International Journal of Social Science And Human Research

ISSN(print): 2644-0679, ISSN(online): 2644-0695

Volume 05 Issue 03 March 2022

DOI: 10.47191/ijsshr/v5-i3-30, Impact factor-5.586

Page No: 950-965

Adjusting Parameters in Optimize Function PSO

Le Thi Bao Tran[1], Nguyen Thu Nguyet Minh[2], Tra Van Dong[3]

[1]Ho Chi Minh City University Of Foreign Languages - Information Technology, Vietnam.

[2,3] Van Lang of University, Vietnam

ABSTRACT*:* Particel Swarm Optimization (PSO) is a form of population evolutionary algorithm introduced in the early 1995 by two American scientists, sociologist James Kennedy and electrical engineer. Russell. This thesis mainly deals with the PSO optimization algorithm and the methods of adaptive adjustment of the parameters of the PSO optimization. The thesis also presents some basic problems of PSO, from PSO history to two basic PSO algorithms and improved PSO algorithms. Some improved PSO algorithms will be presented in the thesis, including: airspeed limit, inertial weighting, and coefficient limit. These improvements are aimed at improving the quality of PSO, finding solutions to speed up the convergence of PSO.

After presenting the basic problems of the PSO algorithm, the thesis focuses on studying the influence of adjusting parameters on the ability to converge in PSO algorithms. PSO algorithms with adaptively adjusted parameters are applied in solving real function optimization problems. The results are compared with the basic PSO algorithm, showing that the methods of adaptive adjustment of the parameters improve the efficiency of the PSO algorithm in finding the optimal solutions.

1. INTRODUCTION

Particle Swarm Optimization (PSO) is a form of population evolutionary algorithms. There have been many proposed and studied population-based evolutionary algorithms such as genetic algorithm (GA), ant colony algorithm (ACO). However, PSO differs from GA in that it favors using the interactions between individuals in a population to explore the search space. PSO is the result of modeling the flight of birds in search of food, so it is often classified as algorithms that use swarm intelligence. Introduced in 1995 at an IEEE conference by James Kennedy and engineer Russell C.Eberhart. Algorithms have many important applications in many fields where it is required to solve optimization problems.

Although the PSO algorithm has been shown to be highly effective in solving many optimization problems in practice, one of the weaknesses of the PSO algorithm is that it has a lot of parameters in the algorithm model. Some parameters can greatly affect the efficiency of the algorithm such as: population size, acceleration coefficients, coefficient of inertia,.... Usually, these parameters will be selected manually by researchers and experimenters of PSO algorithms using previously available knowledge. However, such manual selection does not guarantee that the algorithm will give good results. This topic will focus on studying methods of adaptively adjusting parameters for PSO algorithm in order to improve the efficiency of the algorithm in finding optimal solutions for problems.

Particle Swarm Optimization (PSO) was developed by social psychologist (James Kenned) and electrical engineer (Russel Eberhart) in 1995 [1], starting from initial experiments. on bird behavior by biologists (Frank Heppner). Since then PSO has made strong progress and has many applications in solving practical problems. There have been many studies showing that PSO is more efficient than some other well-known algorithms (eg Gen algorithm) in solving some practical problems.

In Vietnam, the research and application of PSO is relatively small. Therefore, this topic aims to learn about PSO and study methods of adaptive adjustment of parameters of PSO.

2. LITERATURE REVIEW HISTORY OF PSO

Swarm optimization (PSO) refers to a family of algorithms used to find globally optimized solutions. PSO can be easily implemented in many languages and has been proven effective when it comes to solving optimization problems.

PSO was originally developed by a social psychologist (James Kennedy) and an electrical engineer (Russel Eberhart) in 1995 [1], having emerged from previous experiments by a biologist (Frank Heppner) describes the foraging behavior of flocks of birds. According to Heppner [24], the description of a flock of birds has the following characteristics: The birds start by flying around the target but do not know the exact coordinates. If a leader bird flies over an area at the correct location, all the birds in the flock are attracted to follow the leader and fly into that exact location. Each bird tries to fly among the birds near it. They are always drawn (influenced) from the leader and the one next to him. The affected birds follow the leader more and more until the whole flock reaches the right spot. Eberhart and Kennedy in [1] explain Heppner's description as follows: Finding a leader is similar to finding a solution among possible solutions. The way in which a leader bird guides the nearby birds to follow it, increases the range they will find it, it is the focal point for the entire flock of birds to follow it.

A bird, similar to a solution, can fly through a solution space and arrive at the best solution. Particles learn from their successes in the past, just as we learn from our own experiences in the past. Individuals (individuals) learn primarily from the successes of the birds next to them. It is similar to when we compare ourselves to others and we imitate the behavior of others who have had success with the things we care about.

2.1. Full model of pso

PSO simulates the behavior of birds.

Assume the following scenario [24]: a flock of birds is randomly foraging in the area. Only a little food in the area is being sought. The whole flock of birds do not know the exact location of the food, but they do know the distance to the food during each iteration of the search. Therefore, the answer to the question must be sought, what is the best strategy for foraging?

PSO learns from that scenario and uses it to solve optimization problems. In PSO, for each solution called a "Particle", is a bird in the search space. Each particle has a suitable value, evaluated by optimized objective functions, and has a velocity to indicate its flight path. The PSO is initialized with a random pool of particles, and search optimization is performed by updating the next generation of particles. Put xi(t) in the position of Particle i at step t. The Particle's position is changed by adding a velocity, vi(t) to the current position. As a result, the position of Particle i in the (t + 1)th step can be calculated as follows:

xi(t+1) = xi(t) + vi(t+1) (2.1)

It is the velocity vector that affects the flight of each particle. It is also the experience knowledge of the particles and the exchange of common information from its neighbors. The acquired knowledge experience of a particle is often called the cognitive component (learning factor), which is proportional to the distance of the particle from its best position in the past. The social exchange of information is called the social component of the Particle.

The way a Particle communicates depends on the extent of its neighbors. If the neighbor of each particle is the entire swarm, then the particle is said to be the best global PSO; conversely, if the neighbors of each particle are only part of the swarm, then this is called the local best PSO.

2.2. Global PSO

Global optimal PSO (gbest PSO), when the neighbor of each particle is the entire swarm. Information exchanged between particles reflects information obtained from all particles in the swarm

Given gbest PSO, the velocity of Particle i at the iteration at t+1 is calculated as [1]:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}[y'_j(t) - x_{ij}(t)] \qquad (2.2)$$

Condition :

vij(t) is the velocity of Particle i at dimension j = 1,2…nx at step t

xij(t)is the position of Particle i at dimension j = 1,2…, nx at step t

c1 , c2 are learning factors (velocity acceleration factor), they are used to increase the contribution based on the experience and the swarm population in a predetermined order.

r1j(t) ,r2j(t) is the random value in the interval [0,1], r1j(t) ,r2j(t) is the storage environment of the algorithm.

yij(t) is the best individual position of Particle i at dimension j in the first step. It is the best position of Particle i since the first step. The best position of each individual in the t +1 step is calculated as follows:

$$y_{ij}(t+1) = \begin{cases} y_i(t) & if\ f(x_i(t+1)) \geq f(y_i(t)) \\ x_i(t+1) & if\ f(x_i(t+1)) < f(y_i(t)) \end{cases} \qquad (2.3)$$

when f : is a suitable function value, evaluate the approximate value that is close to the optimized solution.

y'j(t) : is the global best position of all particles in the swarm at direction j. The global best value y'(t) at step t is defined by

$$y'(t) = \min\{ f(y_0(t)), f(y_1(t)),\ldots, f(y_{n_s}(t))\} \qquad (2.4)$$

when n_s is all particles in the swarm.

2.3. The algorithm gbest PSO

The velocity of each particle in the swarm is calculated by equation (2.2), the algorithm used to find optimization problems according to [1]: The optimization algorithm is as follows:

```
Step 1: Initialize the swarm S with nx dimensions
            Put  $y_i = x_i$
            Put  $y' = x_1$
  Step 2:
  Repeat:
    For each particle i= 1,2,…,$n_s$ do
            // best position the Particle itself
```

```
                if $f(x_i) < f(y_i)$ then
                        $y_i = x_i$
                end
                // Global best position reset
                if $f(y_i) < f(y')$ then
                        $y' = y_i$
                end
        end
        for each particle i = $1,2,\dots,n_s$ do
              Update the velocity in the formula (2.5)
              Update position in formula (2.1)
end
```

When the condition is met, stop the algorithm.

2.4. Geometry show

Geometric methods are used to describe the motion of particles in global PSO algorithms. For simplicity, we consider a Particle in two-dimensional space as follows:

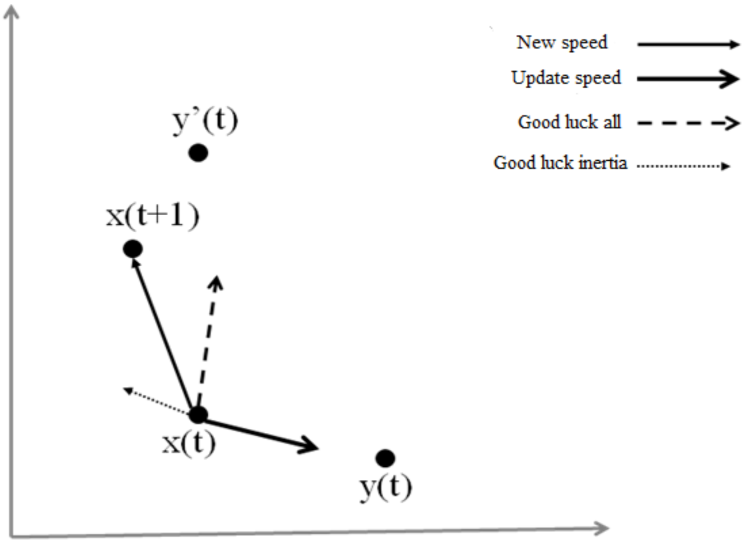

Figure 2.1 : Particle's position at step t+1.[1]

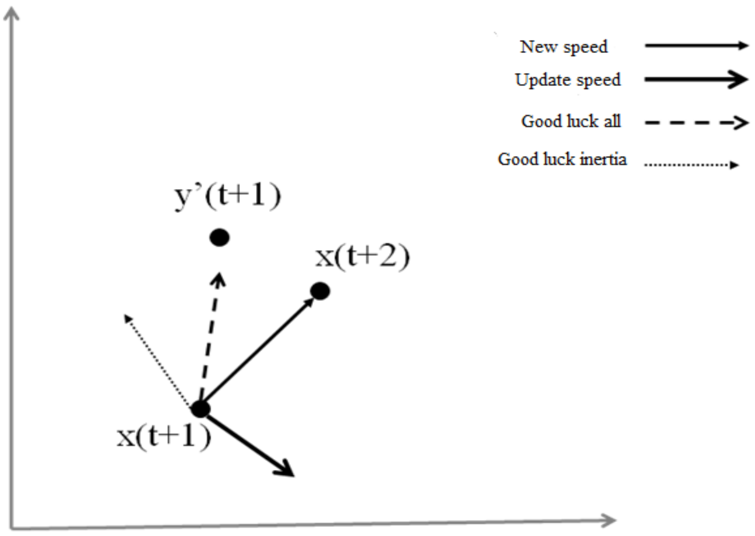

Figure 2.2 : Particle's position at step t+2.[1]

Suppose that in the tth step Particle is at the position shown in Figure 2.1, the best and the best Particle instances globally at the positions shown in the figure are under the influence of three components (update velocity, swarm velocity) , inertial velocity),

the position of the particle at step (t+1) is closer to the global best position. In Figure 2.2, it is shown that the particles move from step (t+1) and to a new position when in step (t+2). In the new location, there can be two cases for a Particle [1] :
Case 1: The solution at the new location is better than the current global best. In this case the new position will become the new global best position, and all the Particles in the swarm will be attracted to the direction of the Particle in this new location.
Case 2: New solution found in new location is still as good as global best Particle. Then, in the next step, herd perceptions of the three-component effects will guide the whole swarm particles back in the direction of the global best.

2.5. Local best PSO

In the best-to-all PSO, the neighbor of each particle is the whole herd. For the local best PSO, the neighbor of each Particle is just a Particle number [1]. The components reflect the exchange of information between Particle's neighbors, in lbest PSO, the rate is calculated as [1] :

$$v_{ij}(t+1) = v_{ij}(t) + c_1r_{1j}[y_{ij}(t) - x_{ij}(t)] + c_2r_{2j}[y'_{ij}(t) - x_{ij}(t)] \quad (2.5)$$

when $y'_{ij}(t)$ is the best position found from the neighbor of the ith particle in the j direction in the tth step. The local best particle position, y'_i, the best position found in the neighbor N_i is defined as

$$y'_i(t+1) \in \{N_i | f(y'(t+1)) = \min\{f(x)\}, \forall x \in N_i\} \quad (2.6)$$

With N_i neighbors are counted

$$N_i = \{y_{i-N_{i+1}}(t), \dots, y_i(t), \dots, y_{i+N_i}(t)\} \quad (2.7)$$

It should be noted that for basic PSO, the selection of neighboring particles is based on the Particle indexes. In fact, there are also other approaches in which neighboring particles are selected based on the distance between particles [9], but this is not considered in this thesis. There are two main reasons why Particle neighbors based on Particle indexes are used [1]: -One is because it is computationally expensive, since it is not necessary to consider the spacing between particles. For these approaches from the distance between particles are used to form neighbors, and usually the Euclidean distance between all pairs of particles, which requires a complexity O(n2) .
Second, because it helps to promote the spread of information from a good solution, obtained for all particles, regardless of their position in the spatial search. It is also important to note that neighboring particles can be overlapped. Each Particle can join several neighborhoods. This connection ensures that the swarm convergence will be at a single point. In fact, the global best PSO is just a special case of the local best PSO when $n_{N_i} = n_s$

The algorithm lbest PSO

The lbest PSO algorithms are the same as gbest PSO, except for some differences in the equation for updating the velocity, which is summarized as follows:

```
Step 1: Khởi tạo bầy S với $n_x$ chiều
            Put $y_i = x_i$
            Put $y'_i = x_i$
Step 2:
  Repeat:
    For each particle i= 1,2,…,$n_s$ do
            // best position the Particle itself
            If $f(x_i) < f(y_i)$ then
                $y_i = x_i$
            end
            // Local best position reset                                    if
f($y_i$) < f($y'_i$) then
                $y'_i = y_i$
            end
    end
    for each particle i = 1,2,…,$n_s$ do
            Update the velocity in the formula (2.5)
  Update position in formula (2.1)
    end
When the condition is met, stop the algorithm.
```

3. COMPONENTS OF BASIC PSO

This section presents some components of the PSO model that can affect its effectiveness. These components include the Particle initialization and the stopping condition of the algorithm, besides the basic parameters are also mentioned.

3.1. Initialize PSO

The first step of the PSO algorithm is to initialize the swarm using the Pseu-dorandom method. Assume that an optimal is placed in the domain defined by two vectors x_{min} and x_{max}, representing the lower and upper bounds in each dimension of the search space, respectively. Then a maximum efficiency of the two vectors x_{max} and x_{min}
in the constructor for Particle positions can be [1]:

$$x_{ij}(0)= x_{min,j} + r_i(x_{max,j} - x_{min,j}), \forall_j=1,\dots,n_x, \forall_i=1,\dots,n_s \quad (3.1)$$

when r_j is a random number in the range [0,1] The initialization velocity can be initialized to 0, that is, $v_{ij} = 0$, it is possible to initialize the velocity to random values, but must be done with care. The initial value of the particle velocity and position is random, the particle velocity is zero (that is, they are fixed), when the particle starts up at a non-zero velocity and with a value that is not too large. A large initial velocity will have a large initial effect leading to a large initial position update. This makes the particle go beyond the boundaries of the search space, causing the swarm to iterate many times before meeting. converge to the optimum point.

3.2. Algorithm stop condition

The second factor affecting the performance of PSO is the stopping condition of the algorithm, which is the criterion used to terminate the iteration in the search process. The following two principles are considered when selecting some of the repetition termination criteria: The stopping condition of the algorithm is not the cause for the early convergence of PSO when the optimal solution is selected. -The stopping condition should not lead to overcomputing for the suitable value of the function, if the stopping condition requires frequent calculation of the value of the suitable function, the computational complexity will increase significantly in search process. There are documents that use some stopping conditions as follows [1]: -Stop the algorithm when the maximum number of repetitions exceeds the specified number of times. It is easy to see that if the number of iterations is too small, before a good solution is found, stopping the algorithm may be too early. So this status is only used to evaluate the best solution found within the specified time period. - Stopping the algorithm when it finds x* is an accepted optimal solution, assuming it is an optimal objective function, then this condition terminates the search as soon as a particle is found that $f(xi) \le |f(x^*) - \varepsilon|$; that is, when with an acceptable error probability threshold found, the threshold ε is chosen carefully, if ε is too large the search process ends up with a solution that does not meet the optimal conditions. On the other hand, if ε is small, the search does not end at all. Assume a stop condition that tells the algorithm about the global optimization value, where the optimal is usually zero.

-Algorithm stops when observation in some iterations still does not improve, Some progress can be the standard by which to judge in some ways [1], such as if the average position changes of particles is small, the swarm can be considered convergent, otherwise if the average velocity of the particle over a number of iterations is approximately zero, only a small position update is performed, then the search may stop. again. The search can also be stopped if after a number of iterations there is still no significant improvement.

-Terminate the algorithm when the radius of the normal swarm is close to zero. The swarm radius is calculated as follows [7] :

$$R_{norm} = \frac{R_{\max}}{diameter(S)} \quad (3.2)$$

When diameter (S) is the diameter of the original swarm and R is the maximum radius.

$$R_{\max} = \left\| x_m - y^{'} \right\|, m = 1,\dots,n_s \quad (3.3)$$

With

$$\left\| x_m - y^{'} \right\| \ge \left\| x_i - y^{'} \right\|, \forall_i = 1,\dots,n_s \quad (3.4)$$

- Stop the algorithm when the objective function value approaches zero. The above conditions only stop to consider the relative position of the particles, but not the slip of the objective function. Changes in the objective function are considered proportionally [7]:

$$f' = (f(y'(t))-f(y'(t-1)) /(f(y',(t)) \quad (3.5)$$

if f'<ε for some continuous iteration, the swarm seems to have converged. The stopping condition has a problem as the search will terminate if some particles are stuck to the local minimum, even though other particles may still be in probing the search space. To solve this problem, this condition can be used in combination with radius methods to check that all particles have indeed converged to the same point in time before terminating the search.

3.3. Basic PSO parameters

This section discusses the parameters that affect two basic PSO algorithms. Those parameters are the swarm size, the distance between the particles in the swarm, the learning factor. The size of the swarm ns is the number of particles in the swarm, the particles in the swarm are larger than the initial diversity of the swarm, provided that a well-distributed and well-distributed initialization method is used to initialize the particles [1]. A large Swarm allows most of the search space to be safe for each iteration. However, with many particles the computational complexity increases. In some real studies, it has been demonstrated that PSO has the ability to find the optimal solution with small swarm size from 10 to 30 particles [4]. Even with the number of elements in the swarm less than 10, success is still obtained [7]. At the same time, experimentally studying a general solution of swarm ns$\in$[10,30], the problem depends on the optimal size of the swarm, a fine search space usually needs a very small number of particles than the rough surface. determining the optimal solution location. Particle Neighbor Size: The size of the swarm determines the mutual interaction between the particles in the swarm. The smaller the neighbor size of the particle, the less interaction occurs and can slow down the convergence of the PSO. These factors give PSO a more reliable convergence with respect to the global optimization method [1], which takes advantage of small and large neighbor sizes. PSOs can be searched with small sizes and gradually increased in number [9]. Learning factor (or acceleration factor): The coefficients c1 and c2 with random vectors r1 and r2, control the effect of starting a process of cognitive and social components on the velocity of the particles. [first]. While c1 represents the confidence of a particle itself, c2 represents the confidence contained in the neighboring particles of a particle. If c1 = c2 = 0 Particles keep their current magnetic velocity until they reach the boundary of the search space. If c1 > 0 and c2 = 0, all particles climb the hill independently. On the other hand, if c2 > 0 and c1 = 0, the whole swarm is attracted to a single point y'. The swarm becomes a random climber . Usually, c1 and c2 are chosen to be equal.

3.4. Performance Indicators

This section presents various measures to quantify the performance of PSO algorithms. Measures, as in other optimization methods, are based on several criteria such as accuracy, reliability, stability, efficiency, diversity and coherence.

3.4.1. accuracy

Accuracy refers to the quality of the solutions obtained. If prior knowledge of optimization is available, quality can be expressed as errors of the global best position. For example, if the optimal value is known, the correctness of the solution y'(t), can be defined as:

$$accuracy(S,t)=|f(y'(t)-f(x^*))| \qquad (3.6)$$

where x* represents the theoretical optimal. If PSO is used to train a neural network, the accuracy is simply the mean squared error (MSE) or squared error (SSE) over the samples in the dataset provided to train the network. neural, f(x*) is usually zero. If there is no information about the optimization, the swarm accuracy at step t is simply the global best-fit Particles, i.e.

$$accuracy(S,t)=f(y'(t) \qquad (3.7)$$

3.4.2. reliability

Since the start-up of the PSO algorithms is random, their performance is evaluated over a sufficiently large number of simulations. Reliability refers to the simulation rate to achieve a certain accuracy (fit or error value). The reliability of a swarm can be defined [1]:

$$reliability(S(t),\varepsilon)=\frac{n_\varepsilon}{N}\times 100 \qquad (3.8)$$

where ε is a predetermined level of precision, N is the total number of simulations, and nε is the number of simulations that reach a certain accuracy. The larger the value of reliability (S(t), ε), the more reliable the flock.

3.4.3. Stability

The stability of the PSO can be measured by the variance of a performance standard across several simulations. The stability of a swarm can be expressed as [1]:

$$robustness\big(S(t)\big)=[\bar{\theta}-\sigma,\bar{\theta}+\sigma] \quad (2.16)$$

where is the average of performance benchmarks over a number of simulation, and σ is the difference in performance criteria.

3.4.4. Effective

The swarm efficiency is expressed as the number of iterations, or the number of matches, to find a solution with specified precision ε. The swarm efficiency represents the relative time to reach a desired solution.

3.4.5. Diversity

Diversity is important for optimization based on the number of algorithms. The great variety ensures that a larger area of the search space can be explored. The diversity of the flock can be calculated as [1]:

$$diversity\big(S(t)\big)=\frac{1}{n_s}\sum_{i=1}^{n_s}\sqrt{\sum_{j=1}^{n_s}\left(x_{ij}(t)-\bar{x}_j(t)\right)^2} \qquad (3.9)$$

trong đó $x_j(t)$ là trung bình của chiều thứ j trên tất cả các Particle, tức là

$$\bar{x}_j(t) = \frac{\sum_{i=1}^{n_s} x_{ij}(t)}{n_s} \qquad (3.10)$$

Một cách cách khác để đánh giá đa dạng của các bầy đã được đưa ra trong [22]

$$diversity(S(t)) = \frac{1}{diameter(S(t))}\frac{1}{n_s}\sum_{i=1}^{n_s}\sqrt{\sum_{j=1}^{n_s}(x_{ij}(t)-\bar{x}_j(t))^2} \qquad (3.11)$$

when diameter (S(t)) is the diameter of the swarm defined as the distance between two distal outer paticles. It can be seen that the construction of diversity in equation (3.11) is independent of the swarm size, the number of dimensions of the search space and the search scope in each dimension.

4. METHODS OF ADJUSTING PARAMETERS FOR PSO

As mentioned above, although the PSO algorithm has been shown to be effective for many optimization problems, one of the limitations of PSO has a lot of parameters in the algorithm model. Often the parameters will be selected manually by the researcher or the PSO user. However, this manual selection does not guarantee that the algorithm will give good results on many problems of interest to the user. Therefore, in order to overcome that drawback, many researchers propose methods to adjust parameters for PSO. In this section we will present some methods of adjusting that parameter. In the following section, we will use the inertial weight model to adjust the parameters of the PSO. As described in Section 2, in the inertial weighted model, the velocities of the particles are updated using the following formula:

$v_{ij}(t+1) = wv_{ij}(t) + c_1r_{1j}[y_{ij}(t) - x_{ij}(t)] + c_2r_{2j}[y'_j(t) - x_{ij}(t)]$ (4.1)

After the velocity of the particles has been updated, the position of that particle will be recalculated according to formula (2.1). It can be seen that, in formula (3.1), if the value of w is chosen to be fixed as 1, then that is the basic PSO model. Thus, in this model, there are three parameters that need attention: inertial weight, w, and acceleration coefficients c1, c2. In section 3.1, we will present the adaptive adjustment method w. The tuning algorithm c1, c2 will be presented

4.1. Adaptive tuning of inertial weights

As mentioned in Chapter 2 when talking about inertial weights, there are many methods for adaptively tuning the inertial weight w. Some of the basic methods will be presented below: -Random tuning: a different number of inertia is randomly chosen each iteration. One common way is to sample from a distributed Gaussian, for example

$$w \sim N(0.71, \sigma) \qquad (4.2)$$

where σ is small enough to ensure that $w \leq 1$. In addition, Peng and colleagues used: [15]

$w = (c_1r_1 + c_2r_2)$ (4.3)

no random scaling of cognitive and social components -Linear reduction: A large value of inertia (usually 0.9) is chosen as at the beginning and then it is reduced to a small value (usually 0.4). It was calculated in [16] as

$$w(t) = (w(0) - w(n_t))\frac{(n_t - t)}{n_t} + w(n_t) \qquad (4.4)$$

where nt is the maximum number of iterations, w(0) is the initial inertia weight, w(nt) is the final value of inertia, and w(t) is the mass of inertia at step t. -Nonlinear reduction: It is similar to linear reduction but the formula for inertia reduction is non-linear. The nonlinear reduction method allows a shorter amount of exploration time than the linear reduction method. Therefore, these methods will be more suitable for smoother search space. The following nonlinear reduction methods have been proposed in the literature:

- Naka et al. [16],

$$w(t+1) = \frac{(w(t) - 0.4)(n_t - t)}{n_t + 0.4} \qquad (4.5)$$

- Venter and Sobieszczanski-Sobieski [23],

$$w(t+1) = \alpha w(t') \qquad (4.6)$$

where α = 0.975, and t' is the time step when the final inertia changes. - Clerc proposed an adaptive inertia approach where the amount of change in the inertial weights is proportional to the improvement in swarm relations [17]. The inertia weights are adjusted according to:

$$w_i(t+1) = w(0) + (w(n_t) - w(0))\frac{e^{m_i(t)} - 1}{e^{m_i(t)} + 1} \qquad (4.7)$$

when the improvement is relative, mi is approximately the same as

$$m_i(t) = \frac{f(y'_i(t)) - f(x_i(t))}{f(y'_i(t)) + f(x_i(t))} \qquad (4.8)$$

with $w(n_t) \approx 0.5$ and $w(0) < 1$

3.2. Adjust the acceleration coefficients c1, c2

This method was introduced by authors Li Guo and Xu Chen in 2009 [6]. This method comes from the observation that the nature of the search process using the PSO algorithm is dynamic and adaptive. Therefore, fixing the acceleration coefficients c1 and c2 is contrary to the nature of the PSO algorithm and can lead to a reduction in the efficiency of the algorithm. This is completely understandable for two reasons: First, even if the particles have good fitness values, fixing the acceleration coefficients can hinder the convergence of these particles. Second, when the PSO algorithm is stuck to local extremes and the PSO algorithm cannot escape these local extremes if the coefficients c1 and c2 are fixed.

Based on that analysis, Guo and Chen propose a new method for adaptive tuning of parameters c1 and c2. Details of this method are presented in the following section.

Particle's velocity is updated according to the formula:

$$v_{id}(k+1) = C.\left[v_{id}(k) + c_i(k).r_1.\left(p_{id}(k) - x_{id}(k)\right) + c_i(k).r_2.g_d(k) - x_{id}(k)\right] \quad (4.9)$$

From formula (9) it can be seen that c1 and c2 are two important parameters affecting the process of finding the optimal solution of PSO. In fact, these two parameters have a role to balance between PSO's local and global search capabilities. Since then, a new method for adaptively tuning those parameters was proposed by Guo and Chen. This new method is called Self-Adaptation Statege of Acceleration Coefficients (SASAC-PSO).

In the SASAC-PSO Method, each particle can have different values of c1, c2, and these values can change over time. This technique helps the PSO algorithm to perform a better balance between global search and local search and thus the efficiency of the algorithm can be improved. The details of the SASAC-PSO algorithm are as follows:

Step 1: Initialize the Particle i swarm randomly. Notation (xi (0), vi (0), c_i (0)) (i = 1, ..., N), where xi (0), v_i (0) and c_i (0) respectively represent for initial position, velocity, acceleration, $x_i(0)$ and $v_i(0)$ are uniformly distributed separately random numbers in domain D and $c_i(0) = 1$.

Step 2: Default $p_i(0)= x_i(0)$ and calculate

$g(0) = \text{argmin}\{f(p_1(0)),\dots,f(p_N(0))\}$,

đặt k = 0.

Step 3: Increase the dimensionality of the position and velocity of each particle.

Step 4:

Deployment Pi(k) (i = 1,2….,n) and g(k)

$p_i(k+1) = \text{argmin}\{f(x_i(k+1), f(p_i(k))\}$

$g(k+1) = \text{argmin}\{f(p_1(k+1),\dots,f\{p_N(k+1))\}$

Step 5:

Check stop condition. If the stop condition is met, exit the program. Otherwise, go to step 6.

Step 6: If g(k + 1) has not changed giving Pnum the number of loops in a row, then go to step 7, otherwise, ci(k + 1) = ci(k) (i = 1, .. ., N) and set k = k + 1. Then return to step 3.

Step 7: Update AC for each Particle

$$c_i(k+1) = \begin{cases} 0.5 \times c_i(k) & i \in I \\ 2 \times c_i(k) & i \in \{1,\dots,N\} - I \end{cases}$$

condition I is the sort index of the first $\left\lfloor \frac{N}{2} \right\rfloor$ Particle

Set k =k+1 and return to step 3.

5. EXPERIMENTAL SETTINGS

The purpose of our experimental part is to compare the effectiveness of the adaptive tuning methods with the standard PSO algorithm and between the adaptive tuning methods. To test the effectiveness of those methods, we use them in finding optimal solutions of real functions in multidimensional space. We use the following 6 functions to test the efficiency of the algorithms.

5.1. Test function

The functions used to test the algorithm are all standard functions (benchmark functions) often used to evaluate the efficiency of approximate optimization algorithms. All these functions are taken from the literature [8]. The following six functions are used in the experimental program to illustrate the algorithms:

f1: Hyper-Ellipsoid:

Hyper-ellipsoid : $f(x) = \sum_{i=1}^{n} i.x_i^2$, $-5.12 \le x_i \le 5.12$, i=1,…,n .

Global minimum f(x)= 0 có thể đạt được, for $x_i= 0$, i=1,…,n

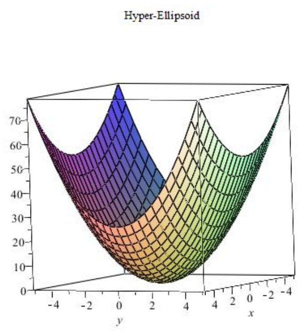

Figure 5.1 graph f1 Hyper-Ellipsoid

f2: Rosenbrock function:

$f(x,y) = (1 - x)^2 + 100(y - x^2)^2$, $-2.048 \leq x,y \leq 2.048$

global minimum $f(x,y) = (1,1) = 0$

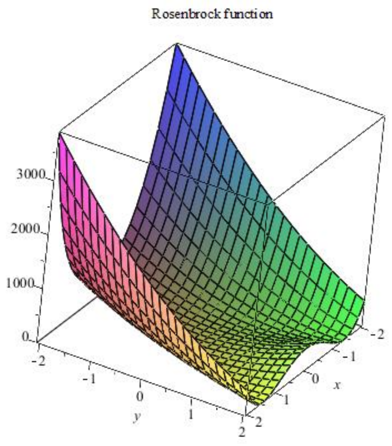

Figure 5.2 graph f2 Rosenbrock

f3: Rastrigin function:

$$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$$

$-2.048 \leq x_i,y_i \leq 2.048$, i=1,…,n . Global minimum $f(x) = 0$, for $x_i = 1$, i=1,…,n

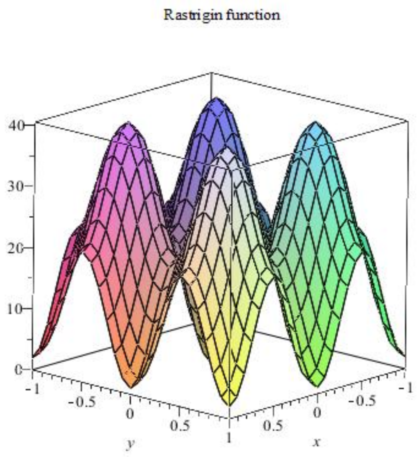

Figure 5.3 Graph f3 Rastrigin

f4: Schwefel function:

$f(x_1, \ldots, x_n) = \sum_{i=1}^{n}\left[-x_i \sin\left(\sqrt{|x_i|}\right)\right]$, $-500 \le x_i \le 500$

global minimum $f(x_1, \ldots, x_n) = -418.9829n$ at $x_i = 420.9687$

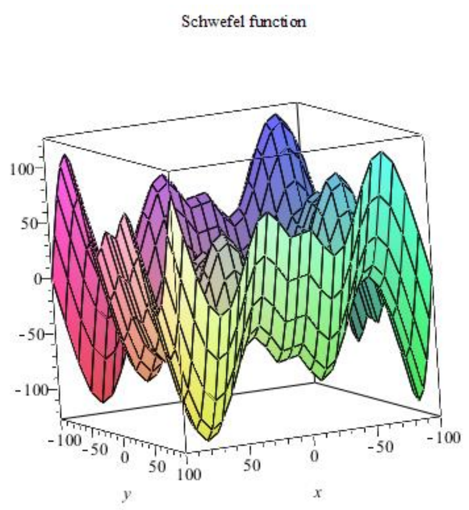

Figure 5.4 graph f4 Swchefel

f5: Griewank function:

$f(x_1, \ldots, x_n) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\frac{x_i}{\sqrt{i}}$, $-600 \le x_i \le 600$

global minimum $f(x_1, \ldots, x_n) = 0$ at $x_i = 0$

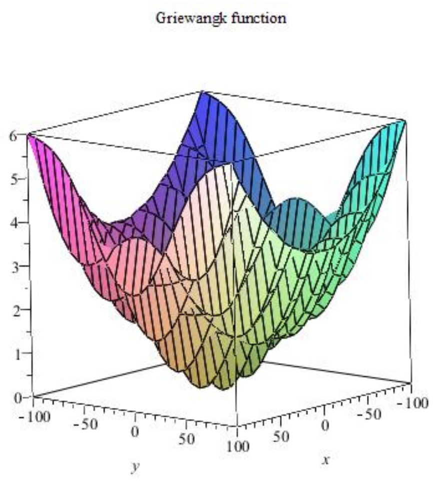

Figure 5.5 graph f4 Griewangk

f6: Ackley function:

$$f(x) = -a.\exp\left(-b.\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(cx_i)\right) + a + \exp(1)$$

Put a =20, b=0.2, c=2π, $-32.768 \le x_i \le 32.768$, i=1,…,n

Global minimum f(x) = 0 , Is obtainable for x_i=0, i=1,…,n

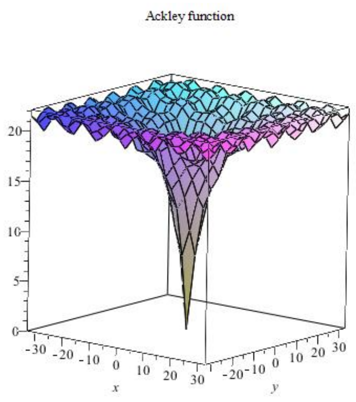

Figure 5.6 graph f6 Ackley

Functions from f1 to f6 have their respective optimal ranges and values according to the following table [8].

Table 5.1: standard values of functions reaching min

function	xi	Threshold value xi	Function value f(x) reach min
f1 (Hyp-ellisoid)	0	[-5.12, 5.12]	0
f2 (Rosenbrock)	1	[-2.048,2.048]	0
f3 (Rastrigin)	0	[-5.12, 5.12]	0
f4 (Schwefel)	420.9687	[-500, 500]	-418.9829*n
f5 (Griewangk)	0	[-600, 600]	0
f6 (Ackley)	0	[-32.768, 3278]	0

5.2. Description of the demo program

To install PSO algorithms, we use Java language, Net Bean IDE 7.4 programming tool. The parameters of the algorithms are chosen the same. As follows:

- The size of the flock is: 100
- Number of generations (number of loops) is: 100
- The number of runs for each configuration is 30 times
- Each function runs the number of dimensions 5, 10, 15, and 20 . respectively
- For the inertial weighted adjustment algorithm (w-PSO), the value of w is initially chosen to be 0.7 and linearly decreases to 0.3 in the final generation.
- For W-SASAC-PSO algorithm, w is also adjusted down linearly like W-PSO, while parameters c1 and c2 are adjusted according to Guo and Chen's method as in SASAC-PSO algorithm.

The inertia weight w at step t is calculated by the formula (2.28)

$$w(t) = \left(w(0) - w(n_t)\right)\frac{(n_t - t)}{n_t} + w(n_t) \qquad (5.1)$$

The inertial weight is adjusted according to the formula

$$w_i(t+1) = w(0) + \left(w(n_t) - w(0)\right)\frac{e^{m_i(t)} - 1}{e^{m_i(t)} + 1} \qquad (5.2)$$

when the improvement is relative, mi is approximately the same as

$$m_i(t) = \frac{f\left(y'_i(t)\right) - f\left(x_i(t)\right)}{f\left(y'_i(t)\right) + f\left(x_i(t)\right)} \qquad (5.3)$$

with $w(n_t) \approx 0.5$ and $w(0) < 1$

Adjusting Parameters in Optimize Function PSO

Parameters c1 and c2 are adjusted according to Guo and Chen's method as in the SASAC-PSO algorithm in step 7 of the algorithm as described above on page 30.

Update AC for each Particle

$$c_i(k+1) = \begin{cases} 0.5 \times c_i(k) & i \in I \\ 2 \times c_i(k) & i \in \{1, \dots, N\} - I \end{cases}$$

condition I is the ordered index of $\left\lfloor \frac{N}{2} \right\rfloor$ Particle prevails first. Set k =k+1 and return to step 3.

The formula Velocity is calculated in algorithms

.Basic PSO:

$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}[y'(t) - x_{ij}(t)]$ (5.4)

.W-PSO:

$v_{ij}(t+1) = w{*}v_{ij}(t) + c_1 r_{1j}[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}[y'(t) - x_{ij}(t)]$ (5.4')

.SASAC-PSO :

$v_{ij}(t+1) = C{*}[v_{ij}(t) + c_i(k) r_1 (y_{ij}(t) - x_{ij}(t)) + c_i(k) r_2 (y'(t) - x_{ij}(t))]$ (5.5)

với $C = 0.729$

- Thus, we see that the formula for updating velocity v common to all PSO algorithms is:

$v = C(w{*}v + c_1 r_1[y_{ij}(t) - x_{ij}(t)] + c_2 r_2[y'(t) - x_{ij}(t)])$ (5.6)

Accordingly, when executing, will choose the input parameter if:

Basic PSO C = 1, w = 1, c1, c2 random

• SASAC-PSO: C = 0.729, w = 1, $c_1 = c_1[k]$, $c_2 = c_2[k]$

In the program, depending on the options selected on the interface, we will get the appropriate values for C, W, c1, c2 to pass in (1). When running experiments from function f1 to f6 in turn through PSO, SASAC-PSO algorithms, with parameters: Size of the swarm is 100, number of iterations 100, number of experimental runs per configuration is 30, through the dimensions 5, 10, 15 and 20 respectively as follows

From the results of 30 runs with corresponding values according to the table, calculate the average time value, the average optimal value of those 30 runs. From the experimental results obtained in Table 5.2

Adjusting Parameters in Optimize Function PSO

Table 5.2 : Result of function f1 (Hyper Ellipsoid), n=5

Number of runs	Number of loops	Running time	Optimal value
1	100	10240	1.57E-28
2	100	10210	8.11E-28
3	100	10165	4.53E-27
4	100	11065	2.44E-28
5	100	13332	6.48E-29
6	100	14556	1.17E-28
7	100	15452	4.42E-27
8	100	16344	2.03E-28
9	100	16541	1.92E-28
10	100	17257	7.16E-28
11	100	19406	8.54E-28
12	100	19940	9.24E-28
13	100	20983	3.72E-29
14	100	21485	4.18E-28
15	100	22801	2.01E-28
16	100	24289	8.77E-28
17	100	23896	9.11E-28
18	100	24801	5.93E-29
19	100	26549	3.81E-28
20	100	27931	4.02E-28
21	**100**	**28716**	**3.04E-29**
22	100	29254	3.11E-28
23	100	29650	1.32E-28
24	100	30955	5.10E-28
25	100	31927	2.20E-28
26	100	33012	4.65E-27
27	100	33448	1.17E-28
28	100	35790	1.05E-27
29	100	35475	8.79E-28
30	100	37492	1.36E-27
	Trung bình	**23098.73**	**0**

Similarly f1 (Hyper_Ellipsoid) uses the PSO algorithm with dimensions of 5, 10,15 and 20 respectively, there will be 04 tables of results similar to Table 5.2. Each algorithm has 04 results tables, each function in turn experiment with 04 algorithms PSO, SASAC-PSO, will give 16 results tables; there are 06 functions from f1 to f6 used for experiment; after running the full experiment in turn, 96 tables of results were obtained, similar to Table 5.2; sequentially fill in the corresponding results in tables 5.3, table 5.4, table 5.5 from which to get the final results of the experiment. To measure the efficiency of those algorithms, we use 3 metrics. The first measure is the average fitness value of the algorithms over 30 runs. The second measure is the best value of each method over 30 runs, and the third measure is the mean time of the methods over 30 runs. The results of the methods with these measures are presented in the following sections.

5.3. Experimental results

Table 5.3 below is the average fitness results of the methods over 30 runs. In this and subsequent tables, the value of the best method in each experimental configuration is highlighted in bold.

Adjusting Parameters in Optimize Function PSO

Table 5.3 Compare average goodness results

function	Algorithm	Dimensions			
		5	10	15	20
f1	PSO	0.0242	0.4082	3.1476	5.2609
	SASAC_PSO(III)	**0**	**0**	**0.0021**	0.1099
f2	PSO	2.1969	9.3667	19.2679	28.5672
	SASAC_PSO(III)	**0.2144**	**6.1470**	**12.6176**	**18.4354**
f3	PSO	3.6723	23.7061	52.1721	87.8797
	SASAC_PSO(III)	5.4101	12.0017	19.3396	**26.8649**
f4	PSO	**-1816.8144**	**-3101.7300**	**-4478.0600**	**-5021.2200**
	SASAC_PSO(III)	-1565.5200	-2638.5400	-4478.0600	-4673.2400
f5	PSO	0.5760	1.3531	1.9903	3.4112
	SASAC_PSO(III)	**0.1270**	0.1598	**0.0776**	**0.1534**
f6	PSO	2.1018	4.0787	5.1683	6.2467
	SASAC_PSO(III)	0.4535	1.7521	2.5902	3.4123

It can be seen from Table 5.3 that all methods of adaptive adjustment of parameters give better results than standard PSO. In all functions and all configurations, these methods are better than standard PSO except on function f4. On this function, PSO is the best. Among the three methods of adaptive adjustment of parameters, it can be seen that the SASAC-PSO method is usually better and the SASAC-PSO method gives quite similar results. Thus, it can be seen that our proposed method is somewhat meaningful to help find better solutions on problems when the solution found by SASAC-PSO has not met the desired requirements.

Table 5.4 Best results in 30 runs of each method

function	Algorithm	Number of experimental dimensions – results			
		5	10	15	20
f1	PSO	0.0034 (3)	0.0218 (2)	0.3031 (29)	0.0151 (2)
	SASAC-PSO(III)	**0** (9)	**0** (30)	**0** (12)	**0** (30)
f2	PSO	0.5080 (14)	8.4709 (14)	14.3711 (15)	19.4059 (7)
	SASAC-PSO(III)	**0** (7)	**4.4205** (11)	10.9482 (11)	**17.3217** (4)
f3	PSO	0.4762 (1)	10.2935 (10)	3.6593 (0)	22.9244 (1)
	SASAC_PSO(III)	0.9950 (27)	**0** (2)	4.9748 (14)	**0.0001** (2)
f4	PSO	-2094.0865 (5)	**-3680.7595** (7)	**-5298.6627** (1)	**-5988.3707** (13)
	SASAC_PSO(III)	-1976.4761 (5)	-3301.5305 (1)	-4290.8913 (14)	-5988.3707 (27)
f6	PSO	0.5541 (28)	1.1733 (2)	3.7323 (10)	4.0750 (30)
	SASAC-PSO(III)	**0** (30)	**0** (15)	1.3404 (1)	1.8400 (20)

Table 5.4 shows the best fitness results of each method in 30 runs, each run is 100 laps. The first column (function) is the name of functions from f1 to f6 (according to table 3.1; standard values of functions reach min[8]); the second column (algorithm) – respectively the algorithms that are conducted PSO experiments, SASAC-PSO with dimensional numbers 5,10,15 and 20 respectively - is represented by the obtained values corresponding to each algorithm, the number recorded in the pair of signs () is the number of runs that the algorithm get the value at that run – eg: function f1, algorithm PSO, at column dimensionality is 5; with the value 0.0034 (3); That is, the function f1 when running the experiment with the PSO algorithm with the number of

dimensions (the number of variables of the function) is 5, the maximum value is equal to 0.0034 at the 3rd run out of 30 experimental runs (The parameters of the algorithms). math are selected the same.Specifically according to the description of the demo program in section 3.4.2). In this table 5.4 we also show the generation in which that best value is found. It can be seen that the results in this table are essentially identical with those in table 5.3 where SASAC-PSO usually gives the best results.

Table 5.5 Average running time of the methods

function	Algorithm	Experimental Dimensions – Time Value			
		5	10	15	20
f1	PSO	18487.94	36392.00	66229.5	111603.1
	SASAC_PSO(III)	**19148.50**	36272.56	67126	110939.3
f2	PSO	16525.81	35457.69	66585.88	114336.6
	SASAC_PSO(III)	**16059**	35457.69	**65636.38**	111697.6
f3	PSO	**16177.75**	**34479.06**	67155.13	111528.5
	SASAC_PSO(III)	16495.31	35493.5	**65382.81**	**109935.6**
f4	PSO	16504.75	36274.25	67754.06	113731.5
	SASAC_PSO(III)	16478.75	35047.19	67754.06	111059.6
f5	PSO	**16188.31**	36170.63	66526.63	**110572.6**
	SASAC_PSO(III)	16231.31	**35075.75**	**66434.31**	112883.8
f6	PSO	16421.44	36036.75	68533.13	250540.8
	SASAC_PSO(III)	17408.69	37907.88	70941.13	116498.0

Table 5.5 is the time-average results over the 30 runs of each method measured in miniseconds. It can be seen from this table that the methods almost all run at the same speed. The difference is insignificant.

6. CONCLUDE

The swarm optimization method is a form of population evolutionary algorithms known before such as genetic algorithm (GA), Ant colony algorithm. However, PSO differs from GA in that it favors using the interactions between individuals in a population to explore the search space. PSO is the result of modeling the flight of birds in search of food, so it is often classified as algorithms that use swarm intelligence. Introduced in 1995 at an IEEE conference by James Kennedy and engineer Russell C.Eberhart. Algorithms have many important applications in all fields where solving optimization problems is required.

In this report, we use the basic PSO algorithm, and the improved PSO algorithms are c-PSO and W-PSO. The use of the basic PSO algorithm and In the future, we want to continue our studies on the methods of adaptively tuning the parameters of the PSO algorithm to further improve the efficiency of the algorithm. We also want to apply these results in solving problems, especially optimizing the storage and search of the Gen-Disease dictionary at our institution.

REFERENCES

1) Engelbrecht, A.P. Fundametals of Computational Swarm Intelligence, John Wiley & Sons, 2005.
2) James Kenedy and Russell Eberhart – Particle swarm optimization, From Proc. IEEE Int’l. Conf. on Neural Networks (Perth, Australia) IEEE Service Center, Piscataway, NJ,IV :1942 -1948.
3) Parsopoulos, K.E and Vrahatis,M . N. Particle Swarm optimization in Noisy and Continuousle Changing Environments. In Proceedings of International Conference on Artificial Intelligence and Soft Computing, 289 – 294, 2002.
4) Brits, R . , Engelbrecht, A. P , anh Van Den Bergh, F. A Niching Particle Swarm Optimization. In Proceedings of the Fourth Asia – Pacific Conference on Simulated Evolution and Learning (SEAL’ 2002), 692 – 696, 2002.
5) Richards, M. and Ventura, D. Choosing a Starting Configuration for Particle Swarm optimization. In Proceedings of the Joint Conference on Newral Networks, 2309 – 2312, 2004.
6) Li Guo and Xu Chen, Institute for Intelligent Computational Science College of Mathematics and Computational Science ShenZhen University, ShenZhen, 518060, China. A Novel Particle Swarm Optimization Based on the Self- Adaptation Strategy of Acceleration Coefficients*

7) F.van der Bergh. An analysis of Particle Swarm Optimizers. PhD thises, Department of Computer Science, university of Pretoria, Pertoria, South Africa, 2002.
8) Marcin Molga, Czeslaw Smutnicki,Test functions for optimization needs 3 kwietnia 2005.
9) P.N. Suganthan. Particle Swarm Optimizer with Neighbourhood Operator. In proceeding of the IEEE Congress on Evolution and Computation, PAGES 1958-1962, IEEE Press, 1999
10) Y.Volkan pehlivanoglu ,Turkish Air Force Academy Aeronautics and Space Engineering Dept. A new particle swarm optimization method for the path planning of uav in 3d environment. Received: 13th January 2012, Accepted: 27th July 2012 .
11) R.C. Eberhart, P.K. Simpson, and R.W. Dobbins. Computational Intelligence PC Tools, Academic Press Professional,first edition, 1996
12) J.F.Schutte and A.A. Groenwold. Sizing Design of Truss Structures using Particle Swarm Optimization. Structural and Multidisciplinary Optimization, 25(4) : 261-269,2003
13) H-Y. Fan. A Modification to Particle Swarm Optimization Algorithm. Engineering Computions, 19(7-8):970-989,2002
14) Y Shi and R.C. Eberhart. A Modified Particle Swarm Optimization. In proceedings of the IEEE Congress on Evolutionary Computation, pages 69-73, May 1998
15) J.Peng,Y. Chen, and R.C. Eberhart. Battery Pack State of Charge Estimator Design using Computational Intelligence Approaches. In Proceedings of the Annual Bttery Conference on Application and Advances, pages 173-179, 2000
16) S.Naka, T.Genji, T.Yura, Y.Fukuyama, and N. Hayashi. Particle Distribution Sate Estimation using Hybrid Particle Swarm Optimization, In IEEE Power Engineering Society Winter Meeting, volume 2, pages 815-820, Jannuary 2001.
17) M.Clerc, Thing locally, Act Locally: The way of Life of Cheap –PSO, an Adaptive PSO, Technical report http://clerc.maurice.free.fr/pso/,2001
18) R.C.Eberhart and Y.Shi.Comparing Inertia Weightand Constriction Factors in Particlelume SwarmOptimization. In Proceedings of the IEEE Congreess evolutionary Computation, volume 1, pages 84 -88, July 2000.
19) A Carlisle and G. Dozier. An off-the-Shelf PSO. In proceeedings of the workshop on ParticleSarm Optimization, pages 429-434,2000
20) J.Kennedy. The Particle Swarm : Social Adapptation of Knowledge. In Proccedings of the IEEE Interational Conference on Evolutionary Computation, pages 303-308, April 1997.
21) R.Brits, A.P.Engelbrecht, and F. vanden Bergh. A Niching Particle Swarm Optimizer. In Procceding of the Fourth Asia – Pacific Conference on Similated Evolution and Learning, pages 692-696, 2002.
22) J.Riget and J.S. Vesterstrom. A Diversity – Guided ParticleSwarm Optimizer- The ARPSO, Technical report, Department of Computer Science, University of Aathus, 2002.
23) G.Venter and Sobiezczanski-Sobieski. Particle Swarm Optimization, Journal for AIAA, 41(8): 1583 -1589,2003
24) http://www.adaptiveview.com/article/ipsopl.html
25) http://www.swarmintelligence.org/tutorrials.php