

COMPARISON OF PARTICLE SWARM OPTIMIZATION VARIANTS

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By
Satyanarayana Daggubati

In Partial Fulfillment
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

October 2012

Fargo, North Dakota

North Dakota State University

Graduate School

Title

Comparison of Particle Swarm Optimization Variants

By

Satyanarayana Daggubati

The Supervisory Committee certifies that this *disquisition* compiles with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Kendall Nygard

Chair

Dr. Simone Ludwig

Advisor

Dr. Simone Ludwig

Dr. Changhui Yan

Dr. Chao You

Approved by Department Chair:

10/29/12

Date

Kenneth Magel

Signature

ABSTRACT

Particle swarm optimization (PSO) is a heuristic global optimization method, which is based on swarm intelligence. It is inspired by the research on the bird and fish flock movement behavior. The algorithm is widely used and can rapidly be implemented with a few parameters to be tuned. In PSO, individuals, referred to as particles, are “flown” through a hyper-dimensional search space. Changes to the position of particles within the search space are based on the social-psychological tendency of individuals to emulate the success of other individuals. The changes to a particle within the swarm are therefore influenced by the experience, or knowledge, of its neighbors. Many different PSO variants have been proposed in the past. This paper describes a few of these variants that have been implemented, and compares them with standard PSO on a number of benchmark functions measuring both the solution quality and execution time.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my advisor Dr. Simone Ludwig for her continued support throughout this paper. I appreciate her time, assistance and continuous guidance. I would also like to thank Dr. Kendall Nygard, Dr. Changhui Yan, and Dr. Chao You for being a part of my graduate supervisory committee. Special thanks to the faculty and staff of the Computer Science Department for their unconditional support throughout my master's program. I am highly obligated to my friends, Dr. Satish and Harsha, without them I wouldn't have come this far.

TABLE OF CONTENTS

ABSTRACT	iii
ACKNOWLEDGEMENTS	iv
LIST OF TABLES	vii
LIST OF FIGURES	vii
1. INTRODUCTION	1
2. RELATED WORK	4
2.1 Basic Particle Swarm Optimization (Basic-PSO)	4
2.2 Global Best PSO	5
2.3 Local Best PSO	6
3. PROBLEM DESCRIPTION	9
3.1 Decreasing Weight Particle Swarm Optimization (DWPSO)	10
3.2 Time-Varying PSO (TVPSO)	11
3.3 Guaranteed convergence PSO (GCPSO)	13
3.4. Time-Varying Acceleration Coefficients PSO (TVAC-PSO)	14
3.5 PSO with Mutation and Time Varying Acceleration Coefficients (MPSO)	15
3.6 Adaptive Particle Swarm Optimization (APSO)	17
3.7 Algorithm Aspects	19
4. IMPLEMENTATION DETAILS	21

4.1. RunPSO Class	21
4.2. ParticleUpdateSimple Class	22
4.3. Swarm Class	22
4.4. FitnessFuntion Class	23
4.5. Benchmark Classes	23
5. BENCHMARK FUNCTIONS AND SETTINGS	25
5.1 Alpine (F1)	25
5.2 Sphere (F2)	25
5.3 Rosenbrock (F3)	26
5.4 Rastrigin (F4)	27
5.6 General Settings	28
6. EXPERIMENTS AND RESULTS	29
6.1 Test Case 1: Varying Number of Iterations	29
6.2 Test Case 2: Varying Number of Dimensions	31
7. CONCLUSION AND FUTURE WORK	35
REFERENCES	36

LIST OF TABLES

<u>Table</u>	<u>Page</u>
3.1: Strategies for the values of c1 and c2	19
4.1: Variables of RunPSO Class	21
4.2: Methods of ParticleUpdateSimple Class	22
4.3: Variables of Swarm Class.....	22
4.4: Methods of Swarm Class	23
4.5: Methods of FitnessFuntion Class.....	23
5.1: Range of Search of Benchmark Functions	28
6.1: Average and variance of each variant: 1 million iterations and 2 dimensions	29
6.2: Execution time in ms of each variant: 1 million iterations and 2 dimensions	30
6.3: Average and variance of each variant: 1 million iterations and 2 dimensions	30
6.4: Shows the execution time for each variant for 3 million iterations and 2 dimensions	31
6.5: Average and variance of each variant: 1 million iterations and 10 dimensions	32
6.6: Execution time in ms of each variant: 1 million iterations and 10 dimensions	32
6.7: Average and variance of each variant: 3 million iterations and 10 dimensions	33
6.8: Execution time in ms of each variant: 3 million iterations and 10 dimensions	33

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
2.1: Flowchart of PSO Algorithm.....	8
5.1: Rosenbrock's Function in 2D [14].....	26
5.2: Rastrigin's Function in 2D [14].....	27

1. INTRODUCTION

The field of optimization is broad and has found applications in many different areas such as logistics, finance, and engineering. The goal of optimization is to find the best solution to a problem. There are many different optimization categories that exist today and nature-inspired algorithms split further into two methodologies referred to as evolutionary computation and swarm intelligence. Evolutionary computation is inspired by processes of biological evolution.

Evolutionary algorithms use a population of individuals, where an individual is referred to as a chromosome. A chromosome defines the characteristics of individuals in the population. Each characteristic is referred to as a gene. The value of a gene is referred to as an allele. For each generation, individuals compete to reproduce offspring. Those individuals with the best survival capabilities have the best chance to reproduce. Offspring are generated by combining parts of the parents, a process referred to as crossover. Each individual in the population can also undergo mutation, which alters some of the allele of the chromosome. The survival strength of an individual is measured using a fitness function which reflects the objectives and constraints of the problem to be solved. After each generation, individuals may undergo culling, or individuals may survive to the next generation (referred to as elitism). Additionally, behavioral characteristics (as encapsulated in phenotypes) can be used to influence the evolutionary process in two ways: phenotypes may influence genetic changes, and/or behavioral characteristics evolve separately [12].

Swarm intelligence originated from the study of colonies, or swarms of social organisms. Studies of the social behavior of organisms (individuals) in swarms prompted the design of very efficient optimization and clustering algorithms. For example, simulation studies of the graceful, but unpredictable, choreography of bird flocks led to the design of the particle swarm optimization algorithm, and studies of the foraging behavior of ants resulted in ant colony optimization algorithms.

Particle swarm optimization (PSO) is a population based stochastic optimization technique developed by Dr. Eberhart and Dr. Kennedy in 1995, inspired by social behavior of bird flocking or fish schooling [1]. PSO incorporates swarming behaviors observed in flocks of birds, schools of fish, or swarms of bees, and even human social behavior, from which the idea has emerged [1]. PSO can be implemented and applied easily to solve various function optimization problems, or problems that can be transformed to function optimization problems. As an algorithm, the main strength of PSO is its fast convergence, which compares favorably with many global optimization algorithms like Genetic Algorithms.

In PSO, individuals are referred to as particles. Changes to the position of particles within the search space are based on the social-psychological tendency of individuals to emulate the success of other individuals. The changes to a particle within the swarm are therefore influenced by the experience, or knowledge, of its neighbors. The search behavior of a particle is thus affected by that of other particles within the swarm.

In this paper, we describe and implement different PSO variants. In particular, decreasing weight PSO, time-varying acceleration PSO, guaranteed convergence PSO, mutation time-varying acceleration PSO, and hybrid PSO are introduced. All PSO variants are implemented

and experiments are conducted using several benchmark functions measuring the solution quality and the execution time.

The arrangement of the chapters is as follows: Chapter 2 briefly introduces basic particle swarm optimization - the global best PSO and local best PSO variants. Chapter 3 describes the different selected PSO variants to be implemented and compared. Chapter 4 explains and discusses the implementation details. Chapter 5 describes the different benchmark functions and PSO settings. Chapter 6 reports the results, and shows the sample outputs. Chapter 7 addresses conclusions drawn and discusses future work.

2. RELATED WORK

When the search space is too large to search exhaustively, population based searches are a good alternative, however, population based search techniques cannot guarantee to find the optimal (best) solution. A population based search technique; PSO is discussed in this paper. The PSO algorithm shares similar characteristics with Genetic Algorithm, however, the manner in which the two algorithms traverse the search space is fundamentally different.

Both Genetic Algorithms and Particle Swarm Optimizers share common elements:

1. Both initialize a population in a similar manner.
2. Both use an evaluation function to determine how fit (good) a potential solution is.
3. Both are generational, that is both repeat the same set of processes for a predetermined number of times.

Nowadays, many scientific experiments such as structural biology and chemistry, neuroscience data analysis, and disaster recovery are conducted through complex and distributed scientific computations that are represented and structured as scientific workflows. Scientific workflows usually need to process huge amount of data and is a computationally-intensive activity. We achieve this by using an iterative method called PSO.

2.1. Basic Particle Swarm Optimization (Basic-PSO)

A PSO algorithm maintains a swarm of particles, where each particle represents a potential solution. A swarm is similar to a population, while a particle is similar to an individual.

In simple terms, the particles are “flown” through a multi-dimensional search space, where the position of each particle is adjusted according to its own experience and that of its neighbors. Initially, particles are distributed throughout the search space and their positions and velocities are modified based on the knowledge of the best solution found thus far by each particle in the ‘swarm’ [2]. Attraction towards the best-found solution occurs stochastically, and uses dynamically-adjusted particle velocities. Let $x_i(n)$ denotes the position of particle i in the search space at time step t ; where t denotes discrete time steps. The position of the particle is changed by adding a velocity, $v_i(n)$, to the current position [3]:

$$x_i(n+1) = x_i(n) + v_i(n+1) \quad (1)$$

The system is initialized with a population of random solutions that searches for the optimum by updating their positions. Each particle keeps track of its coordinates in the problem space, which are associated with the best fitness solution it has achieved so far. This value is called Pbest (Particle best). Another best value that is tracked by the particle swarm optimizer is the best value obtained so far by any particle in the neighborhood. This location is called Lbest. When a particle takes all the population as its neighbors, the best value is a global best and is called Gbest. In PSO, at each time step, the velocity of each particle to achieve its Gbest and Lbest (Global Best PSO and Personal Best Position) is updated.

2.2. Global Best PSO

For the global best PSO, or Gbest PSO, the neighborhood for each particle is the entire swarm. The social network employed by the Gbest PSO reflects the star topology [4]. For the star neighborhood topology, the social component of the particle velocity update reflects

information obtained from all the particles in the swarm. In this case, the social information is the best position found by the swarm, referred to as $\hat{y}(n)$. For Gbest PSO, the velocity of particle i is calculated as:

$$v_{ij}(n+1) = v_{ij}(n) + c_1 r_1(n) [y_{ij}(n) - x_{ij}(n)] + c_2 r_2(n) [\hat{y}(n) - x_{ij}(n)] \quad (2)$$

where $v_{ij}(n)$ is the velocity of particle i in dimension $j = 1, \dots, n_x$ at time step n , $x_{ij}(n)$ is the position of particle i in dimension j at time step n , c_1 and c_2 are positive acceleration constants used to scale the contribution of the cognitive and social components respectively, and $r_1(n)$, $r_2(n)$ are random values in the range $[0, 1]$, sampled from a uniform distribution. These random values introduce a stochastic element to the algorithm. The personal best position y_i associated with particle i is the best position the particle has ever visited.

2.3. Local Best PSO

The local best PSO or Lbest PSO uses a ring social network topology where smaller neighborhoods are defined for each particle. The social component reflects information exchanged within the neighborhood of the particle, reflecting local knowledge of the environment. With reference to the velocity equation, the social contribution to particle velocity is proportional to the distance between a particle and the best position found by the neighborhood of particles [4]. The local best position is also referred to as the Neighborhood best position. The velocity is calculated as:

$$v_{ij}(n+1) = v_{ij}(n) + c_1 r_1(n) [y_{ij}(n) - x_{ij}(n)] + c_2 r_2(n) [\hat{y}(n) - x_{ij}(n)] \quad (3)$$

where $\hat{y}(t)$ is the best position found by the neighborhood of particle i in dimension j . The velocity calculation as given in Equations (2) and (3) consist of three terms:

1. The previous velocity, $v_{ij}(n)$, which serves as a memory of the previous flight direction, i.e. movement in the immediate past.
2. The cognitive component $c_1 r_1(t) [y_{ij}(n) - x_{ij}(n)]$, which quantifies the performance of a particle relative to its past performances.
3. The social component $c_2 r_2(t) [y_{ij}(n) - x_{ij}(n)]$, which quantifies the performance of a particle relative to the group of particles, or neighbors [4].

Figure 2.1 shows the flowchart diagram of the PSO algorithm. First, the particles' position and velocity are randomly initialized. Then, the iteration steps start whereby the fitness of all particles is evaluated, then the global and local best values are updated and until the stopping criterion is not met, the positions of all particles are updated, and the iteration steps are repeated again.

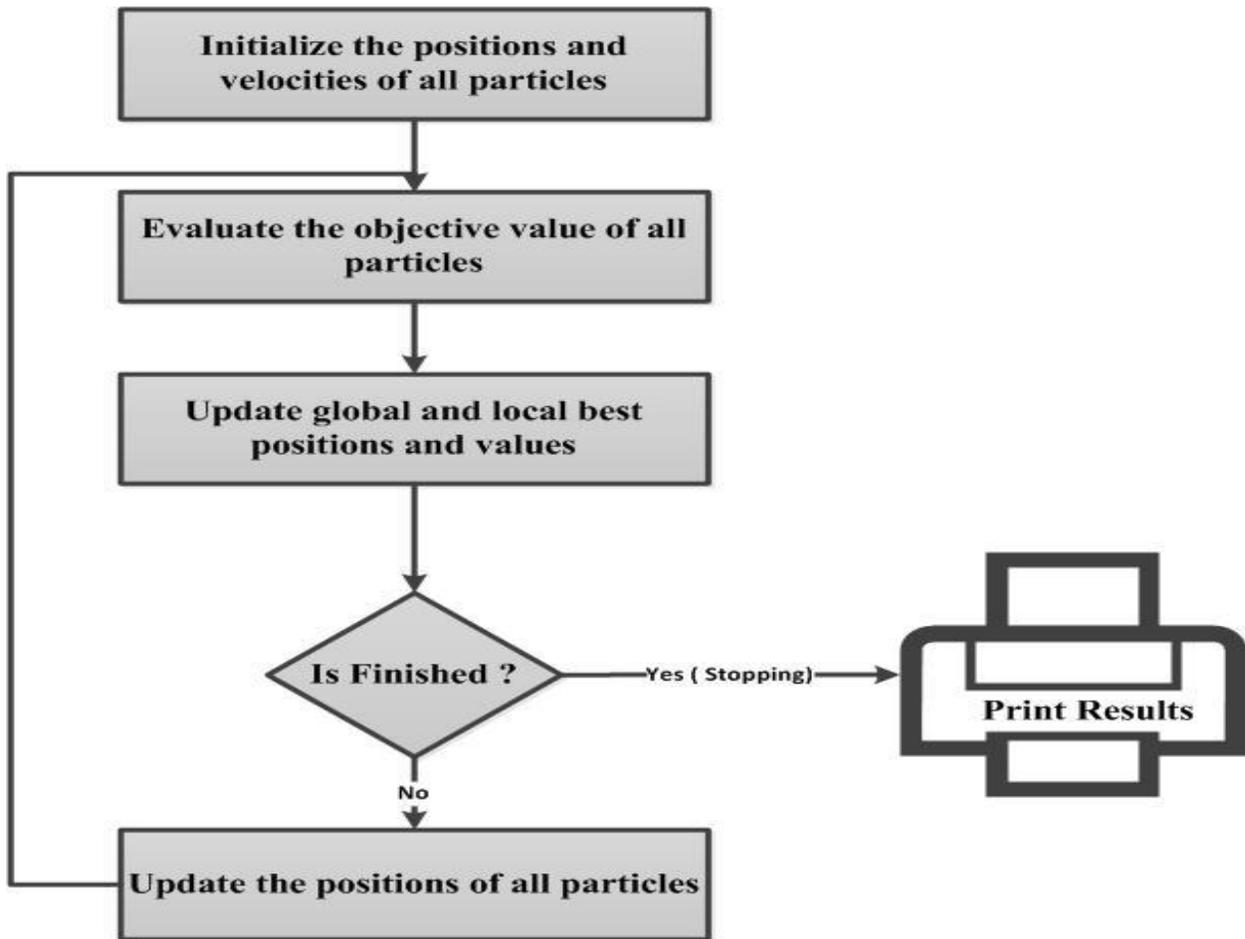


Fig 2.1: Flowchart of PSO Algorithm

3. PROBLEM DESCRIPTION

All PSO variants described in this paper are used to find the optimal solution for PSO to a series of benchmark functions. This chapter deals with the problem definition and detailed description of the all algorithms. In particular, the following PSO variants are described: decreasing weight PSO, time-varying acceleration PSO, guaranteed convergence PSO, mutation time-varying acceleration PSO, and hybrid PSO.

As described in the previous chapter, the position of the i^{th} particle is represented by $x_{ij} = (x_{i1}, x_{i2}, \dots, x_{iD})$. The velocity for the i^{th} particle is written as $v_i = (v_{i1}, v_{i2}, \dots, v_{iD})$. The positions and velocities of these particles are confined within $(x_{\min}, x_{\max})_D$ and $(v_{\min}, v_{\max})_D$ respectively [2].

The best encountered position of the i^{th} particle, denoted as P_{best_i} , is represented by $p_i = (p_{i1}, p_{i2}, \dots, p_{iD})$. The best value of all the individual P_{best_i} values are denoted as the global best position $g = (g_1, g_2, \dots, g_D)$ called g_{best} . The PSO process is initialized with a population of random particles, and the algorithm executes a search for optimal solutions by continuously updating the positions of the particles. At each time of the generation, the position and velocity of the i^{th} particle are updated by P_{best} and G_{best} of the population. The updated equations are formulated as follow:

$$x_{ij}(n+1) = x_{ij}(n) + v_{ij}(n+1) \quad (4)$$

$$v_{ij}(n+1) = w(n) v_{ij}(n) + c_1 r_{1ij}(n) [p_{ij}(n) - x_{ij}(n)] + c_2 r_{2ij}(n) [g_{ij}(n) - x_{ij}(n)] \quad (5)$$

where x_{ij} is the position of particle i and the dimension j , n the current iteration and $n+1$ the next iteration, and $p_{ij}(n) - x_{ij}(n)$ calculates a vector directed towards the personal best position, $g_{ij}(n) - x_{ij}(n)$ calculates a vector directed towards the global best position, and the inertia weight w , the personal best weight c_1 , the global best weight c_2 . Both r_{1ij} , r_{2ij} are random numbers uniformly distributed between 0 and 1. The pseudo code of the Basic-PSO is shown below.

```

01: Begin
02: Randomly initialize particle swarm
03: While (number of iterations, or the stopping criterion is not met)
04:   Evaluate fitness of particle swarm
05:   For n = 1 to the number of particles
06:     Find Pbest
07:     Find Gbest
08:     For d = 1 to the number of dimension of particle
09:       update the position of particles by Equation (5)
10:     Next d
11:   Next n
12: Next generation until the stopping criterion is met
13: End

```

Whenever the particle goes out of range of search space then no update applies to the particle.

3.1. Decreasing Weight Particle Swarm Optimization (DWPSO)

Here, we are linearly decreasing the inertia factor that was introduced into the velocity equation. The rationale behind DWPSO is to focus on diversity in early iterations and convergence in late iterations [2], i.e. decreasing weight PSO balances out the global and local search abilities of the swarm effectively. In this method, w_{new} is the new inertia weight, which linearly decreases from 0.9 to 0.4. Equation (6) for DWPSO is given as:

$$w_{new}(n) = w_s - (w_s - w_e) * n/n_t \quad (6)$$

In Equation (6), w_s (the inertia weight designated for the first iteration) is 0.4, w_e (the inertia weight designated for the last iteration) is 0.9, n is the current iteration, and n_t is the maximum number of iterations. The velocity of the updated equation for DWPSO is formulated as:

$$v_{ij}(n+1) = w_{new}(n) v_{ij}(n) + c_1 r_{1ij}(n) [p_{ij}(n) - x_{ij}(n)] + c_2 r_{2ij}(n) [g_{ij}(n) - x_{ij}(n)] \quad (7)$$

where, the personal best weight is c_1 , the global best weight is c_2 , both r_{1ij} , r_{2ji} are random numbers. The pseudo code of DWPSO is shown below.

```

01: Begin
02: Randomly initialize particle swarm
03: While (number of iterations, or the stopping criterion is not met)
04:   Evaluate fitness of particle swarm
05:   For n = 1 to the number of particles
06:     Find Pbest
07:     Find Gbest
08:     For d = 1 to the number of dimensions of particle
09:       Update the position of particles by Equation (7) and Equation (5)
10:     Next d
11:   Next n
12:   Update the inertia weight value with Equation (6)
13: Next generation until stopping criterion is met
14: End

```

3.2. Time-Varying PSO (TVPSO)

The objective of TVPSO is to enhance the global search in the early part of the optimization and to encourage the particles to converge toward the global optimum at the end of the search. With a large cognitive parameter and small social parameter at the beginning, particles are allowed to move freely around the search space, instead of moving towards the population's best [2]. However, a small cognitive parameter and a large social parameter allow the particles to converge to the global optimum in the latter stages of the optimization. Under this

development, the cognitive parameter c_1 starts with a high value and linearly decreases to a low value, whereas the social parameter c_2 starts with a low value and linearly increases to a high value [3]. The following equations are using to calculate the cognitive parameter and social parameter:

$$c_1(n) = c_{1s} - (c_{1s} - c_{1e}) \times n/n_t \quad (8)$$

$$c_2(n) = c_{2s} - (c_{2s} - c_{2e}) \times n/n_t \quad (9)$$

where c_{1s} is the personal best weight designated for the first iteration and the value is 1.5. c_{1e} is the personal best weight designated for the last iteration and has the value 2.5. c_{2s} is the global best weight designated for the first iteration and the value is 3.0. c_{2e} is the global best weight designated for the last iteration, and the value is 4.0. n_t is the maximum number of iterations, and n is the current iteration. The velocity of the updated equation for TVPSO is formulated as:

$$v_{ij}(n+1) = w \cdot v_{ij}(n) + c_1 r_{1ij}(n) [p_{ij}(n) - x_{ij}(n)] + c_2 r_{2ij}(n) [g_{ij}(n) - x_{ij}(n)] \quad (10)$$

where $v_{ij}(n+1)$ is the new velocity and Pseudo code for TVPSO is shown below.

```

01: Begin
02: Randomly initialize particle swarm
03: While (number of iterations, or the stopping criterion is not met)
04:   Evaluate fitness of particle swarm
05:   For n = 1 to the number of particles
06:     Find Pbest
07:     Find Gbest
08:     For d = 1 to the number of dimensions of particle
09:       Update the position of particles by Equation (10) and Equation (5)
10:     Next d
11:   Next n
12:   Update the cognitive parameter with Equation (8)
13:   Update the social parameter with Equation (9)
14: Next generation until stopping criterion is met
15: End

```

3.3. Guaranteed convergence PSO (GCPSO)

The guaranteed convergence PSO (GCPSO) was introduced by Van den Bergh and Engelbrecht in 2002. Based on the finding that the inertia weight variant of PSO can be non-convergent even on local minimizers, even if velocities become very small, a modified update scheme was introduced for the overall best particle, based on the local search algorithm [4].

More specifically, if the current and the best position of the i^{th} particle at iteration t coincide with the overall best position, i.e., $x_i(t) = p_i(t) = p_g(t)$, then the position update of x_i depends only on the previous velocity term, $w v_i(t)$. Thus, if the velocity is very close to zero, the particle is almost immobilized, since, in the long run, all particles are expected to approximate the global best position. The GCPSO uses the following equation to update the position of the global best particle:

$$v_{ij}(n+1) = -x_{ij}(n) + y_j(n) + w v_{ij}(n) + p(n)(1 - 2r_{3j}) \quad (11)$$

where the random number $r_3=0.1$, and the search radius parameter $p(n)$ is calculated using the following equation:

$$p(n+1) = \begin{cases} 2p(n), & \text{if } s(n) > s_c \\ 0.5p(n), & \text{if } a(n) > a_c \\ p(n), & \text{otherwise} \end{cases} \quad (12)$$

where, $s(n)$ is the number of success, and $a(n)$ is the number of failures. s_c is the success threshold and a_c is the failure threshold. If the number of successes is larger than the success threshold, the radius is doubled. If the number of failures is larger than the failure threshold, the

search radius is halved and the count of success, $s(n)$, and the count of failures, $a(n)$, are calculated using the following equations:

$$s(n+1) = \begin{cases} 0, & \text{if } a(n+1) > a(n) \\ s(n) + 1, & \text{otherwise} \end{cases} \quad (13)$$

$$a(n+1) = \begin{cases} 0, & \text{if } S(n+1) > S(n) \\ a(n) + 1, & \text{otherwise} \end{cases} \quad (14)$$

where the count of successes $s(n+1)$ is incremented by one and the count of failures $a(n+1)$ is reset to zero in case of success, and the count of failures $a(n+1)$ is increased by one and the count of successes $a(n)$ is reset to zero in case of failure. In high dimensional search spaces it is difficult to obtain better values using a random search in only a few iterations, so it is recommended to set the success threshold to $s_c=15$ and the failure threshold to $a_c=5$.

3.4. Time-Varying Acceleration Coefficients PSO (TVAC-PSO)

The basic idea of PSO is the search toward the optimum value based on the cognitive component and the social component. The control of these two components is significant to find the optimum value accurately and efficiently [6].

Linearly decreasing weight and acceleration coefficients were tested over time, but observed that the fixed acceleration coefficients are to generate better solutions. Therefore, a time varying acceleration coefficient as a new parameter automation strategy for the PSO concept is introduced. The objective is to enhance the global search in the early part of the optimization and to encourage the particles to converge toward the global optima at the end of

the search. The acceleration coefficients should be constant at all times. Here, the value of c_{11} is 1.49618, the value of c_2 is 1.49618, and the value of w is 0.7298. The c_1 , c_2 and w were calculated using the following equations:

$$c_1 = (c_{1f} - c_{1i}) \times n / n_t + c_{1i} \quad (15)$$

$$c_2 = (c_{2f} - c_{2i}) \times n / n_t + c_{2i} \quad (16)$$

$$w = (w_1 - w_2) \times (n_t - n) / n_t + w_2 \quad (17)$$

where the value of c_{1f} is 0.5, the value of c_{1i} is 2.5, the value of c_{2f} is 2.5, the value of c_{2i} is 0.5, n_t is the maximum number of iterations, and n is the current iteration, and w_1 and w_2 are constants ($w_1=0.95$ and $w_2=0.4$).

3.5. PSO with Mutation and Time Varying Acceleration Coefficients (MPSO)

In MPSO, to enhance the global search capability of the particles by providing additional diversity, mutation is widely used in most evolutionary optimization methods, such as evolutionary programming and genetic algorithms, to guide and enhance the search toward the global best solution [5]. The velocities and the positions of the particles for the next evaluation were calculated using the following equations:

$$V_{id} = V_{id} + c_1 \times r_1 \times (p_{id} - X_{id}) + c_2 \times r_2 \times (g_{id} - X_{id}) \quad (18)$$

$$X_{id} = X_{id} + V_{id} \quad (19)$$

where x is the position and v the velocity of particle i . p_i is the best position of each particle and p_g is the overall best position. c_1 and c_2 are constants and r_1 and r_2 are two different random numbers in the range of 0 to 1.

In PSO, the lack of diversity of the population, particularly during the latter stages of the optimization, was understood as the dominant factor for the premature convergence of particles to local optimum solutions, the steps or the pseudo code for this algorithm is shown below [7]:

```

1:  Begin
2:  initialize the population
3:  while (termination condition= false)
4:    do
5:      for (i= 1 to number of particles)
6:        evaluate the fitness: =f (x)
7:        update  $P_{id}$  and  $g_D$ 
8:        for d= 1 to number of dimensions
9:          calculate new Velocity: =  $v_{id}$ 
10:         update the position
11:         increase d
12:         increase i
13:         select a random particle: = k
14:         select a random dimension: = l
15:         if ( $\Delta g_{global} \leq 0$ )
16:           if (rand1 (.) <  $p_m$ )
17:             if (rand2 (.) < 0.5)
18:                $v_{kl} = v_{kl} + r_3 (.) * v_{max}/m$ ;
19:             else
20:                $v_{kl} = v_{kl} - r_4 (.) * v_{max}/m$ ;
21:             end if
22:           end if
24:         end if
25:       end for
26:     end do
27:   end

```

Here, the velocities and the positions of the particles for the next evaluation were calculated using Equations (18) and (19).

3.6. Adaptive Particle Swarm Optimization (APSO)

Adaptive particle swarm optimization (APSO) with adaptive parameters and elitist learning strategy based on the evolutionary state estimation (ESE) approach was introduced. The ESE approach develops an ‘evolutionary factor’ using the population distribution information and relative particle fitness information in each generation, and estimates the evolutionary state through a fuzzy classification method. According to the identified state, and taking into account various effects of the algorithm-controlling parameters, adaptive control strategies are developed for the inertia weight and acceleration coefficients for faster convergence speed. Further, an adaptive ‘elitist learning strategy’ (ELS) is designed for the best particle to jump out of possible local optima and/or to refine its accuracy, resulting in substantially improved quality of global solutions [11].

The evolutionary state in each generation is determined by a fuzzy classification method controlled by an evolutionary factor f . These techniques and the estimation process are detailed in the following steps:

Step 1: Find the mean distance of particle i to all the other particles by using the following equation:

$$d_i = \frac{1}{n-1} \sum_{j=1, j \neq i}^n \sqrt{\sum_{k=1}^d (x_i^k - x_j^k)^2} \quad (20)$$

where n and d are the population size and dimension.

Step 2: Compare all d_i values and determine the maximal distance d_{\max} and the minimal distance d_{\min} . Denote d_i of the global best particle by d_g . Define the evolutionary factor f as:

$$f = \frac{d_g - d_{\min}}{d_{\max} - d_{\min}} \in [0,1] \quad (21)$$

which is set to 1, if d_{\max} is equal to d_{\min} , and is also initialized to 1 when the algorithm starts.

Step 3: Based on the evolutionary factor (f), we determine the evolutionary state into four different states, which are convergence, exploitation, exploration and jumping-out states as shown in Table 3.6, and the inertia weight ω is used to balance the global and local search abilities, and was suggested to linearly decrease from 0.9 to 0.4 in every generation. However, it is not necessarily adequate to decrease only with time. Here are the basic equations for inertia weight, acceleration coefficients and particle position:

$$w(f) = \frac{1}{1 + 1.5e^{-2.6f}} \in [0,1] \quad (22)$$

$$c_i(n+1) = c_i(n) \pm \Delta, i=1, 2. \quad (23)$$

where Δ is a uniformly generated random value in the range [0.05, 0.1], as indicated by the empirical study. It should be noticed that we use 0.5δ in Strategies 2 and 3 where “slight” changes are used [11].

The best position, denoted by P_d here, is the formula for the best position, is calculated as follows:

$$p^d = p^d + (X_{\max}^d - X_{\min}^d) \text{Gaussian}(\mu, \sigma^2) \quad (24)$$

Table 3.1: Strategies for the values of c1 and c2

C1	C2	States	Strategies
Increase	Decrease	Exploration	Strategy 1
Slight Increase	Slight Decrease	Exploitation	Strategy 2
Slight Increase	Slight Increase	Convergence	Strategy 3
Decrease	Increase	Jumping-out	Strategy 4

Here, Gaussian (μ, σ^2) represents the Gaussian distribution with a mean $\mu=0$ and a time-varying standard deviation as:

$$\sigma = \sigma_{\max} - (\sigma_{\max} - \sigma_{\min}) \left(\frac{g}{G} \right) \quad (25)$$

where $\sigma_{\min}=0.1$ and $\sigma_{\max}=1.0$ and g is the lbest value and G is the g best value.

3.7. Algorithm Aspects

We have discussed three main aspects that are particle initialization, stopping condition, and function evaluation. With reference to Equations 2 and 3, the optimization process is iterative. Repeated iterations of the algorithms are executed until a stopping condition is satisfied. One such iteration consists of the application of all the steps within the repeat...until loop, i.e. determining the personal best positions and the global best position, and adjusting the velocity and position of each particle. A Function Evaluation (FE) refers to one calculation of the fitness function, which characterizes the optimization problem. For the basic PSO, a total of N_s

function evaluations are performed per iteration, where N_s is the total number of particles in the swarm.

The first step of the PSO algorithm is to initialize the swarm and control parameters. In the context of the basic PSO, the acceleration constants, C_1 and C_2 , the initial velocities, particle positions, and personal best positions need to be specified.

The last aspect of the PSO algorithm is the stopping condition. The stopping condition is to terminate the algorithm when a maximum number of iterations, or FEs, has been reached.

4. IMPLEMENTATION DETAILS

The application developed as a part of this project is a java application consisting of different classes. We briefly discuss those classes later in this chapter. Inputs required for this application are the number of iterations, global increment, and particle increment. All these inputs are given as parameters at the beginning of the algorithm. The basic PSO java code was provided by Dr. Simone Ludwig, and I expanded on the code by implementing the PSO variants.

4.1. RunPSO Class

This class is the main class that calls all other classes of the software. This class has variables which hold values for the number of iterations, inertia and particle increment. All algorithms used in this application are called in this class. The main functionality of this class is to calculate the best fitness, and the variance for each PSO variant, and to measure how much time a PSO variant needs to perform the optimization. Table 4.1 shows the variables that are initialized in this class.

Table 4.1: Variables of RunPSO Class

RETURN TYPE	METHOD	PURPOSE
Void	Begin()	This method is called at the beginning of each iteration.
Void	End()	This method is called at the end of each iteration.
Void	Update()	This method is used to update particle's velocity and position.

4.2. ParticleUpdateSimple Class

This class is responsible for calculating the particle's velocity and position in each iteration for each algorithm. This class has vectors, which hold values for the local update, global update and neighborhood update. Table 4.2 shows the methods that are initialized in this class.

Table 4.2: Methods of ParticleUpdateSimple Class

RETURN TYPE	METHOD	PURPOSE
Void	Begin()	This method is called at the beginning of each iteration.
Void	End()	This method is called at the end of each iteration.
Void	Update()	This method is used to update the velocity and position.

4.3. Swarm Class

This class is called to evaluate the fitness of every particle by using the best fitness so far (global best) and best position so far. This class generates the particle's maximum position, maximum velocity, minimum position and minimum velocity for each dimension. Tables 4.3 and 4.4 shows the methods and variables that are initialized in this class, respectively.

Table 4.3: Variables of Swarm Class

VARIABLE	TYPE	PURPOSE
bestFitness	Double	Stores the best fitness found so far.
bestparticleIndex	Int	Stores the Index of the best particle found so far.
fitnessFunction	FitnessFunction	Stores the Fitness function for this swarm.

Table 4.4: Methods of Swarm Class

RETURN TYPE	METHOD	PURPOSE
Double	maxPosition[]	Stores the maximum position for each dimension
Double	maxVelocity[]	Stores the maximum velocity for each dimension
Double	minPosition[]	Stores the minimum position for each dimension
Double	minVelocity[]	Stores the minimum velocity for each dimension

4.4. FitnessFuntion Class

The main functionality of this class is to evaluate particles position and fitness at a given position and also comparing the fitness values of the current iteration with the pervious iteration by using the isBetterThan () method. Table 4.5 shows the methods that are initialized in this class.

Table 4.5: Methods of FitnessFuntion Class

RETURN TYPE	METHOD	PURPOSE
double	evaluate()	Evaluates a particle's position and fitness.
Boolean	isBetterThan()	Returns true if 'other value' is better than the current fitness.

4.5. Benchmark Classes

In this application, each different benchmark function has its own class. These benchmark functions are used to compare the different PSO variants. We have a total of four different classes for the benchmark functions:

1. Alpine Class

2. Sphere Class

3. Rosenbrock Class

4. Rastrigin Class

We discuss the benchmark functions in more detail in Chapter 5.

5. BENCHMARK FUNCTIONS AND SETTINGS

In the field of evolutionary computation, it is common to compare different algorithms using benchmark functions. However, the effectiveness of an algorithm against another algorithm cannot be measured by the number of problems that it solves better. In general, if we compare two algorithms with all possible benchmark functions, the performance of any two algorithms will be, on average, the same. As a result, attempting to design a perfect test set where all the functions are present in order to determine whether an algorithm is better than another for every function, is a very difficult task. In this paper, all different PSO variants are tested based on the following four benchmark functions. These are the common used benchmarks in PSO.

5.1. Alpine (F1)

A nonseparable function $f(x)$ is called m -nonseparable function if at most m of its parameters x_i are not independent. A nonseparable function $f(x)$ is called fully-nonseparable function if any two of its parameters x_i are not independent.

$$F(\mathbf{x}) = \sum_{i=1}^D |x_i \sin d(x_i) + 0.1x_i| \quad (26)$$

5.2. Sphere (F2)

Sphere is also called first function of De Jong's and is one of the simplest test benchmarks. The function is continuous, convex and unimodal [14].

The Sphere function is defined as follows:

$$F(\mathbf{x}) = \sum_{i=1}^D x_i^2 \quad (27)$$

where D is the dimension and $\mathbf{X} = (X_1, X_2, \dots, X_d)$ is a D -dimensional row vector (i.e., a $1 \times D$ matrix). The Sphere function is very simple and is mainly used for demonstration.

5.3. Rosenbrock (F3)

Rosenbrock's function is also naturally nonseparable and is a classic optimization problem, also known as banana function or the second function of De Jong [14].

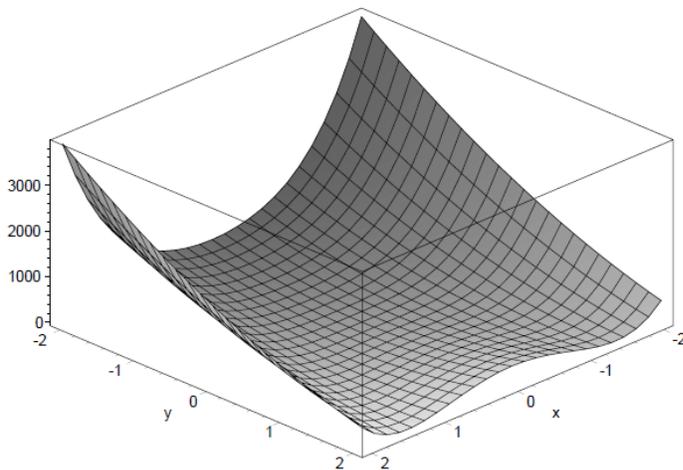


Figure 5.1: Rosenbrock's Function in 2D [14]

The function has the following definition:

$$F(\mathbf{x}) = \sum_{i=1}^{D-1} ((1 - x_i)^2 + 100(x_i^2 - x_{i+1})^2) \quad (28)$$

where $D \geq 2$ is the dimension and $X = (x_1, x_2, \dots, x_d)$ is a D -dimensional row vector (i.e., a $1 \times D$ matrix). Test area is usually restricted to hypercube $-2.048 \leq X_i \leq 2.048, i=1, \dots, n$. Its global minimum is $F(x)=0$ and is obtainable for $X_i, i=1, \dots, n$.

5.4. Rastrigin (F4)

Rastrigin function is based on the function of De Jong with the addition of Cosine modulation in order to produce frequent local minima and is defined as follows:

$$F(x) = \sum_{i=1}^D [g_i(x_i)^2 - 10 \cos(2\pi g_i(x_i))] + 10 \quad (29)$$

where D is the dimension and $x = (x_1, x_2, \dots, x_d)$ is a D -dimensional row vector (i.e., a $1 \times D$ matrix). Rastrigin's function is a classical multimodal problem. It is difficult since the number of local optima grows exponentially with the increase of dimensionality. Test area is usually restricted to hypercube $-5.12 \leq X_i \leq 5.12, i=1, \dots, n$. Its global minimum $F(x) = 0$ is obtainable for $x_i=0, i= 1, \dots, n$.

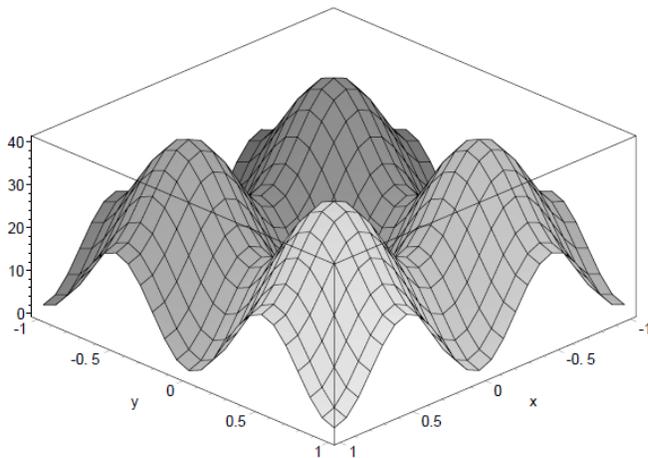


Fig 5.2: Rastrigin's Function in 2D [14]

In Table 5.1, the range of search, and range of initialization space of each benchmark functions is given.

Table 5.1: Range of Search of Benchmark Functions

Function	Range of search
Alpine	$(-10,10)^n$
Sphere	$(-5.12,5.12)^n$
Rosenbrock	$(-2.048,2.048)^n$
Rastrigin	$(-5.12,5.12)^n$

5.6. General Settings

In this implementation, we have the following general settings:

1. Initialization: Uniform random initialization within the search space.
2. Termination: Terminate the algorithm when the maximum number of function evaluations is achieved.
3. Global optimum: All problems have the global optimum within the given bounds, so there is no need to perform the search outside of the given bounds for these problems. The optimum function values are 0 for all problems.
4. Inertia Weight: Throughout this implementation it is set to 0.95 (if not modified by a PSO variant)
5. Global increment and particle increment are set to 0.1 (if not modified by a PSO variant).

6. EXPERIMENTS AND RESULTS

This section describes various test cases that are performed on the different algorithm variants, by changing the parameters, i.e. number of iterations, number of dimensions. The algorithms are compared based on the average best value with variance, as well as execution time. All measurement points are average values taken from 25 independent runs in order to guarantee statistical correctness. The experiments were conducted on an Intel Core 2 Duo (2.4GHz, 3MB L2 cache) running the Java version 1.6.2 JDK runtime environment.

6.1. Test Case 1: Varying Number of Iterations

Experiments are performed by changing the number of iterations and keeping the number of dimensions = 2, number of particles = 25, inertia weight = 0.95, global increment (C1) = 0.1 and particle increment (C2) = 0.1.

Table 6.1: Average and variance of each variant: 1 million iterations and 2 dimensions

Benchmark	Basic-PSO	DWPSO	TVPSO	GCPSO	TVAC-PSO	MPSO	APSO
Alpine	0.0 (0.0)	0.0 (0.0)	1.51E-11 (8.4E-24)	5.97E-7 (1.3E-14)	1.72E-15 (1.9E-31)	7.89E-5 (3.6E-10)	3.5E-5 (1.4E-11)
Rastrigin	0.0254 (2.4E-4)	0.0250 (2.3E-5)	1.75E-12 (9.5E-20)	1.3E-5 (7.9E-12)	1.62E-8 (8.1E-16)	1.83E-3 (2.2E-11)	9.98E-7 (3.1E-15)
Rosenbrock	0.0243 (2.1E-5)	0.00242 (2.2E-7)	2.75E-10 (2.7E-21)	8.336E-6 (2.5E-12)	7.221E-6 (1.9E-12)	3.672E-7 (4.9E-15)	3.2E-3 (6.6E-9)
Sphere	9.1E-15 (3.1E-30)	2.9E-23 (3.2E-47)	5.03E-22 (9.3E-45)	1.19E-13 (5.2E-28)	0.0 (0.0)	2.02E-15 (1.7E-31)	3.9E-11 (1.8E-19)

As can be seen from the table, for the Alpine benchmark, Basic-PSO and DWPSO achieve the best values. For the Rastrigin and Rosenbrock benchmarks, TVPSO achieves the best result. TVAC-PSO achieves the optimum of 0.0 for the Sphere benchmark.

Throughout this test case, the number of dimensions, particles, inertia weight, global increment, and particle increment are constants, based on those average values of the best fitness and the variance is calculated for each variant.

Table 6.2: Execution time in ms of each variant: 1 million iterations and 2 dimensions

Benchmark	Basic-PSO	DWPSO	TVPSO	GCPSO	TVAC-PSO	MPSO	APSO
Alpine	39034	32186	15473	11400	56360	21464	43687
Rastrigin	5409	6944	15746	11506	8851	24763	53623
Rosenbrock	37804	28926	7437	11826	69408	25618	32302
Sphere	22333	21574	10990	8296	14017	22143	21127

Table 6.2 shows the execution time for each algorithm. The above table shows the execution time of all variants for 1 million iterations and 2 dimensions. The execution time of GCPSO is the best for the Alpine and Sphere benchmark functions compared to all other variants. Basic-PSO is the fastest for the Rastrigin benchmark and the execution time of TVPSO is the best for the Rosenbrock function.

Similarly, test results for the number of iterations = 3,000,000, and number of dimensions = 2 are tabulated in Tables 6.3 and 6.4.

Table 6.3: Average and variance of each variant: 1 million iterations and 2 dimensions

Benchmark	Basic-PSO	DWPSO	TVPSO	GCPSO	TVAC-PSO	MPSO	APSO
Alpine	1.11E-15 (4.5E-32)	0.0 (0.0)	1.03E-12 (3.9E-26)	2.5E-10 (2.4E-17)	1.55E-15 (3.9E-32)	7.3E-9 (1.9E-18)	1.8E-5 (5.7E-12)
Rastrigin	4.08E-3 (3.6E-10)	3.4E-5 (2.1E-13)	4.5E-17 (3.9-E31)	3.7E-6 (2.4E-14)	3.8E-10 (3.9E-20)	5.4E-5 (6.2E-15)	2.2E-7 (1.8E-19)
Rosenbrock	5.5E-7 (1.1E-14)	1.4E-09 (7.6E-20)	1.1E-11 (4.8E-24)	2.92E-6 (3.1E-13)	1.08 (0.616)	1.8E-7 (1.3E-15)	7.9E-4 (8.3E-10)
Sphere	1.3E-15 (6.1E-32)	1.02E-07 (3.8E-52)	1.7E-24 (1.1E-49)	1.08E-14 (4.3E-30)	1.2E-18 (6.8E-39)	2.9E-16 (3.2E-33)	9.5E-13 (1.9E-21)

From Table 6.3, the performance of DWPSO is best for the Alpine Benchmark. TVAC achieves the best average values for the Rastrigin, Rosenbrock and Sphere functions.

Table 6.4 Shows the execution time for each variant: 3 million iterations and 2 dimensions

Benchmark	Basic-PSO	DWPSO	TVPSO	GCPSO	TVAC-PSO	MPSO	APSO
Alpine	128587	98993	41064	34789	124004	80312	314056
Rastrigin	118412	69959	45354	38929	29046	76607	125715
Rosenbrock	76588	155984	22416	38316	221868	78432	122996
Sphere	67173	394270	52499	28249	68940	61159	483609

Based on the table, the execution time is shortest for GCPSO running the Alpine and Sphere benchmark functions. TVPSO achieves the best execution time for the Rosenbrock benchmark, and the TVAC-PSO variant takes the least time for the Rastrigin function.

Considering the results in Tables 6.1-6.4, we can summarize that the performance of the DWPSO variant is best for the Alpine benchmark function, and TVPSO achieves the best results for the Rosenbrock and Rastrigin benchmark functions. For the Sphere benchmark, TVAC-PSO and TVPSO achieve the best results.

6.2. Test Case 2: Varying Number of Dimensions

Experiments are performed by changing the number of dimensions and keeping the number of iterations to 1 million, number of particles = 25, inertia weight = 0.95, global Increment (C1) = 0.1 and particle Increment (C2) = 0.1. Table 6.5 shows the average best fitness for every variant for 1 million iterations and 10 dimensions. As can be seen from the table, MPSO achieves the best average value for the Alpine, Rastrigin and Rosenbrock benchmarks. For the Sphere benchmark, DWPSO scores best compared to all other variants.

Table 6.5: Average and variance of each variant: 1 million iterations and 10 dimensions

Benchmark	Basic-PSO	DWPSO	TVPSO	GCPSO	TVAC-PSO	MPSO	APSO
Alpine	3.45652 (0.4404)	0.97544 (0.3507)	3.14649 (0.3649)	3.0438 (3.415)	3.5769 (4.717)	0.124 (5.6E-4)	2.123 (5.236)
Rastrigin	5.1664 (0.9839)	0.41522 (1.1E-4)	3.2941 (0.4002)	5.2913 (0.8732)	9.5127 (3.3359)	0.0312 (2.4E-7)	7.691 (10.627)
Rosenbrock	0.1952 (2.1E-3)	0.174 (1.4E-5)	0.1413 (7.5E-7)	0.284 (3.1E-3)	0.403 (9.3E-4)	0.0831 (2.5E-7)	1.652 (4.1029)
Sphere	1.24E-7 (5.7E-16)	7.70E-11 (2.1E-22)	0.0484 (8.6E-5)	0.0795 (2.3E-4)	0.2924 (3.1E-5)	0.006 (1.4E-6)	0.894 (6.3E-3)

The execution time of each variant for 10 dimensions and 1 million iterations is shown in Table 6.6. The execution time for TVPSO is the shortest for the Alpine, Rosenbrock and Sphere benchmark functions compared to all other variants. Basic-PSO executes fastest for the Rastrigin benchmark.

Table 6.6: Execution time in ms of each variant: 1 million iterations and 10 dimensions

Benchmark	Basic-PSO	DWPSO	TVPSO	GCPSO	TVAC-PSO	MPSO	APSO
Alpine	25212	28210	9409	14245	1030486	2337428	3140561
Rastrigin	8722	10689	1103548	591904	546952	132574	2509451
Rosenbrock	19661	18300	13832	2218020	1788307	5119510	4916527
Sphere	26951	2412109	5930	10972	3480877	3244874	3513081

Experiments are performed by changing the number of dimensions and the number of iterations, and keeping the inertia weight = 0.95, global increment (C1) = 0.1 and particle increment (C2) = 0.1. The experimental results of 3 million iterations and 10 dimensions are showed in Tables 6.7 and 6.8.

Table 6.7: Average and variance of each variant: 3 million iterations and 10 dimensions

Benchmark	Basic-PSO	DWPSO	TVPSO	GCPSO	TVAC-PSO	MPSO	APSO
Alpine	4.909201 (0.88843)	4.72132 (0.82173)	0.65912 (0.01601)	7.1878 (6.6168)	2.79 (1.6724)	0.0079 (1.74E-6)	6.927 (6.521)
Rastrigin	6.3541 (1.9457)	3.401 (0.81109)	6.5607 (2.9851)	6.256 (5.427)	5.893 (6.214)	0.0021 (3.1E-9)	5.812 (4.763)
Rosenbrock	2.0956 (0.1619)	1.9976 (0.1471)	0.06917 (2.2E-8)	0.04653 (2.4E-7)	0.0943 (6.1E-6)	0.00612 (1.3E-9)	3.546 (10.429)
Sphere	1.906E-6 (1.3E-13)	1.232E-9 (5.6E-20)	0.0112 (4.63E-6)	1.073E-4 (4.2E-10)	0.2586 (2.46E-8)	0.0021 (1.69E-7)	0.691 (8.4E-5)

The MPSO scores best for the Alpine, Rastrigin, and Rosenbrock benchmarks, and DWPSO achieve the best average value for the Sphere benchmark function. Comparing these results of with the ones in Table 6.5 (1 million and 10 dimensions) and Table 6.7 (3 million and 10 dimensions), the performance of DWPSO decreases when the number of iterations increases, and MPSO scores better. The execution time of the MPSO variant is also shorter compared to the DWPSO for most of the functions for larger iterations.

Table 6.8: Execution time in ms of each variant: 3 million iterations and 10 dimensions

Benchmark	Basic-PSO	DWPSO	TVPSO	GCPSO	TVAC-PSO	MPSO	APSO
Alpine	3703908	837274	2408058	168263	7867915	8114667	8940564
Rastrigin	48839	599516	4627792	752300	1411713	4321572	4173831
Rosenbrock	1170948	4483442	6035422	173388	140310	8314481	6389964
Sphere	4608681	4308517	117461	175111	1465041	9686968	4285312

The execution time for 3 million iterations and 10 dimensions for all variants is shown in Table 6.8. The shortest execution times are achieved by GCPSO, Basic-PSO, TVAC-PSO, TVPSO for the Alpine, Rastrigin, Rosenbrock, Sphere benchmark functions, respectively.

The experimental results are summarized as follows:

- 1) Basic-PSO is better for small dimensions and smaller numbers of iterations.
- 2) DWPSO and TVPSO achieve the best average values for most of the benchmark functions in Test case 1.
- 3) The performance of MPSO is best for most of the benchmark functions in Test case 2.

7. CONCLUSION AND FUTURE WORK

Particle swarm optimization has two main operators: velocity update and position update. During each iteration, each particle is accelerated toward the particle's previous best position and the global best position. At each iteration a non-velocity value for each particle is calculated based on its current velocity. Particle swarm optimization is an extremely simple algorithm that has proven to be effective for the optimization of a wide range of functions. It is a heuristic optimization method that is based on swarm intelligence.

In this paper, we have implemented and tested a variety of PSO variants such as decreasing weight PSO, time-varying acceleration PSO, guaranteed convergence PSO, mutation time-varying acceleration PSO, and adaptive PSO. Experiments were conducted to measure the solution quality and execution time of the different variants. Based on the results as shown in Chapter 5, the performance of DWPSO is better for most of the test cases with low dimensionality and iterations, and MPSO is found to be better for higher number of iterations and higher dimensions. TVAC-PSO and Basic-PSO also score better in some cases. Finally, we can conclude that based on the four benchmark functions tested, DWPSO and MPSO are the best PSO variants.

Some of the major issues encountered while conducting the experiments of the PSO variants is the execution time measured for each benchmark function. If we only increase the number of dimensions by one, the execution time rises tremendously. Therefore, further experiments need to be run in order to evaluate the effect of higher dimensionality.

REFERENCES

- [1] J. Kennedy and R.C. Eberhart, “Particle Swarm Optimization”, Proceedings of the IEEE International Conference on Neural Networks, Perth, Australia, pp. 1942–1948 ,1995.
- [2] A. Engelbrecht, “Computational Intelligence - An Introduction”, 2nd Edition. Wiley, 2007.
- [3] A. Ratnaweera, S.K. Halgamuge, and H.C. Watson. “Self- organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients”, *IEEE Transaction of Evolutionary Computation*, 8(3):240-255, 2004.
- [4] J. Kennedy and R. Eberhart. Particle swarm optimization. IEEE International Conference on neural Networks, 4:1942 -1948, 1995.
- [5] P.J. Angeline, “Evolutionary optimization verses particle swarm optimization: Philosophy and the performance difference”, Lecture Notes in Computer Science, vol. 1447, Proc. 7th Int. Conf. Evolutionary Programming - Evolutionary Programming VII, pp. 600–610 1998.
- [6] Y. Shi and R. Eberhart, “A modified particle swarm optimizer”, Proc. IEEE Int. Conf. Evolutionary Computation, pp. 69–73, 1998.
- [7] M. Lovbjerg and T. Krink, “Extending particle swarm optimizers with self-organized critically”, Proc. IEEE Int. Congr. Evolutionary Computation, vol. 2, Honolulu, HI, pp. 1588–1593, 2002.
- [8] M. Lovbjerg and T. Krink, “Extending particle swarm optimizers with self-organized critically”, Proc. IEEE Int. Congr. Evolutionary Computation, vol. 2, Honolulu, HI, pp. 1588–1593, 2002.

- [9] M. Lovbjerg, T. K. Rasmussen, and T. Krink, “Hybrid particle swarm optimizer with breeding and subpopulation”, Proc. 3rd Genetic Evolutionary Computation Conf. (GECCO-2001), San Francisco, pp. 469–476, 2001.
- [10] F. van den Bergh, and A. P. Engelbrecht, “Effect of swarm size on cooperative particle swarm optimizers”, Proc. Genetic Evolutionary Computation Conf. (GECCO-2001), pp. 892–899, 2001.
- [11] Z. Zhan and J. Zhang, “Adaptive Particle Swarm Optimization”. In: Department of Computer Science, Sun Yat-sen University, China, 2005.
- [12] A.P. Engelbrecht, “Computational Intelligence an Introduction”, Second Edition, University of Pretoria, South Africa, December 2005.
- [13] J. Kennedy, “Stereotyping: Improving particle swarm performance with cluster analysis”, Proc. IEEE Int. Conf. Evolutionary Computation, vol. 2, pp. 303–308, 2000.
- [14] M. Molga and C. Smutnicki, “Test functions for optimization needs”, 2005.