



Jaypee University of Information Technology
Solan (H.P.)

LEARNING RESOURCE CENTER

Acc. Num. *SP07026* Call Num: ११११११

General Guidelines:

- ◆ Library books should be used with great care.
- ◆ Tearing, folding, cutting of library books or making any marks on them is not permitted and shall lead to disciplinary action.
- ◆ Any defect noticed at the time of borrowing books must be brought to the library staff immediately. Otherwise the borrower may be required to replace the book by a new copy.
- ◆ The loss of LRC book(s) must be immediately brought to the notice of the Librarian in writing.

Learning Resource Centre-JUIT



SP07026

Analysis of convergence behavior of Particle Swarm Optimization

A project report submitted in partial fulfillment of the award of degree of
Bachelor of Technology in Computer Science & Engineering on
by

Mansi (Roll No. 071257)

Kanika choudhary (Roll No. 071286)

Project Guide: Mr Satish Chandra



Jaypee University of Information Technology

Wagnaghat, Solan, India

2011

TABLE OF CONTENTS


Chapter No.	Topics	Page No.
	Certificate from the supervisor	II
	Acknowledgement	III
	Abstract	IV
	List of figures	V
Chapter 1	Introduction	
	1.1 Particle swarm optimization	
	1.2 Aim and objective	
	1.3 Tools and technologies	
Chapter 2	Particle Swarm Optimization	
	2.1 Background: Artificial Life	
	2.2 Standard pso algorithm	
	2.3 Main pso variants	
	2.4 Comparison between genetic algorithm and PSO	
	2.5 PSO control parameter	
Chapter 3	Convergence Behaviour	
	3.1 Premature convergence problem	
	3.2 Causes of premature convergence	
	3.3 Tackling premature convergence	
Chapter 4	Implementation	
	4.1 Applications of pso and current trends	
Chapter 5	Conclusion	
	Appendix	
	References	
	Bio-data	

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY
WAKNAGHAT, SOLAN, HIMACHAL PRADESH

CERTIFICATE

Date:

This is to certify that the work titled "IMPLEMENTATION OF PARTICLE SWARM OPTIMIZATION AND STUDY ITS CONVERGENCE BEHAVIOUR" submitted by "Kanika and Mansi" in partial fulfilment for the award of degree of B. Tech of Jaypee University of Information Technology; Wagnaghat has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor 
Name of Supervisor DR. Satish Chandra
Designation Asstt Professor:.....

ACKNOWLEDGEMENT

It gives us great pleasure in presenting the project report for our project on 'IMPLEMENTING PARTICLE SWARM OPTIMIZATION AND STUDY ITS CONVERGENCE BEHAVIOUR'. We would like to take this opportunity to thank our project guide **Dr. Satish Chandra** for giving us all the help and guidance we needed. We are really grateful to him for his kind support throughout the analysis and design phase. We are also grateful to Brig. S.P. Ghrrera, Head of CSE/I.T Department, Jaypee University of Information Technology and other staff members for giving important suggestions

Signature of the student

Mansi

Karriika

Name of Student

Mansi

Karriika

Date

24/5/11

24/5/11

Abstract

The particle swarm is an algorithm for finding optimal regions of complex search spaces through interaction of individuals in a population of particles . Some results of the particle swarm optimizer, implementing modifications derived from the analysis, suggest methods for altering the original algorithm in ways that eliminate problems and increase the optimization power of the particle swarm.

In order to prevent particle swarm optimization from being trapped in local optima, a new vector called the weighted individual best position of the other particles is introduced. On the one hand, the behavior of each particle is not only influenced by its own best position and the global best position but also by the individual best positions of the others in the swarm. On the other hand, fitness value is used, i.e., each particle weighs the contributions of the others according to their fitness values. So the modified algorithm strengthens cooperation and competition among the particles by making each particle share more useful information of the others. Six benchmark functions are tested and results show that the modified algorithm is more effective than basic particle swarm optimization.

Signature of Student:
Name: Mansi
Date 24/5/11

Mansi

Kanika
Signature of Student
Name Kanika
Date 24/5/11

Schod

Signature of Supervisor
Name Dr. Satish Chandra
Date 24/5/11

LIST OF FIGURES

Sr. No	Name of Diagram
1.1	Bird Swarm
1.2	Graph of local and global optimum
1.3	Flow chart of PSO
2.1	Gbest algorithm
2.2	Lbest algorithm
2.3	DFD
2.4	PSO particles in search space
2.5	Graph of constriction coefficient

Chapter 1

Introduction

Numerical optimization has been widely used in engineering to solve a variety of NP-complete problems in areas such as structural optimization, neural network training, control system analysis and design, and layout and scheduling problems. In these and other engineering disciplines, two major obstacles limiting the solution efficiency are frequently encountered. First, even medium-scale problems can be computationally demanding due to costly fitness evaluations. Second, engineering optimization problems are often plagued by multiple local optima, requiring the use of global search methods such as population-based algorithms to deliver reliable results.

Fortunately, recent advances in microprocessor technology and network technology have led to increased availability of low cost computational power through clusters of low to medium performance.

1.1: Particle Swarm Optimization

Particle swarm optimization (PSO) is a population-based stochastic approach for solving continuous and discrete optimization problems. In particle swarm optimization, simple software agents, called *particles*, move in the search space of an optimization problem. The position of a particle represents a candidate solution to the optimization problem at hand. Each particle searches for better positions in the search space by changing its velocity according to rules originally inspired by behavioral models of bird flocking.

Particle swarm optimization belongs to the class of swarm intelligence techniques that are used to solve optimization problems. Particle swarm optimization was introduced by Kennedy and Eberhart (1995). It has roots in the simulation of social behaviors using tools and ideas taken from computer graphics and social psychology research.

Within the field of computer graphics, the first antecedents of particle swarm optimization can be traced back to the work of Reeves (1983), who proposed particle systems to model objects that are dynamic and cannot be easily represented by polygons

or surfaces. Examples of such objects are fire, smoke, water and clouds. In these systems, particles are independent of each other and their movements are governed by a set of rules. Some years later, Reynolds (1987) used a particle system to simulate the collective behavior of a flock of birds. In a similar kind of simulation, Heppner and Grenander (1990) included a *roost* that was attractive to the simulated birds. Both models inspired the set of rules that were later used in the original particle swarm optimization algorithm.

Social psychology research, in particular the dynamic theory of social impact (Nowak, Szamrej & Latané, 1990), was another source of inspiration in the development of the first particle swarm optimization algorithm (Kennedy, 2006). The rules that govern the movement of the particles in a problem's search space can also be seen as a model of human social behavior in which individuals adjust their beliefs and attitudes to conform with those of their peers (Kennedy & Eberhart 1995).

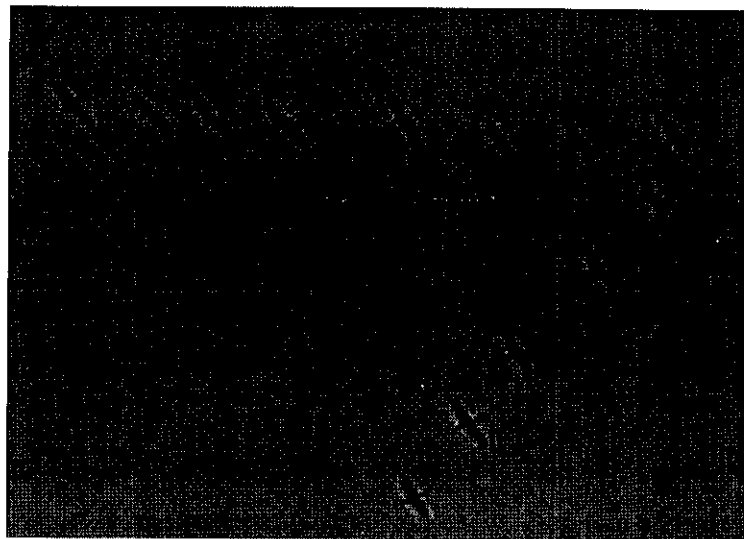


Figure 1.1: Bird Swarm

1.2 Aim and Objective:

- ▣ The project is aimed at implementing the particle swarm optimization and study the convergence behaviour.
- ▣ The project also includes the study about the main causes of premature convergence and the various methods of tackling premature convergence
- ▣ **1.3 Tools and Technologies** The implementation of Particle Swarm optimization has been done using turbo c. There is no as such hardware specification required in this project.

Chapter2

Particle Swarm Optimization

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.

PSO shares many similarities with evolutionary computation techniques such as Genetic Algorithms (GA). The system is initialized with a population of random solutions and searches for optima by updating generations. However, unlike GA, PSO has no evolution operators such as crossover and mutation. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles. The detailed information will be given in following sections.

2.1 Background: Artificial life

The term "Artificial Life" (ALife) is used to describe research into human-made systems that possess some of the essential properties of life. ALife includes two-folded research topic:

1. ALife studies how computational techniques can help when studying biological phenomena
2. ALife studies how biological techniques can help out with computational problems

The focus of this report is on the second topic. Actually, there are already lots of computational techniques inspired by biological systems. For example, artificial neural network is a simplified model of human brain; genetic algorithm is inspired by the human evolution.

Here we discuss another type of biological system - social system, more specifically, the collective behaviors of simple individuals interacting with their environment and each other. Someone called it as swarm intelligence. All of the simulations utilized local processes, such as those modeled by cellular automata, and might underlie the unpredictable group dynamics of social behavior.

Some popular examples are flocks and boids. Both of the simulations were created to interpret the movement of organisms in a bird flock or fish school. These simulations are normally used in computer animation or computer aided design.

There are two popular swarm inspired methods in computational intelligence areas: Ant colony optimization (ACO) and particle swarm optimization (PSO). ACO was inspired by the behaviors of ants and has many successful applications in discrete optimization problems.

The particle swarm concept originated as a simulation of simplified social system. The original intent was to graphically simulate the choreography of birds in a bird flock or fish school. However, it was found that particle swarm model can be used as an optimizer.

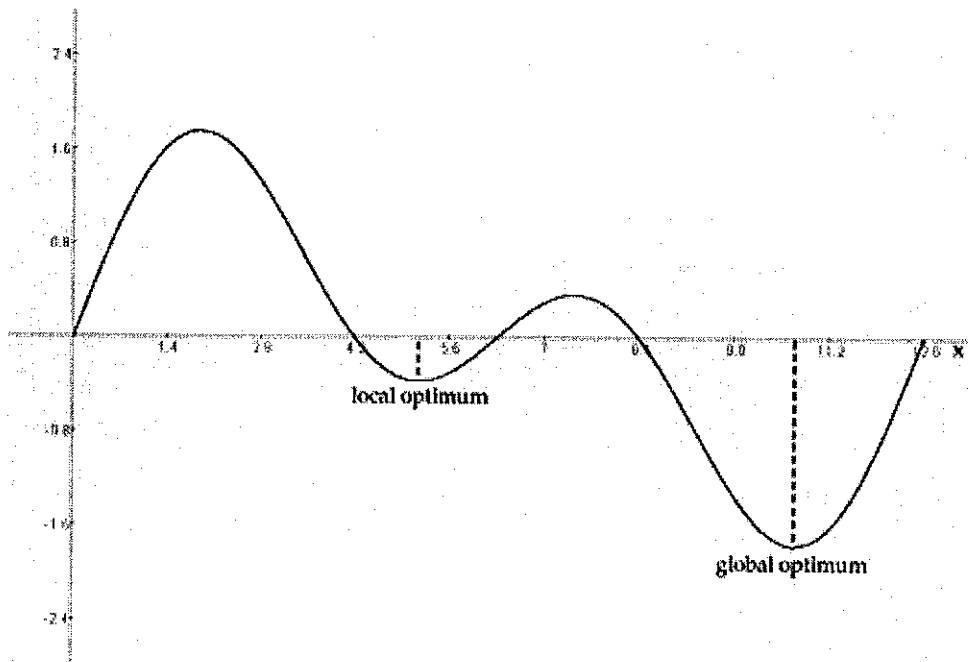


Fig1.2

2.2 Standard PSO algorithm

The algorithm

The PSO algorithm starts by generating random positions for the particles, within an initialization region $\Theta' \subseteq \Theta$. Velocities are usually initialized within Θ' but they can also be initialized to zero or to small random values to prevent particles from leaving the search space during the first iterations. During the main loop of the algorithm, the velocities and positions of the the particles are iteratively updated until a stopping criterion is met.

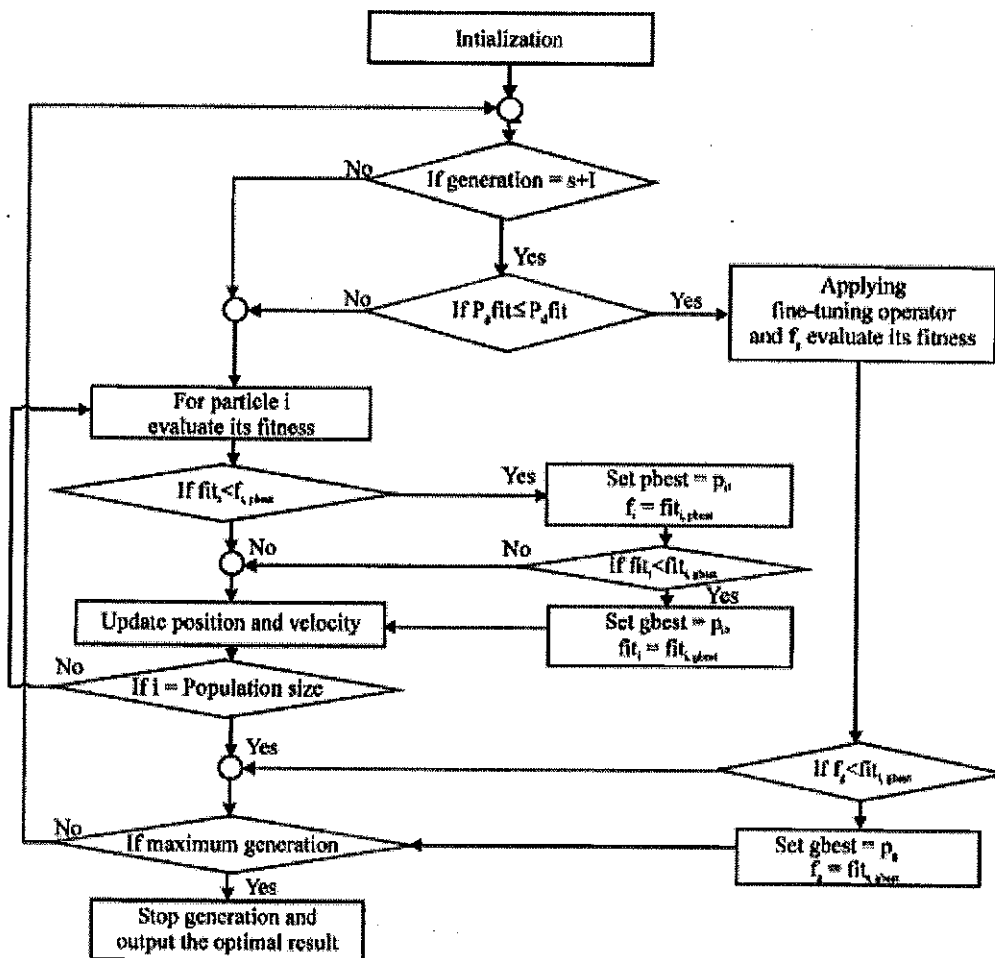


Fig1.3

The update rules are:

$$\vec{v}_i^{t+1} = w\vec{v}_i^t + \varphi_1\vec{U}_1^t(\vec{b}_i^t - \vec{x}_i^t) + \varphi_2\vec{U}_2^t(\vec{l}_i^t - \vec{x}_i^t),$$

$$\vec{x}_i^{t+1} = \vec{x}_i^t + \vec{v}_i^{t+1},$$

where w is a parameter called *inertia weight*, φ_1 and φ_2 are two parameters called *acceleration coefficients*, \vec{U}_1^t and \vec{U}_2^t are two $n \times n$ diagonal matrices in which the entries in the main diagonal are random numbers uniformly distributed in the interval $[0, 1)$. At each iteration, these matrices are regenerated. Usually, vector \vec{l}_i^t , referred to as the *neighborhood best*, is the best position ever found by any particle in the neighborhood of particle p_i , that is, $f(\vec{l}_i^t) \leq f(\vec{b}_j^t) \forall p_j \in \mathcal{N}_i$. If the values of w , φ_1 and φ_2 are properly chosen, it is guaranteed that the particles' velocities do not grow to infinity (Clerc and Kennedy 2002).

The three terms in the velocity-update rule above characterize the local behaviors that particles follow. The first term, called the *inertia* or *momentum* serves as a memory of the previous flight direction, preventing the particle from drastically changing direction. The second term, called the *cognitive component* models the tendency of particles to return to previously found best positions. The third term, called the *social component* quantifies the performance of a particle relative to its neighbors. It represents a group norm or standard that should be attained.

```

Create and initialize an  $n_s$ -dimensional swarm,  $S$ ; repeat

    for each particle  $i=1, \dots, S.n_s$  do
        //set the personal best position
        if  $f(S.x_i) < f(S.y_i)$  then
             $S.y_i = S.x_i$ ;
        end
        //set the global best position if  $f(S.y_i) < f(S.\hat{y})$  then
             $S.y = S.y_i$ ;
        end
    end
end
for each particle  $i=1, \dots, S.n_s$  do
    update the velocity using Eq. (2c);
    update the position using Eq. (2a);
end
until stopping condition is true;

```

Fig. 2: gbest algorithm

```

Create and initialize an  $n_s$ -dimensional swarm,  $S$ ;
repeat
    for each particle  $i=1, \dots, S.n_s$  do
        //set the personal best position
        if  $f(S.x_i) < f(S.y_i)$  then
             $S.y_i = S.x_i$ ;
        end
        //set the neighborhood best position
        if  $f(y_i) < f(\hat{y}_i)$  then
             $S.y = S.y_i$ ;
        end
    end
end
for each particle  $i=1, \dots, S.n_s$  do
    update the velocity using equation (2d);
    update the position using equation (2a);
end
until stopping condition is true;

```

Fig. 3: lbest algorithm

The main algorithm :-

Inputs *Objective function* $f : \Theta \rightarrow \mathbb{R}$, the initialization domain $\Theta' \subseteq \Theta$,

the number of particles $|\mathcal{P}| = k$, the parameters w , φ_1 , and φ_2 , and the stopping criterion S

:Output *Best solution found*

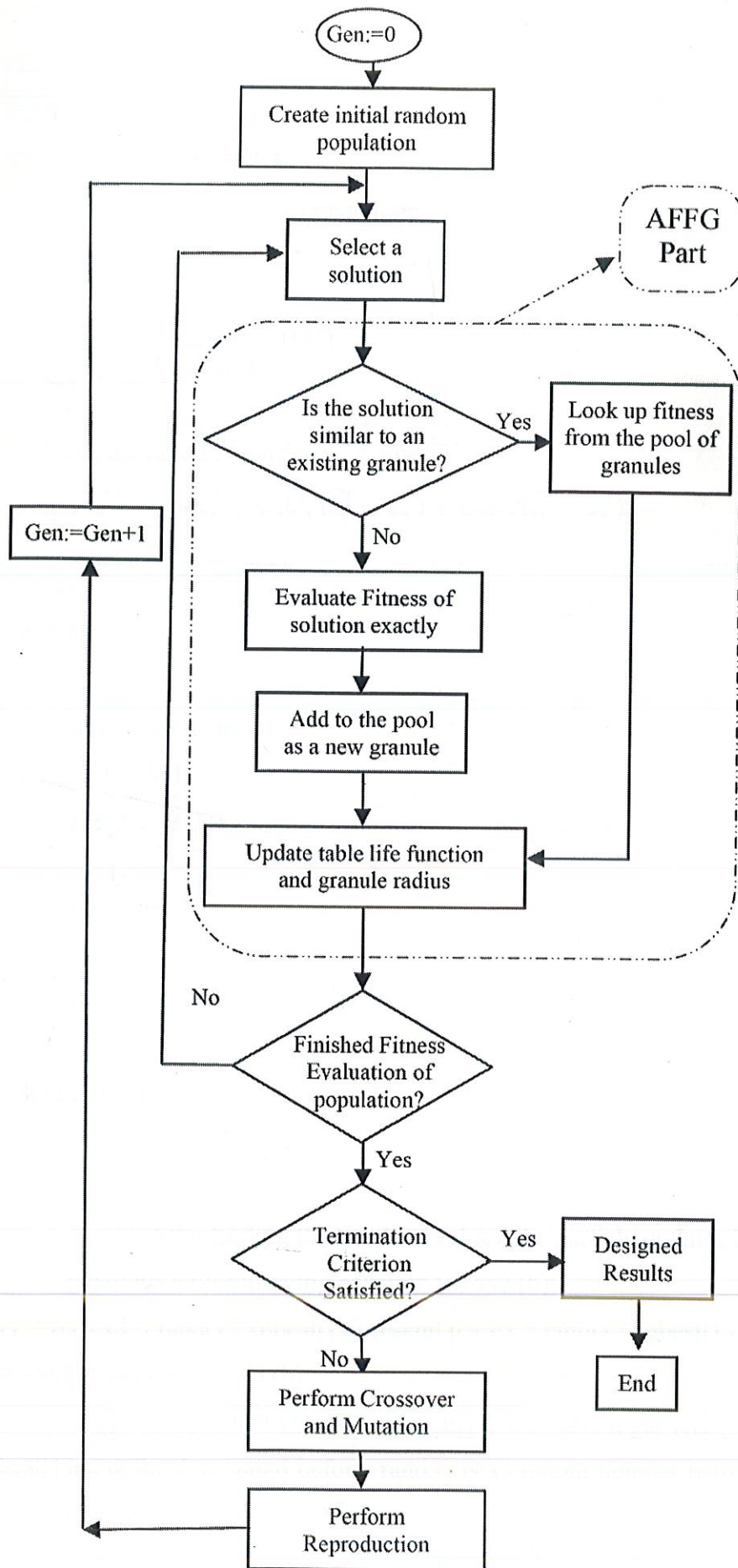
// Initialization

Set $t := 0$

for $i := 1$ to k do

Initialize \mathcal{N}_i to a subset of \mathcal{P} according to the desired topology

Initialize \vec{x}_i^t randomly within Θ'



```

Set
end for
// Main loop
while  $S$  is not satisfied do
    // Velocity and position update loop
    for  $i := 1$  to  $k$  do

        Set  $\vec{l}_i^t := \arg \min_{\vec{b}_j^t \in \Theta \mid p_j \in \mathcal{N}_i} f(\vec{b}_j^t)$ 

        Generate random matrices  $\vec{U}_1^t$  and  $\vec{U}_2^t$ 

        Set  $\vec{v}_i^{t+1} := w\vec{v}_i^t + \varphi_1\vec{U}_1^t(\vec{b}_i^t - \vec{x}_i^t) + \varphi_2\vec{U}_2^t(\vec{l}_i^t - \vec{x}_i^t)$ 

        Set  $\vec{x}_i^{t+1} := \vec{x}_i^t + \vec{v}_i^{t+1}$ 

    end for

    // Solution update loop
    for  $i := 1$  to  $k$  do
        if  $f(\vec{x}_i^t) < f(\vec{b}_i^t)$ 
            Set  $\vec{b}_i^t := \vec{x}_i^t$ 
        end if
    end for

    Set  $t := t + 1$ 

```

end while

After finding the two best values, the particle updates its velocity and positions with following equation (a) and (b).

$$v[] = v[] + c1 * \text{rand}() * (\text{pbest}[] - \text{present}[]) + c2 * \text{rand}() * (\text{gbest}[] - \text{present}[]) \quad (\text{a})$$

$$\text{present}[] = \text{present}[] + v[] \quad (\text{b})$$

$v[]$ is the particle velocity, $\text{present}[]$ is the current particle (solution). $\text{pbest}[]$ and $\text{gbest}[]$ are defined as stated before. $\text{rand}()$ is a random number between (0,1). $c1$, $c2$ are learning factors. usually $c1 = c2 = 2$.

2.3 Main PSO variants:-

The original particle swarm optimization algorithm has undergone a number of changes since it was first proposed. Most of these changes affect the way the velocity of a particle is updated. In the following subsections, we briefly describe some of the most important developments. For a more detailed description of many of the existing particle swarm optimization variants.

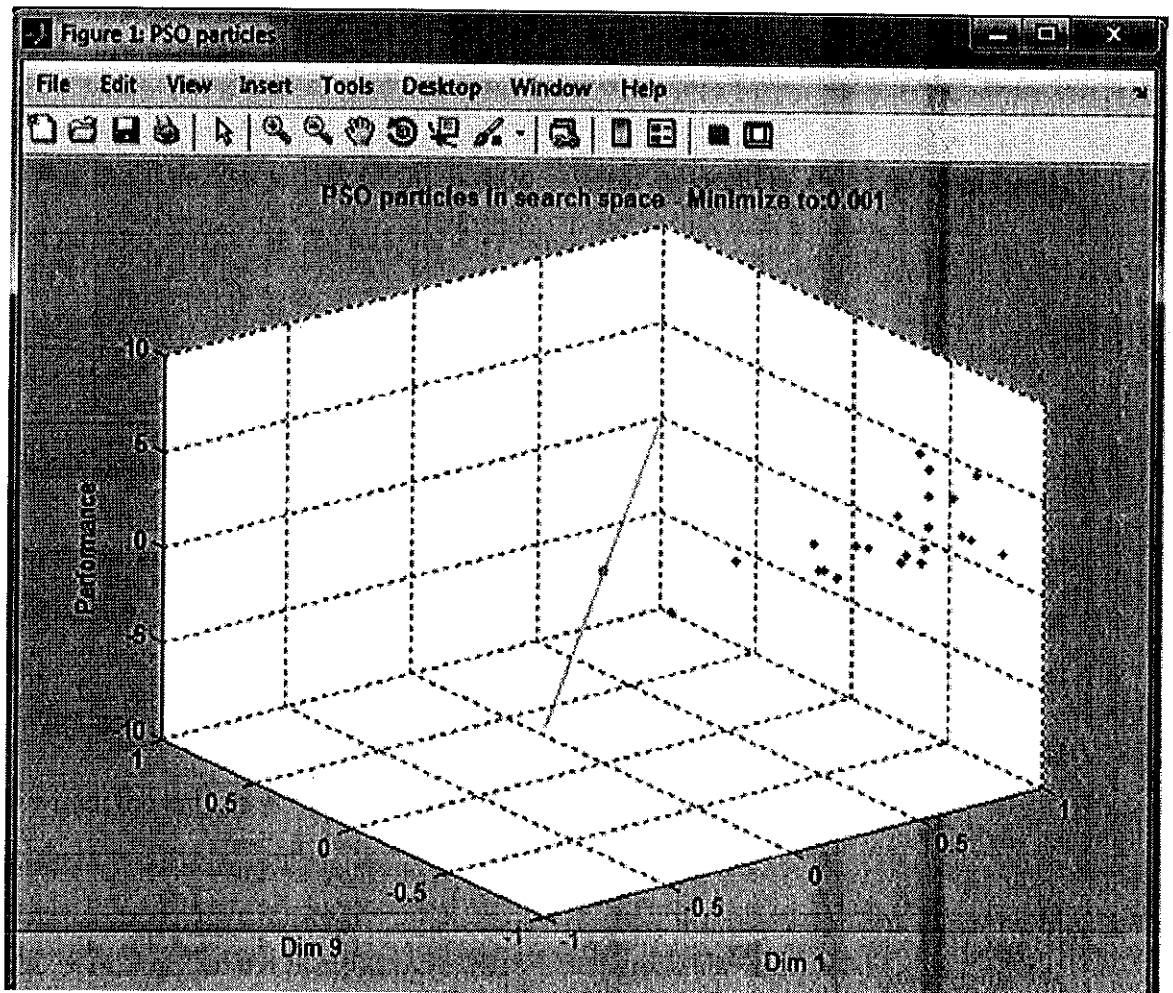


Fig2.4

Discrete PSO

Most particle swarm optimization algorithms are designed to search in continuous domains. However, there are a number of variants that operate in discrete spaces. The first variant proposed for discrete domains was the binary particle swarm optimization algorithm (Kennedy and Eberhart 1997). In this algorithm, a particle's position is discrete but its velocity is continuous. The j th component of a particle's velocity vector is used to compute the probability with which the j th component of the particle's position vector takes a value of 1. Velocities are updated as in the standard PSO algorithm, but positions are updated using the following rule:

$$x_{ij}^{t+1} = \begin{cases} 1 & \text{if } r < \text{sig}(v_{ij}^{t+1}), \\ 0 & \text{otherwise,} \end{cases}$$

where x_{ij} is the j th component of the position vector of particle P_i , r is a uniformly distributed random number in the interval $[0, 1)$ and

$$\text{sig}(x) = \frac{1}{1 + e^{-x}}.$$

Constriction Coefficient

The *constriction coefficient* was introduced as an outcome of a theoretical analysis of swarm dynamics (Clerc and Kennedy 2002). Velocities are constricted, with the following change in the velocity update:

$\vec{v}_i^{t+1} = \chi^t [\vec{v}_i^t + \varphi_1 \vec{U}_1^t (\vec{b}_i^t - \vec{x}_i^t) + \varphi_2 \vec{U}_2^t (\vec{l}_i^t - \vec{x}_i^t)]$, where χ^t is an $n \times n$ diagonal matrix in which the entries in the main diagonal are calculated as

$\chi_{jj}^t = \frac{2\kappa}{|2 - \varphi_{jj}^t - \sqrt{\varphi_{jj}^t(\varphi_{jj}^t - 2)}|}$ with $\varphi_{jj}^t = \varphi_1 U_{1,jj}^t + \varphi_2 U_{2,jj}^t$. Convergence is guaranteed under the conditions that $\varphi_{jj}^t \geq 4 \forall j$ and $\kappa \in [0, 1]$.

2.4 Comparisons between Genetic Algorithm and PSO

Most of evolutionary techniques have the following procedure:

1. Random generation of an initial population
2. Reckoning of a fitness value for each subject. It will directly depend on the distance to the optimum.
3. Reproduction of the population based on fitness values.
4. If requirements are met, then stop. Otherwise go back to 2.

From the procedure, we can learn that PSO shares many common points with GA. Both algorithms start with a group of a randomly generated population, both have fitness values to evaluate the population. Both update the population and search for the optimum with random techniques. Both systems do not guarantee success.

However, PSO does not have genetic operators like crossover and mutation. Particles update themselves with the internal velocity. They also have memory, which is important to the algorithm.

Compared with genetic algorithms (GAs), the information sharing mechanism in PSO is significantly different. In GAs, chromosomes share information with each other. So the whole population moves like a one group towards an optimal area. In PSO, only gBest (or lBest) gives out the information to others. It is a one-way information sharing mechanism. The evolution only looks for the best solution. Compared with GA, all the particles tend to converge to the best solution quickly even in the local version in most cases.

2.5. PSO parameter control

From the above case, we can learn that there are two key steps when applying PSO to optimization problems: the representation of the solution and the fitness function. One of the advantages of PSO is that PSO take real numbers as particles. It is not like GA, which needs to change to binary encoding, or special genetic operators have to be used. For example, we try to find the solution for $f(x) = x_1^2 + x_2^2 + x_3^2$, the particle can be set as (x_1, x_2, x_3) , and fitness function is $f(x)$. Then we can use the standard procedure to find the optimum. The searching is a repeat process, and the stop criteria are that the maximum iteration number is reached or the minimum error condition is satisfied.

There are not many parameter need to be tuned in PSO. Here is a list of the parameters and their typical values.

The number of particles: the typical range is 20 - 40. Actually for most of the problems 10 particles is large enough to get good results. For some difficult or special problems, one can try 100 or 200 particles as well.

Dimension of particles: It is determined by the problem to be optimized,

Range of particles: It is also determined by the problem to be optimized, you can specify different ranges for different dimension of particles.

Vmax: it determines the maximum change one particle can take during one iteration.

Usually we set the range of the particle as the Vmax for example, the particle (x_1, x_2, x_3) x_1 belongs $[-10, 10]$, then $V_{max} = 20$

Learning factors: c_1 and c_2 usually equal to 2. However, other settings were also used in different papers. But usually c_1 equals to c_2 and ranges from $[0,4]$

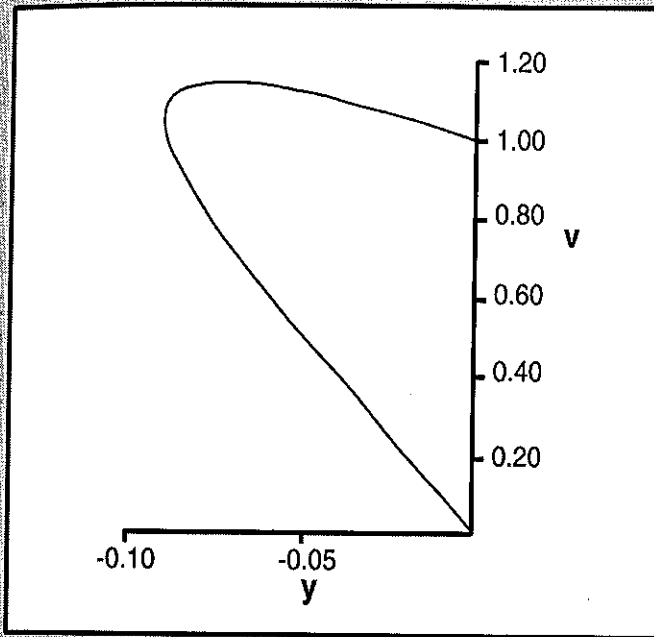


Figure 3.1

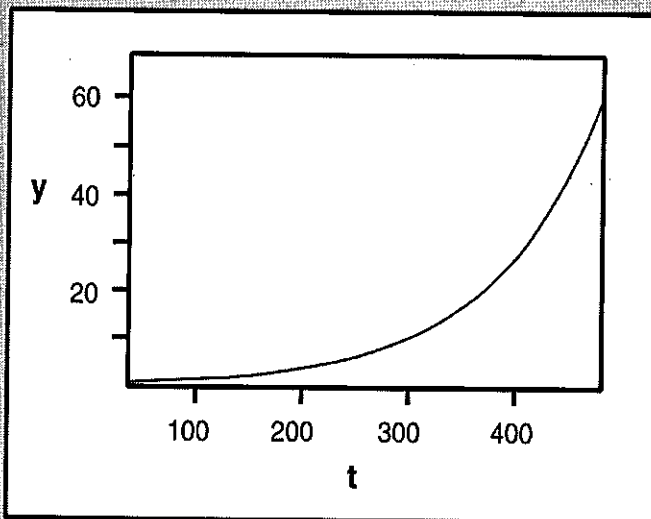


Figure 3.2

Figure 3.1 demonstrates the convergent trajectory in phase space of a particle when $\alpha = \beta = 1$ and $\delta = \eta$, where $\varphi = 4$. Both velocity v and y , the difference between the previous best p and the current position x , converge to 0.0. In Figure 3.2, y increases over time, even when the parameters are real and not complex.

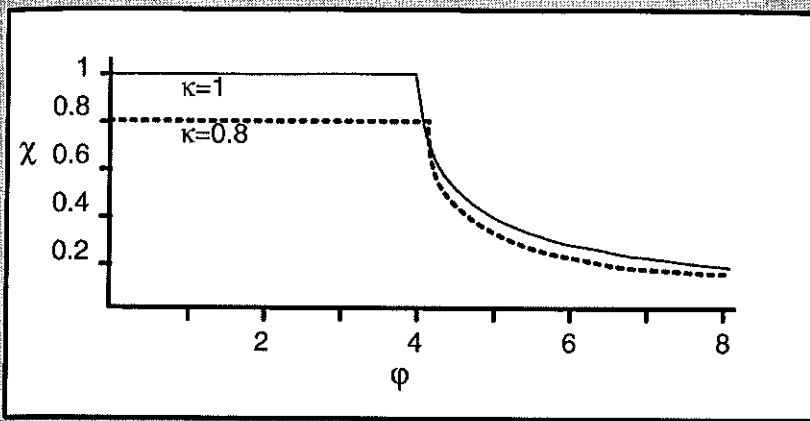


Figure 4.1

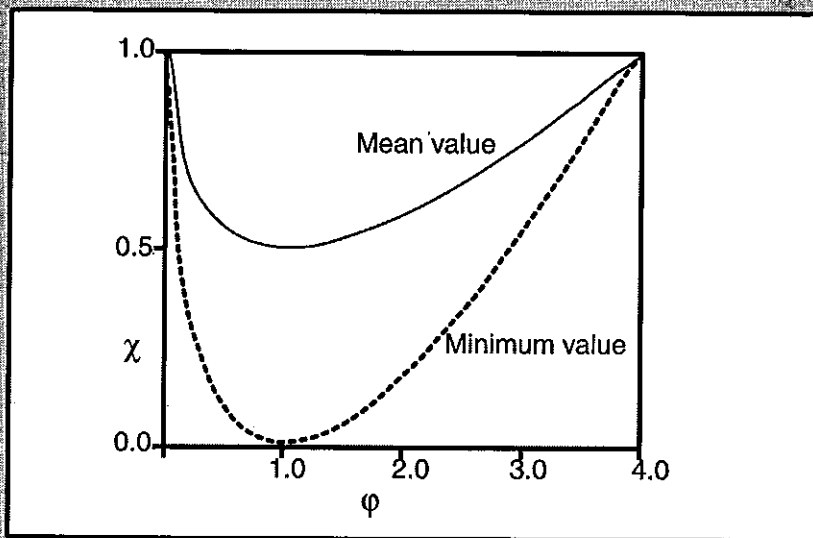


Figure 4.2

Figure 2.5 shows the Type 1 constriction coefficient χ as a function of φ and κ . It drops below κ only when $\varphi > 4.0$. The Type 1' coefficient is less than 1.0 when $\varphi < 4.0$. These coefficients identify the conditions for convergence of the particle system.

Chapter 3

Convergence Behavior

3.1 PREMATURE CONVERGENCE PROBLEM

PSO suffer from premature convergence which occurs when some poor individuals attract the population- due to a local optimum- preventing further exploration of search space. In this population converges before reaching the global optimal solution.

3.2 Causes of Premature Convergence

- For the gbest PSO, particles converge to a single point, which is on the line between the global best and personal best positions. This point is not guaranteed to be even a local optimum.
- Another reason for this problem is the fast rate of information flow between particles, resulting in creation of similar particles (with a loss in diversity) which increases the probability of being trapped in local optima (Riget and Vesterstrom 2002).
- Particle swarm optimization (PSO) is a good optimization algorithm, but it always premature convergence to local optimization, especially in some complex issues like optimization of high-dimensional function. When the sign of premature convergence is arise, search each small area which is defined of all particles by chaotic search, then jump out of local optimization, and avoid premature convergence.
- problem of premature convergence due to the lack of diversity in Estimation of Distributions Algorithms. This problem is quite important for these kind of algorithms since, even when using very complex probabilistic models, they can not solve certain optimization problems such as some deceptive, hierarchical or multimodal ones. There are several which propose different techniques to deal with premature convergence. In most cases, they arise as an adaptation of the techniques used with genetic algorithms, and use randomness to generate individuals. We study a new scheme which tries to preserve the population diversity. Instead of generating individuals randomly, it uses the information

contained in the probability distribution learned from the population. In particular, a new probability distribution is obtained as a variation of the learned one so as to generate individuals with less probability to appear on the evolutionary process. This proposal has been validated experimentally with success with a set of different test functions.

3.3 Tackling Premature convergence

- *Constriction Factor*
- Clerc and Kennedy(2001) proposed using a constriction factor to ensure convergence. The modified velocity update equation is defined as follows:
- $$V_{ij}(t+1) = \chi(V_{ij}(t) + C_1 R_{1,j}(t)(Y_{ij}(t) - X_{ij}(t)) + C_2 R_{2,j}(t))$$
- where χ is the constriction factor

Guaranteed Convergence PSO

- A new version of PSO with guaranteed local convergence was introduced by Van den Bergh(2002), namely GCPSO. In GCPSO, the rate of convergence is faster than PSO, hence it is more likely to be trapped in local optima. However, it has guaranteed local convergence whereas the original PSO does not.

Multi-start PSO(MPSO)

- It is an extension to GCPSO in order to make it a global search algorithm. It works as follows:-
- Randomly initialize all the particles in the swarm.
- Apply the GCPSO until convergence to a local optimum. Save the position of this local optimum.
- Repeat steps 1 and 2 until some stopping criteria are satisfied.

Craziness operator

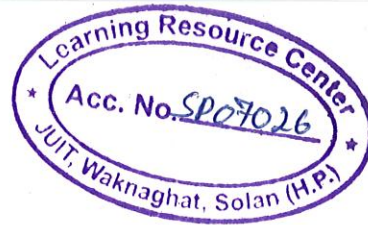
- To avoid premature convergence, Kennedy and Eberhart(1995) introduced the use of a craziness operator with PSO. In each iteration, a few particles far from the center of the swarm are selected.
- According to Venter(2002), the proposed craziness operator does not seem to have a big influence on the performance of PSO.

Self-Organized Critically(SOCPSO)

- In order to increase the population diversity to avoid premature convergence, Loveberg and Krink(2002) extended PSO with Self Organised Critically(SOC). A measure, called critically, of how close particles are to each other is used to relocate the particles and thus increase the diversity of the swarm.

Chapter 4

Implementation



4.1 Applications of PSO and Current Trends :

The first practical application of PSO was in the field of neural network training and was reported together with the algorithm itself (Kennedy and Eberhart 1995). Many more areas of application have been explored ever since, including telecommunications, control, data mining, design, combinatorial optimization, power systems, signal processing, and many others. To date, there are hundreds of publications reporting applications of particle swarm optimization algorithms. For a review, see (Poli 2008). Although PSO has been used mainly to solve unconstrained, single-objective optimization problems, PSO algorithms have been developed to solve constrained problems, multi-objective optimization problems, problems with dynamically changing landscapes, and to find multiple solutions. For a review, see (Engelbrecht 2005).

A number of research directions are currently pursued, including:

- Theoretical aspects
- Matching algorithms (or algorithmic components) to problems
- Application to more and/or different class of problems (e.g., multiobjective)
- Parameter selection
- Comparisons between PSO variants and other algorithms
- New variants

Particle swarm optimization can be and has been used across a wide range of applications.

Areas where PSOs have shown particular promise include multimodal problems and problems for which there is no specialized method available or all specialized methods give unsatisfactory results.

CHAPTER 5

Conclusion

In summary, the parallel Particle Swarm Optimization algorithm presented in this study exhibits excellent parallel performance as long as individual fitness evaluations require the same amount of time. For optimization problems where the time required for each fitness evaluation varies substantially,

an asynchronous implementation may be needed to reduce wasted CPU cycles and maintain high parallel efficiency. When large numbers of processors are available, use of larger population sizes may result in improved convergence rates to the global solution.

An adaptive PSO algorithm that increases population size incrementally may also improve algorithm convergence characteristics

Appendix

```
#define function 700 /* extraction */

#define popsize 100 /* set number of particles in the population */

#define maxgen 100 /* set maximum number of generations */

#define optimization 1 /* set optimization type, 0 for min, 1 for max */

#define archive_size 500 /* set capacity of archive */

#define maxfun 1 /* set maximum number of objective functions

#define maxvar 4 /* set maximum number of variables */

#define verbose 1 /* verbosity level 0,1 */

#define printevery 1 /* how frequently should output be generated */

double PI;

void initialize_rand(void);

void initialize_pop(void);

void initialize_vel(void);

void evaluate(void);

double kita_f1(unsigned int);

double kita_f2(unsigned int);

double kursawe_f1(unsigned int);

double kursawe_f2(unsigned int);

double deb_f1(unsigned int);
```

```
double deb_f2(unsigned int);

void store_pbests(void);

void insert_nondom(void);

void delete_particle(unsigned int);

unsigned int compare_particles(unsigned int, unsigned int);

unsigned int check_constraints(double *);

void crowding(void);

void qsortFitness(unsigned int, unsigned int, unsigned int);

void qsortCrowd(unsigned int, unsigned int);

void compute_distance(unsigned int);

void mutate(unsigned int);

void get_ranges(double *, double *);

void maintain_particles(void);

void compute_velocity(void);

void update_archive(void);

unsigned int check_nondom(unsigned int);

void update_pbests(void);

void save_results(char *);

double DTLZ6_f1(unsigned int);

double DTLZ6_f2(unsigned int);
```

```
double DTLZ6_f3(unsigned int);

double ecm_f1(unsigned int);

double ecm_f2(unsigned int);

double extraction(unsigned int);

double archiveVar[archive_size][maxvar]; /* variable values of particles in the archive */
double archiveFit[archive_size][maxfun]; /* fitness values of particles in the archive */

double popVar[popsize][maxvar]; /* variable values of particles in the population
*/
double popFit[popsize][maxfun]; /* fitness values of particles in the
population */

double pbestsVar[popsize][maxvar]; /* personal bests of particles in the population */
double pbestsFit[popsize][maxfun]; /* personal bests of particles in the population */

double velocity[popsize][maxvar]; /* velocity of particles in the population */

double crowdDist[archive_size]; /* crowding distance values of particles in
archive */

double pMut=0.5; /* probability of mutation */

unsigned int nondomCtr = 0; /* number of nondominated solutions in archive
*/

unsigned int t;

#include <stdlib.h>

#include <stdio.h>
```

```
#include <string.h>

#include <math.h>

#include <time.h>

#define FALSE 0

#define TRUE 1

double u[97],c,cd,cm;

int i97,j97;

int test = FALSE;

void RandomInitialise(int ij,int kl)

{

    double s,t;

    int ii,i,j,k,l,jj,m;

    /*

        Handle the seed range errors

        First random number seed must be between 0 and 31328

        Second seed must have a value between 0 and 30081

    */

    if (ij < 0 || ij > 31328 || kl < 0 || kl > 30081) {

        ij = 1802;
```

```
kl = 9373;

}

i = (ij / 177) % 177 + 2;

j = (ij % 177) + 2;

k = (kl / 169) % 178 + 1;

l = (kl % 169);

for (ii=0; ii<97; ii++) {

    s = 0.0;

    t = 0.5;

    for (jj=0; jj<24; jj++) {

        m = (((i * j) % 179) * k) % 179;

        i = j;

        j = k;

        k = m;

        l = (53 * l + 1) % 169;

        if (((l * m % 64)) >= 32)

            s += t;

        t *= 0.5;

    }

    u[ii] = s;
```



```
}

c = 362436.0 / 16777216.0;

cd = 7654321.0 / 16777216.0;

cm = 16777213.0 / 16777216.0;

i97 = 97;

j97 = 33;

test = TRUE;

}

double RandomUniform(void)

{

double uni;

/* Make sure the initialisation routine has been called */

if (!test)

    RandomInitialise(1802,9373);

uni = u[i97-1] - u[j97-1];

if (uni <= 0.0)

    uni++;

u[i97-1] = uni;

i97--;

if (i97 == 0)
```

```

    i97 = 97;

    j97--;

    if (j97 == 0)

        j97 = 97;

    c -= cd;

    if (c < 0.0)

        c += cm;

    uni -= c;

    if (uni < 0.0)

        uni++;

    return(uni);
}

double RandomGaussian(double mean,double stddev)
{
    double q,u,v,x,y;

    /*
    Generate P = (u,v) uniform in rect. enclosing acceptance region

    Make sure that any random numbers <= 0 are rejected, since
    gaussian() requires uniforms > 0, but RandomUniform() delivers >= 0.

    */

```

```

do {

    u = RandomUniform();

    v = RandomUniform();

    if (u <= 0.0 || v <= 0.0) {

        u = 1.0;

        v = 1.0;

    }

    v = 1.7156 * (v - 0.5);

    /* Evaluate the quadratic form */

    x = u - 0.449871;

    y = fabs(v) + 0.386595;

    q = x * x + y * (0.19600 * y - 0.25472 * x);

    /* Accept P if inside inner ellipse */

    if (q < 0.27597)

        break;

    /* Reject P if outside outer ellipse, or outside acceptance region */

} while ((q > 0.27846) || (v * v > -4.0 * log(u) * u * u));

/* Return ratio of P's coordinates as the normal deviate */

return (mean + stddev * v / u);

}

```

```
/*  
Return random integer within a range, lower -> upper INCLUSIVE
```

```
*/
```

```
int RandomInt(int lower, int upper)
```

```
{
```

```
return((int)(RandomUniform() * (upper - lower + 1)) + lower);
```

```
}
```

```
/*
```

```
Return random float within a range, lower -> upper
```

```
*/
```

```
double RandomDouble(double lower, double upper)
```

```
{
```

```
return((upper - lower) * RandomUniform() + lower);
```

```
}
```

```
int flip(double pf){
```

```
if(RandomDouble(0.0,1.0)<=pf)return 1;else return 0;
```

```
}
```

```
/****** Objective functions for test problems *****/
```

```
double kita_fl(unsigned int i)
```

```
{ return ( -(popVar[i][0] * popVar[i][0]) + popVar[i][1] );
```

```

}

double kita_f2(unsigned int i)

{
    return ( (popVar[i][0] / 2.0) + popVar[i][1] + 1 );
}

double kursawe_f1(unsigned int i)

{
    double r = 0.0;

    unsigned int j;

    for(j = 0; j < 2; j++)

        r += -10.0 * exp(-0.2 *
sqrt(pow(popVar[i][j], 2) + pow(popVar[i][j + 1], 2)) );

    return r;
}

double kursawe_f2(unsigned int i)

{
    double r = 0.0;

    unsigned int j;

    for(j = 0; j < 3; j++)

r += pow(fabs(popVar[i][j]), 0.8) + 5.0 * sin(pow(popVar[i][j], 3));

    return r;
}

double deb_f1(unsigned int i)

```

```

        {

return (popVar[i][0]);

        }

double deb_f2(unsigned int i double g = 2.0 - exp(-pow(((popVar[i][1]-0.2)/0.004),2)) -
0.8 * exp(-pow(((popVar[i][1]-0.6)/0.4),2)));

return ((double)g / popVar[i][0]);

        }

double DTLZ6_f1(unsigned int i)

        {

return (popVar[i][0]);

        }

double DTLZ6_f2(unsigned int i)

        {

return (popVar[i][1]);

        }

double DTLZ6_f3(unsigned int i)

        {

unsigned int j = 0;

unsigned int n = maxvar;

unsigned int k = n - maxfun + 1;

double g, h, s, t;

```

```

s = 0;

for (j = 2; j < maxvar; j++){

s += popVar[i][j] }

g = 1 + (9 / 20) * s;

t = 0;

for (j = 0; j < maxfun-1; j++){

t += (popVar[i][j] * (1 + sin(3 * PI * popVar[i][j]))) / (1 + g);

}

h = 3 - t

return ((1 + g) * h);

}

double ecm_f1(unsigned int i)

{

double Ra;

Ra = 83.8335 - 0.29382*popVar[i][0] - 0.940990*popVar[i][1] - 6.85188*popVar[i][2] -
18.8134*popVar[i][3] + 0.000439621*popVar[i][0]*popVar[i][0] +
0.00802176*popVar[i][1]*popVar[i][1] + 0.290848*popVar[i][2]*popVar[i][2] -
3.41518*popVar[i][3]*popVar[i][3] + 0.000273438*popVar[i][0]*popVar[i][1] +
0.00690625*popVar[i][0]*popVar[i][2] + 0.0459375*popVar[i][0]*popVar[i][3] +
0.0432813*popVar[i][1]*popVar[i][2] + 0.254688*popVar[i][1]*popVar[i][3] -
0.0687500*popVar[i][2]*popVar[i][3];

return (Ra);

}

```

```
double ecm_f2(unsig)
```

```
{
```

```
double MRR;
```

```
MRR = -2.45705 + 0.0178595*popVar[i][0] + 0.0130625*popVar[i][1] +  
0.0929583*popVar[i][2] + 0.38381*popVar[i][3] -  
0.000037953*popVar[i][0]*popVar[i][0] - 0.000245722*popVar[i][1]*popVar[i][1] -  
0.00543155*popVar[i][2]*popVar[i][2] - 0.793155*popVar[i][3]*popVar[i][3] +  
0.000017187*popVar[i][0]*popVar[i][1] + 0.00005*popVar[i][0]*popVar[i][2] -  
0.0005625*popVar[i][0]*popVar[i][3] - 0.0003125*popVar[i][1]*popVar[i][2] +  
0.0034375*popVar[i][1]*popVar[i][3] +0.01*popVar[i][2]*popVar[i][3];
```

```
return (MRR);
```

```
}
```

```
double extraction(unsigned int i)
```

```
{
```

```
double la;
```

```
la = -842.551 + 31.6692*popVar[i][0] + 103.673*popVar[i][1] + 75.2450*popVar[i][2] +  
41.1417*popVar[i][3] - 0.508562*popVar[i][0]*popVar[i][0] -  
35.9412*popVar[i][1]*popVar[i][1] - 13.0713*popVar[i][2]*popVar[i][2] -  
6.66031*popVar[i][3]*popVar[i][3] - 0.38125*popVar[i][0]*popVar[i][1] +  
1.10875*popVar[i][0]*popVar[i][2] - 0.156125*popVar[i][0]*popVar[i][3] +  
0.6875*popVar[i][1]*popVar[i][2] + 2.93375*popVar[i][1]*popVar[i][3] +  
4.61875*popVar[i][2]*popVar[i][3];
```

```
return la;
```

```
}
```

```
int main(int argc, char *argv[]
```



```
{  
  
char archiveName[20];  
  
unsigned int i, j;  
  
clock_t startTime, endTime;  
  
double duration, clocktime;  
  
FILE *outfile,*plotfile;  
  
PI = 4.0*atan(1.0);  
  
sprintf(archiveName,"archive.out");  
  
outfile = fopen("output.out","w");  
  
plotfile = fopen("plot.out","w");  
  
/* Initialize random number generator */  
  
initialize_rand();  
  
startTime = clock();  
  
/* Initialize generation counter */  
  
t=0;  
  
/* Initialize population with random values */  
  
initialize_pop();  
  
/* Initialize velocity */  
  
initialize_vel();  
  
/* Evaluate particles in population */
```

```
evaluate();

/* Store initial personal bests (both variable and fitness values) of particles */

store_pbests();

/* Insert nondominated particles in population into the archive */

insert_nondom();

        /****** MAIN LOOP *****/

while(t <= maxgen)

{

clocktime = (clock() - startTime)/(double)CLOCKS_PER_SEC;

    if(verbose > 0 && t%printevery==0 || t == maxgen) {

        fprintf(stdout,"Generation %d Time: %.2f sec\n",t,clocktime);

        fflush(stdout);

    }

    if(t%printevery==0 || t == maxgen) {

        fprintf(outfile,"Generation %d Time: %.2f sec\n",t,clocktime);

    }

for(i=0; i <= archive_size; i++){

crowdDist[i] = i*rand();

    crowdDist[i]/=10;

}
```

```
/* Compute crowding distance of each particle in the archive */

/* Only when there are at least 3 particles in the archive */

if(nondomCtr > 2)

    crowding();

/* Compute new velocity of each particle in the population */

compute_velocity();

/* Maintain particles in the population within the search space */

maintain_particles();

/* Mutate particles in the population */

if(t < maxgen * pMut)

    mutate(t);

/* Evaluate particles in the population */

evaluate();

/* Insert new nondominated particles in pop into archive */

update_archive();

/* Update personal bests of particles in the population */

update_pbests();

/* Print Best So Far */

if(t%printevery==0 || t == maxgen) {

    fprintf(outfile, "Size of Pareto Set: %d\n", nondomCtr);
```

```
    for(i = 0; i < nondomCtr; i++) {  
  
        fprintf(outfile, "Function Values: ");  
  
        for(j = 0; j < maxfun; j++)  
  
            fprintf(outfile, "%.6f ", archiveFit[i][j]);  
  
        fprintf(outfile, "\n");  
  
        for(j = 0; j < maxvar; j++)  
  
            fprintf(outfile, "%.6f ", archiveVar[i][j]);  
  
        fprintf(outfile, "\n");  
  
    }  
  
    fprintf(outfile, "\n\n");  
  
    fflush(outfile);  
  
}  
  
if(t == maxgen) {  
  
    fprintf(plotfile, "# GNU Plot\n");  
  
    for(i = 0; i < nondomCtr; i++) {  
  
        for(j = 0; j < maxfun; j++)  
  
            fprintf(plotfile, "%.4f ", archiveFit[i][j]);  
  
        fprintf(plotfile, "\n");  
  
    }  
  
    fflush(plotfile);  
  
}
```

```
    }

    /* Increment generation counter */

    t++; }

/* Write results to file */

save_results(archiveName);

/* Compute time duration */

endTime = clock();

duration = ( endTime - startTime ) / (double)CLOCKS_PER_SEC;

fprintf(stdout, "%lf sec\n", duration);

fclose(outfile);

fclose(plotfile);

return EXIT_SUCCESS;

}

void initialize_rand() /* Initialize random number generator from randomlib.h */

{ unsigned int i, j;

srand((unsigned int)time((time_t *)NULL));

i = (unsigned int) (31329.0 * rand() / (RAND_MAX + 1.0));

j = (unsigned int) (30082.0 * rand() / (RAND_MAX + 1.0));

RandomInitialise(i,j);

}
```

```
void initialize_pop() /* Initialize population variables */  
  
{  
  
    unsigned int i, j;  
  
    switch(function){  
  
        case 100: /* Kita test function*/  
  
            for(i=0; i < popsize; i++)  
  
                for(j =0; j < maxvar; j++)  
  
                    popVar[i][j] = RandomDouble(0.0, 7.0);  
  
            break;  
  
        case 200: /* Kursawe test function */  
  
            for(i=0; i < popsize; i++)  
  
                for(j =0; j < maxvar; j++)  
  
                    popVar[i][j] = RandomDouble(-5.0, 5.0);  
  
            break;  
  
        case 300: /* Deb test function */  
  
            for(i=0; i < popsize; i++)  
  
                for(j =0; j < maxvar; j++)  
  
                    popVar[i][j] = RandomDouble(0.1, 1.0);//0.8191);  
  
            break;  
  
        case 500: /* DTLZ6 test function */
```

```
for(i=0; i < popsize; i++)

    for(j =0; j < maxvar; j++)

        popVar[i][j] = RandomDouble(0.0, 1.0);

break;

case 900: /*ECM */

    for(i=0; i < popsize; i++)

        for(j =0; j < maxvar; j++)

            {

                if(j==0)

                    popVar[i][j] = RandomDouble(200.0, 280.0);

                if(j==1)

                    popVar[i][j] = RandomDouble(20.0, 36.0);

                if(j==2)

                    popVar[i][j] = RandomDouble(5.0, 9.0);

                if(j==3)

                    popVar[i][j] = RandomDouble(0.1, 0.5);

                if(j==4)

                    popVar[i][j] = RandomD
```

```
ouble(10.0, 20.0);

    }

    case 700: /*extraction */

    for(i=0; i < popsize; i++)

    for(j =0; j < maxvar; j++)

    {

        if(j==0)

            popVar[i][j] = RandomDouble(25.0, 45.0);

        if(j==1)

            popVar[i][j] = RandomDouble(0.5, 2.5);

        if(j==2)

            popVar[i][j] = RandomDouble(4.5, 6.5);

        if(j==3)

            popVar[i][j] = RandomDouble(3.0, 7.0);

        if(j==4)

            popVar[i][j] = RandomDouble(10.0, 20.0);

    }

    break;

/* Insert particle in pop if it is feasible and nondominated */
```



```

if(insertFlag == 1){

/* If memory is not yet full, insert particle */

    if (nondomCtr < archive_size) {

for(j = 0; j < maxvar; j++)

        archiveVar[nondomCtr][j] = popVar[i][j];

        for(j = 0; j < maxfun; j++)

            archiveFit[nondomCtr][j] = popFit[i][j];

        nondomCtr += 1;

    } else{ /* If memory is full, select particle to replace */
        /* Compute crowding distance values of particles in archive*/

crowding();

bottom = (unsigned int)((nondomCtr-1) * 0.90);

/* Randomly select which particle from the most areas to replace */

k = RandomInt(bottom, nondomCtr-1);

/* Insert new particle into archive */

for(j = 0; j < maxvar; j++)

    archiveVar[k][j] = popVar[i][j];

for(j = 0; j < maxfun; j++)

    archiveFit[k][j] = popFit[i][j];

    }

}

```

```

} /* Finished comparing particles in pop with particles in archive */

}

void delete_particle(unsigned int k) /* Delete particle in archive */

{

    unsigned int j;

    if( (nondomCtr == 1) || (k == (nondomCtr-1)) ) {

        nondomCtr -= 1;

    } else {

        for(j = 0; j < maxvar; j++)

            archiveVar[k][j] = archiveVar[nondomCtr-1][j];

        for(j = 0; j < maxfun; j++)

            archiveFit[k][j] = archiveFit[nondomCtr-1][j];

        nondomCtr -= 1;

    }

}

unsigned int check_constraints(double *consVar) /* Check for constraint to determine
feasibility */

{

    unsigned int violations = 0;

    switch(function){

        else if(sum == 0){ /* If particle in archive is dominated, delete it */

```

```
for(j = 0; j < maxvar; j++)

    archiveVar[h][j] = archiveVar[nondomCtr-1][j];

        for(j = 0; j < maxfun; j++)

            archiveFit[h][j] = archiveFit[nondomCtr-1][j];
nondomCtr -= 1;

}

}

case 700: /*Extraction*/

for(i = 0; i < maxvar; i++)

{

    if(i==0)

    {

        minvalue[i] = 25.0;

        maxvalue[i] = 45.0;

    }

    if(i==1)

    {

        minvalue[i] = 0.5;

        maxvalue[i] = 2.5;

    }

}
```

```
if(i==2)

{

    minvalue[i] = 4.5;

    maxvalue[i] = 6.5;

}

if(i==3)

{

    minvalue[i] = 3.0;

    maxvalue[i] = 7.0;

}

}

break;

}

}

void maintain_particles() /* Maintain particles in the population within the search space
*/

{

    unsigned int i, j;

    double minvalue[maxvar], maxvalue[maxvar];

    return; /* This statement checks the stability of the swarm by allowing particles to fly
past their boundaries */
```

```
/* Stable swarms do not explode even without boundary constraints. */
```

```
/* Comment this out if you are not happy with this type of checking */
```

```
switch (function){
```

```
case 100: /* Kita test function */
```

```
for(i = 0; i < maxvar; i++){
```

```
    minvalue[i] = 0.0;
```

```
    maxvalue[i] = 7.0;
```

```
}
```

```
break;
```

```
case 200: /* Kursawe test function */
```

```
for(i = 0; i < maxvar; i++) {
```

```
    minvalue[i] = -5.0;
```

```
    maxvalue[i] = 5.0;
```

```
}
```

```
break;
```

```
case 300: /* Deb test function */
```

```
for(i = 0; i < maxvar; i++){
```

```
    minvalue[i] = 0.1;
```

```
    maxvalue[i] = 1.0//0.8191;
```

```
}  
  
break;  
  
case 500: /* DTLZ6 test function */  
  
for(i = 0; i < maxvar; i++){  
  
    minvalue[i] = 0.0;  
  
    maxvalue[i] = 1.0;  
  
}  
  
break;  
  
case 900: /*ECM*/  
  
for(i = 0; i < maxvar; i++){  
  
    if(i==0)  
  
    {  
  
minvalue[i] = 200.0;  
  
    maxvalue[i] = 280.0;  
  
    }  
  
    if(i==1)  
  
    {  
  
minvalue[i] = 20.0;  
  
    maxvalue[i] = 36.0;  
  
    }  
  
}
```

```
if(i==2)

{

    minvalue[i] = 5.0;

    maxvalue[i] = 9.0;

}

if(i==3)

{

    minvalue[i] = 0.1;

    maxvalue[i] = 0.5;

}

}

case 700: /*Extraction*/

for(i = 0; i < maxvar; i++)

{

    if(i==0)

    {

        minvalue[i] = 25.0;

        maxvalue[i] = 45.0;

    }

    if(i==1)
```

```
{  
  
    minvalue[i] = 0.5;  
  
    maxvalue[i] = 2.5;  
  
}  
  
if(i==2)  
  
    {  
  
        minvalue[i] = 4.5;  
  
        maxvalue[i] = 6.5;  
  
    }  
  
if(i==3)  
  
    {  
  
        minvalue[i] = 3.0;  
  
        maxvalue[i] = 7.0;  
  
    }  
  
    }  
  
break;  
  
/** Add boundary constraints here! **/  
  
}  
  
for(i = 0; i < popsize; i++) {  
  
    for(j = 0; j < maxvar; j++) {
```



```

/* If particle goes beyond minimum range value */

if(popVar[i][j] < minvalue[j]){

    /* Set it to minimum range value */

    popVar[i][j] = minvalue[j];

    /* Change to opposite direction */

    velocity[i][j] = -velocity[i][j];

}

/* If particle goes beyond maximum range value */

if(popVar[i][j] > maxvalue[j]){

    /* Set it to maximum range value */

    popVar[i][j] = maxvalue[j];

    /* Change to opposite direction */

    velocity[i][j] = -velocity[i][j];

}

}

}

}

void update_pbests() /* Update personal bests of particles in the population */

{

    unsigned int i, j, sum, better;

```

```
for(i = 0; i < popsize; i++) {  
  
    sum = 0;  
  
    for(j = 0; j < maxfun; j++){  
  
        if( ((popFit[i][j] < pbestsFit[i][j]) && (optimization == 0))  
  
            || ((popFit[i][j] > pbestsFit[i][j]) && (optimization == 1)))  
  
            sum += 1;  
  
    }  
  
    if (sum == maxfun) {  
  
        better = 0;  
  
    } else {  
  
        if (sum == 0)  
  
            better = 1;  
  
        else  
  
            better = RandomInt(0, 1);  
  
    }  
  
    if (better == 0){  
  
        for(j = 0; j < maxfun; j++)  
  
            pbestsFit[i][j] = popFit[i][j];  
  
        for(j = 0; j < maxvar; j++)  
  
            pbestsVar[i][j] = popVar[i][j];  
  
    }  
  
}
```

```
    }  
  }  
}  
  
void save_results(char *archiveName) /* Write results to file */  
{  
  unsigned int i, j;  
  
  FILE *fp;  
  
  /* Open file for writing */  
  
  fp = fopen(archiveName, "w");  
  
  for(i = 0; i < nondomCtr; i++) {  
  
    for(j = 0; j < maxfun; j++)  
  
      fprintf(fp, "%.6f ", archiveFit[i][j]);  
  
      fprintf(fp, "\n");  
  
  }  
  
  fclose(fp);  
  
}
```

CHAPTER8

REFERENCES

References :-

- M. Clerc and J. Kennedy. The particle swarm-explosion, stability and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58-73, 2002.
- A. P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. John Wiley & Sons, Chichester, UK, 2005.
- F. Heppner and U. Grenander. A stochastic nonlinear model for coordinated bird flocks. *The Ubiquity of Chaos*. AAAS Publications, Washington, DC, 1990.
- J. Kennedy. Bare bones particle swarms. In *Proceedings of the IEEE Swarm Intelligence Symposium*, pages 80-87, IEEE Press, Piscataway, NJ, 2003.
- J. Kennedy. Swarm Intelligence. In *Handbook of Nature-Inspired and Innovative Computing: Integrating Classical Models with Emerging Technologies*. A. Y. Zomaya (Ed.) , pages 187-219, Springer US, Secaucus, NJ, 2006.
- J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, pages 1942-1948, IEEE Press, Piscataway, NJ, 1995.
- J. Kennedy and R. Eberhart. A discrete binary version of the particle swarm algorithm. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pages 4104-4108, IEEE Press, Piscataway, NJ, 1997.
- J. Kennedy, and R. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, San Francisco, CA, 2001.
- R. Mendes, J. Kennedy, and J. Neves. The fully informed particle swarm: simpler, maybe better. *IEEE Transactions on Evolutionary Computation*, 8(3):204-210, 2004.

- A. Nowak, J. Szamrej, and B. Latane. From private attitude to public opinion: A dynamic theory of social impact. *Psychological Review*, 97(3):362-376, 1990.
- R. Poli. Analysis of the publications on the applications of particle swarm optimisation. *Journal of Artificial Evolution and Applications*, Article ID 685175, 10 pages, 2008.
- R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization. An overview. *Swarm Intelligence*, 1(1):33-57, 2007.
- W. T. Reeves. Particle systems--A technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics*, 2(2):91-108, 1983.
- C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *ACM Computer Graphics*, 21(4):25-34, 1987
- Mark S. Voss, and Xin Feng, "Anna Model Selection Using Particle Swarm Optimization and Aic Criteria", 15th Triennial World Congress, Barcelona, Spain, 2002.
- K. E. Parsopoulos, and M. N. Vrahatis, "Particle Swarm Optimization Method in Multiobjective Problems", Proceedings of the 2002 ACM Symposium on Applied Computing, pp. 603-607, 2002.
- F. Van den Bergh, and A. P. Engelbrecht, "Cooperative Learning in Neural Networks using Particle Swarm Optimizers", South African Computer Journal, Vol 26, pp. 84-90, 2000.
- Shi Y., and Eberhart, R. C., "A Modified Particle Swarm Optimizer", IEEE International Conference of Evolutionary Computation, Anchorage, Alaska, May 1998.
- Yuhui Shi, and Eberhart, R. C., "Fuzzy Adaptive Particle Swarm Optimization", Proceeding of Congress on Evolutionary Computation, Seoul , Korea, pp. 101-106, 2001.

- Keiichiro Yasuda, "Adaptive Particle Swarm Optimization using Velocity Information of Swarm", 2004 IEEE International Conference on Systems, Man and Cybernetics, Tokyo, NJ, pp. 3475-3479.
- Clerc, M., "The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization", Proceedings of the IEEE Congress on Evolutionary Computation, pp. 1951-1957, 1999.
- Cunrui Wang, Xiaodong Duan, and etc., "A Modified Basic Particle Swarm Optimization Algorithm", Computer Engineering, Vol 30, No. 21, pp. 35-37, Nov. 2004.
- J. J. Liang, and P. N. Suganthan, "Adaptive Comprehensive Learning Particle Swarm Optimizer with History Learning", The 6th International Conference on Simulated Evolution and Learning, Hefei, pp. 213-220, October 2006
- Kennedy, J. and Eberhart, R.C., "Particle Swarm Optimization", Proceeding of IEEE International Conference on Neural Networks, Perth, Australia, pp. 1942-1948, 1995.
- Eberhart, R. C., and Kennedy, J., "A new optimizer using particle swarm theory", Proceeding of the Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan, pp. 39-43, 1995.
- Hirotaka, Yoshida, and Kenichi, "A particle Swarm Optimization for Reactive Power and Voltage Control Considering Voltage Stability", Proceeding of IEEE International Conference on Intelligent System Applications to Power Systems, Rio de Janeiro, pp.117-121, April 1999.

Bio Data

Mansi

Department of Computer Science Engineering
Jaypee University of Information Technology,
Waknaghat, District Solan,
Himachal Pradesh-173215
E-mail : mansisingh07@gmail.com
Mobile: +91-9816808052

OBJECTIVE

To work hard towards achieving common goals for the organization and myself, by applying all the knowledge that I have, and at same time imbibing anything new and good that comes my way and also to incorporate my hard and soft skills in serving the organization towards expected goals and targets.

Academics

Standard	College/School	Year	CGPA/Percentage
B.Tech (CSE)	Jaypee University of Information Technology, Solan.	2011	6.8 (74%) (Up till 7 th semester)
12 th (CBSE)	Dewan Public School Meerut	2007	81.8%
10 th (CBSE)	Dayawati Modi Academy -1 Meerut	2005	84.6%

TECHNICAL SKILLS

Language: C, C++

Operating Systems: Windows, Linux

Basic Knowledge of Java, Web Technologies, DBMS, software engineering, Networking, software testing and debugging, network security .

PROJECTS UNDERTAKEN

1. Currently working on the final year project " **Particle swarm optimization and study it's convergence behavior** " .

2 Student registration system

Language Involved : C

Responsibility : Development, Testing and Debugging.

Description of Project : This Project was regarding Student Registration at the start of the Semester. All arrears, credits of subjects according to semester and Stream are taken care of. Got appreciation from HOD of Computer Science.

3. Railway reservation system

Language Involved C++

Role & Responsibility

- Software Development, Testing and Debugging.

Description of Project : This Project was done to implement Object Orientation in Real Life.

All reservations for any Destination was made available at any station and according to age groups and Services, different discounts were given. All Information about trains was present. All this was implemented by creating different classes and very high level of abstraction was there to hide the function definitions and only the functionality was exposed.

PERSONAL ACHIEVEMENTS

- Represented School at state level in Basketball.
- Won many Debates and Extempore Competitions.
- Outstanding academic performance and for being among the top 0.1% of successful candidates and getting a "SCHOLAR BLAZER" in class 10th.
- 7 scholar badges consecutively for 7 years in school.
- Was in anti-ragging squad in college .
- Was in organizing committee in our college fest .for 3 years .

HOBBIES

- Hanging out with friends.
- Travelling
- Swimming
- Playing basketball
- Playing with my pets
- Dancing
- Sketching

PERSONAL DETAILS

Father's Name
Mother's Name

DR.V.S.Singh
Mrs. Vandana Singh

Date of Birth
Permanent Address

13th January 1989
C-1 Damodar colony , Meerut, U.P

KANIKA CHOUDHARY

Department of Information Technology,
Jaypee University of Information Technology,
Waknaghat, Solan (H.P.) - 173215
Cell: +91-8894329294
E-mail: kaniks_choudhary@yahoo.com

OBJECTIVE

To work hard towards achieving common goals for an organization and myself, by applying all the knowledge that I have, and at same time imbibing anything new and good that comes my way.

ACADEMIC QUALIFICATIONS

Pursuing B.Tech. in Computer Science, IV year, Jaypee University of Information Technology, Waknaghat (Solan).

Standard	College/School	Year	CGPA/Percentage
B.Tech (CSE)	JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, SOLAN.	2011	6.4/71% (Up till 7th sem)
12 th (C.B.S.E)	O.P. JINDAL MODERN SCHOOL, HISAR, HARYANA	2007	80.4
10 th (C.B.S.E)	O.P. JINDAL MODERN SCHOOL, HISAR, HARYANA	2005	87.8

1. Currently working on our final year project "Particle Swarm Optimization and Study the Convergence Behavior".
2. Developed website for Online Registration System.
3. Developed a website for Childcare system.

Industrial Training

Organization: DU PONT
Place : Gurgaon
Duration : 01.06.2010 TO 14.07.2010

TECHNICAL SKILLS

Programming languages: C, C++
Software Packages: Macromedia flash player 8.0, Matlab, Turbo c++.

AREAS OF INTEREST

- Computer Networks
- Software testing
- Data mining and information retrieval
- Project Management

PERSONAL ACHIEVEMENTS

- Participated in round square international conference at BCS(2004)
- Participated in state level debate competition and won 2nd prize(2003)
- Active participation in literary works in school and won several awards.
- Was the sports captain of my house (2006-07)
- Awarded certificate of good conduct for academic year 2002-03

HOBBIES

- Swimming
- Cooking
- Travelling

PERSONAL DETAILS

Date of Birth : 22nd december,1988
Father's Name : Mr. Ishwar singh
Mother's Name : Mrs. Darshna