle with its anti-particle. The *i*PSO just replaces the worst particle with its opposite. The results show that, in general, *i*OPSO and *i*PSO outperformed the other approaches. In addition, the results show that using OBL enhances the performance of PSO without requiring additional parameters. The ideas introduced in this paper could also be used with any PSO variant.

Future research will investigate the effect of noise on the performance of the proposed approaches. Furthermore, a scalability study will be conducted. The use of opposition-based learning with other PSO variants will be explored. Finally, applying the proposed approaches to real-world problem will be investigated.

References

[1] R. Eberhart and J. Kennedy. A New Optimizer using Particle Swarm Theory. In Proceedings of the Sixth International Symposium on Micromachine and Human Science, pp. 39—43, 1995.

[2] L. Han and X. He. A novel Opposition-based Particle Swarm Optimization for Noisy Problems. In the Proceedings of the Third International Conference on Natural Computation, IEEE Press, vol. 3, pp. 624 – 629, 2007.

[3] J. Kennedy and R.C. Eberhart. Particle Swarm Optimization. In Proceedings of the IEEE International Joint Conference on Neural Networks, pp. 1942—1948, IEEE Press, 1995.

[4] A. Engelbrecht, Fundamentals of Computational Swarm Intelligence, Wiley & Sons, 2005.

[5] J. Kennedy and R.C. Eberhart and Y. Shi. Swarm Intelligence. Morgan Kaufmann, 2001.

[6] H. Tizhoosh. Opposition-based Learning: A New Scheme for Machine Intelligence. In Proc. Int. Conf. Comput. Intell. Modeling Control and Autom, Vienna, Austria, vol. I, pp. 695—701, 2005.

[7] S. Rahnamayan, H. Tizhoosh and M. Salama. Opposite-based Differential Evolution. IEEE Trans. On Evolutionary Computation, vol. 12(1), pp. 107—125, 2008.

[8] R. Storn and K. Price. Differential Evolution – A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces. Technical Report TR-95-012, International Computer Science Institute, Berkeley, CA, 1995.

[9] Y. Shi and R. Eberhart. A Modified Particle Swarm Optimizer. In Proceedings of the IEEE Congress on Evolutionary Computation, pp. 69—73, 1998.

[10] J. Kennedy. Small Worlds and Mega-Minds: Effects of Neighborhood Topology on Particle Swarm Performance. In Proceedings of the IEEE Congress on Evolutionary Computation, vol. 3, pp. 1931—1938, 1999.

[11] J. Kennedy and R. Mendes. Population Structure and Particle Performance. In Proceedings of the IEEE Congress on Evolutionary Computation, pp. 1671—1676, IEEE Press, 2002.

[12] E. Peer and F. van den Bergh and A. Engelbrecht. Using Neighborhoods with the Guaranteed Convergence PSO. In Proceedings of the IEEE Swarm Intelligence Symposium, pp. 235—242, IEEE Press, 2003.

[13] M. Clerc and J. Kennedy. The Particle Swarm-Explosion, Stability, and Convergence in a Multidimensional Complex Space. IEEE Transactions on Evolutionary Computation, vol. 6(1), pp. 58—73, 2002.

[14] F. van den Bergh. An Analysis of Particle Swarm Optimizers. Department of Computer Science, University of Pretoria, Pretoria, South Africa, 2002.

[15] F. van den Bergh and A. Engelbrecht. A Study of Particle Swarm Optimization Particle Trajectories. Information Sciences, vol. 176(8), pp. 937—971, 2006.

[16] H. Wang, Y. Liu, S. Zeng, H. Li, C. Li. Opposition-based Particle Swarm Algorithm with Cauchy Mutation. In the Proceedings of the IEEE Congress on Evolutionary Computation, pp. 4750-4756, IEEE Press, 2007.

TABLE I
MEAN AND STANDARD DEVIATION (±SD) OF THE FUNCTION OPTIMIZATION RESULTS

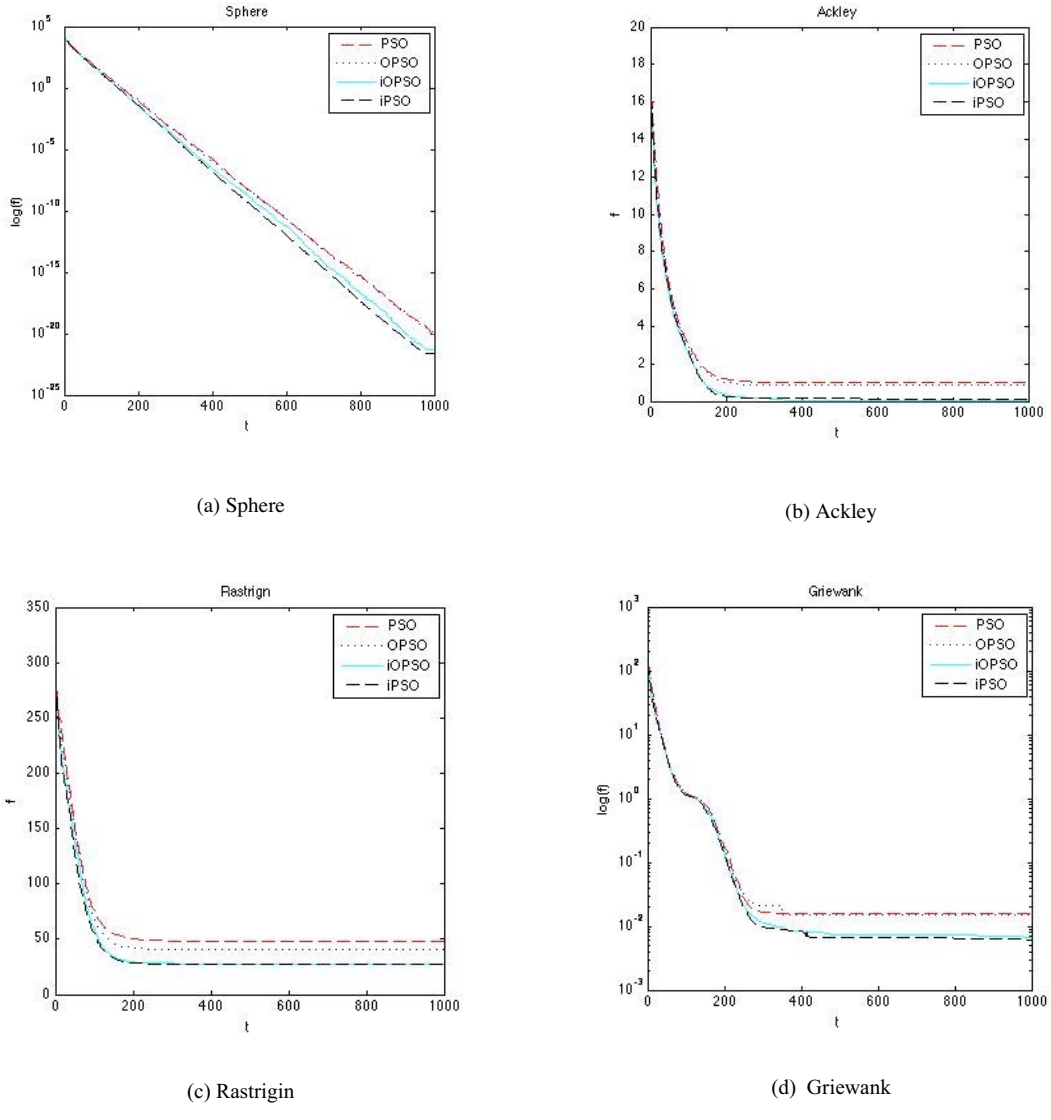
	PSO	OPSO	*i*OPSO	*i*PSO
Sphere	0(0)	0(0)	0(0)	0(0)
Rosenbrock	22.191441 (1.741527)	21.968519 (1.967754)	**20.761261 (0.396935)**	**20.645323 (0.426212)**
Step	0(0)	0(0)	0(0)	0(0)
Rastrigin	48.487584 (14.599249)	40.992256 (10.878502)	**27.858826 (12.185685)**	**27.460845 (11.966896)**
Ackley	1.096863 (0.953266)	0.833034 (0.999683)	**0(0)**	**0(0)**
Griewank	0.015806 (0.022757)	0.015158 (0.018313)	0.006668 (0.015211)	**0.006163 (0.009966)**
Salomon	0.446540 (0.122428)	0.499873 (0.146217)	**0.109884 (0.030509)**	**0.113207 (0.034575)**
Camel-back	-1.031628 (0)	-1.031628 (0)	-1.031628 (0)	-1.031628 (0)

TABLE II
MEAN AND STANDARD DEVIATION (±SD) OF THE NUMBER OF FUNCTION EVALUATIONS

	PSO	OPSO	*i*OPSO	*i*PSO
Sphere	19793.333333 (1276.088672)	19353.333333 (1263.610954)	19096.8 (1005.987194)	18841.8 (1075.69889)
Rosenbrock	50000(0)	50000(0)	50000(0)	50000(0)
Step	14650.0 (6392.722155)	13098.333333 (3877.910105)	9175.6 (1451.790253)	9126.3 (1462.34998)
Rastrigin	50000(0)	50000(0)	49234.533333 (3924.249550)	50000(0)
Ackley	42586.666667 (9932.747418)	39645.0 (9900.020463)	28599.8 (1787.104080)	27982.7 (1132.35605)
Griewank	38021.666667 (14939.883461)	41085.0 (13777.800423)	29464.433333 (13704.273604)	33144.2 (14660.289133)
Salomon	50000(0)	50000(0)	50000(0)	50000(0)
Camel-back	2108.333333 (221.703177)	1986.666667 (332.424047)	1957.4 (316.545862)	2117.2 (359.062515)

for each particle $i \in 1,...,s$ **do**
 for each dimension $j \in 1,...,N_d$ **do**
  $\boldsymbol{x}_{i,j} = LB_j + r_j \times (UB_j - LB_j)$
 endloop
endfor
for each particle $i \in 1,...,s$ **do**
 for each dimension $j \in 1,...,N_d$ **do**
  $\boldsymbol{ox}_{i,j} = LB_j + UB_j - \boldsymbol{x}_{i,j}$
 endloop
endfor
Select s fittest particles in X and OX as the initial swarm
for each particle $i \in 1,...,s$ **do**
 Set $\boldsymbol{v}_i$ to zero
 Set $\boldsymbol{y}_i = \boldsymbol{x}_i$
endfor
Repeat
 for each particle $i \in 1,...,s$ **do**
  Evaluate the fitness of particle i, $f(\boldsymbol{x}_i)$
  Update $\boldsymbol{y}_i$
  Update $\hat{\boldsymbol{y}}$ using equation (3)
  for each dimension $j \in 1,...,N_d$ **do**
   Apply velocity update using equation (2)
  endloop
  Apply position update using equation (1)
 endloop
Let $\boldsymbol{x}_b$ be the particle with the lowest fitness
for each dimension $j \in 1,...,N_d$ **do**
 $\boldsymbol{x}_{b,j} = LB_j + UB_j - \boldsymbol{x}_{b,j}$
endloop
$\boldsymbol{v}_b = \boldsymbol{0}$
$\boldsymbol{y}_b = \boldsymbol{x}_b$
if $f(\boldsymbol{x}_b) < f(\hat{\boldsymbol{y}})$
 $\hat{\boldsymbol{y}} = \boldsymbol{x}_b$
endif
Until some convergence criteria is satisfied

Algorithm 1: General pseudo-code for *i*OPSO



(a) Sphere

(b) Ackley

(c) Rastrigin

(d) Griewank

Fig. 1. Performance Comparison of the different methods when applied to selected functions