An Improved Particle Swarm Optimizer with Momentum

Tao Xiang, Jun Wang, and Xiaofeng Liao

Abstract—In this paper, an improved particle swarm optimization algorithm with momentum (mPSO) is proposed based on inspiration from the back propagation (BP) learning algorithm with momentum in neural networks. The momentum acts as a lowpass filter to relieve excessive oscillation and also extends the PSO velocity updating equation to a second-order difference equation. Experimental results are shown to verify its superiority both in robustness and efficiency.

I. INTRODUCTION

PARTICLE swarm optimization (PSO) was firstly introduced by Kennedy and Eberhart in 1995. It is a metaphor of the social behavior of animals such as bird flocking and fish schooling[1], [2]. Investigations and applications have proved its capability and efficiency for various optimization problems.

PSO represents each potential solution of target function by particles directly. These particles are conceptual, and they do not possess volume or weight, but each of them has its own position coordinate and velocity. Their positions and velocities are randomly initialized firstly, all particles then “fly” through the solution space and update their positions until they find the optimal solution. During this iterative process, each particle’s velocity is adjusted according to its own experience and social cooperation.

Since the introduction of PSO, numerous variants have been emerged to enhance its capabilities in tackling various complex optimization problems. Almost all variants are proposed to balance the global exploration and local exploitation, keep the diversity of particles, and accelerate or guarantee the convergence. In this paper, an improved PSO algorithm with momentum is proposed. This inspiration is drawn from the improvement of BP learning algorithm with momentum in neural networks. The proposed method is tested on benchmarks and compared with other prevalent variants. Conclusions are made at the end.

II. STANDARD PSO ALGORITHM AND ITS VARIANTS

In this section, the standard PSO algorithm and two important variants are described in detail.

A. Standard PSO Algorithm

PSO is a population-based iterative stochastic optimization algorithm. In standard PSO (SPSO) algorithm, all particles compose a swarm. In an $n$-dimensional search space, at the $t$th-iteration, taking the $i$th particle into account, the position vector and the velocity vector can be represented as $X_i^t = (x_{i,1}^t, x_{i,2}^t, \ldots, x_{i,n}^t)$ and $V_i^t = (v_{i,1}^t, v_{i,2}^t, \ldots, v_{i,n}^t)$, respectively.

The velocity and position updating rules are respectively given as follows:

$$v_{i,j}^{t+1} = v_{i,j}^t + c_1 r_1(\hat{y}_{i,j}^t - x_{i,j}^t) + c_2 r_2(\hat{y}_j^t - x_{i,j}^t) \quad (1)$$

$$x_{i,j}^{t+1} = x_{i,j}^t + v_{i,j}^{t+1} \quad (2)$$

for $i = 1, 2, 3 \ldots p$; $j = 1, 2, 3 \ldots n$, where $p$ is the swarm size (i.e., the number of particles in the swarm), $n$ is the dimension of search space, $r_1$ and $r_2$ are uniformly distributed random numbers between 0 and 1 (i.e., $r_1, r_2 \in U(0, 1)$), $c_1$ and $c_2$ are positive constants referred as acceleration constants (a recommended value for them is $c_1 = c_2 = 2[1]$), $y_{i,j}^t$ is the $j$th element of the best position found so far by the $i$th particle, $\hat{y}_j^t$ is the $j$th element of the global best position found so far by all particles of the swarm. $y^t$ and $\hat{y}^t$ are updated as follows:

$$y_{i,j}^{t+1} = \begin{cases} y_{i,j}^t, & \text{if } f(x_{i,j}^{t+1}) \leq f(x_{i,j}^t) \\ x_{i,j}^{t+1}, & \text{if } f(x_{i,j}^{t+1}) > f(x_{i,j}^t) \end{cases} \quad (3)$$

$$\hat{y}_{j}^{t+1} = \arg\min_{y \in \{y_1, y_2, \ldots, y_p\}} f(y) \quad (4)$$

Generally, a maximum velocity ($V_{\text{max}}$) for each dimension of velocity is defined in order to clamp the excessive roaming of particles. This limitation is described in as follows:

$$v_{i,j}^t = \text{sign}(v_{i,j}^t) \cdot \min(V_{\text{max}}, \text{abs}(v_{i,j}^t)) \quad (5)$$

B. PSO Algorithm with Inertia Weight

The PSO algorithm with inertia weight (WPSO) was introduced in [3]. The inertia weight ($\omega$) is employed to balance global exploration and local exploitation. Large value of inertia weight facilitates better global exploration ability while small value of it enhances local exploitation capability. The velocity updating rule of this scheme is given as follows:

$$v_{i,j}^{t+1} = \omega v_{i,j}^t + c_1 r_1(\hat{y}_{i,j}^t - x_{i,j}^t) + c_2 r_2(\hat{y}_j^t - x_{i,j}^t) \quad (6)$$

It is pointed out that WPSO takes the least average number of iterations to find global optimum when $\omega$ takes the value in $[0.9, 1.2][3]$. In another paper[4], Shi and Eberhart empirically studied the parameter selection of inertia weight and suggested linearly decrease $\omega$, which substantially improved the performance of WPSO. At the beginning of iterations, large inertia weight helps particles explore new promising areas independently, while in the latter iterations, decreased
inertia weight facilitates fine-tuning the consensual area so as to accelerate convergence. A commonly used setting is to linearly decrease $\omega$ from 0.9 to 0.4[4].

C. PSO Algorithm with Constriction Factor

The other prevalent concept is constriction factor, which was introduced in [5]. The constriction factor ($\chi$) is a constant utilized to constrict the magnitude of flying velocity of particles. In this way, the excessive roaming out of the defined search space is controlled, convergence rate is accelerated meanwhile. Velocity updating rule of the PSO with constriction factor (CPSO) is given as follows:

$$v_{i,j}^{t+1} = \chi \left( v_{i,j}^t + c_1 r_1 (y_{i,j}^t - x_{i,j}^t) + c_2 r_2 (y_{i,j}^t - x_{i,j}^t) \right)$$

(7)

where

$$\chi = \frac{2\kappa}{2 - \phi - \sqrt{\phi^2 - 4\phi}},$$

(8)

$\phi = c_1 + c_2$ and $\kappa \in [0,1]$. The constant $\kappa$ controls the speed of convergence. When $\kappa \approx 0$, fast convergence to stable point is obtained, whereas a $\kappa \approx 1$ results in slow convergence[6]. In practice, $\kappa = 1$, $\phi = 4.1$ and $\chi = 0.729$ was recommended by Clerc et al.[5].

III. IMPROVING PSO WITH MOMENTUM

In this section, the PSO algorithm with momentum (mPSO) is introduced. This inspiration is drawn from the improvement of back-propagation (BP) learning algorithm with momentum in neural networks.

The BP algorithm is one of the most prevalent algorithms used in training multi-layer feedforward neural networks [7]. It uses gradient descent method to minimize the error between actual output and expected output, but it tends to stagnate or oscillate in the superficial local minima and fail to converge to global minimum. The momentum was introduced to deal with this issue[8]. It acts as a lowpass filter to smoothen the alteration of weight.

Its mathematical principle can be simply explained as follows:

$$y^t = (1-\lambda)x^t + \lambda y^{t-1}$$

(9)

for $0 \leq \lambda < 1$, where $x$ denotes the signal sequence needed to be smoothened, and $x^t$ the discrete signal value at time $t$. $y^t$ denotes filter’s output at time $t$. $\lambda$ is so called a momentum factor. The greater value $\lambda$ takes, the stronger smoothing capability it has.

Inspired by this, a momentum is introduced in the velocity updating equation. For simplicity, the momentum is applied in SPSO. The new velocity updating rule is given as follows:

$$v_{i,j}^{t+1} = (1-\lambda)[v_{i,j}^t+c_1 r_1 (y_{i,j}^t-x_{i,j}^t)+c_2 r_2 (y_{i,j}^t-x_{i,j}^t)]+\lambda v_{i,j}^{t-1}$$

(10)

Compared (10) with the original SPSO velocity updating equation (1), $\lambda v_{i,j}^{t-1}$ (i.e. the momentum), is added. $\lambda$, the momentum factor, indicates the effect of the momentum.

This variant is unprecedented because to the best of our knowledge all velocity updating rules in existing schemes are first-order difference equations about particle velocity, but in this variant, the momentum is employed to extend it to a second-order difference equation. mPSO has the following three advantages at least: Firstly, it is just as simple as SPSO both in representation and implementation because it does not introduce any complicated operator or data structure. Secondly, it smoothens the movement of particles, which can eliminate the oscillation at the end of iterations as well as the stagnation in local minima. Finally, it is a generalized technique which can be incorporated in all existing PSO schemes that have explicit velocity updating equations. For example, mPSO can be incorporated in hierarchical PSO[9], hybrid PSO, etc. where SPSO, WPSO, and CPSO are employed currently.

IV. EXPERIMENTS AND DISCUSSIONS

This section lays out experimental configurations, numerical results, as well as their discussions.

A. Experimental Setup

Six benchmarks, which are widely used in literature, are utilized to test our scheme and compare with other variants. Their mathematical representations and related parameters configuration are listed in Tables I. The threshold column in Table I indicates the tolerable error range for judging whether a running of an algorithm find priori global minimizer successfully.

The population size is assigned to 30 throughout the following experiments. Each algorithm is run for 5000 iterations in a running. In order to eliminate stochastic discrepancy, each algorithm is repeated 50 times. All criteria data will be calculated after each running and then their average values insist of final criteria data.

B. Criteria

Generally speaking, there are four criteria to evaluate the performance of a PSO algorithm; i.e., convergence precision error, success ratio, convergence speed, and convergence curve.

The convergence precision error ($\epsilon$) denotes the difference between priori minimum of an objective function and actual convergent value. The smaller, the better. $\epsilon$ equals to the final best value because the priori minimum of each benchmarks are all equal to 0.

The success ratio ($\gamma$) which indicates the robustness of a PSO algorithm can be defined as follows:

$$\gamma = \frac{\text{times of successful running}}{\text{times of total running}}$$

(11)

where a successful running means an algorithm converges to global minimizer at the global minimizer of a benchmark within its error threshold after a running.

The convergence speed indicates the cost of time to converge to global minimizer. But due to the discrepancy of test functions (because fitness evaluation is time-consuming and the difference of time overhead among evaluating disparate functions is significant), this criterion is often measured by the time of evaluation before convergence ($\tau$) instead, which
is exactly adopted in this paper. If an algorithm cannot converge within the error threshold on a benchmark, this value is asserted to inf.

Convergence curve plots the tendency of global best value of cost function during iterations. This criterion is widely given in literature to observe the convergence situation of a PSO algorithm or compare performance among several algorithms.

C. Results and Discussions

In our experiments, different values of λ (from 0.1 to 0.9 with 0.1 increment) are deployed and compared to other prevalent variants. It is found that mPSO gains best performance when λ takes the values between 0.2 and 0.4. Therefore, only the results of mPSO with λ = 0.2, 0.3, and 0.4 are shown here. The detailed results are presented in Table II and Fig. 1.

In Table II, as a whole, the criteria data of mPSO are better than those of SPSO, WPSO, and CPSO on average. In particular, mPSO with λ = 0.2 has the best robustness as its success ratio is the highest on five benchmarks except for f6, and it gains 100% success ratio on four benchmarks (f1 to f4). But its best cost function value is not noticeable as it only wins in Rastrigin and Ackley functions. As for the convergence speed, mPSO with λ = 0.3 and λ = 0.4 has its advantages, mPSO with λ = 0.4 wins on four benchmarks (f1, f2, f3, and f6) and that with λ = 0.3 wins on two benchmarks (f2 and f5). When the best cost function value is considered, victors are distributed more dispersely, mPSO with λ = 0.2 to 0.4 still win on four benchmarks (f1, f2, f3, and f5).

In order to give a detailed performance view for the algorithms, the convergence curve are also drawn in Fig. 1. For a benchmark, the mPSO whose ε is the best is plotted. In Fig. 1, the convergence curves of mPSO always descend rapidly during iterations and reaches the lowest point compared to those of SPSO, WPSO, and CPSO.

V. Conclusion

In this paper, an improved PSO algorithm with momentum is proposed. The momentum acts as a lowpass filter to relieve the excessive oscillation of particles in superficial minima and accelerate convergence rate. It also extends PSO velocity updating equation to a second-order difference equation. In order to investigate its performance, a series of experiments are conducted on the benchmarks. The results are compared to other prevalent versions (SPSO, WPSO, CPSO) under several criteria. From experimental results, it is concluded that the mPSO algorithm with λ = 0.2 to 0.4 has superior performance both in efficiency and robustness. It gains the best robustness when λ = 0.2, whereas greater value of λ (λ = 0.3 or 0.4) gives it more efficiency in finding better resolution in less iterations.

The proposed mPSO algorithm does not introduce any time-consuming operation, complicated topological structure, or profound velocity updating rule. It actually belongs to a foundational variant like WPSO and CPSO, but outperforms them significantly. Undoubtedly, mPSO can also be used in all extended variants where SPSO, WPSO, and CPSO are employed currently.

REFERENCES

Fig. 1. Convergence curves of mPSO, SPSO, WPSO, and CPSO on different benchmarks.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$J_1$</th>
<th>$J_2$</th>
<th>$J_3$</th>
<th>$J_4$</th>
<th>$J_5$</th>
<th>$J_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>mPSO ($\lambda=0.2$)</td>
<td>4.3695e-055</td>
<td>18.1544</td>
<td>1.7942</td>
<td>20.6266</td>
<td>20.0180</td>
<td>17.3432</td>
</tr>
<tr>
<td>mPSO ($\lambda=0.3$)</td>
<td>1.6099e-080</td>
<td>3.0106</td>
<td>1.7942</td>
<td>20.6266</td>
<td>20.0180</td>
<td>17.3432</td>
</tr>
<tr>
<td>mPSO ($\lambda=0.4$)</td>
<td>1.9148e-087</td>
<td>20.6266</td>
<td>20.0180</td>
<td>17.3432</td>
<td>17.3432</td>
<td>17.3432</td>
</tr>
</tbody>
</table>

2007 IEEE Congress on Evolutionary Computation (CEC 2007) 3345