Empirical Study of Particle Swarm Optimization

Yuhui Shi  
EDS Indianapolis Technology Center  
12400 N. Meridian Street  
Carmel, IN 46032  
Yuhui.shi@mail.eds.com  
Phone: 317-705-6740  
Fax: 317-705-6710

Russell C. Eberhart  
Department of Electrical Engineering  
Purdue School of Engineering and Technology  
799 W. Michigan Street  
Indianapolis, IN 46202  
Phone: 317-278-0255  
Eberhart@engr.iupui.edu

Abstract- In this paper, we empirically study the performance of the particle swarm optimizer (PSO). Four different benchmark functions with asymmetric initial range settings are selected as testing functions. The experimental results illustrate the advantages and disadvantages of the PSO. Under all the testing cases, the PSO always converges very quickly towards the optimal positions but may slow its convergence speed when it is near a minimum. Nevertheless, the experimental results show that the PSO is a promising optimization method and a new approach is suggested to improve PSO’s performance near the optima, such as using an adaptive inertia weight.

1 Introduction

Through cooperation and competition among the population, population-based optimization approaches often can find very good solutions efficiently and effectively. Most of the population based search approaches are motivated by evolution as seen in nature. Four well-known examples are genetic algorithms [1], evolutionary programming [2], evolutionary strategies [3] and genetic programming [4]. Particle swarm optimization (PSO), on the other hand, is motivated from the simulation of social behavior. Nevertheless, they all work in the same way, that is, updating the population of individuals by applying some kinds of operators according to the fitness information obtained from the environment so that the individuals of the population can be expected to move towards better solution areas.

The PSO algorithm was first introduced by Eberhart and Kennedy [5, 6, 7, 8]. Instead of using evolutionary operators to manipulate the individuals, like in other evolutionary computational algorithms, each individual in PSO flies in the search space with a velocity which is dynamically adjusted according to its own flying experience and its companions’ flying experience. Each individual is treated as a volume-less particle (a point) in the D-dimensional search space. The ith particle is represented as $X_i = (x_{i1}, x_{i2}, \ldots, x_{id})$. The best previous position (the position giving the best fitness value) of the ith particle is recorded and represented as $P_i = (p_{i1}, p_{i2}, \ldots, p_{id})$. The index of the best particle among all the particles in the population is represented by the symbol g. The rate of the position change (velocity) for particle i is represented as $V_i = (v_{i1}, v_{i2}, \ldots, v_{id})$. The particles are manipulated according to the following equation:

$$v_{id} = v_{id} + c_1 \cdot \text{rand()} \cdot (p_{id} - x_{id}) + c_2 \cdot \text{Rand()} \cdot (p_{gd} - x_{id})$$ (1a)

$$x_{id} = x_{id} + v_{id}$$ (1b)

where $c_1$ and $c_2$ are two positive constants, and rand() and Rand() are two random functions in the range [0,1].

Unlike in genetic algorithms, evolutionary programming, and evolution strategies, in PSO, the selection operation is not performed [9, 10]. All particles in PSO are kept as members of the population through the course of the run (a run is defined as the total number of generations of the evolutionary algorithms prior to termination) [9]. It is the velocity of the particle which is updated according to its own previous best position and the previous best position of its companions. The particles fly with the updated velocities. PSO is the only evolutionary algorithm that does not implement survival of the fittest [9].

By considering equation (1b) as similar to a mutation operation, the PSO algorithm is similar to the evolutionary programming algorithm since neither algorithm performs a crossover operation. In evolutionary programming, each individual is mutated by adding a random function (the most commonly used random function is either a Gaussian or Cauchy function) [11, 12], while in PSO each particle (individual) is updated according to its own flying experience and the group’s flying experience. In other words, at each generation, each particle in PSO can only fly in a limited number of directions which are expected to be good areas to fly toward according to the group’s experience; while in evolutionary programming, each individual has the possibility to “fly” in any direction. That
is to say, PSO performs a kind of "mutation" operation with a "conscience" [13]. Theoretically speaking, evolutionary programming has more chance to "fly" into an area around the global optimal position while the PSO has more chance to "fly" into the better solution areas more quickly when the "conscience" provides helpful information.

In evolutionary programming, the balance between the global and local search is adjusted through adapting the variance (strategy parameter) of the Gaussian random function or step size, which can even be encoded into the chromosomes to undergo evolution itself. In PSO, a parameter called inertia weight is brought in for balancing the global and local search and equation (1) is changed to:

\[ v_i = w \cdot v_i + c_1 \cdot \text{rand} - x_i + c_2 \cdot \text{Rand} - x_i \]  \hspace{1cm} (2a)
\[ x_i = x_i + v_i \] \hspace{1cm} (2b)

where \( w \) is the inertia weight [13, 14]. The inertia weight has characteristics that are reminiscent of the temperature parameter in the simulated annealing [9]. A large inertia weight facilitates a global search while a small inertia weight facilitates a local search. By linearly decreasing the inertia weight from a relatively large value to a small value through the course of the PSO run, the PSO tends to have more global search ability at the beginning of the run while having more local search ability near the end of the run. The simulation results on the benchmark problem of Schaffer’s F6 function illustrate that an inertia weight starting with a value close to 1 and linearly decreasing to 0.4 through the course of the run will give the PSO the best performance compared with all fixed inertia weight settings [13].

In [15], Angeline compared the philosophy and performance differences between the evolutionary programming algorithm and PSO algorithm by conducting experiments on four non-linear functions well studied in the evolutionary optimization literature. The evolutionary programming algorithm employed is the one with a combination of Gaussian and Cauchy functions as strategy parameters' update function. This version of algorithm was first reported in [12] and has been shown to be superior to other update functions for the strategy parameters [16]. The PSO algorithm employed is the original one described by equation (1a) and (1b). Through adapting the strategy parameters to adjust the mutation step size, the evolutionary programming algorithm employed ideally has the ability to fine tune the search area around the optima. Since only the original version of PSO algorithm is involved in the comparison, no mechanisms are utilized in the PSO to adjust its velocity step size, therefore the PSO may lack some fine tuning ability. The experimental results reported in [15] shows that generally speaking the PSO has a quick convergence ability but a slow fine tuning ability while the evolutionary programming algorithm is the opposite. From the results, it can be excepted that by employing a dynamically adapting velocity step size approach, the PSO performance can be improved to have better fine tuning ability similar to that of evolutionary programming.

By introducing a linearly decreasing inertia weight into the original version of PSO, the performance of PSO has been greatly improved through experimental study on the benchmark problem of Schaffer’s F6 function [13, 14]. In order to further illustrate the effect of this linearly decreasing inertia weight, in this paper experimental results with the four non-linear testing functions used in [15] are reported and discussed.

## 2 Experimental Setting

For comparison, four non-linear functions used in [15] are used here. The first function is the Sphere function described by equation (3):

\[ f_1(x) = \sum_{i=1}^{n} x_i^2 \] \hspace{1cm} (3)

where \( x = [x_1, x_2, \ldots, x_n] \) is an \( n \)-dimensional real-valued vector. The second function is the Rosenbrock function described by equation (4):

\[ f_2(x) = \sum_{i=1}^{n-1} (100(x_{i+1} - x_i)^2 + (x_i - 1)^2) \] \hspace{1cm} (4)

The third function is the generalized Rastrigrin function described by equation (5):

\[ f_3(x) = \sum_{i=1}^{n} (x_i^2 - 10 \cos(2\pi x_i)) + 10 \] \hspace{1cm} (5)

The last function is the generalized Griewank function described by equation (6):

\[ f_4(x) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos \left( \frac{x_i}{\sqrt{i}} \right) + 1 \] \hspace{1cm} (6)

Following the suggestion in [11] and for the purpose of comparison, the asymmetric initialization method used in [15] is adopted here for population initialization. Table 1 lists the initialization ranges of the four functions.

**Table 1: Asymmetric initialization ranges.**

<table>
<thead>
<tr>
<th>Function</th>
<th>Asymmetric Initialization Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_1 )</td>
<td>(50,100) ( n )</td>
</tr>
<tr>
<td>( f_2 )</td>
<td>(15,30) ( n )</td>
</tr>
<tr>
<td>( f_3 )</td>
<td>(2.56,5.12) ( n )</td>
</tr>
<tr>
<td>( f_4 )</td>
<td>(300,600) ( n )</td>
</tr>
</tbody>
</table>

As in [15], for each function, three different dimension sizes are tested. They are dimension sizes: 10, 20 and 30.
The maximum number of generations is set as 1000, 1500 and 2000 corresponding to the dimensions 10, 20 and 30, respectively. In order to investigate whether the PSO algorithm scales well or not, different population sizes are used for each function with different dimensions. They are population sizes of 20, 40, 80, and 160. A linearly decreasing inertia weight is used which starts at 0.9 and ends at 0.4, with $c_1=2$ and $c_2=2$. $V_{max}$ and $X_{max}$ are set to be equal and their values for each function are listed in Table 2. A total of 50 runs for each experimental setting are conducted.

### Table 2: $V_{max}$ and $X_{max}$ values for each function.

<table>
<thead>
<tr>
<th>Function</th>
<th>$V_{max}$</th>
<th>$X_{max}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_0$</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>$f_1$</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>$f_2$</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>$f_3$</td>
<td>600</td>
<td></td>
</tr>
</tbody>
</table>

### 3 Experimental Results and Discussion

Figures 1 to 4 show the results for the sphere function with four different population sizes, respectively. Table 3 lists the mean fitness values of the best particle found for the 50 runs for the four functions. It is easy to see for the Sphere function, that PSO can find the optima very fast and the PSO algorithm also scales very well. In Table 3, since only four digits after the decimal are recorded, the values shown here are also zeros which can be seen from the Figures.

Figures 5 to 8 show the results for the Rosenbrock function with four different population sizes, respectively. Figures 9 to 12 show the results for the generalized Rastrigrin function with four different population sizes, respectively. Figures 13 to 16 show the results for the generalized Griewank function with four different population sizes.

Tables 4 to 6 list the mean fitness values of the best particle found for the 50 runs for the other three functions, respectively.

#### Table 3: The mean fitness values for the sphere function.

<table>
<thead>
<tr>
<th>Popu. Size</th>
<th>Dimension</th>
<th>Generation</th>
<th>Mean Best Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>10</td>
<td>1000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>1500</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>2000</td>
<td>0.0000</td>
</tr>
<tr>
<td>40</td>
<td>10</td>
<td>1000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>1500</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>2000</td>
<td>0.0000</td>
</tr>
<tr>
<td>80</td>
<td>10</td>
<td>1000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>1500</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>2000</td>
<td>0.0000</td>
</tr>
<tr>
<td>160</td>
<td>10</td>
<td>1000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>1500</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>2000</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

#### Table 4: Mean fitness values for the Rosenbrock function.

<table>
<thead>
<tr>
<th>Popu. Size</th>
<th>Dimension</th>
<th>Generation</th>
<th>Mean Best Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>10</td>
<td>1000</td>
<td>96.1715</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>1500</td>
<td>214.6764</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>2000</td>
<td>316.4468</td>
</tr>
<tr>
<td>40</td>
<td>10</td>
<td>1000</td>
<td>70.2139</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>1500</td>
<td>180.9671</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>2000</td>
<td>299.7061</td>
</tr>
<tr>
<td>80</td>
<td>10</td>
<td>1000</td>
<td>36.2945</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>1500</td>
<td>87.2802</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>2000</td>
<td>205.5596</td>
</tr>
<tr>
<td>160</td>
<td>10</td>
<td>1000</td>
<td>24.4477</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>1500</td>
<td>72.8190</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>2000</td>
<td>131.5866</td>
</tr>
</tbody>
</table>

#### Table 5: Mean fitness values for the generalized Rastrigrin function.

<table>
<thead>
<tr>
<th>Popu. Size</th>
<th>Dimension</th>
<th>Generation</th>
<th>Mean Best Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>10</td>
<td>1000</td>
<td>5.5572</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>1500</td>
<td>22.8892</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>2000</td>
<td>47.2941</td>
</tr>
<tr>
<td>40</td>
<td>10</td>
<td>1000</td>
<td>3.5623</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>1500</td>
<td>16.3504</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>2000</td>
<td>38.5250</td>
</tr>
<tr>
<td>80</td>
<td>10</td>
<td>1000</td>
<td>2.5379</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>1500</td>
<td>13.4263</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>2000</td>
<td>29.3063</td>
</tr>
<tr>
<td>160</td>
<td>10</td>
<td>1000</td>
<td>1.4943</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>1500</td>
<td>10.3696</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>2000</td>
<td>24.0864</td>
</tr>
</tbody>
</table>

#### Table 6: Mean fitness values for the generalized Griewank function.

<table>
<thead>
<tr>
<th>Popu. Size</th>
<th>Dimension</th>
<th>Generation</th>
<th>Mean Best Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>10</td>
<td>1000</td>
<td>0.0919</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>1500</td>
<td>0.0303</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>2000</td>
<td>0.0182</td>
</tr>
<tr>
<td>40</td>
<td>10</td>
<td>1000</td>
<td>0.0862</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>1500</td>
<td>0.0286</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>2000</td>
<td>0.0127</td>
</tr>
<tr>
<td>80</td>
<td>10</td>
<td>1000</td>
<td>0.0760</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>1500</td>
<td>0.0288</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>2000</td>
<td>0.0128</td>
</tr>
<tr>
<td>160</td>
<td>10</td>
<td>1000</td>
<td>0.0628</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>1500</td>
<td>0.0300</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>2000</td>
<td>0.0127</td>
</tr>
</tbody>
</table>

By looking at the shapes of the curves in all the figures, it is easy to see that the PSO converges quickly under all cases but will slow its convergence speed down when reaching the optima. This may be due to the use of the linearly decreasing inertia weight. By using the linearly decreasing inertia weight, the PSO lacks global search ability at the end of run even when the global search ability is required to
jump out of the local minimum in some cases. Nevertheless, the results shown illustrate that by using a linearly decreasing inertia weight, the performance of PSO can be improved greatly and have better results than that of both PSO and evolutionary programming reported in [15]. From the figures, it is also clear that the PSO with different population sizes has almost the similar performance. Similar to the observation for Sphere function, the PSO algorithm scales well for all four functions.

4 Conclusion

In this paper, the performance of the PSO algorithm with linearly decreasing inertia weight has been extensively investigated by experimental studies of four non-linear functions well studied in the literature. The experimental results illustrate that the PSO has the ability to quickly converge, the performance of PSO is not sensitive to the population size, and PSO scales well.

The results also illustrate that the PSO may lack global search ability at the end of a run due to the utilization of a linearly decreasing inertia weight. The PSO may fail to find the required optima in cases when the problem to be solved is too complicated and complex. But to some extent, this can be overcome by employing a self-adapting strategy for adjusting the inertia weight.

References
