The Gregarious Particle Swarm Optimizer (G-PSO)

Srinivas Pasupuleti and Roberto Battiti
Dept. of Computer Science and Telecommunications
University of Trento, Italy
{srinivas, battiti}@dit.unitn.it

ABSTRACT

This paper presents a *gregarious* particle swarm optimization algorithm (G-PSO) in which the particles explore the search space by aggressively scouting the local minima with the help of only social knowledge. To avoid premature convergence of the swarm, the particles are re-initialized with a random velocity when stuck at a local minimum. Furthermore, G-PSO adopts a "reactive" determination of the step size, based on feedback from the last iterations. This is in contrast to the basic particle swarm algorithm, in which the particles explore the search space by using both the individual "cognitive" component and the "social" knowledge and no feedback is used for the self-tuning of algorithm parameters. The novel scheme presented, besides generally improving the average optimal values found, reduces the computation effort.

Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: Optimization—*Global Optimization*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Algorithms

Keywords

Particle swarm algorithm, Repeated Affine Shaker, Differential Evolution

1. INTRODUCTION

The problem being addressed is that of *minimizing* a function f mapping a vector $\mathbf{x} \in D \subseteq \mathbb{R}^d$ of d real numbers into a real number,

$$\min_{\boldsymbol{x} \in D} \quad f(\boldsymbol{x})$$
$$\text{where} \quad f : D \subseteq \mathbb{R}^d \to \mathbb{R}$$

where d is the dimension of the search space D. The goal is to find a global optimum $\mathbf{x}^*$, such that

$$f(\mathbf{x}^*) \le f(\mathbf{x}), \forall \mathbf{x} \in D$$

Population-based stochastic local search techniques are a new paradigm in the field of function optimization. Particle Swarm Optimization [11], PSO for short, is one of the simplest algorithms using a population of searchers, inspired by bird flocking phenomena and social behavior of humans. We refer to one of the earlier and highly used version of PSO presented in [18] as the Basic-PSO throughout the paper. In this paper we propose a *gregarious* particle swarm optimization algorithm (G-PSO), where the only information shared by the swarm members is the best value found during the previous search of the swarm up to the current iteration. To make this very simple version effective, an adaptive online tuning of the step size is executed during the search, motivated by the Reactive Search approach presented in [4], and a simple individual restart strategy is activated for each particle if there is evidence of stagnation. The experimental results confirm the original findings in [9] that "social-only" versions can easily outperform the complete model in certain cases. When coupled with the "reactive" and restart mechanisms considered in this paper the conclusion seems to hold not only for the simple case considered in [9], but for more general optimization tasks.

A brief description of Basic-PSO is presented in Section 2. The issues and related work done to improve the performance are discussed in Section 3. In Section 4 the G-PSO algorithm is presented. Different optimization algorithms used for comparison are discussed in Section 5. The simulation settings and results are presented in Section 6 and Section 7.

2. BASIC-PSO ALGORITHM

This brief summary of PSO is intended to fix the notation and to specify the precise version used in the comparison. We refer the readers to the cited bibliography for more details, see for example [18, 19]. The position and velocity vectors of the i-th particle in the d-dimensional search space at time t are denoted as $\mathbf{x}_i(t) \equiv \big(x_{i1}(t), \ldots, x_{id}(t)\big)$ and $\mathbf{v}_i(t) \equiv \big(v_{i1}(t), \ldots, v_{id}(t)\big)$. The best position of the i-th particle up to time t is referred to as $\mathbf{p}_i(t)$ and the global best position among all particles in the population up to time t is called $\mathbf{g}(t)$. The positions of particles are initialized by picking points at random and with uniform probability in an initialization range delimited by a lower bound L'_j and an upper bound U'_j: $x_{ij}(0) = rand(L'_j, U'_j)$, where

$L_j \leq L'_j < U'_j \leq U_j$. Note that the initialization range (L'_j, U'_j) is a subset of the over all search range (L_j, U_j) for reasons explained later in the paper. The initial velocity $\mathbf{v}_i(0)$ of particle i is chosen with uniform probability within suitable limits: $v_{ij}(0) = rand(-V_{max}, V_{max})$. The velocity and position update equations are as follows:

$$\begin{aligned} v_{ij}(t) &= w \cdot v_{ij}(t-1) + c_1 \cdot r_{1,j} \cdot (p_{ij}(t-1) - x_{ij}(t-1)) \\ &\quad + c_2 \cdot r_{2,j} \cdot (g_j(t-1) - x_{ij}(t-1)) \qquad (1) \\ x_{ij}(t) &= x_{ij}(t-1) + v_{ij}(t) \qquad (2) \end{aligned}$$

where $r_{1,j}$ and $r_{2,j}$ are different random numbers in the range $[0,1]$ following the uniform distribution. The parameters c_1 and c_2 are known as acceleration coefficients. The second and third terms on the right hand side of eqn. (1) represent the "cognitive" and "social" components, respectively. The velocity components of the particle v_{ij} are limited to a maximum allowable modulus V_{max}, as follows:

$$v_{ij} \longleftarrow \begin{cases} -V_{max} & \text{if } v_{ij} < -V_{max} \\ V_{max} & \text{if } v_{ij} > V_{max} \\ v_{ij} & \text{otherwise.} \end{cases} \qquad (3)$$

The value of V_{max} is defined as one half of the total search range. The term inertia weight w in equation (1) is decreased linearly with time as suggested in [17]:

$$w = (w_1 - w_2) \times \frac{(MAXITER - t)}{MAXITER} + w_2 \qquad (4)$$

where w_1 and w_2 are the initial and final values, respectively. The inertia weight controls the impact of the previous velocity: a large inertia weight favors exploration, while a small inertia weight favors exploitation. Thus, a high inertia weight at the beginning of the search helps in exploring the search space by avoiding local minima, while decreasing the inertia weight as the search proceeds helps in exploiting the search space and converging to the optimal solution [18]. t is the current iteration number and $MAXITER$ is the maximum number of allowed iterations before termination. Following the velocity and position updates, the personal best of a particle at time step t is updated as:

$$\mathbf{p}_i(t) \longleftarrow \begin{cases} \mathbf{p}_i(t-1) & \text{if } f(\mathbf{x}_i(t)) \geq f(\mathbf{p}_i(t-1)) \\ \mathbf{x}_i(t) & \text{otherwise} \end{cases} \qquad (5)$$

The global best position is asynchronously updated [6], as soon as each new best position is evaluated, by using the following equation:

$$\mathbf{g}(t) = arg\min\{f(\mathbf{p}_1(t)), f(\mathbf{p}_2(t)), \ldots, f(\mathbf{p}_s(t))\} \qquad (6)$$

where s is the swarm size. Fig. 1 summarizes the Basic-PSO algorithm.

```
Algorithm : BASIC-PSO($\mathbf{L}, \mathbf{U}, s, V_{max}, MAXITER$)

$t \leftarrow 1$
for $i \leftarrow 1$ to $s$
  do
    for $j \leftarrow 1$ to $d$
      do
        $x_{ij} \leftarrow rand(L'_j, U'_j)$
        $v_{ij} \leftarrow rand(-V_{max}, V_{max})$
    $\mathbf{p}_i \leftarrow \mathbf{x}_i$
$\mathbf{g} \leftarrow arg\min\{f(\mathbf{p}_0), f(\mathbf{p}_1), \ldots, f(\mathbf{p}_s)\}$
while $t \leq MAXITER$
  do
    for $i \leftarrow 1$ to $s$
      do
        for $j \leftarrow 1$ to $d$
          do
            Update Velocity using Eq: 1
            Limit Velocity using Eq: 3
            Update Position using Eq: 2
        Update personal best applying Eq: 5
        Update global best applying Eq: 6
    $t \leftarrow t + 1$
return ($\mathbf{g}$)
```

Figure 1: Basic PSO Algorithm

3. ISSUES AND RELATED WORK PSO

The major advantage of PSO over basic Evolutionary Algorithms (EAs) is that, in PSO, each individual benefits from its history whereas no such mechanism exists in the EAs [2, 7]. Each particle memorizes its previous velocity and the previous best position and uses them in its movement.

According to [2], although PSO finds good solutions much faster than other evolutionary algorithms, it usually cannot improve the quality of solutions as the number of iterations increases and it suffers from premature convergence when strongly multi-modal problems are being optimized. One of the main reasons is that all particles converge to a single point as the speed of the particles is decreased with time, thus forcing them to converge to the global best point found so far, which is not guaranteed even to be a local minimum. If a particle's current position is very close to the global best position, the particle will move away from this point only if its previous velocity and inertia weight w are non-zero.

Stochastic search techniques are usually problem dependent and thus an efficient parameter setting forms an important part of the algorithm. PSO is no exception: modifying a single parameter may result in a large effect [6]. For example, increasing the value of the inertia weight w will increase the speed of the particles resulting in more exploration and less exploitation. Many researchers tried to improve the performance by tuning the parameters of Basic-PSO [20], or by adding new parameters such as mutation [17]. The combination of PSO with other evolutionary algorithms has introduced new parameters and increased the computational effort [13], with varying degree of success for different problems. In [16] genetic programming is used as a tool to derive PSO variants.

We follow the opposite direction of research, which considers simplifications, elimination of parameters or automated self-tuning schemes so that the final user is not charged with the burden of a preliminary tuning phase. Ofcourse the complexity of the algorithm increases because of the internal feedback scheme, but this burden is on the shoulders of the researcher and not on the end user. As an example, a simplified version called "social-only" has been presented in [9], in which the particles are only guided by the social knowledge. The version is shown to perform better than the Basic-PSO for a very simple XOR problem. But the algorithm has not been widely considered and analyzed. Our proposal, G-

PSO, differs from this version because it does not take into account the previous velocity, the step size is adjusted in an adaptive manner during the search, therefore avoiding a preliminary tuning, and the velocity is re-initialized to displace the particles away from the local minimum when there is evidence of stagnation.

4. THE GREGARIOUS-PSO (G-PSO) ALGORITHM

In G-PSO, the population is attracted by the global best position and each particle is re-initialized with a random velocity if it is stuck close to the global best position. In this manner, the algorithm proceeds by aggressively and greedily scouting the local minima whereas Basic-PSO proceeds by trying to avoid them. Therefore a re-initialization mechanism is needed to avoid the premature convergence of the swarm.

In every iteration, each particle will either take a step along the direction towards the global best position or be re-initialized if it gets very close to the global best position. The particle's velocity is re-initialized in the range $[-V_{max}, V_{max}]$, if the Euclidean distance between its current position and the global best position is less than ϵ. The $d-$dimensional velocity vector $\mathbf{v}_i(t)$ is thus updated as follows:

$$\begin{array}{l} if(||\mathbf{x}_i(t-1) - \mathbf{g}(t-1)|| \leq \epsilon) \\ \qquad \forall j \; v_{ij}(t) = rand_j(-V_{max}, V_{max}) \\ \text{else} \\ \qquad \forall j \; v_{ij}(t) = \gamma \cdot rand_j(0,1) \cdot (x_{ij}(t-1) - g_j(t-1)) \end{array} \tag{7}$$

where $rand_j(-V_{max}, V_{max})$ and $rand_j(0,1)$ are independent random numbers for every dimension in the range $[-V_{max}, V_{max}]$ and $[0,1]$, respectively. Because the best position found by the swarm is the only shared information in the swarm, we call the algorithm the *gregarious* particle swarm optimization algorithm. The factor γ determines the step size of each particle in the direction of the global best position. Large values of γ will make the current particle i go past the global best position, thus resulting in oscillations and small values of γ will result in small step sizes which will lead to slow convergence to the global best position. Thus, γ is reactively adjusted by checking the improvement on the global best value found at the end of every iteration. The value of γ is bounded by the limits $[\gamma_{min}, \gamma_{max}]$ and is linearly adjusted at the end of every iteration, as follows:

$$\gamma \longleftarrow \begin{cases} max(\gamma - \delta, \gamma_{min}) & \text{if } (f(\mathbf{g}(t)) < f(\mathbf{g}(t-1)) \\ min(\gamma + \delta, \gamma_{max}) & \text{otherwise} \end{cases} \tag{8}$$

where δ is a constant. The values of the limits of γ and of δ used in our simulations are [2,4] and 0.5, respectively. The initial value of γ is set to 3.0. While the choice of the step size δ is robust, the motivation for determining the limits is as follows. Each component of the difference $(\mathbf{x}_i(t-1) - \mathbf{g}(t-1))$ in eqn. (7) is multiplied by a mean factor of 1.5 initially. If there is improvement in the global best value, the value of γ is decreased up to a minimum value equal to 2. Thus, as the performance improves, the particles tend to converge to the global best position, resulting in an aggressive search for local minima. If the global best value in not improved over the iterations, the value of γ is increased up to a maximum value 4.0, which would help the particles to explore the search space by oscillating around the current minimum, in order to find better directions. The G-PSO algorithm is summarized in the pseudo-code of Fig. 2. The difference between Figs. 1 and 2 is that there is no update of the personal best here, as the particles do not memorize their previous search history and the velocity update equation used is (7) instead of (1).

Algorithm : G-PSO($\mathbf{L}, \mathbf{U}, s, V_{max}, \gamma, MAXITER$)

```
$t \leftarrow 1$
for $i \leftarrow 1$ to $s$
    do for $j \leftarrow 1$ to $d$
        do $\{x_{ij} \leftarrow rand(L'_j, U'_j)$
$\mathbf{g} \leftarrow arg\min\{f(\mathbf{x}_0), f(\mathbf{x}_1), \ldots, f(\mathbf{x}_s)\}$
while $t \leq MAXITER$
    do for $i \leftarrow 1$ to $s$
           do for $j \leftarrow 1$ to $d$
                  do Update Velocity using Eq: 7
                     Limit Velocity using Eq: 3
                     Update Position using Eq: 2
              Update global best applying Eq: 6
       Adjust the step size $\gamma$ applying Eq: 8
       $t \leftarrow t + 1$
return ($\mathbf{g}$)
```

Figure 2: The Gregarious PSO Algorithm

The advantage of G-PSO is two-fold. Due to its greediness in scouting local minima, it finds promising regions rapidly during the initial phase of the search, while due to re-initialization with random velocities, the particles don't loose the global exploration capability and thus they will tend to find better optima as the search continues.

An analogy of the principle of G-PSO with nature can be as follows: consider a flock of birds looking for food. If one of the birds locates a region rich with food, all the birds get to that place, although with some exploratory oscillations around the place. As they see that there is no more food left, one after another they leave the place in search of more food, thereby exploring different food regions. The velocity update equation (7) makes all the particles converge close to the global best position found and the re-initialization with random velocities makes the particles explore the search region as they get very close to the global best position found.

5. OPTIMIZATION ALGORITHMS USED FOR COMPARISON

We compare the G-PSO with Basic-PSO, a variant of PSO called HPSO-TVAC [17] and two other stochastic local search techniques: Repeated Affine Shaker [5] and Differential Evolution [21].

5.1 HPSO-TVAC

HPSO-TVAC [17] is a self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. In this, the inertia weight term is set to zero in equation (1) and hence the movement of the particle is guided only by the "cognitive" and "social" components. If the particles stagnate along any dimension i.e., $v_{ij} \equiv 0$, the particles are re-initialized with a random velocity. The acceleration coefficients c_1 and c_2 are varied linearly with time to enhance the global search in the early part of the optimization and to make the particles converge toward global optima at the end of the search. The optimal performance has been observed when changing c_1 from 2.5 to 0.5 and changing c_2 from 0.5 to 2.5, over the full range of the search. We thus consider the same values in our simulation with re-initialization velocity equal to the maximum limit of the velocity V_{max}.

5.2 Repeated Affine Shaker

The Affine Shaker algorithm (AS) proposed in [5] is an adaptive random search algorithm based only on the knowledge of function values. The algorithm starts by choosing an initial point $\mathbf{x}$ in the configuration space and an initial search region R surrounding it. At every iteration, a new point is chosen randomly within the search region R. The search region is then modified according to the value of the function at the new point by using an affine transformation. It is compressed if the new function value is greater than the current one (unsuccessful sample) or expanded otherwise (successful sample). If the sample is successful, the new point becomes the current point, and the search region R is translated so that the current point is at its center for the next iteration. Once an improvement in the function value is found, the search region grows in the promising direction, causing a faster movement along that direction. The search region is compressed in every step that fails to find a better point along the chosen search direction. Each AS run is terminated when the size of steps for 8 consecutive times is less than 10^{-8}. By design, AS searches for local minimizers and is stopped as soon as one is found. A simple variant of AS is the Repeated Affine Shaker algorithm, in which the search is continued after a local minimum is found by restarting from a different initial random point. This leads to a "population" of AS searchers, but in this case each member of the population is independent, completely unaware of what other members are doing.

5.3 Differential Evolution

The Differential Evolution (DE) algorithm [21] is a stochastic search algorithm which uses a population of potential solutions to exploit the search space. DE borrows the concepts of *mutation*, *crossover* and *selection* to choose potential solutions for each generation. In the initial population the corresponding vectors are chosen randomly in the entire search space. For each generation, a mutant vector for a chosen target vector is generated by adding the weighted difference between two population vectors to a third vector. The mutated vector's parameters are then mixed with the parameters of the target vector, to yield a trial vector. If the trial vector is a better solution compared to the target vector, then the target vector is replaced by the trial vector. DE has been shown to be an efficient algorithm for global optimization with few control parameters namely, the *scaling factor* (F) and the *crossover factor* (CF). There are many variants of DE based on the choice of vector to be mutated, the number of difference vectors used and the type of crossover scheme. We adopt a well known variant of DE, "$DE/best/1/exp$" [15], in which the best vector is mutated with only one difference vector and the crossover scheme is exponential. The value of F is 0.5 and CF is 0.8.

6. EXPERIMENTAL SETTINGS

The benchmark functions are given in Table 1. The first two functions are unimodal whereas the next functions are multimodal. The global minimum for the first five functions is 0.0. The Schaffer f_6 function is maximized for a global maximum of 1.0 at origin. The global minimum for Shekel's foxholes f_7 function is at (-31.95,-31.95) with value equal to 0.998, approximated to three decimal places.

Table 1: Benchmarks for simulations, where d is the dimension of the function

Function Name	d	Mathematical Representation
Sphere $f_1(x)$	30	$\sum_{i=1}^{d} x_i^2$
Rosenbrock $f_2(x)$	30	$\sum_{i=1}^{d} \big(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\big)$
Rastrigin $f_3(x)$	30	$\sum_{i=1}^{d} \big(x_i^2 - 10\cos 2\pi x_i + 10\big)$
Griewank $f_4(x)$	30	$\frac{1}{4000}\sum_{i=1}^{d} x_i^2 - \prod_{i=1}^{d} \cos\frac{x_i}{\sqrt{i}} + 1$
Ackley $f_5(x)$	30	$-20e^{-0.2\sqrt{(\frac{1}{d}\sum_{i=1}^{d} x_i^2)}}$ $-e^{\frac{1}{d}\sum_{i=1}^{d} cos2\Pi x_i} + 20 + e$
Schaffer's $f_6(x)$	2	$0.5 - \frac{(\sin\sqrt{x^2+y^2})^2 - 0.5}{(1.0 + 0.001(x^2+y^2))^2}$
Shekel's foxholes $f_7(x)$	2	$(\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6})^{-1}$

The initialization and search ranges, already used by other researchers, are given in Table 2. The asymmetric initialization method adopted for the experiments suggests deliberately initializing the population in regions that do not include the global optimum for a fair evaluation [2, 3]. The constants c_1 and c_2 of Basic-PSO are both fixed at 2.0, and the inertia weight w is varied from $w_1 = 0.9$ at the beginning of the search to $w_2 = 0.4$ at the end [10]. The swarm size s is equal to 40 for the G-PSO, Basic-PSO and HPSO-TVAC, as suggested in [6]. For DE, the population size is equal to ten times the dimensionality of the search region [21]. For the Repeated Affine Shaker, there is only one searcher, and the algorithm is restarted when the modulus of subsequent steps becomes less than 10^{-8}. The parameter ϵ used to measure the proximity of global best position and current position of a particle in G-PSO is 10^{-8}. The algorithms are run for 100 trials and the average optimum value $f(\mathbf{g})$ is measured as a function of the number of function evaluations. The number of function evaluations is given by the product of MAXITER and swarm size s. The termination criterion is 200000 function evaluations for each run. The experiments are performed on a computer with Intel Xeon CPU 2.80GHz and 1GB of RAM, by using the gcc compiler.

Table 2: Search range and initialization range for the benchmark functions

Func.	Search range $[L_1, U_1] \times \cdots \times [L_d, U_d]$	Initialization range $[L'_1, U'_1] \times \cdots \times [L'_d, U'_d]$
f$_1$	$[-100, 100]^d$	$[50, 100]^d$
f$_2$	$[-100, 100]^d$	$[15, 30]^d$
f$_3$	$[-10, 10]^d$	$[2.56, 5.12]^d$
f$_4$	$[-600, 600]^d$	$[300, 600]^d$
f$_5$	$[-32, 32]^d$	$[15, 32]^d$
f$_6$	$[-100, 100]^2$	$[15, 30]^2$
f$_7$	$[-65.536, 65.536]^2$	$[0, 65.536]^2$

7. RESULTS

Figs. 3-9 show the comparison between G-PSO, Basic-PSO and the other stochastic algorithms. The plots are on a log-log scale. The graphs plot the average optimum value found as a function of the number of function evaluations except for the Schaffer function whose y-axis is one minus the average optimum, i.e., the distance from the optimal value of 1.

Except for the Schaffer and Griewank functions, G-PSO outperforms the Basic-PSO and HPSO-TVAC for all the benchmark functions. G-PSO improves the quality of the average optimum found as a function of the number of function evaluations and also leads to faster convergence against Basic-PSO. The Griewank function is a non-convex function with over 1000 optima in the range of interest, and the density of local minima increases as one approaches the global optimum. G-PSO, though faster at the beginning, tends to get stuck in one of the local minima.

The Schaffer function is a 2-dimensional maximization function with many circular valleys surrounding the global maximum of 1 at $(0, 0)$. It is designed to trap algorithms searching for local minima. In G-PSO the searchers tend to settle down on the first valley very close to the global maximum. The Ashaker and DE also find it hard to cross the valleys and converge prematurely. The 2-dim Shekel function contains 25 foxholes of varying depth surrounded by a relatively flat surface. The DE algorithm is stuck in the first foxhole it falls into. And, although the convergence rate is faster at the beginning in the case of Basic-PSO, both G-PSO and Basic-PSO reach the global minimum at approximately the same time.

The average optimal value obtained at the end of 200000 function evaluations is presented in Table 3. The error on the average is obtained by dividing the standard deviation of results by the square root of the number of runs. If the average optimum value or error is less than 10^{-6}, it is shown as 0 in the table. It can be observed that G-PSO tends to yield better average optimal values when compared to other well-known algorithms, with least error on the average.

The CPU time for each algorithm for 200000 function evaluations is presented in Table 4. DE has the least computational time of all the algorithms except for the last two functions. With functions f_5 and f_6, DE gets stuck in the first local minima it lands on and it spends lot of time on selection and crossover operations. The computation time of G-PSO is close to DE and is better than DE for the functions f_6 and f_7, due to the simplicity of the velocity update

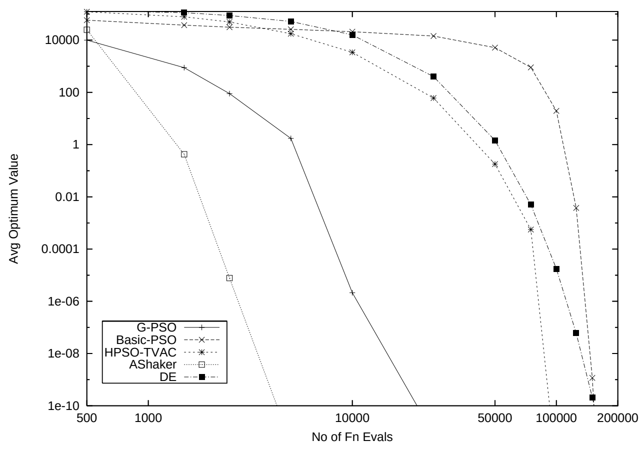

Figure 3: Experimental results on Sphere

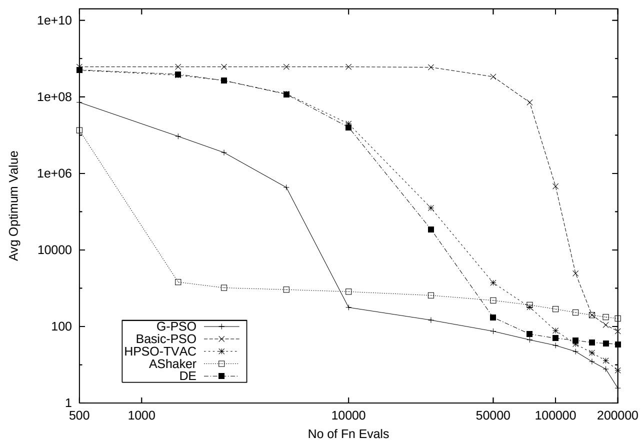

Figure 4: Experimental results on Rosenbrock

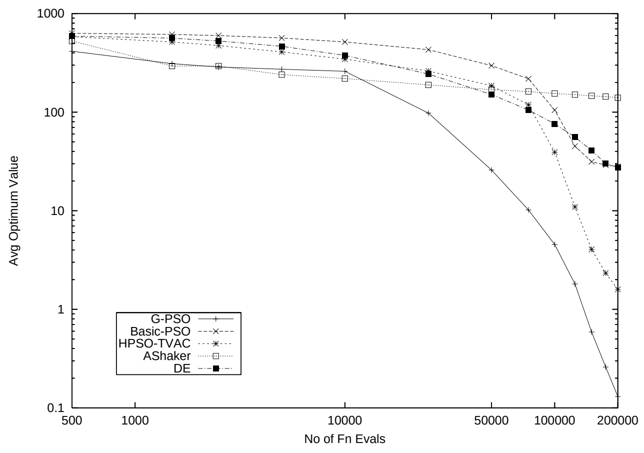

Figure 5: Experimental results on Rastrigin

Table 3: Average optimum value and standard deviation at the end of 200000 function evaluations

Function	G-PSO	Basic-PSO	HPSO-TVAC	AShaker	DE
f_1	**0** **(0)**	**0** **(0)**	**0** **(0)**	**0** **(0)**	**0** **(0)**
f_2	**2.46** **(10.14)**	75.30 (131.269)	7.14 (17.25)	161.921 (212.328)	34.35 (26.62)
f_3	**0.13** **(0.36)**	28.25 (7.75)	1.59 (3.63)	140.045 (16.95)	27.43 (16.4193)
f_4	0.066 (0.05)	0.0150 (0.016)	0.0157 (0.018)	**0** **(0)**	0.0035 (0.0057)
f_5	0.037 (0.205)	3.423 (7.564)	19.381 (0.287)	20.559 (2.04)	**0.0008** **(0.0047)**
f_6	0.002 (0.0039)	**0** **(0)**	0.00793 (0.00588)	0.0034 (0.0046)	0.0488 (0.089)
f_7	**0.998004** **(0)**	**0.998004** **(0)**	**0.998004** **(0)**	**0.998004** **(0)**	15.471 (6.605)

Table 4: CPU time in seconds at the end of 200000 function evaluations

Func.	G-PSO	Basic-PSO	HPSO-TVAC	AShaker	DE
f_1	0.52	0.80	0.89	24.66	**0.39**
f_2	0.55	0.80	0.89	26.37	**0.44**
f_3	1.03	1.28	1.34	24.91	**0.93**
f_4	1.27	1.51	1.59	26.14	**1.25**
f_5	1.09	1.35	1.39	24.87	**0.98**
f_6	**0.07**	0.09	0.09	0.12	0.30
f_7	**0.25**	0.27	0.27	0.27	1.52

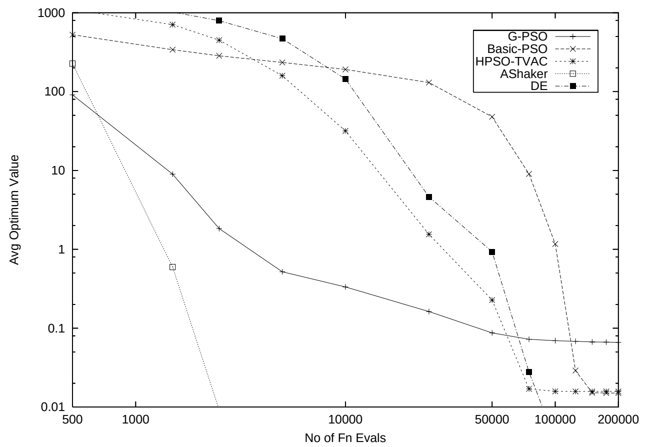

Figure 6: Experimental results on Griewank

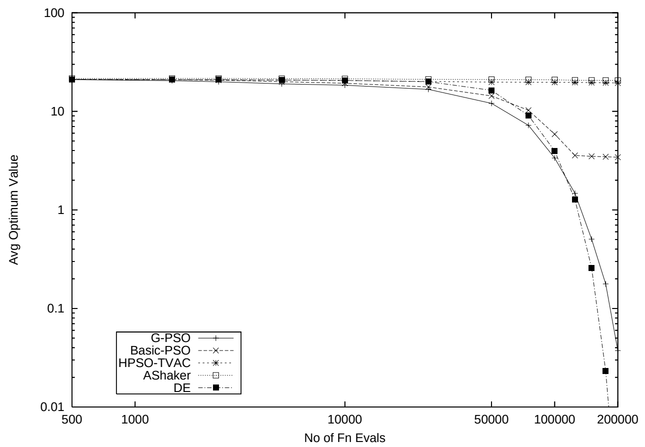

Figure 7: Experimental results on Ackley

Table 5: Number of runs(out of 100) to optimality and corresponding mean number of function evaluations rounded to nearest integer

Function	G-PSO	Basic-PSO	HPSO-TVAC	AShaker	DE
f_1	100 (9322)	100 (138515)	100 (84262)	**100 (2762)**	100 (112602)
f_2	**100 (295539)**	–	–	–	89 (462051)
f_3	**100 (177331)**	–	92 (189747)	–	–
f_4	12 (204027)	34 (144599)	45 (97071)	**100 (12356)**	68 (80436)
f_5	**100 (139772)**	83 (155829)	48 (285132)	–	100 216795
f_6	100 (134330)	**100 (48881)**	36 (176996)	100 (200070)	18 (120)
f_7	**100 (2572)**	100 (13881)	100 (26008)	100 (8667)	1 (440)

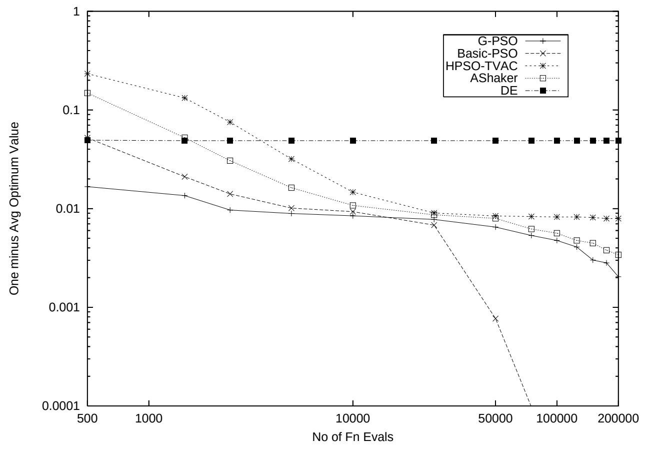

Figure 8: Experimental results on Schaffer

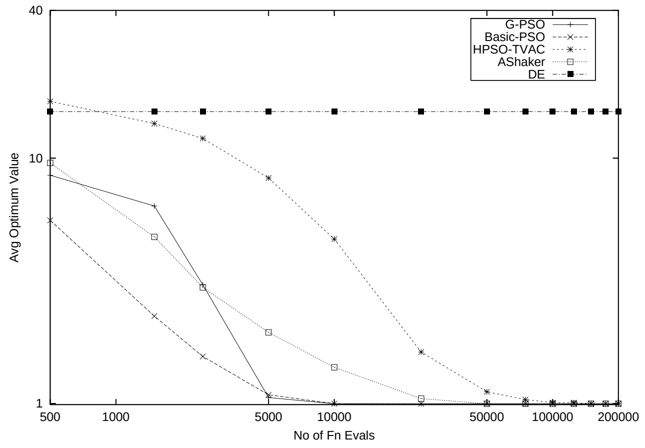

Figure 9: Experimental results on Shekel's foxholes

equation. The Affine Shaker gives better average optimal values for a couple of functions (Table 3), but it consumes a large CPU time as the algorithm involves high dimensional matrix-vector multiplications during the affine transformation of the search region.

To check whether the algorithms converge to global optimum or not, they are run until either the average optimum value is less than 10^{-6} (global minimum of all functions is zero except for Shekel function whose minimum is 0.998004) or until a maximum of 2×10^6 function evaluations. The algorithms are run for 100 trials for each benchmark function. Table 5 shows the number of runs which lead to convergence and the average number of function evaluations for the successful runs. The hyphen(-) mark indicates that none of the runs reached the optimal value. Except for the Griewank function which contains a huge number of local minima, G-PSO converges for all the 100 runs. Also, the mean number of function evaluations required for convergence tends to be less when compared with that of the other algorithms.

8. CONCLUSION

The motivation for considering the proposed gregarious PSO algorithm has been to see whether a simplified version of PSO, coupled with a dynamic tuning of the step size and a randomized restart when each particle is stuck could be as affective as more complex versions. In particular, the simplifying assumption is that the only information shared by the swarm particles is the best optimum found so far by the entire swarm.

Therefore the particles, instead of aiming at a compromise between social knowledge and personal knowledge, are "gregarious" as they aggressively search in the neighborhood of the best optimum found by the entire swarm, with randomized oscillations around the best position. The dynamic re-initialization allows the particles to explore the search space and find new local minima when they are stuck at a local optimum. The reactive adjustment of step size helps in exploiting and exploring the search space based on the success of the last iterations. The gregarious particle swarm optimization algorithm has a fast convergence rate, yielding in many cases better average optimal values with the least error on the average, when compared to the basic particle swarm algorithm and other well-known optimization algorithms. These findings generalize the preliminary results of [9] obtained for a simple case, reduce the emphasis on the "cognitive" aspects of PSO, and motivate a future more extended analysis. The ongoing work includes extensive comparison of G-PSO with different variants of Basic-PSO [1, 14, 22, 8] and evaluating the effectiveness of G-PSO on more benchmark functions [12, 23].

9. ACKNOWLEDGMENTS

This work is supported in part by the project CASCADAS (IST-027807) funded by the FET Program of the European Commission. The authors would like to thank the anonymous reviewers whose comments helped to increase the clarity of this paper.

10. REFERENCES

[1] Particle swarm central. *http://www.particleswarm.info/Programs.html*. Online; accessed 19-April-2006.

[2] P. J. Angeline. Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. *Evolutionary Programming VII*, pages 601–610, 1998.

[3] P. J. Angeline. Using Selection to Improve Particle Swarm Optimization. *IEEE International Conference on Evolutionary Computation*, pages 84–89, May 1998.

[4] R. Battiti and M. Brunato. Reactive search: Machine learning for memory-based heuristics. Technical Report DIT-05-058, Università di Trento, Sept. 2005. To appear as a chapter in the book: Teofilo F. Gonzalez (Ed.), Approximation Algorithms and Metaheuristics, Taylor & Francis Books (CRC Press), 2006.

[5] R. Battiti and G. Tecchiolli. Learning with first, second and no derivatives: A case study in high energy physiscs. *Neurocomp*, pages 181–206, 1994.

[6] A. Carlisle and G. Dozier. An off-the-shelf pso. *Proceedings of the Particle Swarm Optimization Workshop*, pages 1–6, April 2001.

[7] R. C. Eberhart and Y. H. Shi. Comparison between genetic algorithms and particle swarm optimization. In *Proceedings of the 7th International Conference on Evolutionary Programming*, volume 1447, pages 611–616. Springer, 1998.

[8] S. Janson and M. Middendorf. A hierarchical particle swarm optimizer and its adaptive variant. In *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, volume 35, pages 1272–1282, December 2005.

[9] J. Kennedy. The particle swarm: Social adaptation of knowledge. *IEEE International Conference on Evolutionary Computation*, pages 303–308, 1997.

[10] J. Kennedy. The behavior of particles. *7th Annual Conference on Evolutionary Programming*, pages 581–590, 1998.

[11] J. Kennedy and R. C. Eberhart. Particle Swarm Optimization. *IEEE Int. Conf. Neural Networks*, pages 1942–1948, 1995.

[12] J. J. Liang, P. N. Suganthan, and K. Deb. Novel composition test functions for numerical global optimization. *IEEE Swarm Intelligence Symposium*, pages 68–75, June 2005.

[13] M. Lovbjerg, T. K. Rasmussen, and T. Krink. Hybrid particle swarm optimizer with breeding and subpopulation. *Genetic And Evolutionary Computation Conference (GECCO'01)*, pages 469–476, July 2001.

[14] R. Mendes, J. Kennedy, and J. Neves. The fully informed particle swarm: simpler, maybe better. In *IEEE Transactions on Evolutionary Computation*, volume 8, pages 204–210, June 2004.

[15] N. Noman and H. Iba. Enhancing differential evolution performance with local search for high dimensional function optimization. *Genetic And Evolutionary Computation Conference (GECCO'05)*, pages 967–974, 2005.

[16] R. Poli, C. D. Chio, and W. B. Langdon. Exploring extended particle swarms: a genetic programming approach. *Genetic And Evolutionary Computation Conference (GECCO'05)*, pages 169–176, 2005.

[17] A. Ratnaweera, S. Halgamuge, and H. Watson. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. In *IEEE Transactions on Evolutionary Computation*, volume 8, pages 240–255, 2004.

[18] Y. H. Shi and R. C. Eberhart. Parameter selection in particle swarm optimization. *Annual Conference on Evolutionary Programming*, pages 591–600, March 1998.

[19] Y. H. Shi and R. C. Eberhart. Empirical study of particle swarm optimization. *Proc. IEEE Int. Congr. Evolutionary Computation*, pages 101–106, 1999.

[20] Y. H. Shi and R. C. Eberhart. Fuzzy adaptive particle swarm optimization. *Proc. IEEE Int. Congr. Evolutionary Computation*, pages 101–106, 2001.

[21] R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. In *Journal of Global Optimization*, volume 11, pages 341–359, 1997.

[22] F. van den Bergh and A. P. Engelbrecht. A cooperative approach to particle swarm optimization. In *IEEE Transactions on Evolutionary Computation*, volume 8, pages 225–239, June 2004.

[23] X. Yao, Y. Liu, and G. Lin. Evolutionary programming made faster. In *IEEE Transactions on Evolutionary Computation*, volume 3, pages 82–102, July 1999.