CHAPTER 3

Particle Swarm Optimization Algorithms

3.1 ORIGINS OF PARTICLE SWARM OPTIMIZATION

This section provides a general outline of how the concepts of particle swarming can be translated into practical algorithms. This type of algorithm arose from the work of Kennedy and Eberhart (1995) which was related to the flocking of birds and the behavior of fish schools. The word boids is frequently used to represent flocking creatures and arises from a shortening of the term birdoid objects. We will briefly discuss the nature of flocking algorithms which simulate flocking behavior. Using these concepts we describe the development of the particle swarm optimization (PSO) method.

The key features of flocking are that each boid has a specific location and velocity and pursues the following aims:

Alignment. A boid will attempt to align itself with other boids by matching their velocity.

Cohesion. The individual boid will tend to steer towards the center of mass of the flock.

Separation. Clearly each boid attempts to avoid bumping into to its near neighbors by keeping a reasonable distance apart, sufficient for collisions to be avoided while not losing sight of the flock.

To implement these concepts as an algorithm we generate, for each boid, an initial set of locations and velocities randomly then the continuous updating of the values of velocity and location is achieved using relatively simple formula that reflects the properties of alignment, cohesion and separation. The result is the simulation of boid flocking over a defined area which enables the flock to explore the region they occupy in an efficient and cohesive way. A real life example of this area of research is work done in Oxford University on the flight of ibis flocks which fly in a Vee formation. Each member of the flock takes it in turn to lead. The ibis work in pairs to achieve an efficient, perhaps optimum, distribution of effort during the flight which leads to a Vee alignment as the optimal choice.

It was noted by Kennedy and Eberhart (1995) that these concepts could be applied to non-linear optimization problems and in particular to the class of very difficult problems where multiple optima are present and where it is important to obtain the global optimum for the problem: clearly the roaming nature of the boid flocking process will

DOI: 10.1016/B978-0-12-803636-5.00003-7

help in the exploration phase of the optimization process which aims to avoid being trapped at local optimum.

The basis of the method of applying these concepts to non-linear optimization problems is achieved by considering a range of candidate solutions for the optima which are the locations of boids within the flock or as a general concept the location of particles in a swarm. These candidate solutions are updated and move around the search region randomly, thus exploring the region for improved solutions which provide values for the optimum of the objective or fitness function. An important feature of these algorithms is that not only may the local optima be located but the global optima can be obtained. Indeed this is usually the main aim, unlike gradient methods where only local optima may be found. In the PSO algorithm, the updating formulae used reflect the flocking behavior. These formulae take account not only of the behavior of individual particles in relation to their near neighbors, but also the global behavior of the group particles as a whole.

The key features which control the efficiency of the algorithm are how thoroughly the region is explored and how the accuracy of solutions is improved and the balance between the two. This process is sometimes called exploration and exploitation.

3.2 THE PSO ALGORITHM

The key properties of the particles of the PSO algorithm are position and velocity. The position of a population of particles is randomly chosen within the boundaries of the search region and the value of the objective function calculated for each particle position. The initial values of the velocities are taken as zero or selected randomly within specific limits. Thus if v_i are the velocities of the i particles then a possible initial setting is:

$$\mathbf{v}_i^{(0)} = 0 \text{ for } i = 1,\ 2,\ \ldots,\ n_{pop} \tag{3.1}$$

where n_{pop} is the size of the population.

There are many alternatives for assigning the initial velocities. Appropriate formula is then used to update the velocity of each particle. Using these values of velocity, the positions of the particles are updated. Then it is determined if improvements to the objective function have been achieved. If improvements are obtained the values of the positions of the particles are accepted as the new positions. In addition the global best position for all particles of the swarm is updated. This process is repeated until a preset convergence criterion is met. Then the process is stopped and the solutions provided. We now give the specific formula for updating the values of velocity and position.

The velocity of each particle is calculated from the following equation at time t; in practice the t values are equivalent to the iteration or generation number of the process, starting with $t = 0$:

$$\mathbf{v}_i^{(t+1)} = \mathbf{v}_i^{(t)} + a\mathbf{r}_1 \circ [\mathbf{x}_{lopt}^{(t)} - \mathbf{x}_i^{(t)}] + b\mathbf{r}_2 \circ [\mathbf{x}_{gopt}^{(t)} - \mathbf{x}_i^{(t)}] \tag{3.2}$$

where the values a and b are in general positive constants set by the user, usually at the start of the process. These values must be carefully selected since they have a significant influence on the success of the process and the nature of convergence of the method giving proper emphasis to the local and global search aspects of the algorithm. The $\mathbf{x}_{lopt}^{(t)}$ vector is the best position vector for the particles calculated using the objective function $f(\mathbf{x}_i)$ in the local region. The $\mathbf{x}_{gopt}^{(t)}$ vector is the current global best position vector. This vector is continuously updated at each iteration and will provide the final optimum location vector. The $\mathbf{v}_i^{(t)}$ and $\mathbf{x}_i^{(t)}$ are the current values of the velocity and position vectors. The values $\mathbf{r}_1$, $\mathbf{r}_2$ are selected from the uniform random distribution vector $\mathbf{r}_u$ in the range 0 to 1 and they are reselected at each step of the algorithm. Note that $\circ$ in (3.2) denotes element by element multiplication, or the Hadamard or Schur multiplication. For example if there **a** and **b** are vectors of two elements then

$$\mathbf{a} \circ \mathbf{b} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \circ \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} a_1 b_1 \\ a_2 b_2 \end{bmatrix}$$

It is important to emphasize that the last two terms of the velocity adjustment equation, (3.2), reflect the two competing features of the algorithm. Firstly to ensure sufficient exploration of the local region to find an acceptably accurate value of the local minimum. Secondly to ensure that the whole region is explored so that a global minimum can be found and consequently avoid getting stuck at a local minimum. Randomness plays a key role in this process. Thus the choice of a and b in (3.2) is crucial in ensuring the compatibility of these two aims and their selection is not a simple matter.

Once the velocity values are updated the new position values can be calculated from:

$$\mathbf{x}_i^{(t+1)} = \mathbf{x}_i^{(t)} + \mathbf{v}_i^{(t+1)} \tag{3.3}$$

This equation drives the process on from point to point, and the new point can then be tested for improvement. Large values of velocity mean the region is explored rapidly but perhaps missing key optima. If changes to $\mathbf{x}_i$ are too small the progress of the algorithm may be very slow. Careful choice of parameters can ensure balanced progress. We now describe the basic steps that are used in the PSO method.

1. Initialize values for velocity variables. Define the constants a and b. Generate randomly an initial population of $\mathbf{x}_i$ values confined to the region of interest. Thus

$$\mathbf{x}_i = \mathbf{x}_{lo} + \mathbf{r}_u \circ (\mathbf{x}_{hi} - \mathbf{x}_{lo})$$

ensures $\mathbf{x}_i$ is in range $\mathbf{x}_{lo}$ to $\mathbf{x}_{hi}$. For the $\mathbf{x}_i$ values, obtain the corresponding objective function values and select the best value of the objective function.

2. Calculate $\mathbf{x}_{lopt}^{(t)}$ and $\mathbf{x}_{gopt}^{(t)}$. Then update the velocity values for each particle using (3.2)
3. Update location values for each particle using (3.3).
4. Calculate the new objective function values and calculate the best values for individual particles and the global best value.
5. Repeat the steps from step 2 until convergence has been achieved.

This completes the algorithm description.

Here we provide MATLAB like pseudo-code for showing the details of the implementation of the calculation of the values $\mathbf{x}_{lopt}^{(t)}$ and $\mathbf{x}_{gopt}^{(t)}$. This assumes an objective function to be minimized has been defined as $f(\mathbf{x})$. For an each value of i.

Algorithm 1 Current Local Best Values.

```
if $f(x(i,:)) < f(xlopt(i,:))$ then
    $xlopt(i,:) = x(i,:)$
end if
if $f(x(i,:)) < f(xgopt(i,:))$ then
    $xgopt(:) = x(i,:)$
end if
```

This algorithm is called the global best particle swarm algorithm because it uses in the update equations the current best overall optima of all the particles i.e. $\mathbf{x}_{gopt}^{(t)}$. A particle swarm algorithm very similar to this one has been developed in parallel with this called the local best particle swarm algorithm.

The basic algorithm we have described has been tested extensively by many researchers on a range of test problems and on many industrial applications and found to be generally successful. However some researchers noted certain shortcomings in the behavior of the algorithm and consequently many suggestions have been made for its improvement. Some of these are described in the next section.

3.3 DEVELOPMENTS OF THE PSO ALGORITHM

Clearly the algorithm has a relatively simple structure but its efficiency depends crucially on the choice of parameters a and b. These values determine the balance between global and local exploration of the region. There clearly is the possibility of an over rapid exploration of the region or too slow an exploration of the region. A very useful and comprehensive review of modifications to the basic PSO algorithm is provided by Engelbrecht (2005). The velocity values drive the rate of exploration and one early change to the structure of the algorithm was to modify the velocity factor by introducing an inertial weight to allow the user to modify the rate and nature of exploration of the search region. We designate w as the inertial weight.

Then we modify the velocity updating formula as follows:

$$\mathbf{v}_i^{(t+1)} = w\mathbf{v}_i^{(t)} + a\mathbf{r}_1 \circ [\mathbf{x}_{lopt}^{(t)} - \mathbf{x}_i^{(t)}] + b\mathbf{r}_2 \circ [\mathbf{x}_{gopt}^{(t)} - \mathbf{x}_i^{(t)}] \tag{3.4}$$

With this relatively simple modification we can adjust the value of w until we are satisfied with the convergence of the algorithm. Larger values of w will promote global exploration of the given region since this will produce a larger increase in the velocity value, whereas smaller values of w, where $0 < w < 1$, will promote the effective and accurate evaluation of the local region. For a detailed explanation of this approach see Chatterje and Siarry (2006). Since this presents us with the problem of selecting the parameter w, a further suggestion for improving the PSO algorithm is that the inertial weight is changed as the algorithm proceeds. Thus we need only initially provide a broad general range of values for the w parameter. The first method we discuss (Eberhart and Shi, 2000) provides the means of gradually adjusting w so the effect of the first term decreases as the optimization proceeds. Thus at iteration t we use the formula:

$$w^{(t)} = (w^{(0)} - w^{(t_{max})})\frac{(t_{max} - t)}{t_{max}} + w^{(t_{max})} \tag{3.5}$$

which provides a linear decreasing value for w. The value of t_{max} is the preset maximum number of iterations and $w^{(0)}$ and $w^{(t_{max})}$ are the initial and final values of w. For example, $w^{(0)} = 0.9$, $w^{(t_{max})} = 0.1$.

The second method modifies the value of w based on a measure of the relative improvement in the function values as approximations to the optimum value are calculated, see Eberhart and Shi (2000). A relative improvement factor $K^{(t)}$ is calculated at each iteration t where:

$$K_i^{(t)} = \frac{f(\mathbf{x}_{gopt}^{(t)}) - f(\mathbf{x}_i^{(t)})}{f(\mathbf{x}_{gopt}^{(t)}) + f(\mathbf{x}_i^{(t)})} \tag{3.6}$$

Then the weights are determined from:

$$w_i^{(t+1)} = w^{(0)} + (w^{(t_{max})} - w^{(0)})\frac{e^{K_i^{(t)}} - 1}{e^{K_i^{(t)}} + 1} \tag{3.7}$$

We note that as the difference between the current function value and the global optimum value becomes smaller on convergence, thus the value of K_i tends to zero hence the exponential value of K_i tends to one. Consequently the second term of (3.7) tends to zero and the weight values change less and less.

Another issue that arises and is related to values of the velocity becoming too high and the particles rapidly diverging. To avoid this the idea of velocity clamping was introduced. One way is to use a constriction coefficient (C) which was introduced as a multiplier of the right hand side of the velocity equation, Clerc and Kennedy (2002). This is calculated from

$$C = \frac{2k}{|2 - \phi - \sqrt{\phi(\phi - 4)}|} \tag{3.8}$$

where $\phi = ar_1 + br_2$. Consequently the velocity equation is modified to include this factor as follows

$$\mathbf{v}_i^{(t+1)} = C\left(\mathbf{v}_i^{(t)} + a\mathbf{r}_1 \circ [\mathbf{x}_{lopt}^{(t)} - \mathbf{x}_i^{(t)}] + b\mathbf{r}_2 \circ [\mathbf{x}_{gopt}^{(t)} - \mathbf{x}_i^{(t)}]\right) \tag{3.9}$$

Using (3.9) which depends on ϕ, if $\phi >= 4$ and $k \in [0, 1]$ then Clerc and Kennedy (2002) show the swarm is guaranteed to converge. The value k controls the nature of the exploration process. If k is small then fast local convergence is achieved if k is large then the region is more thoroughly explored.

Another way to restrict the rate of global exploration of the region by the particles is to introduce a limit on the velocity directly. This is fairly easy to implement and is often used. We consider the velocity components for each particle and impose the following simple restrictions:

$$v_{ij} = \begin{cases} v_{ij} & \text{if } v_{ij} < v_{max,j} \\ v_{max,j} & \text{otherwise} \end{cases} \tag{3.10}$$

The initial values of the $v_{max,j}$ are set arbitrarily by the user, or set to the difference between the maximum and minimum values of the positions of the particle in the jth dimension. Similarly a slow exploration of the region can be avoided by imposing a minimum value of the velocity which it should not fall below. For a detailed explanation of this procedure see Shahzad et al. (2009) and also Ghalia (2008). Clearly the choice of $\mathbf{v}_{max}$ is important and various suggestions have been proposed for its value which may be problem dependent.

The danger of this procedure is that all values may be assigned the same maximum value or the same minimum value and consequently get stuck at these points. Modifications have been introduced to this clamping method that allow the maximum value of the velocity to be changed as the algorithm proceeds. A relatively simple approach is to adjust the clamping velocity according to the following equation:

$$\mathbf{v}_{max}^{(t+1)} = \left(1 - \left(\frac{t}{t_{max}}\right)^p\right)\mathbf{v}_{max}^{(t)} \tag{3.11}$$

Depending on the value of p this will reduce the value of $\mathbf{v}_{max}^{(t)}$ at different rates so taking $p = 1$ provides a linear rate of reduction and as the algorithm proceeds t tends to t_{max} and

$\mathbf{v}_{max}$ tends to zero. The argument being that as we approach the optimum only small changes in velocity are required. Larger values of p will provide more gradual changes in v_{max}. This approach was suggested by Fan (2002). An alternative approach reduces v_{max} according to its current performance in obtaining reductions in the optimum value. Thus if there is no reduction in the current best value after a number of iterations then the value of $\mathbf{v}_{max}$ is changed. This was suggested by Schutte and Groenwold (2003). Additional control parameters have been proposed to avoid premature convergence, see Van den Berg and Engelbrecht (2002).

A further suggested improvement on the basic PSO method is to consider the problem of the selection of good values for the quantities a and b. This can be done on an experimental basis by checking the effect on convergence of different combinations of values of a and b, but this is time consuming. Alternatively these coefficients may be adjusted from iteration to iteration. This method was suggested by Ratnaweera et al. (2004). They studied the use of the formulae:

$$a = (a_{min} - a_{max})\frac{t}{t_{max}} + a_{max} \tag{3.12}$$

$$b = (b_{min} - b_{max})\frac{t}{t_{max}} + b_{max} \tag{3.13}$$

Here b_{max} and b_{min} are values selected to provide a bound on the values that b can take. Similarly a_{max} and a_{min} are values selected to give an appropriate range of values for a. Clearly as the process continues, t, the current iteration number, will approach t_{max}. This is defined as the maximum number of iterations allowed for the execution of the algorithm in a specific case and thus b will tend to b_{min} and similarly a will tend to a_{min} as the iterations proceed.

One major problem that occurs with the PSO method is that of premature convergence in this case a solution is reached but is not the true global optimum. To deal with this important problem a major alteration to the original PSO algorithm has been suggested, called the guaranteed convergence PSO or GCPSO.

The guaranteed convergence PSO (GCPSO) method requires updating formulae for both the position and velocity. However, the updating formulae have major differences when compared with the original PSO updating formulae and for a detailed description of the method see Van den Berg (2006). The key feature of this modification is that adjustments are made which by random variation provoke further search in the region of an optimal location to avoid premature convergence. It should be noted that this modification of the algorithm may not lead to faster convergence but may improve the consistency of the performance of the algorithm over the whole range of problems.

3.4 SELECTED NUMERICAL STUDIES USING PSO

The range of modifications of the basic PSO method we have described clearly need to be examined to see how effective they are, indeed some are alternative ways of doing the same thing.

We now provide some practical insight into the nature of the PSO by using it to solve some standard non-linear optimization test problems. It should be emphasized that these studies are intended to be illustrative rather than exhaustive. Studies are now undertaken on three test problems illustrating specific features of optimization. The first problem, Rosenbrock's function which has only one local minimum, the second problem due to Styblinski and Tang has a small number of local optima and a specific global minimum the third, Rastrigin's function has many local minima and consequently presents a harder test for finding the global minima. None of these problems demand excessive computer time for low dimensional problems and consequently may be easily replicated by the reader.

Rosenbrock's function is a classic test problem for gradient optimization methods and because the single optimum is located in a long shallow curved valley, it presents significant difficulties and takes many iterations to locate the optimum. It is used in these tests to show that the methods can also solve difficult single optima problems. The minimum value is $f(x) = 0$ and is obtained when $x_i = 1$ for $i = 1, 2$. The problem is usually defined for test purposes in the range $[-5, 5]$ for each variable, although the range $[-5, 10]$ is sometimes used.

The Styblinski–Tang function for two variables is not a demanding problem but has various local minima so it is a useful test to see that the global minimum is found at the point $x_1 = -2.903534$ and $x_2 = -2.903534$. With global minimum at -78.3323. (See Figure 3.6.) The problem is usually tested in the range of values for the independent variables x_1 and x_2 in the range $[-5, 5]$. This problem can easily extended to many dimensions.

The final problem we can consider in this section is Rastrigin's function which is a more demanding problem since it has many closely packed local minima of similar value and a significant danger of an optimization method becoming trapped in a particular local minimum rather than finding the global minimum. This has a global minimum at $\mathbf{x} = [0, 0]^\top$ with $f(x) = 0$. The problem is usually tested in the range $[-5, 5]$ for the independent variables.

We now use a modified version of the PSO algorithm and test it on the functions we have described in various ways. Figure 3.1 shows the convergence for the much more difficult problem of minimizing the Rastrigin function in 6 dimensions or variables. It shows a gradual reduction in the function value over the 2000 iterations and appears to indicate periods where little change occurs in the current best value of the objective function as the area is searched for improvements.

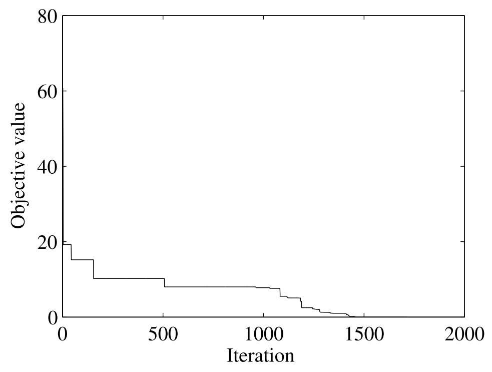

Figure 3.1 Minimizing Rastrigin's function with 6 variables, showing progress of convergence.

Table 3.1 Effect of number of iterations on the estimate of the minimum value of the function. $a = 2.05$ and $b = 2.05$

Iterations	ROS2	S-T2	RAS2	RAS4
200	3.3629×10^{-4}	-78.3323	1.1267×10^{-9}	2.2052
400	4.4627×10^{-7}	-78.3323	0	0.020526

Table 3.2 Effect of swarm size on the estimate of the minimum value of the function after 200 iterations. $a = 2.05$ and $b = 2.05$

Swarm Size	ROS2	S-T2	RAS2	RAS4
20	3.3629×10^{-4}	-78.3323	1.1267×10^{-9}	2.2052
10	6.3392×10^{-4}	-78.3323	5.7565×10^{-7}	2.2799

The most basic features that effect the performance of the algorithm are the number of iterations and the size of the swarm. In Table 3.1 we give some results for the test problems that show the difference between using 200 iterations and 400 iterations. In Tables 3.1 and 3.2, ROS2 indicates Rosenbrock's function with two variables, S-T2 indicates the Styblinski–Tang function in two variables and RAS2 and RAS4 indicates Rastrigin's function in two and four variables, respectively.

Recalling that the solution of the Rosenbrock and the two and four variable Rastrigin functions are zero and the solution of the Styblinski and Tang is −78.3323, Table 3.1 shows small or no improvements in the minimum values except for the last problem which is a more demanding one with four variables and shows a very large improvement with 400 iterations.

Similarly the effect of swarm size can be studied. Various researchers have recommended different swarm sizes, usually 10, 20 or 30 but sometimes more, depending on the problem. Swarm size is very important parameter since too few members of the swarm give poor results rapidly but too many members generally give more accu-

Table 3.3 Minimization of the function RAS4. 800 iterations

Swarm Size	Mean	Best	Worse	St Dev
10	0.5972	0	1.9899	0.7500
20	0.3447	0	5.8967	1.3255
30	1.0658×10^{-15}	0	2.1361×10^{-14}	4.7665×10^{-13}

Table 3.4 Minimization of the function RAS4. 2000 iterations

Swarm Size	Mean	Best	Worst	St Dev
10	0.0497	0	0.9950	0.2225
20	2.5743×10^{-12}	0	5.1486×10^{-11}	1.1513×10^{-11}
30	0	0	0	0

Table 3.5 Minimization of the function RAS6. 2000 iterations

Swarm Size	Mean	Best	Worst	St Dev
10	1.3318	0	37479	1.0550
20	0.5472	0	1.9899	0.7553
30	0.1996	0	0.9950	0.4080

rate results more slowly. Indeed a very large swarm size amounts to little more than enumeration of the possible solutions and a complete loss of efficiency.

Table 3.2 sets the number of iterations at 200 and illustrates the effect of different swarm sizes. The table shows only slightly less accurate results for the smaller swarm size for those problems which are not demanding.

A more thorough test is to perform many runs of the algorithm and take the mean of the results, thus smoothing out some of the effects of randomness. The results shown in Table 3.3 and 3.4 for the minimization of the RAS4 test function, using 20 runs. Two tests are performed, one with 800 iterations and one with 2000 iterations. Since the minimum value of this function is zero, it is clear that the larger number of iterations improves the accuracy of the solution and the larger the swarm size the better the optimum obtained for this particular function. To provide further information about the performance of the method we have included the best and worst results, together with the mean and the standard deviation of these results.

As a further illustration we minimize the Rastrigin function but with six variables (RAS6), a much harder problem. In this set of runs we have used 2000 iterations and swarm sizes 10, 20 and 30. Taking 20 runs gives the results shown in Table 3.5. Contrast these results with those of Table 3.4 which illustrates how the algorithm performs on the same function but with only four variables rather than six. The results shown in Table 3.4 are clearly much better and illustrate how increasing the number of variables in

Table 3.6 Effect of parameters *a* and *b* on optimization of RAS4. 800 iterations

Set	Mean	Best	Worst	St Dev
Set 1	0.0565	0	0.9950	0.2230
Set 2	0.2985	0	0.9950	0.4678

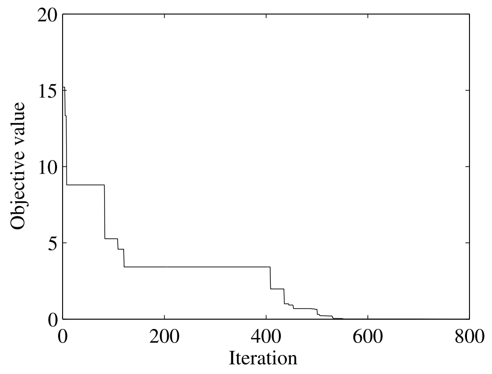

Figure 3.2 Changes in the objective function value for parameter set 1.

a problem disproportionately increases the problems difficulty. This is frequently called the "curse of dimensionality".

We now consider the sensitivity of the performance of the algorithm to other parameter choices and this will be illustrated using sample test problems. We will also study the effects of some of the modifications to the algorithm suggested by workers in the field. This will be achieved by considering the effects of these modifications individually on the behavior of the algorithm and comparing the results for some test problems. Specifically we will consider the importance of the choice of the parameters a and b on the performance of the algorithm and the alternatives methods for setting the values of the inertial weights, w. The first study compares the performance of the PSO algorithm using values of the parameters $a = 2.05$ and $b = 2.05$, called set 1, with the use of $a = 0.7$ and $b = 1.4$, called set 2. A swarm size of 30 is used with 800 iterations. The function considered is the Rastrigin function with 4 variables, denoted by RAS4. The results for 20 runs is given in Table 3.6. There is significant improvement with set 1.

As a graphical illustration, Figure 3.2 and Figure 3.3 show the progress of the iterations for an individual run for the two parameter sets. There is significant difference between the convergence paths but this could be because of the random nature of the process since it only involves one run of the PSO algorithm.

A further test considers how taking a specific value for the inertial weight compares with the continuous adjustment of the weight as the procedure continues. A swarm size

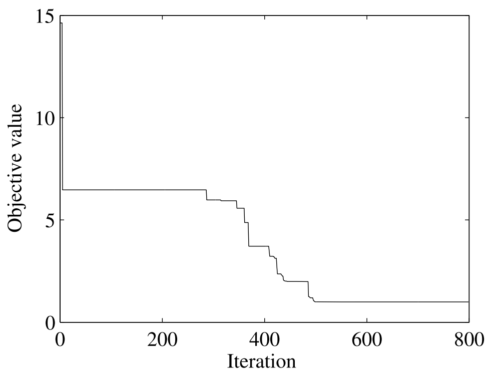

Figure 3.3 Changes in the objective function value for parameter set 2.

Table 3.7 Effect of constant and variable weight, *w* in finding the minimum of RAS4

Set	Mean	Best	Worst	St Dev
0.7	0.0497	0	0.9950	0.2225
Variable w	1.0452×10^{-7}	0	2.0905×10^{-6}	4.6744×10^{-7}

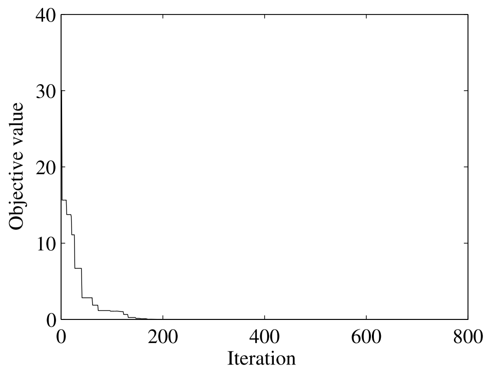

Figure 3.4 Iteration to determine the minimum of RAS4. $w = 0.7$.

of 30 is used with 800 iterations. The function considered is the Rastrigin function with 4 variables, RAS4. The results for 20 runs of the algorithm are given in Table 3.7. The table shows that there does seem to be a significant difference between the results, although further extensive testing should be performed before drawing hard and fast conclusions.

Figure 3.4 illustrates the progress of algorithm for the RAS4 function using w fixed at 0.7. Figure 3.5 shows the progress using w varied as the process proceeds.

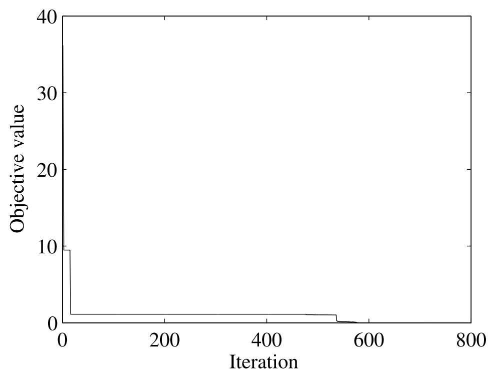

Figure 3.5 Iteration to determine the minimum of RAS4 using a variable w.

There is a difference between the nature of the graphs which may reflect the progress of the algorithm in the two cases. With the varying w the progress is initially rapid, then pauses for many iterations, until convergence is achieved, perhaps reflecting more searching for a better solution. Many more studies would be required to confirm this.

We now illustrate the performance of the swarm of particles graphically as they converge to the global minimum. This is shown by plotting the swarm particle locations on contour graphs of the Styblinski–Tang, Rastrigin and Rosenbrock functions in two variables. The position of the points at various stages of the iteration process is plotted on a series of contour graphs.

We begin with the Styblinski–Tang function by plotting the initial particle position values, then the position of the particles after fifty, one hundred and three hundred iterations, each set of particles on a separate contour plot of the function. This is shown in Figure 3.6.

We note that these graphs show rapid convergence to the global minimum of the function at the point $(-2.9035,\ -2.9035)$. In Figure 3.6, 3.7 and 3.8 the individual particles are shown by filled circles.

A second example illustrates the convergence behavior for the Rastrigin function which has many closely packed minima in the region $x = -5$ to 5, $y = -5$ to 5. Figure 3.7 shows the convergence of the swarm. For clarity only the locations of the many minima are shown, rather than a conventional contour plot. Effective convergence to the global minimum at $(0,\ 0)$ is achieved after three hundred iterations and convergence to the many nearby local minima is avoided.

As a final example we show how the algorithm behaves with Rosenbrock's function which has a single unique minimum at $(1,\ 1)$. The results for a run with the PSO algorithm for the optimization of Rosenbrock's function are shown in Figure 3.8. The contour graph of Rosenbrock's function shows a shallow banana shaped valley, it rep-

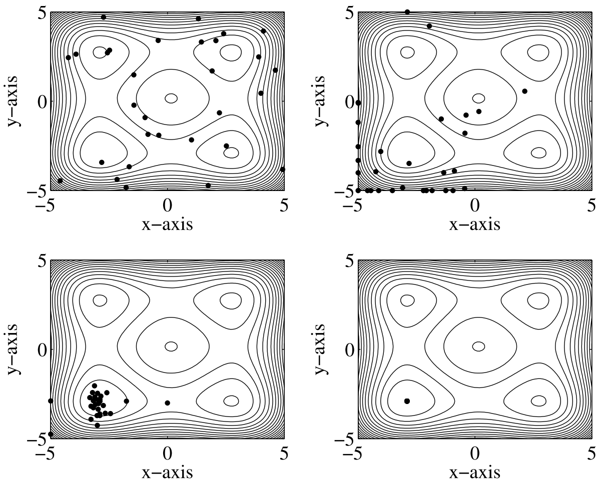

Figure 3.6 Contour plot of the Styblinski-Tang function in two variables, showing the progress of the particle swarm to convergence for 0, 50, 100 and 300 iterations.

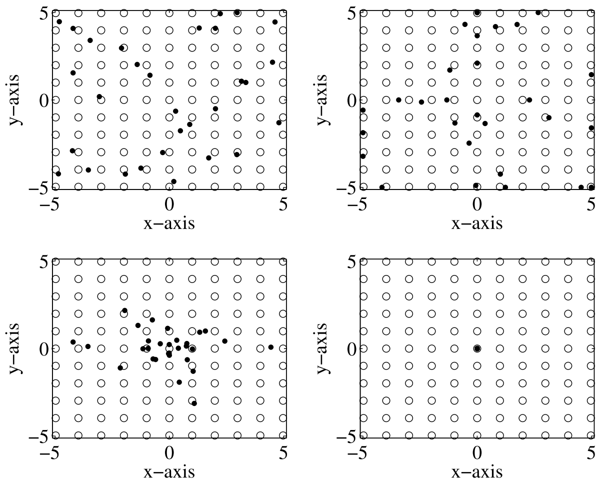

Figure 3.7 Rastrigin function in two variables, showing the progress of the particle swarm to convergence, for 0, 50, 100 and 300 iterations. The open circles show the many local minima.

resents a significant challenge for gradient methods because of its shallowness, however the PSO method copes well with this function. Although convergence is more gradual for this function, the points are converging on the required single minimum.

MATLAB scripts have been used throughout to implement these tests and to generate the graphical representation.

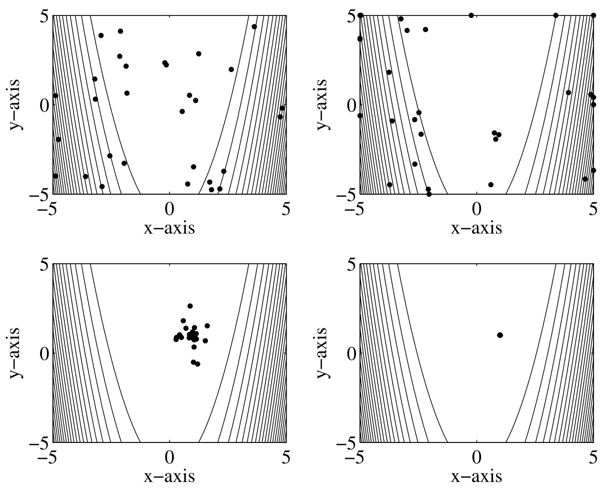

Figure 3.8 Contour plot of the Rosenbrock function in two variables, showing the progress of the particle swarm to convergence, for 0, 50, 100 and 300 iterations.

3.5 A REVIEW OF SOME RELEVANT DEVELOPMENTS

The method outlined in this chapter is one of the earlier biologically inspired algorithms to be used for optimization and has provided inspiration for the development of new algorithms. In this section we will consider recent developments in the use of the algorithm and describe the comparative studies of the particle swarm algorithm with other competitive methods.

It is important to test any new development on a standard set of test problems together with equitable setting of parameters so that the tests are fair. Tests need to be cleverly devised so that they check the key feature of the algorithm. That is to find global optima with good accuracy and reasonable efficiency. A key feature is to avoid becoming stuck at a local optima; a particularly challenging problem where local optima are tightly packed or the function profile provides rapid or near discontinuous change. Standard test problems reflect these features.

An interesting comparative study has been made between particle swarm optimization and GAs by Hassan et al. (2005). This study compared performance on some standard test problems, on a telescope array optimization and a satellite design problem and found both methods discovered good solutions for the problems but the particle swarming method was more efficient.

It is not possible to cover all the most recent developments in the field of PSO but we have selected papers that provide in our judgment interest and value and deserve further study. The following references outline some interesting developments:

We first consider the work of Arasomwan and Adewumi (2013). This work considers the problem of the best choice of inertial weights for the Particle Swarm Optimization algorithm. The use of inertial weights was an early development in the PSO algorithm which is generally accepted to provide an improvement to the basic algorithm. However over the years researchers proposed various alternative as to how these weights should be set and adjusted. The Arasomwan and Adewumi paper provides an assessment of these competing techniques, the authors point out it is important to test these competing methods in a completely equitable way, a difficult aim to achieve. They review and compare a range of methods which we will briefly describe.

Rather than using a fixed weight an early development was proposed by Eberhart and Shi (2000) which suggested a simple linear adjustment to the weight which reduced its value as the algorithm proceeded: called Linear Decreasing Inertia Weight (LDIW-PSO) procedure. This modification has been described earlier in this chapter. However it is found with this modification that the algorithm may still get stuck at a local optimal point rather than find a global optimum. Consequently a simple adjustment was suggested to the LDIW method. In this method introduced by Feng et al. (2007) a further chaotic factor is introduced that modifies the end weight value. The authors of the method called the Chaotic Descending Inertial Weight PSO (CDIW-PSO) reported significant improvement in the algorithms performance. Another method suggested by Gao and Duan (2007) called the Random Inertia Weight and Evolutionary Strategy PSO, (REPSO), introduced an interesting combination of PSO and Simulated Annealing. The weights being chosen at each stage using an annealing probability. Again Gao and Duan reported good results for this algorithm. Optimization problems of high dimension and a multi peak profile present challenging optimization problems. The method of Dynamic Particle Swarm Optimization, (DAPSO) introduced by Xin et al. (2009) was introduced to deal with premature convergence and provide good convergence speed for this type of problem. It modified the standard linear weight adjustment formula using factors using estimates of particle group fitness at each iteration. Promising results were reported for this method. Another method considered in this paper was adaptive swarm optimization introduced by Alfi and Modares (2011). In this algorithm the weights were adjusted at each iteration using a function of the fitness of the current best solution. In addition, particles are chosen for mutation and a Gaussian random value added to the particle. The final two variants considered by Arasomwan et al. were Dynamic Nonlinear and Dynamic Logistic Chaotic Map PSO, (DLPSO2). These were introduced by Liu et al. (2009). One method introduced a dynamic nonlinear factor to the standard linear weight adjustment formula. In addition it used a variant of the chaotic map technique given in the CDIW-PSO method, providing a multi-faceted search procedure. Liu et al. reported good results for this method. Clearly an independent analysis of these competing claims would be useful.

Arasomwan et al. embarked on a carefully designed and detailed comparative study on the methods we have briefly described above. The authors maintain that to provide a fair comparison for the LDIW method its parameters must be properly set. With this setting of the LDIW parameters the authors provide a range of numerical results for the comparison of the methods described with the LDIW method for range of test problems. Their conclusion is that the linear dynamic inertia weight adjustment introduced by Shi and Eberhart (1998) with appropriate parameter settings performs competitively with all the variants described above.

The reader is advised to make their own judgments about algorithm performance by referring to the papers we have cited. The paper by Arasomwan et al. does highlight the difficulties of comparative studies and the different conclusions that may be reached.

Another paper we discuss addresses the major problem of global optimization algorithms, that is how to avoid premature convergence to a local optimum. The work of Napoles et al. (2012) addresses the fundamental problem of global optimization methods which is premature convergence to a global optimum, from which the algorithm is unable to escape. This is frequently possible since in many problems the locations of the optima may be tightly packed or distantly isolated from the other minima. Two questions arise when dealing with this problem: these are:

1. How can the fact that the algorithm is trapped near a local optimum be detected?
2. How can the algorithm be stimulated to move to search a wider area of territory in which the function is defined?

A simple approach to the first question is to declare that the algorithm is trapped, if after a large number of iterations, there is no improvement in the value of the function value, but this could mean the optimum has been reached and it is unclear how to set the specific value of the large number of iterations. Since this approach has obvious shortcomings a more subtle approach is needed. One method involves the use of the criteria for judging if premature convergence has occurred. This is by using the maximum radius of the swarm. Figure 3.6 illustrates how the swarm gradually clusters around the optimum point. Clearly this may reduce the diversity of the search process and it is important to be able to detect this state. To do this, according to Napoles et al., we define the point $\mathbf{x}_g$ as the global best point in the neighborhood and $\mathbf{x}_i^{(k)}$ as the position of the other particles for $i = 1, \ ,\ldots,\ n_{pop}$ at iteration k in the swarm then the expression:

$$\| \mathbf{x}_i^{(k)} - \mathbf{x}_g \| \text{ for } i = 1,\ 2,\ \ldots,\ n_{pop} \tag{3.14}$$

gives the Euclidean distance between the global best point and each of the particles in the swarm. Here n_{pop} is the population size. Consequently the expression:

$$\max_{i=1:n_{pop}} \| \mathbf{x}_i^{(k)} - \mathbf{x}_g \| \tag{3.15}$$

represents the maximum distance of a typical particle in the swarm from the global best particle found. Let r_{max} and r_{min} be the range of possible values for particle position values, then the we can normalize the maximum distance as follows:

$$s^{(k)} = \frac{\max_{i=1:n_{pop}} \parallel \mathbf{x}_i^{(k)} - \mathbf{x}_g \parallel}{|r_{max} - r_{min}|} \tag{3.16}$$

Thus if the value of $s^{(k)}$, the current radius of the swarm, is less than a preset value, premature convergence is deemed to have occurred since the points are relatively closely clustered around the current best point. This is used to distinguish a premature convergence state from a global one.

Napoles et al. (2012) proposed to deal with the second question of moving to a better optimum by a process he calls random sampling in variable neighborhoods. The concept of separate neighborhoods was introduced by Hansen and Mladenovic (2001). The aim of this is to allow the particles to move so that a better point may be found giving an improved objective function value. However the movement must be conducted in an organized way, so that good information is not lost and that the process is not trapped again.

Napoles et al. defined the ranges for the new sub-regions and selects uniformly distributed random values for each dimension of the points in each of these new ranges. Taking the union of these points or suitable subsets of these points produces the new set of points for the swarm. The selected subsets are described by the Napoles et al. as points selected as good enough particles by an elitist criterion. The reader is referred to Napoles et al. for details of this algorithm.

Napoles et al. also discussed a simple variant of the algorithm we have described which instead of using the radius of the swarm uses preset maximum number of function evaluations to determine if the algorithm is trapped. This is a less costly procedure in terms of computation time and they report it to be fully competitive with the form of the algorithm using the swarm radius. Further tests were performed by the author to compare the performance of the algorithm with other variants of the PSO algorithm, the attraction-repulsion based PSO, the Quadratic Interpolation based PSO, the Gaussian Mutation based PSO and a hybrid variant of the PSO using simulated annealing. Napoles et al. state that the new algorithm produces superior results in most cases.

3.6 SOME APPLICATIONS OF PARTICLE SWARM OPTIMIZATION

Particle Swarm Optimization (PSO) is a well established algorithm and is often cited in the literature and reported to have been applied to solve efficiently numerous problems which arise in real life. Here we indicate the nature of a small selection of these. In papers by Kang et al. (2012) and Gökdağ and Yildiz (2012) beams are modeled using the finite element method and damage is expressed by stiffness reduction at a global level. It is reported that the resultant optimization problem has been successful solved using the PSO algorithm.

In a paper by Hassan et al. (2005) the authors describe an application to minimize the weight of a simple gearbox for a light aircraft. In addition, they describe the optimum design of the distribution of a group of small telescopes so that they jointly provide the resolution of a larger, single telescope. This is an important problem for astronomers dealing with scarce financial resources and requiring optimum observational results.

In his MSc thesis, Talukder (2011) has produced a useful listing of applications of the PSO method, include many Biomedical applications; for example the diagnosis of Parkinson's through body tremors. He also described applications in robotics, including the development of the robot running process, and to problems in graphics, including character recognition.

3.7 SUMMARY

The method outlined here is one of the earlier biologically inspired algorithms to be used for optimization of non-linear functions. It has provided a useful impetus in the development of new algorithms inspired from many other facets of the manner in which the natural world agents seek to solve problems efficiently.

We have described the nature of the basic PSO algorithm and some of the many suggested changes to this algorithm to improve its performance. To examine the efficiency of the basic PSO algorithm we have tested its performance on a number of standard test problems, some of which have multiple minima, and have examined how effective the method is in finding the best of these minima, the global minima.

In addition the some of the modifications we have described have been tested systematically on the same standard test problems to detect if any significant improvements have been achieved. It is always dangerous to draw hard and fast conclusions from such tests but they give some insight into the nature of the algorithm and the effects of specific modifications. These studies are illustrations rather than pieces of research.

3.8 PROBLEMS

3.1 We wish to minimize the objective function $f(\mathbf{x}) = x_1^2 + x_2^2$. Assuming an initial particle swarm population $\mathbf{x}_1 = [2.5,\ 1]^\top$ and $\mathbf{x}_2 = [2,\ 1]^\top$ calculate the objective function values for this population and hence $\mathbf{x}_{lopt}$ and $\mathbf{x}_{gopt}$.

3.2 Use the same objective function and initial population as in Problem 3.1. Take the initial velocity vector as $[0,\ 0]^\top$ and set $a = 1$, $b = 1$. Use the random vectors $\mathbf{r}_1 = [0.23\ 0.45]^\top$ and $\mathbf{r}_2 = [0.42\ 0.65]^\top$ and perform one iteration of the PSO algorithm using (3.2) and (3.3). Note: After the value of $\mathbf{x}$ has been updated you must update $\mathbf{x}_{lopt}^{(0)}$ and $\mathbf{x}_{gopt}^{(0)}$.

3.3 Taking $w^{(0)} = 1$ and $w^{(t_{max})} = 0.2$ where $t_{max} = 10$ use equation (3.5) to generate a range of weights between the values 1 and 0.2.

3.4 Plot a graph showing how v_{max} varies with iteration t using formula (3.11). Take $t_{max} = 10$ and hence $t = 0 : 10$. Draw three graphs taking: $p = 1,\ 2,\ 3$.