Recent Advances in Adaptive Particle Swarm Optimization Algorithms

The Jin Ai and Voratas Kachitvichyanukul
School of Engineering and Technology, Asian Institute of Technology
P.O. Box 4, Klong Luang, Pathumthani 12120, Thailand
E-mail thejin.ai@ait.ac.th, voratas@ait.ac.th

Abstract

This paper reviews recent literature on the mechanisms for adapting parameters of particle swarm optimization (PSO) algorithm. The review covers the mechanisms for adaptively setting such parameters as inertia weight, acceleration constants, number of particles and number of iterations. In additions, a more thorough review of the mechanisms for adapting inertia weight and acceleration constants is made. Detailed description of two specific mechanisms that are reviewed in detailed include the velocity index pattern for adapting the inertia weight and the relative gaps between various learning terms and the best objective function values for adapting the acceleration constants. An example to demonstrate the mechanisms are illustrated by using the GLNPSO for a specific optimization problem, namely, the vehicle routing problem. The preliminary experiment indicates that the addition of the proposed adaptive mechanisms can provide good algorithm performance in terms of solution quality with a slightly slower computational time.

Keywords: Particle swarm optimization, metaheuristic, algorithm’s parameter, adaptive PSO, VRP.

1. Introduction

Particle swarm optimization (PSO) [1, 2, 3] has recently been successfully applied to solve many combinatorial optimization problems including job shop scheduling problem [4] and vehicle routing problem [5, 6]. Similar to other evolutionary computing methods, PSO has several parameters that must be properly set in order to yield good algorithm performance. The main PSO parameters include inertia weight, acceleration constants, number of particles and number of iterations. Finding the best parameter set for a specific optimization problem is not an easy task, since the same parameter set may yield different performance on different problem cases. Usually, many experiments are required over many problem cases to determine proper values of these parameters. However, there is no guarantee that the selected parameter set will always yield the best algorithm performance, especially when the algorithm is applied to solve a new problem case.

A novel approach to replace the parameter-tuning experiments is through an adaptive mechanism that can adapt PSO parameters autonomously whenever it is applied to solve a problem instance. It is noted that the concept of adaptive algorithm is also present in the wider scope of evolutionary computing method, i.e. in the genetic algorithm [7, 8]. Also, some earlier works in PSO, which will be further reviewed and discussed in Section 3 of this paper, have dealt with the issue of how to adaptively set its parameters.

The main objective of this paper is to give a perspective on adaptive PSO algorithms. To start with, the GLNPSO, a PSO Algorithm with multiple social learning structures [9], is briefly reviewed in Section 2, altogether with the main role of its parameters. Section 3 reviews the existing adaptive mechanism in the literature and presents other alternative mechanisms. Some selected mechanisms are finally embedded into the GLNPSO Algorithm and applied to a specific optimization problem in Section 4. Finally, Section 5 summarizes the material presented in this paper and recommends further works.

2. GLNPSO Algorithm

The GLNPSO Algorithm is a PSO Algorithm with multiple social learning structures [9]. Instead of using only the global best, it also incorporates the local best and near-neighbor best as additional social learning factors. Therefore, in the velocity updating equation, it requires three different acceleration constants related to each social learning factor. The detail of the GLNPSO Algorithm is presented below.

Notation:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau$</td>
<td>Iteration index, $\tau = 1\ldots T$</td>
</tr>
<tr>
<td>$l$</td>
<td>Particle index, $l = 1\ldots L$</td>
</tr>
<tr>
<td>$h$</td>
<td>Dimension index, $h = 1\ldots H$</td>
</tr>
<tr>
<td>$u$</td>
<td>Uniform random number in the interval $[0,1]$</td>
</tr>
<tr>
<td>$w$</td>
<td>Inertia weight</td>
</tr>
<tr>
<td>$\omega_{lh}(\tau)$</td>
<td>Velocity of the particle $l$ at the dimension $h$ in the iteration $\tau$</td>
</tr>
<tr>
<td>$\theta_{lh}(\tau)$</td>
<td>Position of the particle $l$ at the dimension $h$ in the iteration $\tau$</td>
</tr>
<tr>
<td>$\psi_{lh}$</td>
<td>Personal best position (pbest) of the particle $l$ at the dimension $h$</td>
</tr>
<tr>
<td>$\psi_{gb}$</td>
<td>Global best position (gbest) at the dimension $h$</td>
</tr>
<tr>
<td>$\psi_{lbh}$</td>
<td>Local best position (lbest) of the particle $l$ at the dimension $h$</td>
</tr>
<tr>
<td>$\psi_{nbest}$</td>
<td>Near neighbor best position (nbest) of the particle $l$ at the dimension $h$</td>
</tr>
<tr>
<td>$c_p$</td>
<td>Personal best position acceleration constant</td>
</tr>
<tr>
<td>$c_g$</td>
<td>Global best position acceleration constant</td>
</tr>
<tr>
<td>$c_l$</td>
<td>Local best position acceleration constant</td>
</tr>
<tr>
<td>$c_n$</td>
<td>Near neighbor best position acceleration constant</td>
</tr>
</tbody>
</table>
GLNPSO Algorithm (For minimization):
1. Initialize a swarm with \( L \) particles; generate the particle \( l \) with random position \( \Theta_i \) in the range \([\theta^\text{min}, \theta^\text{max}]\), velocity \( \Omega = 0 \) and personal best \( \Psi_i = \Theta_i \) for \( l = 1 \ldots L \). Set iteration \( \tau = 1 \).

2. For \( l = 1 \ldots L \), decode \( \Theta_i(\tau) \) to a problem specific solution \( R_i \).

3. For \( l = 1 \ldots L \), compute the performance measurement of \( R_i \), and set this as the fitness value of \( \Theta_i \), represented by \( Z(\Theta_i) \).

4. Update pbest: For \( l = 1 \ldots L \), update \( \Psi_i = \Theta_i \), if \( Z(\Theta_i) < Z(\Psi_i) \).

5. Update gbest: For \( l = 1 \ldots L \), update \( \Psi_g = \Psi_i \), if \( Z(\Psi_i) < Z(\Psi_g) \).

6. Update lbest: For \( l = 1 \ldots L \), among all pbest from \( K \) neighbors of the particle \( l \), set \( \psi^g_{l, o} = \psi^g_{l, o} \) that maximizing fitness-distance-ratio \( FDR \) for \( o = 1 \ldots H \). Where \( FDR \) is defined as

\[
FDR = \frac{Z(\Theta_i) - Z(\Psi_g)}{\theta^\text{max} - \psi^g_{l, o}} \quad \text{which} \quad l \neq o
\]

7. Generate nbest: For \( l = 1 \ldots L \), and \( h = 1 \ldots H \), set \( \psi^g_{l, h} = \psi^g_{l, h} \) that maximizing fitness-distance-ratio \( FDR \) for \( o = 1 \ldots H \). Where \( FDR \) is defined as

\[
FDR = \frac{Z(\Theta_i) - Z(\Psi_g)}{\theta^\text{max} - \psi^g_{l, o}} \quad \text{which} \quad l \neq o
\]

8. Update the velocity and the position of each particle \( l \) :

\[
\begin{align*}
\omega_l(\tau+1) & = c_a(\psi^g_{l, o} - \theta_l(\tau)) + c_r(\psi^g_{l, o} - \theta^p_l(\tau)) + \omega_l(\tau) \\
\theta_l(\tau+1) & = \theta_l(\tau) + \omega_l(\tau+1) \\
\end{align*}
\]

If \( \theta_l(\tau+1) < \theta^\text{min} \), then

\[
\begin{align*}
\theta_l(\tau+1) & = \theta^\text{min} \\
\omega_l(\tau+1) & = 0
\end{align*}
\]

9. If the stopping criterion is met, i.e. \( \tau = \tau^\text{stop} \), stop. Otherwise, \( \tau = \tau + 1 \) and return to step 2.

It can be seen from the algorithm above that there are some parameters that are required by GLNPSO, including inertia weight \( w \), acceleration constants \( c_a, c_r, c_p \), number of particles \( L \) and number of iterations \( T \). The inertia weight and acceleration constants play very important role in the velocity updating equation (Eq. 2). Since the velocity drives the movement of particles from one position to the next (Eq. 3), it implies that the movement of the swarm of particles as a searching agent in PSO is affected by these parameters. Movement of the swarm is closely linked to the algorithm performance, since each distinct position may correspond to different solution and the final solution obtained by PSO must be one of the positions that have been visited by the swarm. Therefore, the number of particles and the number of iterations are also related to the algorithm performance, since these parameters partially determine the number of positions visited by the swarm. However, simply increase the number of particles and number of iterations does not always improve the algorithm performance, since the velocity updating mechanism also depends on the cognitive learning (pbest) and social learning (gbest, lbest, and nbest). In addition, the number of particles also has influence on the social information values and theirs updating behavior.

3. Parameters Adaptation

3.1 Inertia Weight

Among PSO parameters, inertia weight has received enormous attention since the early development of PSO. The proper setting of inertia weight is believed to lead to good performance of PSO algorithm. Instead of using constant inertia weight, A linear decreasing function has been proposed for setting the inertia weight [10]. The GLNPSO algorithm described above is implemented by using following expression as the inertia weight \( w \) in Eq. 2:

\[
w(\tau) = w(T) + \frac{\tau - T}{1 - T}[w(1) - w(T)]
\]

where

\[
w(\tau) : \text{Inertia weight in iteration } \tau
\]

Similar to this approach, a nonlinear decreasing function was proposed for setting inertia weight [11]. With these decreasing inertia weight settings, it is expected that the particles are able to explore the problem space more widely at the beginning of iteration steps and to exploit promising solution in the later iteration steps. As seen in Eq. 2, the inertia term is the product of the inertia weight times the previous velocity. Therefore, applying large inertia weight at the beginning causes the particles to maintain their previous velocity and makes the particles move more aggressively in the early iteration. When this inertia weight is step by step reduced at the later iteration steps, the particles are influence less by previous velocity and their movements are influence more by theirs cognitive and social learning information.
Adaptive PSO was proposed [12] that alternating its inertia weight between a high value and a low value in order to control the swarm’s velocity. For this purpose, the velocity index of the swarm (\(\tilde{v}\)) is defined by the expression given in Eq. 9. The index can be continuously observed from iteration to iteration:

\[
\tilde{v} = \frac{\sum_{i=1}^{L} \sum_{h=1}^{H} |v_{hi}|}{L \cdot H}
\]  
\[(9)\]

Then, the swarm velocity index is compared with the target velocity (\(\omega^*\)), which is a linear decreasing function:

\[
\omega^* = \left(1 - \frac{\tau}{T}\right) \omega^{\text{max}}
\]  
\[(10)\]

Whenever the velocity index is bigger than the target velocity, the low value of inertia weight is selected. Conversely, the inertia weight is set at the high value when the velocity index is smaller than the target velocity.

A study of the dynamic behavior of the swarm in PSO was carried out in [13]. The main idea in [13] stated that different pattern should be used in order to achieve balance between exploration and exploitation process. This balance is often mentioned as the key to a good performance of PSO. The velocity index pattern is used as mechanism to balance between the two phases so that half of iterations focused on exploration and the other half concentrated on exploitation. For example, two-step linear decreasing pattern can be selected to portray this condition, in which the target velocity follows this expression:

\[
\omega^* = \begin{cases} 
1.8r & 0 \leq \tau \leq T/2 \\
0.2r & T/2 \leq \tau \leq T
\end{cases}
\]  
\[(11)\]

By using Eq. 11, the target velocity index is gradually decreased from \(\omega^{\text{max}}\) at the first iteration to \(0.1\omega^{\text{max}}\) at the first half of iterations. It is expected that the problem space is well explored by the swarm in this phase, so that the swarm is able to exploit the existing solutions during the second half of iterations when the desired velocity index is small enough and is slowly reduced from \(0.1\omega^{\text{max}}\) to 0.

The inertia weight can also be set in the range of minimum (\(\omega^{\text{min}}\)) and maximum (\(\omega^{\text{max}}\)). The updating principle is similar to the existing work: whenever the swarm velocity index is lower than the desired velocity index, the inertia weight is increased, and reversely when the swarm velocity index is greater than the desired velocity index, the inertia weight is decreased. It can be defined that the amount of increases or decreases of inertia weight depends on the difference between the velocity index of the swarm and the target velocity index. An example of equations that are used to update inertia weight is as follow:

\[
\Delta \omega = \omega^{\text{max}} - \omega^{\text{min}}
\]  
\[(12)\]

\[
w = w + \Delta w
\]  
\[(13)\]

\[
w = \begin{cases} 
\omega^{\text{max}} & w > \omega^{\text{max}} \\
\omega^{\text{min}} & w < \omega^{\text{min}}
\end{cases}
\]  
\[(14)\]

\[
w = \begin{cases} 
\omega^{\text{max}} & \tau < \tau^{\text{max}} \\
\omega^{\text{min}} & \tau > \tau^{\text{max}}
\end{cases}
\]  
\[(15)\]

Other proposed mechanisms to adaptively adjust the inertia weight are based on the value of local best and global best at a particular iteration [14] or the population diversity of the swarm [15, 16, 17]. In addition, fuzzy logic rules based on the swarm fitness values also had been proposed to adaptively adjust the inertia weight [18, 19].

Instead of using single value of inertia weight for the whole swarm, Feng et al. [20] used the velocity and the acceleration component of each particle to set the individual inertia weight. Panigrahi et al. [21] proposed a method to spread the inertia weight between the range of minimum and maximum value, in which particle with the best performance is given the smallest weight so that it moves the slowest and particle with the worst performance is given the biggest weight so that it moves the fastest. It is noted that setting inertia weight for each individual particle in the swarm required more computational effort than setting single weight for whole swarm. Therefore, the effectiveness of this mechanism should be carefully studied in order to evaluate whether the additional computational effort can significantly improve the performance of the algorithm.

### 3.2 Acceleration Constants

The values of local best and global best at a particular iteration are proposed as the basis for updating the acceleration constants [14]. Alternatively, a time-varying acceleration coefficient (TVAC) is proposed to replace the same constant during the whole iteration process [22]. In TVAC, the cognitive acceleration constant is linearly reduced and the social acceleration constant is linearly increased through the iterations.

One mechanism for adaptively setting the important weight of each acceleration term is presented in [23]. As an illustration, there are four cognitive/social terms that are taken into consideration in the GLNPSO presented above: personal best, global best, local best, and near-neighbor best. The acceleration constant gives relative importance of respective term when the velocity is updated. A heavier weight for a specific term means that term is more dominant than the others and the particles tend to move more toward the direction of this term. The adaptive mechanism in [23] reverses this property by first determining a relative importance of the cognitive/social term from the current swarm characteristics before setting the acceleration constants.

The importance measurement that is employed here is the difference between the corresponding objective function of particle’s position and the objective function of respective term. For a minimization problem, a bigger difference on a particular term implies that particles have opportunity to gain more improvement in its objective function if moving towards this term. However, negative difference is avoided since it will lead to worsening objective function. As shown in Fig 1, for a single particle which is located at position \(\Theta\) and surrounded in its corresponding cognitive/social terms (personal best \(\Psi\), global best \(\Psi^g\), local best \(\Psi^L\), and near neighbor best \(\Psi^N\)), the degree of importance of each term can be defined as

\[
\max \left\{ Z(\Theta) - Z(\Psi), 0 \right\}, \quad \max \left\{ Z(\Theta) - Z(\Psi^g), 0 \right\},
\]

\[
\max \left\{ Z(\Theta) - Z(\Psi^L), 0 \right\}, \quad \max \left\{ Z(\Theta) - Z(\Psi^N), 0 \right\}
\]

respectively for personal best, global best, local best, and near neighbor best.
Adaptive Particle Swarm Optimization Algorithms

Fig. 1 Particle position and its corresponding social terms. Then, the acceleration constants can be determined as the proportion of respective degree of importance to the constant $c^*$, which is defined as the sum of the acceleration constants. The expression for the degree of importance of a single particle can be expanded for the whole swarm which consists of $L$ particles by combining all particles properties, as follow:

$$\Delta Z_p = \sum_{l=1}^{L} \max \{Z(\Theta_l) - Z(\Psi_l), 0\}$$  \hspace{1cm} (16)

$$\Delta Z_g = \sum_{l=1}^{L} \max \{Z(\Theta_l) - Z(\Psi_g), 0\}$$  \hspace{1cm} (17)

$$\Delta Z_l = \sum_{l=1}^{L} \max \{Z(\Theta_l) - Z(\Psi_l^*), 0\}$$  \hspace{1cm} (18)

$$\Delta Z_n = \sum_{l=1}^{L} \max \{Z(\Theta_l) - Z(\Psi_n^*), 0\}$$  \hspace{1cm} (19)

where:

$\Delta Z_p$ : Degree of importance for personal best  
$\Delta Z_g$ : Degree of importance for global best  
$\Delta Z_l$ : Degree of importance for local best  
$\Delta Z_n$ : Degree of importance for near neighbor best

Finally, the acceleration constants can be determined as the proportion of degree of importance. Also, in order to avoid rapid changing of parameters, the acceleration constant is updated using exponential weighted moving average technique:

$$\Delta Z = \Delta Z_p + \Delta Z_g + \Delta Z_l + \Delta Z_n$$  \hspace{1cm} (20)

$$c_p = \alpha c_p + (1-\alpha) \frac{\Delta Z_p}{\Delta Z} c^*$$  \hspace{1cm} (21)

$$c_g = \alpha c_g + (1-\alpha) \frac{\Delta Z_g}{\Delta Z} c^*$$  \hspace{1cm} (22)

$$c_l = \alpha c_l + (1-\alpha) \frac{\Delta Z_l}{\Delta Z} c^*$$  \hspace{1cm} (23)

$$c_n = \alpha c_n + (1-\alpha) \frac{\Delta Z_n}{\Delta Z} c^*$$  \hspace{1cm} (24)

### 3.3 Number of Particles

There are few published work related to adapting the number of particles. Recently PSO with adaptive population size was proposed [24]. The total iteration steps are divided into some ladders with same number of iterations. At the end of each ladder, the diversity of swarm is measured, and then the population size is adjusted based on the measured diversity. If the swarm diversity is lower than a threshold value, the population size is increased. Otherwise, if the swarm diversity is higher than the threshold, the population size is decreased.

### 3.4 Other Parameters

Although existing adaptive mechanism for some parameters is not yet available in the literature, such as number of Iterations and number of neighbor, there are also possibilities to set these parameters adaptively.

### 4. Illustrative Example

An illustrative example of adaptive PSO algorithm is presented in this section. In this example some adaptive features reviewed above are added to the GLNPNSO algorithm in which the algorithm is only slightly modified and the computational effort is not significantly increased. To be more specific, the example algorithm can adaptively set the inertia weight and acceleration constants. Therefore, the only change to the GLNPNSO is in the Step 8, in which it is updated to:

a. Update inertia weight following Eq. 9, 11–15.

b. Update accelerations constant following Eq. 16–24.

c. Update the velocity and the position of each particle following Eq. 2–7.

In order to save some computational effort, the adaptive mechanism of inertia weight (step a) and acceleration constants (step b) is not performed in every iteration, but only performed every fixed number of iterations, for example 10 iterations.

To make the adaptive feature works, the following initialization is required. For the inertia weight, the $\omega_{\text{max}}$ is taken from the velocity index at the first iteration. Also, the $w^\text{max}$ and $w^\text{min}$ are being set as 0.9 and 0.4, respectively. For the acceleration constants, the value of $c^*$ is 4 and $\alpha$ is 0.8. Initially, equal acceleration constant is employed, i.e. $c_p = c_g = c_l = c_n = 1$.

For a test case, this adaptive PSO algorithm is applied to solve vehicle routing problem (VRP), in which the solution representation of this problem for PSO and the corresponding decoding method have been proposed before using GLNPNSO [5]. It is noted that the adaptive PSO algorithm can be applied to any optimization problem that have been solved by respective decoding method have been proposed before using GLNPNSO [5].

In a typical run with 1000 iterations, the velocity index pattern of both GLNPNSO algorithm (without adaptive feature) and the adaptive PSO algorithm are displayed in Fig. 1. It can be seen that the velocity index pattern of GLNPNSO is steadily decreasing, so that in the first half of the run (approximately from the first iteration to the 500th iteration) velocity index of adaptive PSO algorithm is bigger than velocity index of GLNPNSO. Also, in the second half of the run velocity index of adaptive PSO algorithm is smaller than velocity index of GLNPNSO. This pattern implies that the adaptive PSO algorithm is better at exploring the solution space in the first half of the run than GLNPNSO. Also, the adaptive PSO algorithm is better than the GLNPNSO algorithm to exploit the solution space in the second half of the run.
In the same algorithm run, the dynamic behavior of the best objective function values (the objective function of gbest) is presented in Fig 2. Following the pattern of velocity index, the best objective function of GLNPSO is improving steadily. However, the objective function values from GLNPSO are better than those from the adaptive PSO only in the early part of run. After about the 600th iteration, the adaptive PSO provides better objective function values. Finally, the best objective function values found are 2720.86 and 2671.59 for the original version of GLNPSO and from the adaptive version of GLNPSO, respectively.

The computational time of the adaptive PSO is not significantly bigger than the GLNPSO. It is empirically shown for the typical run tested above, the computational time of both algorithms are 08:12 and 08:21 minutes, respectively for GLNPSO and adaptive PSO.

5. Conclusion and Further Works

Recent effort to enable particle swarm optimization algorithm to self-adapt its parameter are reviewed in this paper. For illustrative purpose, a demonstrated example of adaptive PSO algorithm is proposed to adaptively set the inertia weight and acceleration constants.

Preliminary computational experiment on a typical vehicle routing instance implies that the adaptive PSO algorithm may perform better than GLNPSO algorithm in terms of solution quality but with slightly slower computational time. However, more computational experiment is required in order to generalize the result. Also, further works is still needed to explore more mechanisms for adapting other PSO parameters such as: number of particles, number of neighbors, and number of iterations.

Acknowledgments

The research is a part of the research program of the High Performance Computing Group at Asian Institute of Technology (AITHPC) which received support from the Thai GRID project. The authors wish to thank the AITHPC and the Thai GRID Center for the access of computing facility and the technical support.

References


