Particle Swarm Optimization - A Tutorial

Alaa Tharwat[1,*], Tarek Gaber[2,*], Aboul Ella Hassanien[3,*], and Basem E. Elnaghi[1]

[1]*Faculty of Engineering, Suez Canal University, Ismailia, Egypt*

[2]*Faculty of Computers and Informatics, Suez Canal University, Ismailia, Egypt*

[3]*Faculty of Computer and Information, Cairo University, Egypt*

[*]Scientific Research Group in Egypt (SRGE)

http://www.egyptscience.net

ABSTRACT

Optimization algorithms are necessary to solve many problems such as parameter tuning. Particle Swarm optimization (PSO) is one of these optimization algorithms. The aim of PSO is to search for the optimal solution in the search space. This paper highlights the basic background needed to understand and implement the PSO algorithm. This paper starts with basic definitions of the PSO algorithm and how the particles are moved in the search space to find the optimal or near optimal solution. Moreover, a numerical example is illustrated to show how the particles are moved in convex optimization problem. Another numerical example is illustrated to show how the PSO trapped in a local minima problem. Two experiments are conducted to show how the PSO searches for the optimal parameters in one-dimensional and two-dimensional spaces to solve machine learning problems.

Particle Swarm Optimization (PSO), Optimization problems, k-NN, Support Vector Machine (SVM), Meta-heuristic optimization Algorithms

INTRODUCTION

Swarm optimization techniques are recent techniques used to solve many optimization problems. They are employed in image processing (Mostafa et al., 2015; Ali et al., 2016), machine learning (Yamany et al., 2015a; Tharwat et al., 2015c; Ibrahim & Tharwat, 2014; Tharwat et al., 2015b), power electronics (Yoshida et al., 2000), and numerical problems (Vesterstrom & Thomsen, 2004), and mechanical problems (dos Santos Coelho, 2010). The optimization can be defined as a mechanism through which the maximum or minimum value of a given function or process can be found. The optimization is usually used in different fields such as economics, physics, and engineering where the main aim is achieve maximum production, efficiency, or some other measure. In addition, there are many scientific, social, and commercial problems which have various parameters which if they have been adjusted can produce a more desirable outcome. Generally, the optimization could be used to achieve maximization or minimization where the maximization of a given function f is equivalent to the minimization of this function opposite, -f (Van Den Bergh, 2006; Gaber et al., 2016).

In the past years, a wide range of optimization techniques were proposed to solve such optimization problems. These techniques generally make use of two main kinds of learning techniques in the literature: stochastic or random versus deterministic. First, in deterministic techniques, the algorithm results in the same accuracy if the algorithm uses the same initial values. On the contrary, stochastic methods employ stochastic optimization algorithms and it achieved different results due to randomness. There are two main problems of the stochastic algorithms due to randomness. First, the goodness of the obtained solution depends mainly on the initial random solution. Hence, different initialization may lead to different solutions. Second, there exist greater probabilities of local optima problem. This problem is common in almost all optimization algorithms (Ho et al., 2007; Tharwat et al., 2016).

The optimization algorithms have two main phases: exploration and exploitation. In the exploration phase, the goal is to explore the search space to search for the optimal or near optimal solutions. Hence, exploration may lead to new search regions that may contain better solutions. In the exploitation phase, the aim is to search locally around the current good solution(s) this is so-called exploitation. The optimization algorithms should make balance between a random selection and greedy selection to bias the search towards better solutions, i.e. exploitation, while exploring the search space to find different solutions, i.e. exploration (Yamany et al., 2015a,b; Tharwat et al., 2015c,a).

One of the widely used swarm-based optimization techniques is the Particle Swarm Optimization (PSO). PSO has a small number of parameters which control the movements of the particles inside the search space. PSO has been used in many applications such as machine learning (Subasi, 2013), image processing (Maitra & Chatterjee, 2008), and power electronics (Miyatake et al., 2011). In general, PSO algorithm lends itself strongly to exploitation phase which may lead to local optima problem. Hence, in some cases, PSO fails to find the global optimal solution.

This chapter aims to give a detailed tutorial about the PSO algorithm and it is divided into five sections. Section 2 summarizes the related work of different applications that used PSO algorithm. A clear definition of the PSO algorithm and its background are highlighted in Section 3. Section 4 illustrates numerical examples to show how the particles are moved and how the PSO algorithm trapped into a local optima problem. In Section 5, two experiments are conducted to show how the PSO algorithm solved machine learning in one-dimensional and two-dimensional spaces. Finally, concluding remarks will be given in Section 6.

RELATED WORK

PSO is widely used in image processing. For example, Maitra et al. used PSO algorithm to find the optimal thresholding value in image segmentation (Maitra & Chatterjee, 2008). Moreover, Akhilesh et al. used a new variant of the PSO algorithm for image in segmentation (Chander et al., 2011). This new variant adapting social and momentum components of the velocity equation for particle move updates and their proposed algorithm outperformed genetic Algorithm (GA) and the standard PSO algorithm. Forouzanfar et al. used PSO to search for the optimal parameters of Fuzzy c-means algorithm brain MR image segmentation (Forouzanfar et al., 2010). They used PSO to find the optimum value of degree of attraction and they found that PSO achieved results better and converged faster than GA.

In machine learning, PSO algorithm is widely used to search for the optimal parameters of learning algorithms to enhance the classification performance. For example, PSO was used to search for the Support Vector Machine (SVM) parameters (Subasi, 2013). Moreover, PSO was used to adjust the weights of Back Propagation Neural Networks (BPNN) and it achieved good results compared with BP algorithm (Bashir & El-Hawary, 2009). In addition, PSO was used for feature selection (Lin et al., 2008).

In clustering, PSO was used to search for the centroids of a user specified number of clusters and the PSO achieved good results compared with K-means clustering (Van der Merwe &Engelbrecht, 2003).

PSO algorithm is also used to solve mathematical problems. Vesterstrom et al. compared PSO with Differential Evolution (DE) algorithm using widely used bench mark functions (Vesterstrom & Thomsen, 2004). They found that DE achieved results better than PSO. In another research, Zhao Xinchao proposed perturbed particle swarm algorithm (pPSA) algorithm (Xinchao, 2010). He used the new algorithm to search for the optimal solutions for 12 functions and he found that the proposed algorithm achieved results better than PSO algorithm. Moreover, Ho et al. used a modified PSO to solve multimodal functions of inverse problems and they achieved good results compared with Tabu search (Ho et al., 2007).

In power applications, PSO algorithm was used for different purposes. Miyatake et al. used PSO algorithm to find the maximum power point tracking of multiple photovolatic and they found that PSO took two seconds to find the global maximum power point (Miyatake et al., 2011). Sidhartha Pandan and Narayana Prasad Padhy used PSO algorithm was used to locate the optimal location of the static synchronous compensator (STATCOM) and its coordinated design with power system stabilizers (PSSs) (Panda & Padhy, 2008). They found that the PSO algorithm improved the stability of the system. Moreover, M. A. Abido used PSO algorithm to optimal design of multimachine power system stabilizers (PSSs) (Abido, 2002).

PARTICLE SWARM OPTIMIZATION (PSO)

PSO algorithm was discovered by Ryenolds and Heppner and the algorithm was simulated by Kennedy and Eberhart (Heppner & Grenander, 1990; Reynolds, 1987; Eberhart & Kennedy, 1995). The PSO is an easy algorithm; hence, it has been used in a wide range of applications (Kennedy, 2010; Kulkarni & Venayagamoorthy, 2011; Akay, 2013; Ishaque & Salam, 2013; Elbedwehy et al., 2012). The main goal of the PSO algorithm is to search in the search space for the positions/ locations that are close to the global minimum or maximum solution(s). The dimension of the search space is determined by the number of parameters that are needed to optimize. For example, if the search space has n dimensions, so the numbers of variables in the objective function is also n. The PSO algorithm has many parameters. The current position of each particle is used to calculate the fitness value at that location. Each particle has three parameters, namely, position ($x^i \in R^n$), velocity, i.e. rate of position change, (v^i) and the previous best positions (p^i). In addition, the position of the particle that has the best fitness value is called global best position and it is denoted by G (Yang, 2014).

The position of each particle is denoted by xi and it is represented by a set of coordinates that represents a point in the search space. During the searching process, the current positions of all particles are evaluated using the fitness function. The fitness value of each particle is then compared with the current position and the best position is stored in the previous best positions (p^i). In other words, the previous best positions store the positions of the particles that have better values.

$$x^i(t+1) = x^i(t) + v^i(t+1) \quad (1)$$

Each particle is moved by adding the velocity to the current position as follows:

$$v^i(t+1) = wv^i(t) + C_1 r_1(p^i(t) - x^i(t)) + C_2 r_2(G - x^i(t)) \quad (2)$$

where w is the inertia weight, C_1 represents the cognition learning factor, C_2 indicates the social learning factors, r_1, r_2 are the uniformly generated random numbers in the range of [0,1], and p^i is the best solution of the i^{th} particle. Since the velocity of the particles depends on random variables; hence, the

particles are moved randomly. Thus, the motion of the particles is called random walk (Yang, 2014). From Equation (2), the new velocity of each particle in the search space is determined by:
1) The original velocity or current motion of that particle (wv^i (t)).
2) The position of the previous best position of that particle that is so-called particle memory or cognitive component. This term as shown in Equation 2 is used to adjust the velocity towards the best position visited by that particle ($C_1 r_1$ (p^i (t) - x^i (t))).
3) The position of the best fitness value is called social component ($C_2 r_2$ (G - x^i(t))) and it is used to adjust the velocity towards the global best position in all particles (Hassan et al., 2005).

High values of the updated velocity make the particles very fast, which may prevent the particles from converging to the optimal solution; thus, the velocity of the particles could be limited to a range [$-V_{max}$, V_{max}]. This is much similar the learning rate in the learning algorithms. A large value of V_{max} expands the search area; thus, the particles may move away from the best solution and it cannot converge correctly to the optimal solution. On the other hand, a small value of V_{max} causes the particles to search within a small area, but it may lead to slow convergence. The positions and velocity of all particles are changed iteratively until it reaches a predefined stopping criterion (Eberhart & Kennedy, 1995; Kennedy, 2010).

The new positions of all particles are then calculated by adding the velocity and the current position of that particle as in Equation (1). The PSO algorithm utilizes the current x^i,p^i,v^i, and G, to search for better positions by keep moving the particles towards the optimal solution. The details of the PSO algorithm are summarized in Algorithm (1).

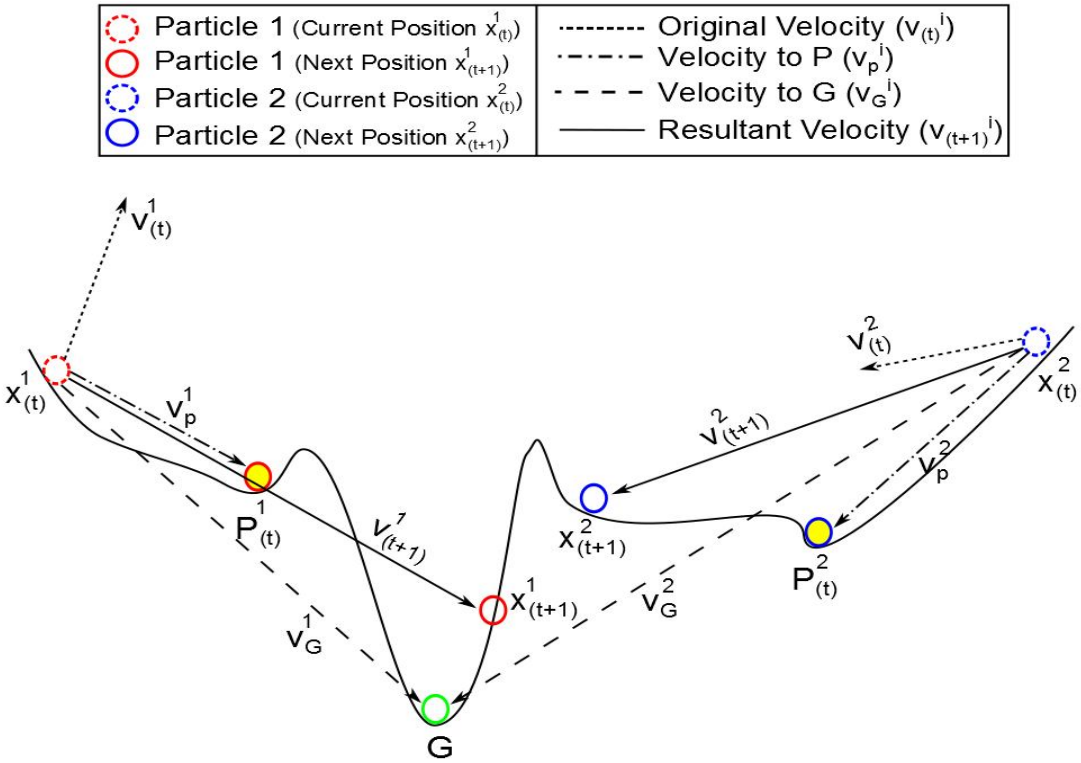


Figure. 1: Illustration of the movement of two particles using PSO algorithm in one-dimensional space.

Figure 1 shows an example of the movement of two particles in the search space. As shown, the search space is one dimensional, and the first, x^1 (t), and second, x^2 (t), particles are represented by the dotted red and blue circles, respectively. Moreover, the two particles have two different previous best positions,

p^1 (t) and p^2(t), and one global solution (G). As shown in the figure, there are three different directions or directed velocity, namely; (1) the original direction (v^1 and v^2), (2) the direction to the previous best positions ($v^1{}_p$ and $v^2{}_p$), and (3) the direction to the best position ($v^1{}_G$ and $v^2{}_G$). The velocity of the particles are calculated as in Equation (2) and it will be denoted by (v^1 (t+1) and v^2 (t+1)). As shown in the figure, the positions of the two particles in the next iteration (t + 1) become closer to the global solution.

Algorithm 1 : Particle Swarm Optimization (PSO)

Initialize the particles' positions (x^i), velocity (v^i), previous best positions (p^i), and the number of particles N.
while (t < maximum number of iterations (T)) **do**
  for all Particles (i) **do**
    Calculate the fitness function for the current position x_i of the i^{th} particle (F(x^i)).
    if (F(x^i) < F(p^i)) **then**
    $p^i = x^i$ **end if**
    if (F(x^i) < F(G)) **then**
      G= x^i
    end if
    Adjust the velocity and positions of all particles according to Equations (1 and 2).
  end for
  Stop the algorithm if a sufficiently good fitness function is met.
14: **end while**

NUMERICAL EXAMPLES

In this section, two numerical examples were illustrated to show how the particles are moved in the search space to find the optimal solution. In the second example, the problem of local minimum is simulated to show how the particles are trapped in one or more local solutions instead of the global solution.

First Example: PSO Example

In this section, the PSO algorithm is explained in details to show how the particles are moved and also to show the influence of each parameter on the PSO algorithm. Assume the PSO algorithm has five particles (p^i, i = 1,…, 5), and the initial positions of the particles are presented in Table 1. The velocities of all particles are initialized to be zeros. Moreover, the initial best solutions of all particles are set to 1000 as shown in Table 1. In this example, De Jong function (see Equation 3) was used as a fitness function.

$$\min F(x, y) = x^2 + y^2 \quad (3)$$

where x and y are the dimensions of the problem. Figure 2 shows the surface and contour plot of the De Jong function. As shown, the function is a strictly convex function1. Moreover, the optimal solution is zero and it is found at the origin. Moreover, the lower and upper boundaries of both x and y dimensions were -1 and 1, respectively. Moreover, in PSO algorithm, the inertia (w) was 0.3, and the values of the cognitive and social constants were as follows, $C_1 = 2$ and $C_2 = 2$.

TABLE 1: Initial positions, velocity, and best positions of all particles.

Particle No.	Initial Positions		Velocity		Best Solution	Best Position		Fitness Value
	x	y	x	y		x	y	
P_1	1	1	0	0	1000	-	-	2
P_2	-1	1	0	0	1000	-	-	2
P_3	0.5	-0.5	0	0	1000	-	-	0.5
P_4	1	-1	0	0	1000	-	-	2
P_5	0.25	0.25	0	0	1000	-	-	0.125

In this example, PSO iteratively searches for the optimal solution, and in each iteration the movement for each particle was calculated including its position, velocity, and fitness function as follows:

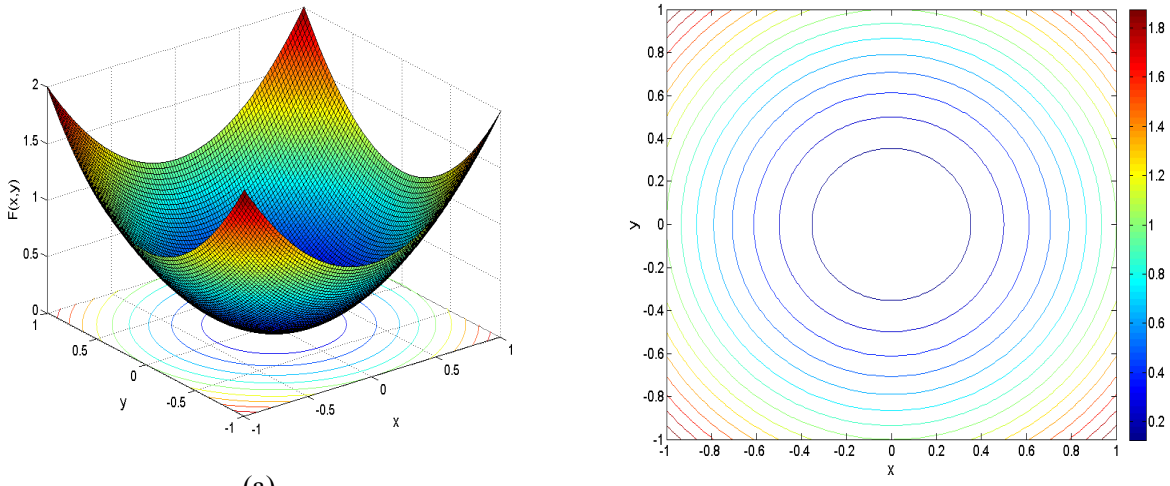


Figure 2: The surface and contour plot of De Jong function in Equation 3, (a) Surface (b) Contour plot.

First Iteration: The first step in this iteration was to calculate the fitness value for all particles, and if the fitness value of any particle ($F(p^i)$) was lower than the corresponding previous best position (p^i), then save the position of this particle. As shown from Tables 1, the first particle was located at (1, 1); hence, the fitness value is 12 + 12 = 2. Similarly, the fitness values of all particles were calculated as in Table 1. As shown, the fitness values of all particles were lower than the current best solutions; hence, the values of p^i were then updated with the best positions as shown in Table 2, and the best solutions were also updated. Moreover, the fifth particle achieved the best solution, i.e. minimum fitness value; G = (0.25, 0.25). As shown in Table 1, the initial velocities of all particles were zero; thus, the particles will not move in this iteration.

TABLE 2: The positions, velocity and best positions of all particles after the first iteration.

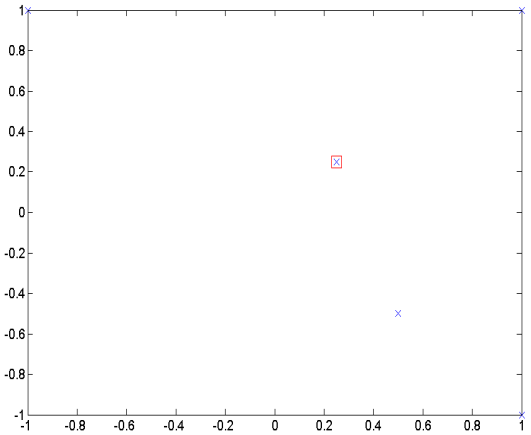
Particle No.	Initial Positions		Velocity		Best Solution	Best Position		Fitness Value
	x	y	x	y		x	y	
P_1	1	1	-0.75	-0.75	2	1	1	2
P_2	-1	1	1.25	-0.75	2	-1	1	2
P_3	0.5	-0.5	-0.25	0.75	0.5	0.5	-0.5	0.5
P_4	1	-1	-0.75	1.25	2	1	-1	2
P_5	0.25	0.25	0	0	0.125	0.25	0.25	0.125

TABLE 3: The positions, velocity and best positions of all particles after the second iteration.

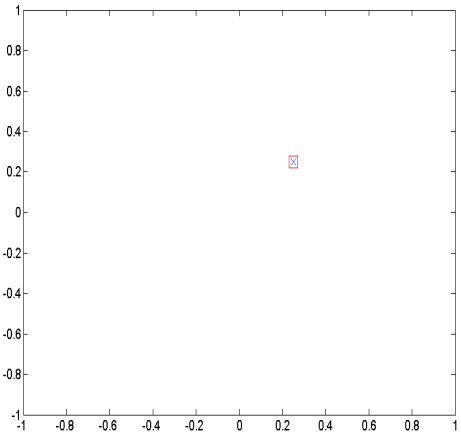
Particle No.	Initial Positions		Velocity		Best Solution	Best Position		Fitness Value
	x	y	x	y		x	y	
P_1	0.25	0.25	-0.3750	-0.3750	2	1	1	0.125
P_2	0.25	0.25	0.6250	-0.3750	2	-1	1	0.125
P_3	0.25	0.25	-0.1250	0.3750	0.5	0.5	-0.5	0.125
P_4	0.25	0.25	-0.3750	0.6250	2	1	-1	0.125
P_5	0.25	0.25	0	0	0.125	0.25	0.25	0.125

The velocity of each particle was calculated as in Equation (2). In this experiment, assume the two random numbers r_1 and r_2 were equal to 0.5. Since the initial velocity of all particles was zero as shown in Table 1 and the previous best positions and the current positions were equal; thus, the first two terms of

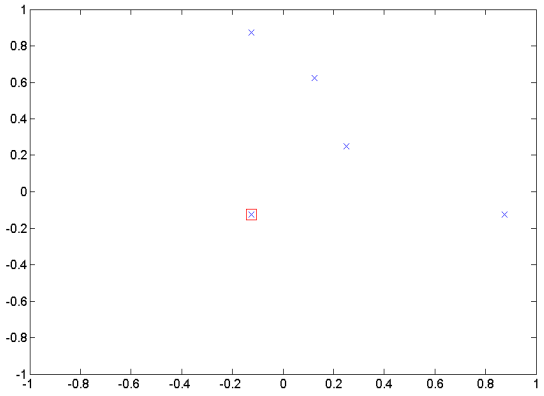
the velocity as shown in Equation (2) were equal to zero. Hence, the velocity in this iteration depends only on the global best position. The velocity of the first particle was calculated as follows, v^i (t+1) = $C_2 r_2$ (G - x^i (t)) = 2 * 0.5 * ((0.25 - 1), (0.25-1)) = (0.75, 0.75), and similarly the velocity of all particles were calculated and the values of all velocities are shown in Table 2. As shown in Table 2, the velocity of the fifth particle was (0, 0) because the fifth particle is the global best solution; hence, it remained at the same position at this iteration. Figure 3 shows the positions of all particles in this iteration. Moreover, Figure 4 shows how the best solution converged to the optimal solution during iterations.



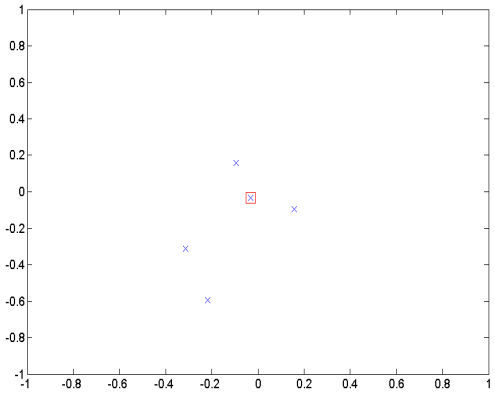
(a) First Iteration



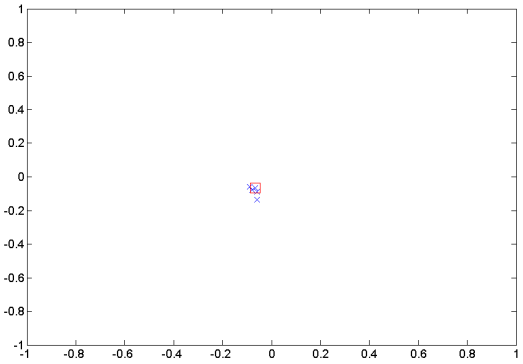
(b) Second Iteration



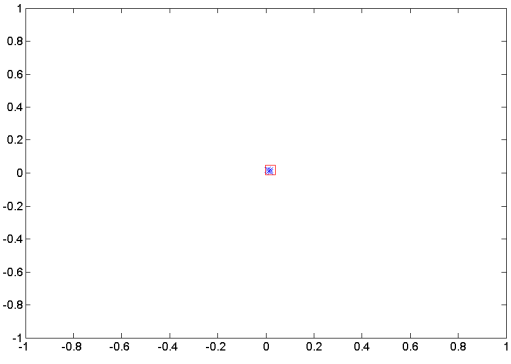
(c) Third Iteration



(d) Fourth Iteration



(e) Seventh Iteration



(f) Thirteenth Iteration

Figure 3: Visualization of the positions of all particles of the PSO algorithm in different iterations.

Second Iteration: In this iteration, the particles were moved to the new locations inside the search space using the positions and velocity that were calculated in the first iteration (see Section 4.1.1). The new positions of all particles are listed in Table 3. The velocity of all particles are then calculated (see Table 3) to move the particles in the next iteration. Moreover, the fitness values of all particles were calculated.

TABLE 4: The positions, velocity and best positions of all particles after the third iteration.

Particle No.	Initial Positions		Velocity		Best Solution	Best Position		Fitness Value
	x	y	x	y		x	y	
P_1	-0.1250	-0.1250	-0.1875	-0.1875	0.125	0.25	0.25	0.0313
P_2	0.8750	-0.1250	-1.3125	0.1875	0.125	0.25	0.25	0.7813
P_3	0.1250	0.6250	-0.1875	-0.9375	0.125	0.25	0.25	0.4063
P_4	-0.1250	0.8750	0.1875	-1.3125	0.125	0.25	0.25	0.7813
P_5	0.2500	0.2500	-0.3750	-0.3750	0.125	0.25	0.25	0.1250

Third Iteration: In this iteration, the particles continue moving towards the optimal solution. At the beginning, the particles were moved to the new positions and their fitness values were calculated as in Table 4. As shown, the first particle became much closer to the optimal solution than the other particles and its fitness value was 0.0313. As shown in this iteration, the velocities of all particles were not zero; in other words, the particles will be moved in the next iterations.

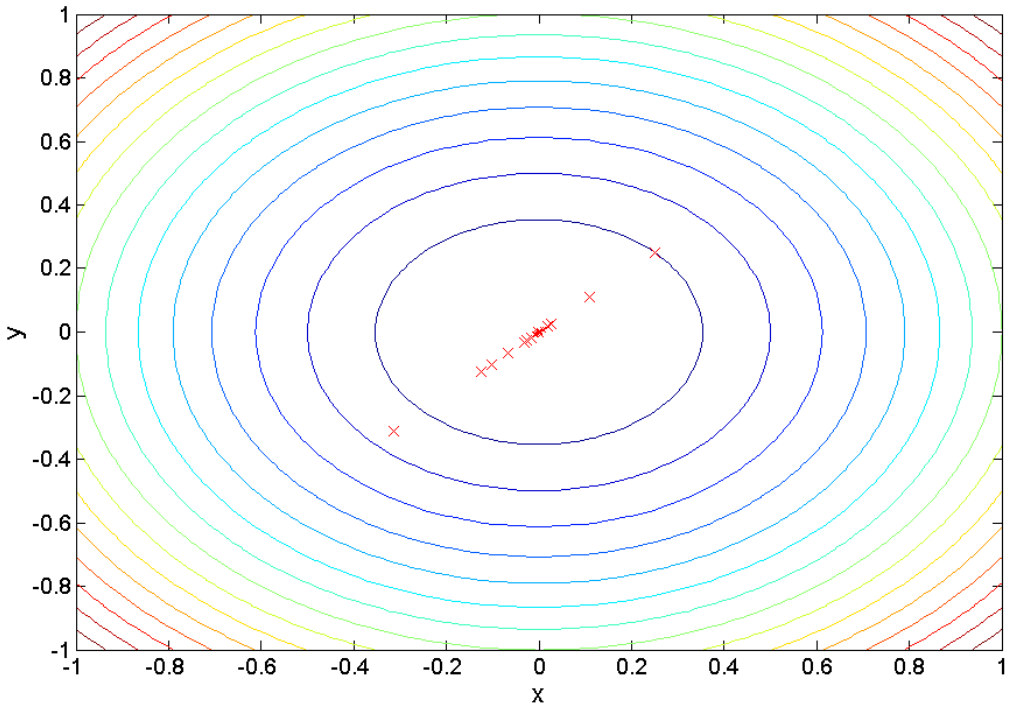


Figure 4: Visualization of the best positions during iterations on the contour plot.

Discussion: The results in Tables 1, 2, 3 and 4 show how the PSO algorithm converges to the global solution. Figure 3 shows the positions of particles in different iterations. As shown, the five particles were initialized at random positions and iteratively converge to the global solution. After the first iteration, the fifth particle was the best particle by achieving 0.125 fitness value and it was located at (0.25, 0.25), and this particle guides the other three particles to move to the better solutions. After the second iteration, all particles were at the same position (0.25, 0.25) and their fitness value was 0.125. After the third iteration,

the first particle was located at (-0.125,-0.125) and it achieved the best fitness value 0.0313. Hence, as the iterations proceed, the PSO algorithm converged to the optimal solution. Another important note is that the convergence rate depends mainly on the values of PSO parameters and the initial positions or solutions. Figure 5 shows the convergence curve of the PSO algorithm in our example. In addition, from Tables 1, 2, 3 and 4 it can be seen that the velocity of the particles was much high in the first iterations than the last iterations. Figure 6 shows the total velocity of the particles in our example. The velocity of the particles in the first iteration was 6.5, and the velocity of the 25^{th} iteration was approximately zero, which reflects that the first iterations in PSO algorithm were faster than the last iterations. This is because two reasons, (1) the PSO algorithm uses linearly decreasing inertia weight; (2) the new velocity of any particle depends on the distance to the previous best position and the best position that may be close to that particle in the last iterations than the first iterations. This simple numerical example shows how the PSO algorithm is simple algorithm and it rapidly converges to the global solution.

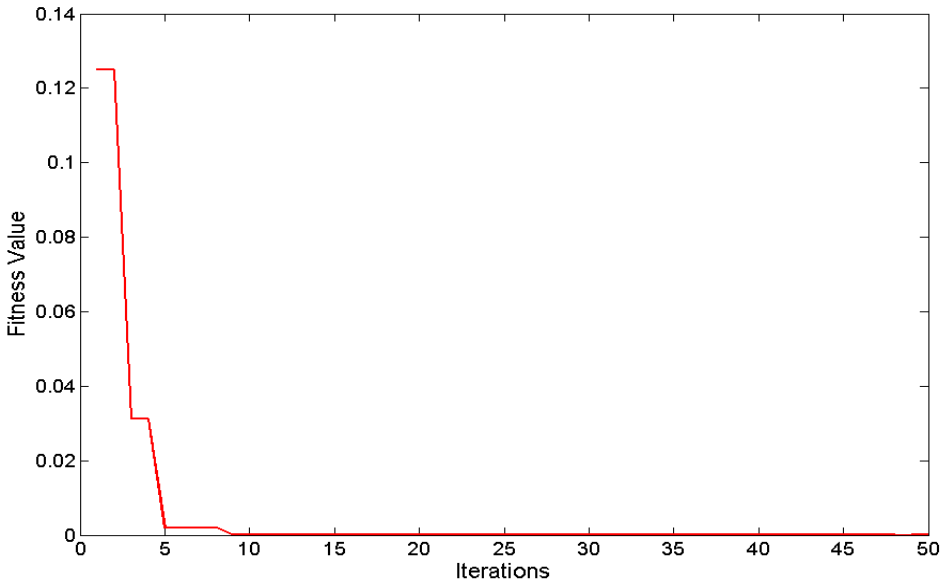


Figure 5: Convergence curve of our numerical example.

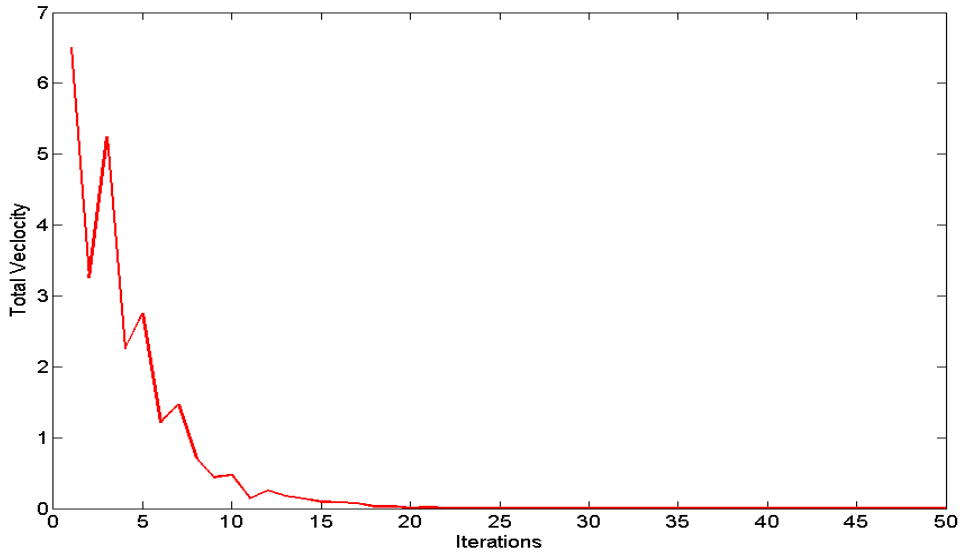


Figure 6: Total velocity of all particles during iterations.

Second Example: Local Optima Problem

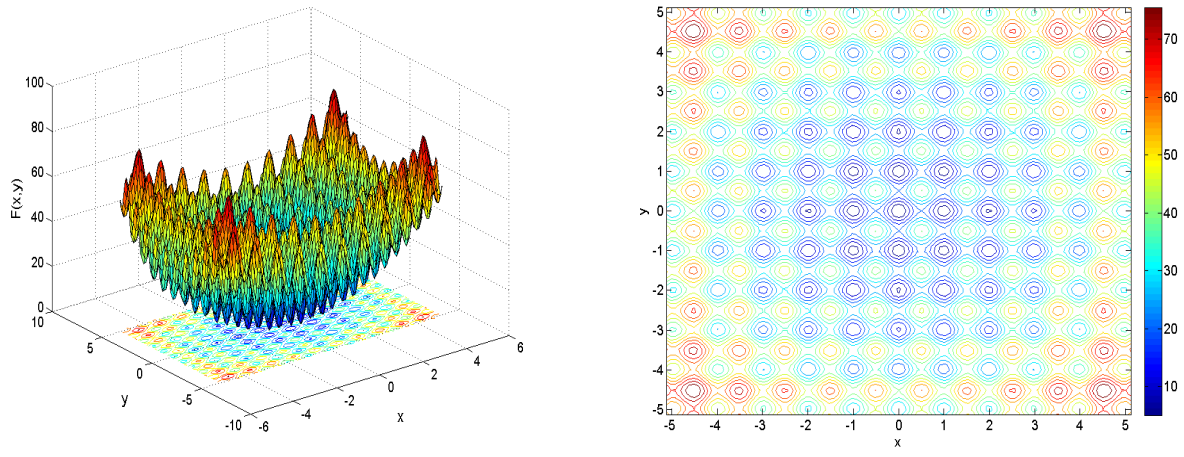
In this section, a numerical example is explained to explain the local optima problem of the PSO algorithm. Assume the PSO algorithm in this example has five particles (P^i, i = 1, ….,5), and the initial positions of the particles are initialized randomly. The velocity of all particles is initialized to be zeros. Moreover, the initial best solutions of all particles are set to 1000 as in the first example. In this example, Rastrigin function (see Equation 4) was used as a fitness function.

$$\min F(x,y) = 10n + \sum_{t=1}^{D}(x_i{}^2 - 10.\cos(2.\pi.x_i) \quad (4)$$

Figure 7 shows the surface and contour plot of the Rastrigin function. As shown, the function is not convex function and it has many local optimal solutions and the optimal solution is located at the origin and the optimal value is zero. Moreover, the limits of the search space for both x and y dimensions are bounded by -5.12 and 5.12. Moreover, in this example, the inertia (w) was 0.3, and the values of the cognitive and social constants were as follows, $C_1 = 2$ and $C_2 = 2$.

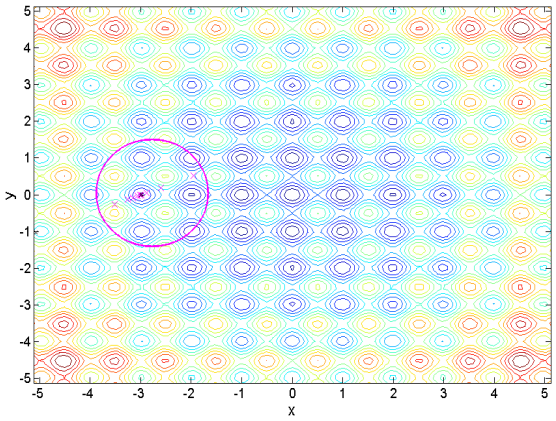
TABLE 5: Initial positions and optima; values of the PSO algorithm in five different runs

Particles No.	First Run		Second Run		Third Run		Fourth Run		Fifth Run	
	x	y	x	y	x	y	x	y	x	y
P_1	-4.028	3.775	-0.984	0.770	1.913	-4.289	3.850	-2.035	-4.415	3.277
P_2	4.730	-4.255	-4.132	-4.508	-3.241	4.397	0.514	-0.298	-1.847	2.236
P_3	-5.073	-1.026	-3.769	-2.716	-1.347	2.823	1.254	-2.760	0.316	4.799
P_4	2.815	-2.459	4.527	-1.504	1.286	-0.135	0.891	3.526	1.582	0.321
P_5	3.249	3.073	4.671	3.289	2.870	-0.657	-2.993	-3.126	-0.946	-1.791
Best Solution	8.96		1.99		5.14		10.62		3.99	

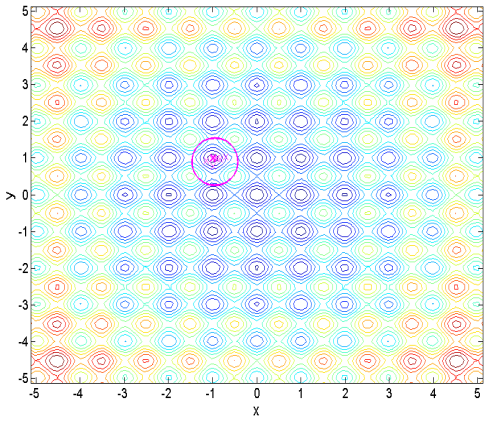


(a) (b)

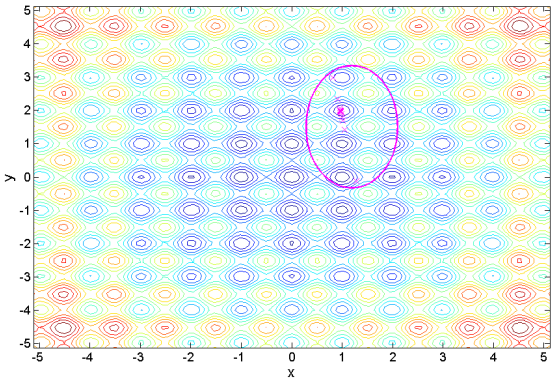
Figure 7: The surface and contour plot of Rastrigin function in Equation 4, (a) Surface (b) Contour plot.



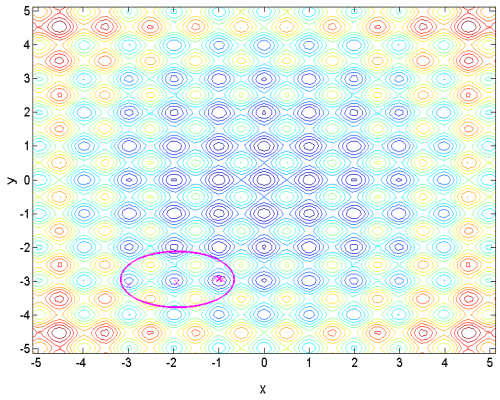
(a) First Run



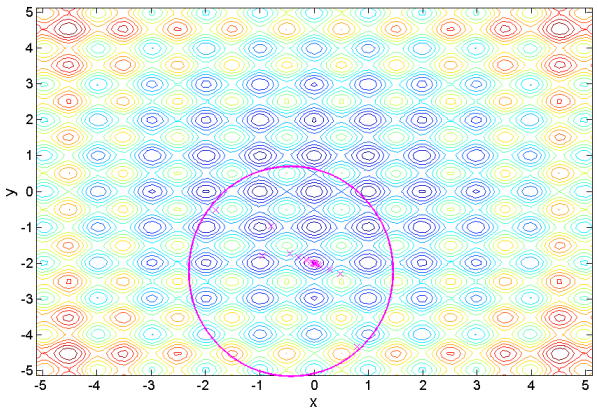
(b) Second Run



(c) Third Run

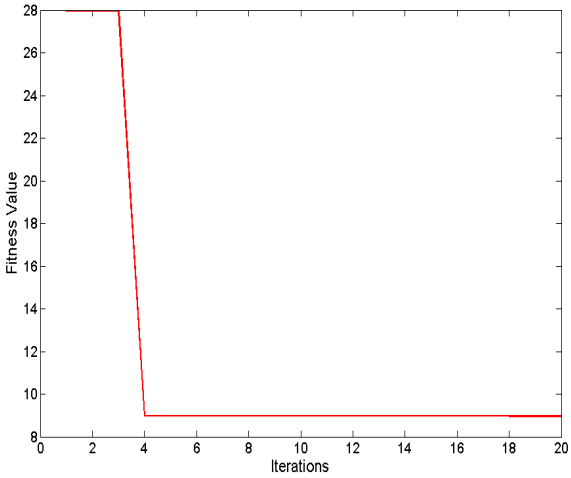


(d) Fourth Run

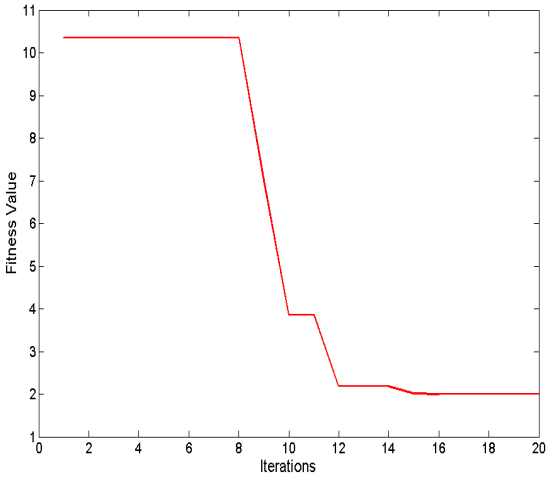


(e) Fifth Run

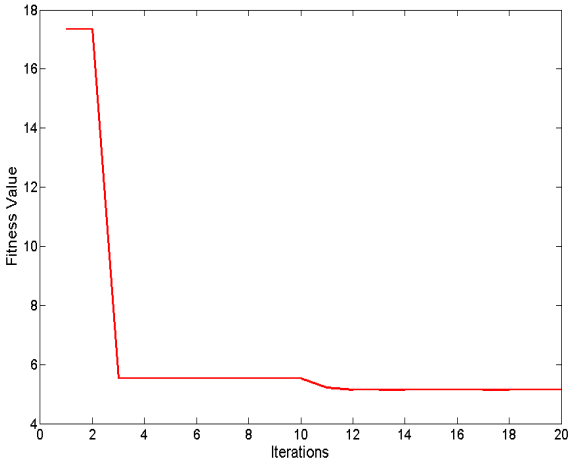
Figure 8: Visualization of the convergence of the PSO particles in different runs.



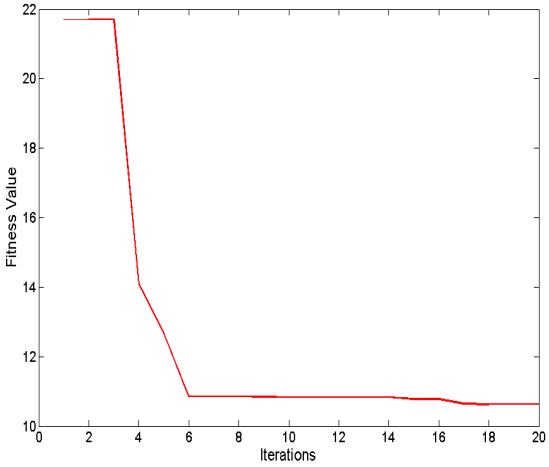
(a) First Run



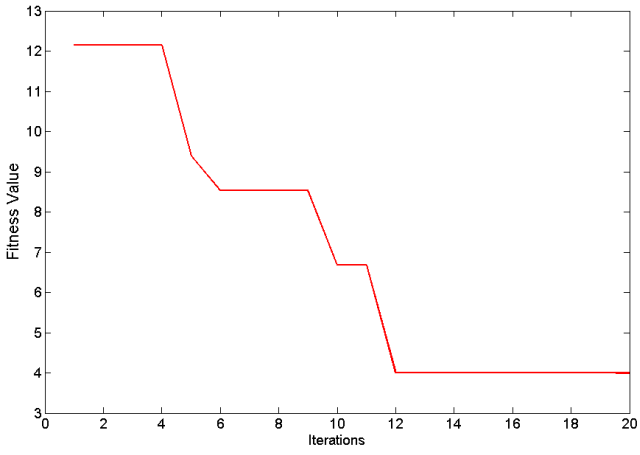
(b) Second Run



(c) Third Run



(d) Fourth Run



(e) Fifth Run

Figure 9: Convergence curves of the PSO particles in different runs.

PSO with Different Runs: In this example, on the contrary to the first example, we will not explain each step in the PSO algorithm because it is already explained in the first example. Instead, in this example, the PSO algorithm has been run five times. In each run, the particles are randomly initialized as shown in Table 5. The particles' positions are then iteratively moved using the PSO algorithm. In this example, the maximum number of iterations was 20. The results of this example are shown in Figures 8 and 9.

Discussion: Figure 8 shows the convergence of the particles in each run. As shown, the five runs converged to different optimal solutions, and all of them did not reach to the optimal solution. Therefore, in each run, the PSO algorithm achieved different optimal value. As shown in Table 5, the best value of the first run was 8.96, and the best values of the second; third, fourth and fifth runs were 1.99, 5.14, 10.62 and 3.99, respectively. This is because the PSO in each run was trapped in different local minimum solution. Figure 9 shows the convergence curves of the PSO algorithms in all runs. As shown, the convergence rate depends mainly on the initial solutions. For example, in Figure 9a, the PSO was trapped starting from the fourth iteration; hence, the PSO algorithm cannot explore different regions in the search space and then trapped in the local solution. On the other hand, the PSO algorithm in the fifth run (see Figure 9e) searched for the optimal solution til it trapped in the local solution after the twelfth iterations.

To conclude, the PSO algorithm can be trapped into local optimal solution; hence, it cannot find better solutions because its exploration capability is very limited, this problem is common in many optimization algorithms and it is called Stagnation. However, many other solutions for this problem were used such as combining PSO with other optimization algorithms which make a balance between the exploration and exploitation phases. Approximately all the recent optimization algorithms solved this problem but they not guarantee to reach to the same solution in each run due to the stochastic nature of the optimization algorithms.

5 Experimental Results and Discussion

In this section, two experiments were conducted to show how the PSO algorithm is used to solve machine learning problems. In the first experiment, PSO was used to find the optimal value for the k parameter in the k-Nearest Neighbor (k-NN) classifier. In this experiment, the PSO searches for the optimal value in one-dimensional space. In the second experiment, the PSO algorithm was used to search for the penalty and kernel parameters of the SVM classifiers. In this experiment, the PSO searched in two-dimensional search space.

Experimental setup

The platform adopted to develop the PSO-SVM algorithm is a PC with the following features: Intel(R) Core (TM) i5-2400 CPU@3.10GHz, 4G RAM, a Window 7 operating system, and Matlab 7.10.0 (R2010a). To evaluate the proposed algorithm four standard classification datasets were used. The datasets were obtained from University of California at Irvin (UCI) Machine Learning Repository (Blake & Merz, 1998). The descriptions of all datasets are listed in Table 6. These datasets are widely used to compare the performance of different classification problems in the literature. As shown in Table 6, all the datasets have only two classes.

TABLE 6: Datasets description.

Dataset	Dimension	# Samples	# Classes
Ionosphere	34	351	2
Liver-disorders	6	345	2
Sonar	60	208	2
Tic-Tac-Toc	9	958	2

In all experiments, k-fold cross-validation tests have used. In k-fold cross-validation, the original samples of the dataset were randomly partitioned into k subsets of (approximately) equal size and the experiment is run k times. For each time, one subset was used as the testing set and the other k-1 subsets were used as the training set. The average of the k results from the folds can then be calculated to produce a single estimation. In this study, the value of k was set to 10. Since the number of samples in each class is not a multiple of 10 (see Table 6), the dataset cannot be partitioned fairly. However, the ratio between the number of samples in the training and testing sets was maintained as closely as possible to 9: 1.

TABLE 7: Accuracy and CPU time of the PSO-kNN and GA-k-NN algorithms.

Dataset	PSO-kNN		GA-kNN	
	Accuracy (%)	CPU time (sec)	Accuracy (%)	CPU time (secs)
Iono	3.12±0.12	38.58	4.15±0.4	52.36
Liver	11.12±2.1	62.13	12.31±1.5	74.26
Sonar	9.45±2.1	50.92	9.87±1.9	59.23
Tic-Tac-Toc	10.26±1.9	47.2	12.35±2.4	62.3

TABLE 8: Accuracy and CPU time of the PSO-SVM and GA-SVM algorithms.

Dataset	PSO-SVM		GA-SVM	
	Accuracy (%)	CPU time (sec)	Accuracy (%)	CPU time (secs)
Iono	2.65±0.45	421.33	3.15±0.32	612.32
Liver	10.93±0.56	1235.32	11.23±0.45	1456.65
Sonar	8.45±0.76	496.32	9.15±0.35	634.56
Tic-Tac-Toc	9.25±1.1	2845.62	11.2±1.5	3056.23

Experimental Scenarios

The first experiment was conducted to compare the pro- posed PSO-SVM algorithm with Genetic Algorithm (GA) algorithm. Table 7 shows the results of this experiment. As shown in the table, the PSO algorithm achieved results better than GA. Moreover, in terms of the CPU time, the PSO algorithm required CPU time lower than the GA algorithm.

In the second experiment, PSO algorithm was used to optimize SVM classifier. In this experiment, the PSO- SVM algorithm was compared with GA-SVM algorithm. Table 8 shows the results of this experiment. As shown in the table, the PSO algorithm achieved results better than GA. In terms of the CPU time, the PSO algorithm required CPU time lower than the GA algorithm.

To conclude, the PSO algorithm converged to the optimal or near optimal solutions faster than GA. Hence, the PSO algorithm achieved results better than GA algorithm.

CONCLUSION

Optimization algorithms are used recently in many applications. Particle Swarm Optimization (PSO) is one of the well-known algorithms. It is simple and easy to implement algorithm. This paper explains the steps of calculating the PSO algorithm, and how the particles are converged to the optimal solution. In addition, two numerical examples were given and graphically illustrated to explain the steps of PSO algorithm and to focus on the local optima problem and how PSO algorithm trapped in local minima problem. Moreover, using standard datasets, two experiments were conducted to show how the PSO algorithm is used to optimize k-NN classifier, where the search space in one-dimensional, and how the PSO optimize SVM where the search space is two-dimensional space.

REFERENCES

Abido, M. (2002). Optimal design of power-system stabilizers using particle swarm optimization. IEEE transactions on energy conversion, 17, 406-413.

Akay, B. (2013). A study on particle swarm optimization and artificial bee colony algorithms for multilevel thresholding. Applied Soft Computing, 13, 3066-3091.

Ali, A. F., Mostafa, A., Sayed, G. I., Elfattah, M. A., & Hassanien, A. E. (2016). Nature inspired optimization algorithms for ct liver segmentation. In Medical Imaging in Clinical Applications (pp. 431-460). Springer.

Bashir, Z., & El-Hawary, M. (2009). Applying wavelets to short-term load forecasting using pso-based neural networks. IEEE transactions on power systems, 24 , 20-27.

Blake, C., & Merz, C. J. (1998). fUCIg repository of machine learning databases,

Chander, A., Chatterjee, A., & Siarry, P. (2011). A new social and momentum component adaptive pso algorithm for image segmentation. Expert Systems with Applications, 38 , 4998-5004.

Eberhart, R. C., & Kennedy, J. (1995). A new optimizer using particle swarm theory. In Proceedings of the 6th international symposium on micro machine and human science (pp. 39-43). New York, NY volume 1.

Elbedwehy, M. N., Zawbaa, H. M., Ghali, N., & Hassanien, A. E. (2012). Detection of heart disease using binary particle swarm optimization. In Proceedings of the Federated Conference on Computer Science and Information Systems 360 (FedCSIS), Wrosw, Poland, September 9-12 (pp. 177-182). IEEE.

Forouzanfar, M., Forghani, N., & Teshnehlab, M. (2010). Parameter optimization of improved fuzzy c-means clustering algorithm for brain mr image segmentation. Engineering Applications of Artificial Intelligence, 23 , 160-168.

Gaber, T., Tharwat, A., Hassanien, A. E., & Snasel, V. (2016). Biometric cattle identification approach based on weber's local descriptor and adaboost classifier. Computers and Electronics in Agriculture , 122 , 55-66.

Hassan, R., Cohanim, B., De Weck, O., & Venter, G. (2005). A comparison of particle swarm optimization and the genetic algorithm. In Proceedings of the 1st AIAA multidisciplinary design optimization specialist conference, 370 Honolulu, Hawaii, April 23 - 26 (pp. 1-13).

Heppner, F., & Grenander, U. (1990). A stochastic nonlinear model for coordinated bird flocks. American Association For the Adavancement of Science, Washington, DC (USA).

Ho, S., Yang, S., Ni, G., & Wong, K. (2007). An improved pso method with application to multimodal functions of inverse problems. IEEE transactions on magnetics, 34 (4), 1597-1600.

Ibrahim, A., & Tharwat, A. (2014). Biometric authentication methods based on ear and finger knuckle images. International Journal of Computer Science Issues (IJCSI), 11, 134.

Ishaque, K., & Salam, Z. (2013). A deterministic particle swarm optimization maximum power point tracker for photovoltaic system under partial shading condition. IEEE Transactions on Industrial Electronics , 60 , 3195-3206.

Kennedy, J. (2010). Particle swarm optimization. In Encyclopedia of Machine Learning (pp. 760_766). Springer.

Kulkarni, R. V., & Venayagamoorthy, G. K. (2011). Particle swarm optimization in wireless-sensor networks: A brief survey. Proceedings of IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews , 41, 262-267.

Lin, S.-W., Ying, K.-C., Chen, S.-C., & Lee, Z.-J. (2008). Particle swarm optimization for parameter determination and feature selection of support vector machines. Expert systems with applications , 35 , 1817-1824.

Maitra, M., & Chatterjee, A. (2008). A hybrid cooperative comprehensive learning based pso algorithm for image segmentation using multilevel thresholding. Expert Systems with Applications , 34 , 1341-1350.

Van der Merwe, D., & Engelbrecht, A. P. (2003). Data clustering using particle swarm optimization. In Evolutionary Computation, 2003. CEC'03. The 2003 Congress on (pp. 215_220). IEEE volume 1.

Miyatake, M., Veerachary, M., Toriumi, F., Fujii, N., & Ko, H. (2011). Maximum power point tracking of multiple photovoltaic arrays: a pso approach. IEEE Transactions on Aerospace and Electronic Systems , 47, 367-380.

Mostafa, A., Fouad, A., Elfattah, M. A., Hassanien, A. E., Hefny, H., Zhu, S. Y., & Schaefer, G. (2015). Ct liver segmentation using artificial bee colony optimisation. Procedia Computer Science, 60, 1622-1630.

Panda, S., & Padhy, N. P. (2008). Optimal location and controller design of statcom for power system stability improvement using pso. Journal of the Franklin Institute, 345, 166-181.

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. ACM Siggraph Computer Graphics , 21 , 25-34.

dos Santos Coelho, L. (2010). Gaussian quantum-behaved particle swarm optimization approaches for constrained engineering design problems. Expert 410 Systems with Applications , 37 , 1676-1683.

Subasi, A. (2013). Classification of emg signals using pso optimized svm for diagnosis of neuromuscular disorders. Computers in biology and medicine, 43 , 576-586.

Tharwat, A., Gaber, T., Awad, Y. M., Dey, N., & Hassanien, A. E. (2016). Plants identification using feature fusion technique and bagging classifier. In The 1st International Conference on Advanced Intelligent System and Informatics (AISI2015), November 28-30, 2015, Beni Suef, Egypt (pp. 461-471). Springer.

Tharwat, A., Gaber, T., & Hassanien, A. E. (2015a). Two biometric approaches for cattle identification based on features and classifiers fusion. International Journal of Image Mining, 1, 342-365.

Tharwat, A., Ibrahim, A., Hassanien, A. E., & Schaefer, G. (2015b). Ear recognition using block-based principal component analysis and decision fusion. In International Conference on Pattern Recognition and Machine Intelligence 425 (pp. 246-254). Springer.

Tharwat, A., Zawbaa, H. M., Gaber, T., Hassanien, A. E., & Snasel, V. (2015c). Automated zebrafish-based toxicity test using bat optimization and adaboost classifier. In 2015 11th International Computer Engineering Conference (ICENCO) (pp. 169-174). IEEE.

Van Den Bergh, F. (2006). An analysis of particle swarm optimizers . Ph.D. thesis University of Pretoria.

Vesterstrom, J., & Thomsen, R. (2004). A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In Evolutionary Computation, 2004. CEC2004. Congress on (pp. 1980-1987). IEEE volume 2.

Xinchao, Z. (2010). A perturbed particle swarm algorithm for numerical optimization. Applied Soft Computing, 10 , 119-124.

Yamany, W., Fawzy, M., Tharwat, A., & Hassanien, A. E. (2015a). Moth-Flame optimization for training multi-layer perceptrons. In 2015 11th International Computer Engineering Conference (ICENCO) (pp. 267-272). IEEE.

Yamany, W., Tharwat, A., Hassanin, M. F., Gaber, T., Hassanien, A. E., & Kim, T.-H. (2015b). A new multi-layer perceptrons trainer based on ant lion optimization algorithm. In 2015 Fourth International Conference on Information Science and Industrial Applications (ISI) (pp. 40-45). IEEE.

Yang, X.-S. (2014). Nature-inspired optimization algorithms. Elsevier, First Edition.

Yoshida, H., Kawata, K., Fukuyama, Y., Takayama, S., & Nakanishi, Y. (2000). A particle swarm optimization for reactive power and voltage control considering voltage security assessment. IEEE Transactions on power systems, 15, 1232-1239.

APPENDIX A

In this section, a Matlab code for the PSO algorithm is introduced.

```
clc
clear all

Max_iterations=50;  % Maximum Number of Iterations
correction_factor = 2.0; % Correction factor
inertia = 1.0; % Ineritia Coeffecient
swarm_size = 5; % Number of particles
LB=[-5.12 -5.12]; % Lower Boundaries
UB=[5.12 5.12];   % Upper Boundaries
xrange=UB(1)-LB(1);
yrange=UB(2)-LB(2);

% Initial Positions
swarm(:, 1, 1)=rand(1,swarm_size)*xrange+LB(1);
swarm(:, 1, 2)=rand(1,swarm_size)*yrange+LB(2);

% Initial best value so far
swarm(:, 4, 1) = 1000;

% Initial velocity
swarm(:, 2, :) = 0;
for iter = 1 : Max_iterations

    % Calculating fitness value for all particles
    for i = 1 : swarm_size
```

```
        swarm(i, 1, 1) = swarm(i, 1, 1) + swarm(i, 2, 1)/1.3;      %update x
        position
        swarm(i, 1, 2) = swarm(i, 1, 2) + swarm(i, 2, 2)/1.3;      %update y
        position


        % The fitness function (DeJong) F(x,y)=x^2+y^2
        Fval = (swarm(i, 1, 1))^2 + (swarm(i, 1, 2))^2;   % fitness
        evaluation (you may replace this objective function with any function
        having a global minima)

        % If the fitness value for this particle is better than the
        % best fitness value of that particle exchange both values
        if Fval < swarm(i, 4, 1)                  % if new position is better
             swarm(i, 3, 1) = swarm(i, 1, 1);    % Update the position of the
        first dimension
             swarm(i, 3, 2) = swarm(i, 1, 2);    % Update the position of the
        second dimension
            swarm(i, 4, 1) = Fval;               % Update best value
        end
    end

    % Search for the global best solution
    [temp, gbest] = min(swarm(:, 4, 1));         % global best position

    % Updating velocity vectors
    for i = 1 : swarm_size
        swarm(i, 2, 1) = rand*inertia*swarm(i, 2, 1) +
        correction_factor*rand*(swarm(i, 3, 1) - swarm(i, 1, 1)) +
        correction_factor*rand*(swarm(gbest, 3, 1) - swarm(i, 1, 1));   %x
        velocity component
        swarm(i, 2, 2) = rand*inertia*swarm(i, 2, 2) +
        correction_factor*rand*(swarm(i, 3, 2) - swarm(i, 1, 2)) +
        correction_factor*rand*(swarm(gbest, 3, 2) - swarm(i, 1, 2));   %y
        velocity component
    end

    % Store the best fitness valuye in the convergence curve
    ConvergenceCurve(iter,1)=swarm(gbest,4,1);
    disp(['Iterations No. ' int2str(iter) ' , the best fitness value is '
     num2str(swarm(gbest,4,1))]);
end

% Plot convergence curve
plot(ConvergenceCurve,'r-')
title('Convergence Curve')
xlabel('Iterations')
ylabel('Fitness Value')
```