VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belgaum-590014, Karnataka, INDIA

PROJECT REPORT
On

"DISCRETE OPTIMIZATION USING NATURE INSPIRED ALGORITHMS"

Submitted in partial fulfillment of the requirements for the VIII Semester

Bachelor of Engineering
In
ELECTRONICS AND COMMUNICATION ENGINEERING

For the Academic Year
2015-2016

BY

ARUN DUGGANI	**1PE12EC022**
KATARI ROHITH SAI	**1PE12EC065**
LAKSHMI R	**1PE13EC411**

UNDER THE GUIDANCE OF
Mr. AVISHEK GHOSH
Assistant Professor
Dept. of ECE, PESIT (BSC).

Department of Electronics and Communication Engineering
PESIT - Bangalore South Campus
Hosur Road, Bangalore-560100

PESIT - Bangalore South Campus

Hosur Road, Bangalore-560100
Department of Electronics and Communication Engineering

CERTIFICATE

This is to certify that the project work entitled "DISCRETE OPTIMIZATION USING NATURE INSPIRED ALGORITHMS" carried out by Arun Duggani, Katari Rohith Sai , Lakshmi R ,bearing USNs 1PE12EC022, 1PE12EC065, 1PE13EC411, respectively in partial fulfillment for the award of Degree of Bachelors(Bachelors of Engineering)in Electronics and communication Engineering of Visvesvaraya Technological University, Belgaum during the year 2015-2016.

It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the Report. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for said degree.

_____	_____	_____
Signature of guide **Mr. Avishek Ghosh** **Assistant Professor** **Dept. of ECE**	**Signature of HOD** **Dr. Subhash Kulkarni** **Head Of Dept.** **Dept. of ECE**	**Signature of the Principal** **Dr. J Surya Prasad** **Principal / Director** **PESIT- BSC**

External Viva

Name of the Examiners	**Signature with date**
1	_____
2.	_____

ABSTRACT

Nature has the ability to solve very complex problems in its own distinctive way. The problems around us are becoming more and more complex in the real time and at the same instance our mother nature is guiding us to solve these natural problems. Nature gives some of the logical and effective ways to find solution to these problems. Nature acts as an optimizer for solving the complex problems. The algorithms imitate the processes running in nature and hence are called as "***Nature Inspired Algorithms***". These algorithms have proved to be very efficient and thus have become popular tools for solving real-world problems. To optimize(maximize or minimize) such problems, optimization tools have to be used, though there is no guarantee that the optimal solution can be obtained. Hence some new algorithms have been designed and two such algorithms which have a high efficiency are Genetic Algorithm and Particle Swarm Optimization.

ACKNOWLEGEMENT

We would like express our sincere gratitude to all the lecturers and staff of the department of Electronics and Communication Engineering for extending their help and guidance towards our project.

We would like to thank the college management and express our sincere gratitude to **Dr. J. Surya Prasad,** Director/Principal of PESIT(BSC) have given me the opportunity for the completion of this project.

We would like to thank **Dr. Subhash Kulkarni,** Head of Department Electronics and Communication, PESIT(BSC) for giving us the support and encouragement that was necessary for the completion of this report.

We convey our thanks to the project coordinator **Mr. Pattabhi Raman,** Associate Professor for providing us all the needful information and facilities which was of a great help to complete this project successfully.

We would like to thank our project guide **Mr. Avishek Ghosh,** Assistant Professor for providing us required assistance, encouragement and constant support which helped us a lot.

Last but not least; the project would not have been a success without the support of our parents and friends.

CONTENTS

LIST OF FIGURES

CHAPTER 1

CHAPTER 1

INTRODUCTION

Real-world optimization problems are often very challenging to solve, as a matter of fact for many problems there are no definite algorithms to solve the problems. Hence, many problems are solved by trial and error method which consumes a lot of time. As a result, we have devised algorithms from the nature which help to optimize these problems efficiently.

1.1 NATURE INSPIRED ALGORITHMS

Real-world optimization problems are often very challenging to solve. Nature has inspired many researchers in many ways and thus is a rich source of inspiration. Nowadays, most new algorithms are nature-inspired, because they have been developed by drawing inspiration from nature. In the most generic term, the main source of inspiration is Nature.

Therefore, almost all new algorithms can be referred to as nature-inspired. By far the majority of nature-inspired algorithms are based on some successful characteristics of biological system. Therefore, the largest fraction of nature-inspired algorithms are bio-inspired for short.

Among bio-inspired algorithms, a special class of algorithms have been developed by drawing inspiration from swarm intelligence. Therefore, some of the bio inspired algorithms can be called swarm-intelligence based. In fact, algorithms based on swarm intelligence are among the most popular. Good examples are ant colony optimization , particle swarm optimization etc.

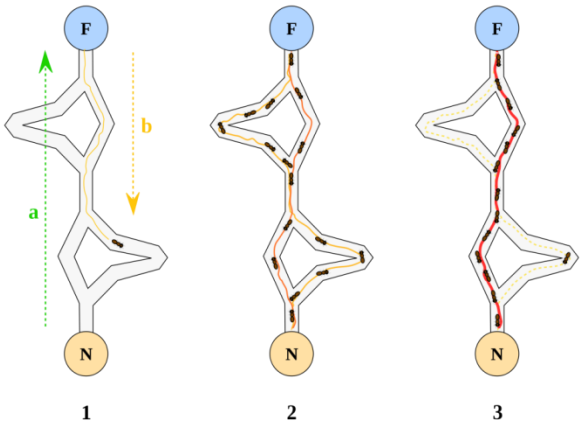

Figure 1.1 : Ant Colony Optimization

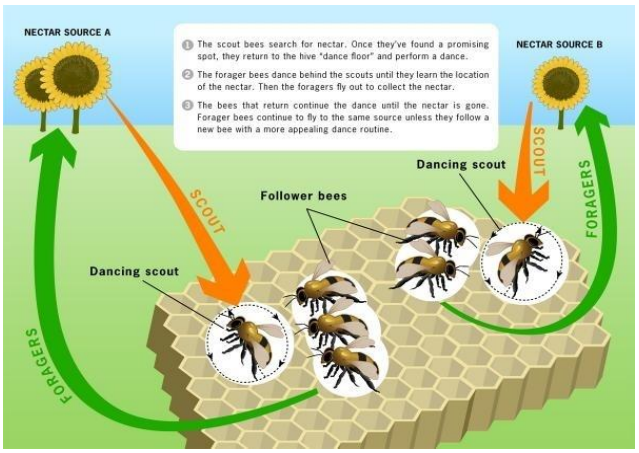

Figure 1.2: Bee Colony Optimization

Nature has the ability to solve very complex problems in its own distinctive way. The problems around us are becoming more and more complex in the real time and at the same instance our mother nature is guiding us to solve these natural problems. Nature gives some of the logical and effective ways to find solution to these problems. Nature acts as an optimizer for solving the complex problems.

Nature has four powerful features which are basic building blocks are ***self optimization, self healing, self learning and self processing*.** Nature as Self optimizer is that it can automatically manage its resources in an efficient manner to meet enterprise need. Nature as self healer is as the components of nature on seeing any problem finds a solution and come out of it. Self learning and self processing are two related terms. They go hand in hand and moved together. Nature and its components self processes the changing conditions in the environment learn from the past and present conditions to evolve in the changed environment in natural evolution.

As the individuals of nature have the capability to evolve according to the changing environment so in present scenario it is indeed required that computers and their intelligence to learn and involve as per changing conditions and solve highly complex problems as nature does. To fulfill this desire, we want our algorithms to adopt the techniques and features from nature and become more effective and efficient too.

There are two most important algorithms which we use in our project namely

1. Genetic Algorithm
2. Particle Swarm optimization

CHAPTER 2

CHAPTER 2
GENETIC ALGORITHM

2.1 EVOLUTIONARY ALGORITHMS

Over many stages of life, biological organisms develop according to the principles of natural selection like "Survival Of The Fittest" to attain some astounding form of accomplishment. The best example of natural evolution can be seen in generation of human beings. So if it works so admirably in nature, it should be interesting to imitate natural evolution and try to procure a technique which may solve existing search and optimization problems.

An EA uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection. Candidate solutions to the optimization problem play the role of individuals in a population, and the fitness function determines the quality of the solutions (see also loss function). Evolution of the population then takes place after the repeated application of the above operators. Evolutionary algorithms often perform well approximating solutions to all types of problems because they ideally do not make any assumption about the underlying fitness landscape.

In most real applications of EAs, computational complexity is a prohibiting factor. In fact, this computational complexity is due to fitness function evaluation. Fitness approximation is one of the solutions to overcome this difficulty. However, seemingly simple EA can solve often complex problems; therefore, there may be no direct link between algorithm complexity and problem complexity.

Different types of Evolutionary Algorithm are:

1. Genetic algorithm – This is the most popular type of EA. One seeks the solution of a problem in the form of strings of numbers (traditionally binary, although the best representations are usually those that reflect something about the problem being solved), by applying operators such as recombination and mutation (sometimes one, sometimes both). This type of EA is often used in optimization problems.

2. Genetic programming – Here the solutions are in the form of computer programs, and their fitness is determined by their ability to solve a computational problem.

3. Evolutionary programming – Similar to genetic programming, but the structure of the program is fixed and its numerical parameters are allowed to evolve.

4. Evolution strategy – Works with vectors of real numbers as representations of solutions, and typically uses self-adaptive mutation rates.

2.2 INTRODUCTION TO GENETIC ALGORITHM

Genetic algorithms were invented by Holland in the 60's and developed later in the 70's. This method was defined as a way to move from one population of chromosomes to another by utilizing natural selection and the operator of crossover, mutation, and inversion.

Nature provides us different things. When we try to think about nature, there are complex mechanisms, operations involved in it. There are many scientific theories proposed on specific part of nature. One among the successful theory is DARWIN'S theory of EVOLUTION. The most important is that it predicted the need for a biological way for passing information between generations. That ultimately led to the discovery of the DNA molecule and within half a century the mapping of the human genome as well as that of other animals.

In the other direction the ideas of evolution have given computer scientists ideas for new ways to program - the notion of genetic algorithms. Computer science is about coming up with solutions to problems and that is exactly what nature does over time - adapt animal species via natural selection to allow them to survive better in their changing environments. The idea is that to solve a problem into "digital DNA" and evolve a solution.

In recent years, there was been widespread interaction among researchers from varied evolution-based studies and as a result, we find some breakdown in the boundaries that define and separate the fields of genetic algorithms, evolution strategies, and evolutionary programming. Often, the term - genetic algorithm - is used to describe something very different from what was originally defined.

In a genetic algorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible.

The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a *generation*. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population

A typical genetic algorithm requires:

1. a genetic representation of the solution domain,
2. a fitness function to evaluate the solution domain.

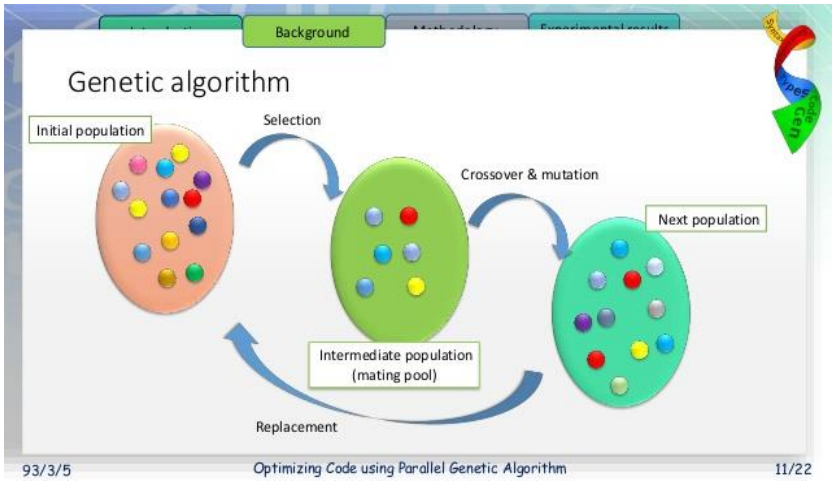

Figure 2.1: Genetic Algorithm

A standard representation of each candidate solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in genetic programming and graph-form representations are explored in evolutionary programming; a mix of both linear chromosomes and trees is explored in gene expression programming.

Once the genetic representation and the fitness function are defined, a GA proceeds to initialize a population of solutions and then to improve it through repetitive application of the mutation, crossover, inversion and selection operators.

2.3 USE OF GENETIC ALGORITHM

"Genetic Algorithms are nondeterministic stochastic search/optimization methods that utilize the theories of evolution and natural selection to solve a problem within a complex solution space".

Genetic algorithms are basically computer-based problem solving systems which use computational models of some of the known mechanisms in "evolution" as key elements in their design and implementation. They are a member of a wider population of algorithm Evolutionary Algorithms (EA).

Genetic Algorithms are heuristic, which means it estimates a solution. We won't know if we get the exact solution, but that may be a minor concern.

GA's work within a Complex solution space: GAs can be used where optimization is needed. I mean that where there are complex large solutions to the problem but we have to find the best one. Like we can use GAs in finding best moves in chess, mathematical problems, financial problems and in many more areas.

There are many tasks for which we know fast (polynomial) algorithms. There are also some problems that are not possible to be solved algorithmically. For some problems was proved that they are not solvable in polynomial time.

NP stands for nondeterministic polynomial and it means that it is possible to "guess" the solution (by some nondeterministic algorithm) and then check it, both in polynomial time. If we had a machine that can guess, we would be able to find a solution in some reasonable time.

Studying of NP-complete problems is for simplicity restricted to the problems, where the answer can be yes or no. Because there are tasks with complicated outputs, a class of problems called NP-hard problems has been introduced. This class is not as limited as class of NP-complete problems

For NP-problems is characteristic that some simple algorithm to find a solution is obvious at a first sight - just trying all possible solutions. But this algorithm is very slow (usually O(2^n)) and even for a bit bigger instances of the problems it is not usable at all.

Today nobody knows if some faster exact algorithm exists. Proving or disproving this remains as a big task for new researchers. Today many people think, that such an algorithm does not exist and so they are looking for some alternative methods - example of these methods are "GENETIC ALGORITHMS".

2.4 GENETIC OPERATERS

A **genetic operator** is an operator used in genetic algorithms to guide the algorithm towards a solution to a given problem. There are three main types of operators (mutation, crossover and selection), which must work in conjunction with one another in order for the algorithm to be successful. Genetic operators are used to create and maintain genetic diversity (mutation operator), combine existing solutions (also known as chromosomes) into new solutions (crossover) and select between solutions (selection)

2.4.1 CROSSOVER

In genetic algorithms, **crossover** is a genetic operator used to vary the programming of a chromosome or chromosomes from one generation to the next. Its a process of producing a child(a new better solution) by taking two parent solutions.

Crossover is the process of taking more than one parent solutions (chromosomes) and producing a child solution from them. By recombining portions of good solutions, the genetic algorithm is more likely to create a better solution. As with selection, there are a number of different methods for combining the parent solutions, including the edge recombination operator (ERO) and the 'cut and splice crossover' and 'uniform crossover' methods. The crossover method is often chosen to closely match the chromosome's representation of the solution; this may become particularly important when variables are grouped together as building blocks, which might be disrupted by a non-respectful crossover operator. Similarly, crossover methods may be particularly suited to certain problems; the ERO is generally considered a good option for solving the travelling salesman problem.
Cross over is a recombination operator that proceeds in three steps:

- The reproduction operator chooses at random a pair of two individual strings for the mating.
- A cross site is then decided at random along the string length.
- Finally, the selected position values are swapped between the two strings following the cross site.

Different types of crossover are :

1. Single Point Crossover
   Chooses a random integer n between 1 and Number of variables, and selects the vector entries numbered less than or equal to n from the first parent, selects genes numbered greater than n from the second parent, and concatenates these entries to form the child.

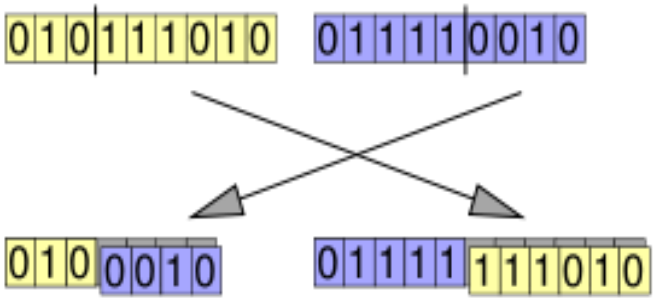

Figure 2.2.: Single Point Crossover

2. Two Point Crossover
   Two point selects two random integers m and n between 1 and Number of variables. The algorithm selects genes numbered less than or equal to m from the first parent, selects genes numbered from m+1 to n from the second parent, and selects genes numbered greater than n from the first parent. The algorithm then concatenates these genes to form a single gene.

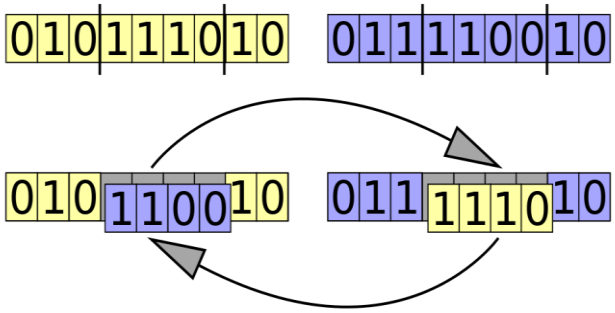

Figure 2.3: Two Point Crossover

3. Cut and Splice
   In the crossover "cut and splice" approach, cutting points are selected separately for each parent. This may result in a change in length of the child organisms.

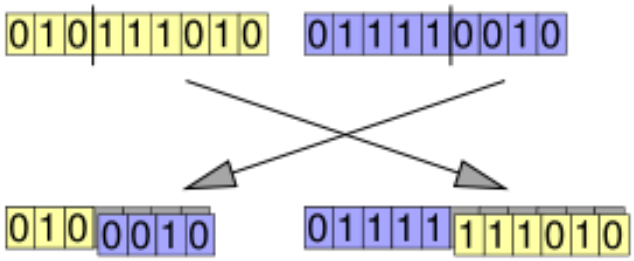

Figure 2.4: Cut and Splice

2.4.2 MUTATION

For prevention of the algorithm to be trapped in a local minimum. Its use is to change the value of a random position in a string. Lastly the new generation which is evolved if contains a solution that produces an output to the problem that is very close to the desired result would be actual solution. If the problem does not reach to its solution *i.e.*, after whole procedure the solution is not obtained, then the new generation goes through the same procedure as their parents did. This will continue until a solution is achieved. After a crossover is performed, mutation take place. This is to prevent falling all solutions in population into a local optimum of

solved problem. Mutation changes randomly the new offspring. For binary encoding we can switch a few randomly chosen bits from 1 to 0 or from 0 to 1.

Mutation occurs during evolution according to a user-definable mutation probability. This probability should be set low. If it is set too high, the search will turn into a primitive random search. The classic example of a mutation operator involves a probability that an arbitrary bit in a genetic sequence will be changed from its original state. A common method of implementing the mutation operator involves generating a random variable for each bit in a sequence. This random variable tells whether or not a particular bit will be modified. This mutation procedure, based on the biological point mutation, is called single point mutation. Other types are inversion and floating point mutation.

The purpose of mutation in GAs is preserving and introducing diversity. Mutation should allow the algorithm to avoid local minima by preventing the population of chromosomes from becoming too similar to each other, thus slowing or even stopping evolution. This reasoning also explains the fact that most GA systems avoid only taking the fittest of the population in generating the next but rather a random (or semi-random) selection with a weighting toward those that are fitter.
The most common type of mutation used is

Bit String Mutation
The mutation of bit strings ensue through bit flips at random positions.

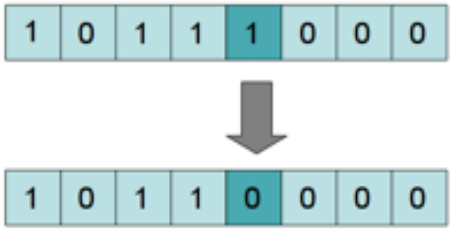

Figure 2.5: Bit string Mutation

2.4.3 SELECTION

Selection is the stage of a genetic algorithm in which individual genomes are chosen from a population for later breeding.
The procedure for selection is as follows:

1. The fitness function is evaluated for each individual, providing fitness values, which are then normalized. Normalization means dividing the fitness value of each individual by the sum of all fitness values, so that the sum of all resulting fitness values equals 1.The population is sorted by descending fitness values.
2. Accumulated normalized fitness values are computed (the accumulated fitness value of an individual is the sum of its own fitness value plus the fitness values of all the previous individuals). The accumulated fitness of the last individual should be 1 (otherwise something went wrong in the normalization step).
3. A random number *R* between 0 and 1 is chosen.

4. The selected individual is the first one whose accumulated normalized value is greater than *R*

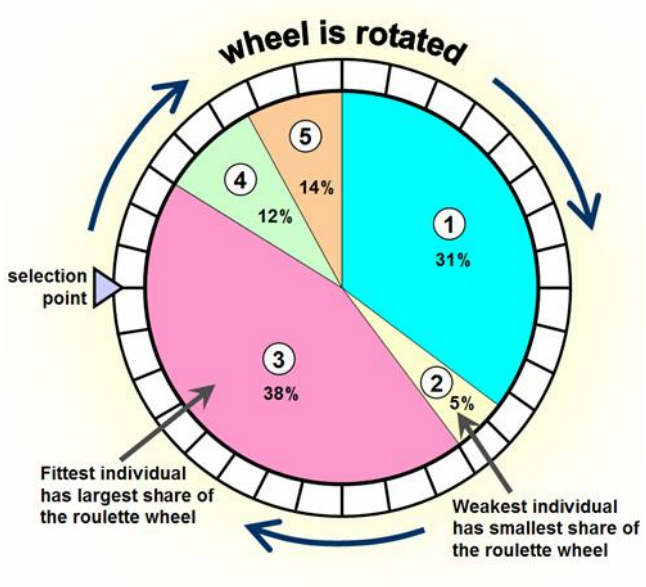

Figure 2.6: Roulette Wheel Selection

If this procedure is repeated until there are enough selected individuals, this selection method is called fitness proportionate selection or roulette-wheel selection. If instead of a single pointer spun multiple times, there are multiple, equally spaced pointers on a wheel that is spun once, it is called stochastic universal sampling. Repeatedly selecting the best individual of a randomly chosen subset is tournament selection. Taking the best half, third or another proportion of the individuals is truncation selection.

There are other selection algorithms that do not consider all individuals for selection, but only those with a fitness value that is higher than a given (arbitrary) constant. Other algorithms select from a restricted pool where only a certain percentage of the individuals are allowed, based on fitness value.

2.5 ALGORITHM AND FLOW CHART

1. First arbitrarily create an initial set of population of independent computer programs made of functions and terminals.
2. Execute each program in the population and calculate fitness value for each program using a criterion decided in starting.
3. Select one or two independent program(s) from the population with using a probabilistic approach for fitness to take part in the genetic operations.
4. Create new independent program(s) for the population by applying the following genetic
    i. **Reproduction:** Replicate the selected independent programs to the new population

ii. **Crossover:** Construct new offspring program(s) for the new population by recombining arbitrarily chosen parts from the selected programs.

iii. **Mutation:** Construct a single new offspring program for the new population by arbitrarily mutating an arbitrarily chosen part of one selected program

5. After the stopping criterion is fulfilled, the single best program in the population created during the process is gathered and classified as the result of execution. By taking the union if we get the desired result, which is a solution (or approximate solution) to the problem.

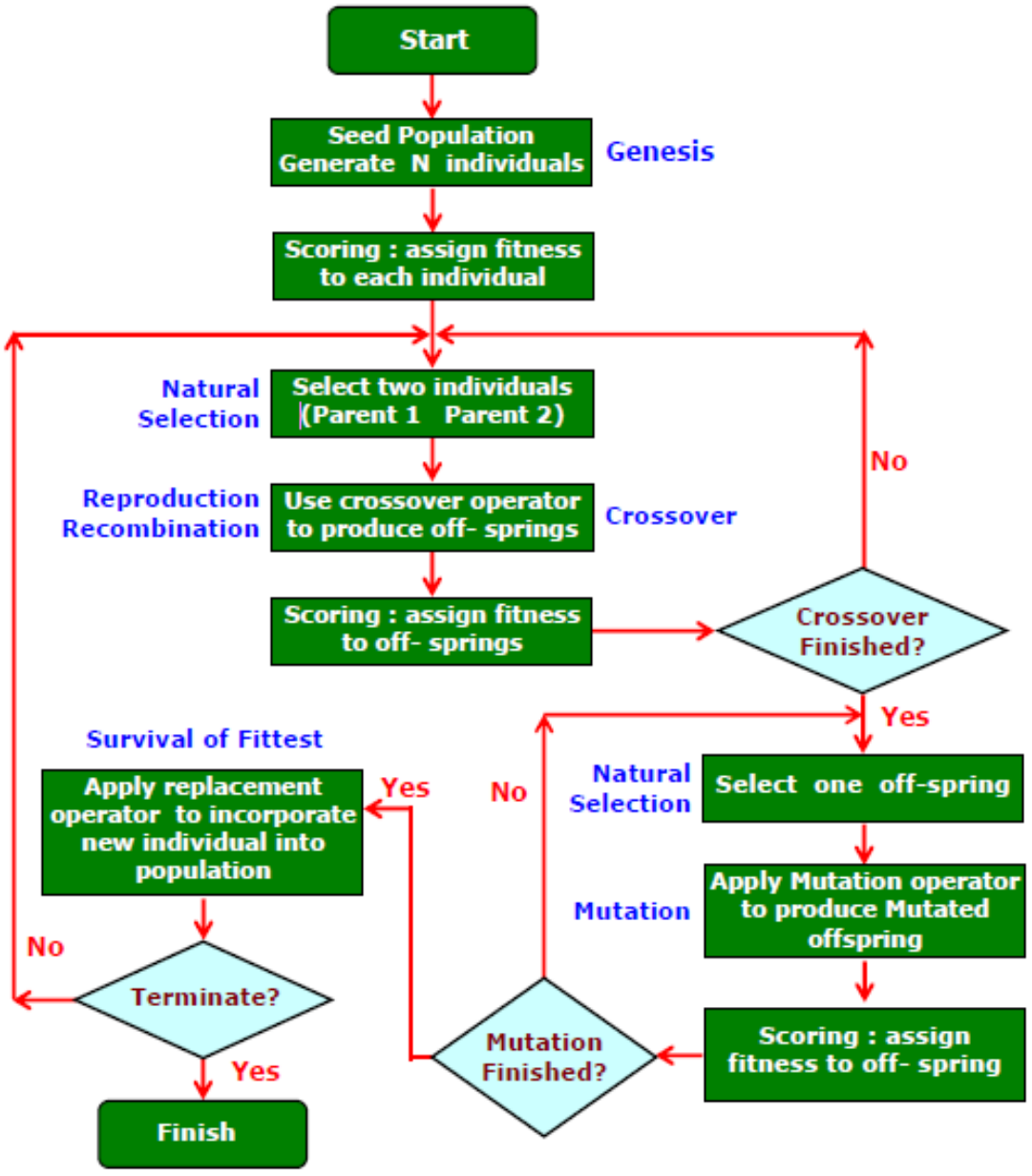

Figure 2.7: Flow Chart of Genetic Algorithm

2.6 ADVANTAGES OF GENETIC ALGORITHM

- Parallelism : GA works with multiple offspring's thus making it ideal for large problems where evaluation of all possible solutions in serial would be too time taking, if not impossible.
- It can quickly scan a vast solution set. The inductive nature of the GA means that it doesn't have to know any rules of the problem it works by its own internal rules. This is very useful for complex or loosely defined problems
- If the objective function is not smooth.
- The number of parameters is large

2.7 DISADVANTAGES OF GENETIC ALGORITHM

- Certain optimization problems cannot be solved by means of genetic algorithms. This occurs due to poorly known fitness functions which generate bad chromosome blocks in spite of the fact that only good chromosome blocks cross-over.
- There is no absolute assurance that a genetic algorithm will find a global optimum. It happens very often when the populations have a lot of subjects.
- You usually need a decent sized population and a lot of generations before you see good results. And with a heavy simulation, you can often wait days for a solution.
- Fine tuning all the parameters for the GA, like mutation rate, crossover parameters, fitness normalization/selection parameters, etc, is often just trial and error.

2.8 APPLICATIONS

- Traveling salesman problem and its applications.
- Clustering, using genetic algorithms to optimize a wide range of different fit-functions,
- Container loading optimization: Minimizing of wastage of space while loading the container.
- Multimodal optimization: It deals with optimization tasks that involve finding all or most of the multiple (at least locally optimal) solutions of a problem, as opposed to a single best solution.

2.9 CONCLUSION

Genetic Algorithms are mainly inspired by DARWIN's theory of evolution. The whole concept is derived from single sentence "survival of the fittest". Nature itself provides best robust solutions to complex problems that we see day today in our life. If we understand those solutions and implement in complex problems, there might be no solution better than that.

Genetic algorithms explore a far greater range of solutions to a problem than do conventional programs techniques like greedy techniques etc. They can be used when there is no other known efficient problem solving strategy, and the problem domain is NP-complete. These algorithms are extremely efficient, and are used in fields as stock exchange, production scheduling or programming of assembly robots in the automotive industry.

Finally, using Genetic Algorithms, there would be a no problem that remains unsolved even considering performance criteria.

CHAPTER 3

CHAPTER 3

PARTICLE SWARM OPTIMIZATION

3.1 INTRODUCTION TO PSO

Inspired by the flocking and schooling patterns of birds and fish, Particle Swarm Optimization (PSO) was invented by Russell Eberhart and James Kennedy in 1995. Originally, these two started out developing computer software simulations of birds flocking around food sources, then later realized how well their algorithms worked on optimization problems.

Particle Swarm Optimization might sound complicated, but it's really a very simple algorithm. Over a number of iterations, a group of variables have their values adjusted closer to the member whose value is closest to the target at any given moment. Imagine a flock of birds circling over an area where they can smell a hidden source of food. The one who is closest to the food chirps the loudest and the other birds swing around in his direction. If any of the other circling birds comes closer to the target than the first, it chirps louder and the others veer over toward him. This tightening pattern continues until one of the birds happens upon the food. It's an algorithm that's simple and easy to implement.

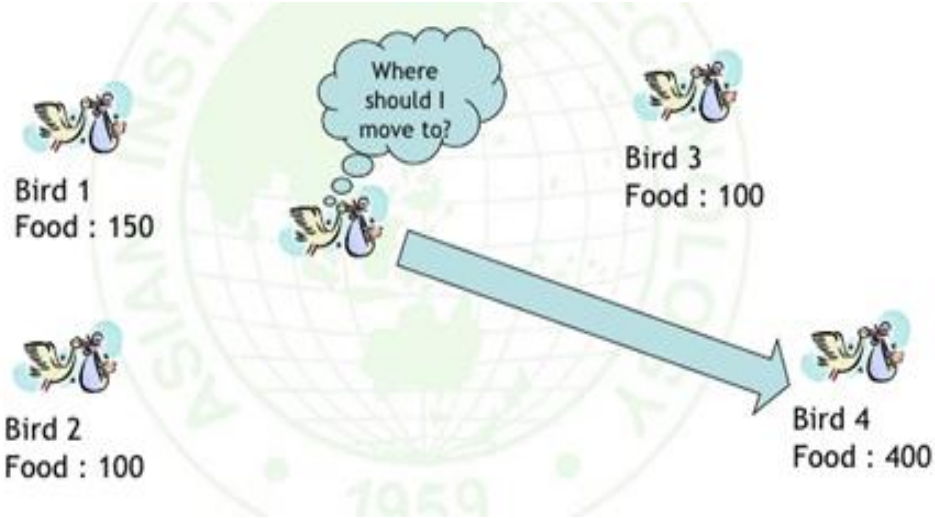

Figure 3.1: Flock of birds finding food source

In computer science, particle swarm optimization (PSO) is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. It solves a problem by having a population of candidate solutions, here dubbed particles, and moving these particles around in the search-space according to simple mathematical formulae over the particle's position and velocity. Each particle's movement is influenced by its local best known position but, is also guided toward the best known positions in

the search-space, which are updated as better positions are found by other particles. This is expected to move the swarm toward the best solutions.

Many areas in power systems require solving one or more nonlinear optimization problems. While analytical methods might suffer from slow convergence and the curse of dimensionality, heuristics-based swarm intelligence can be an efficient alternative. Particle swarm optimization (PSO), part of the swarm intelligence family, is known to effectively solve large-scale nonlinear optimization problems. For each application, technical details that are required for applying PSO, such as its type, particle formulation (solution representation), and the most efficient fitness functions are also discussed**.**

Their flexibility can be used to incorporate algorithms or heuristics for global optimization, in a way that the resulting direct or pattern search method inherits some of the properties of the imported global optimization technique, without jeopardizing the convergencefor local stationary mentioned before.

The aim of optimization is to determine the best-suited solution to a problem under a given set of constraints. Several researchers over the decades have come up with different solutions to linear and non-linear optimization problems. Mathematically an optimization problem involves fitness function describing the problem, under a set of constraints representing the solution space for the problem. Unfortunately, most of the traditional optimization techniques are centered around evaluating the first derivatives to locate the optima on a given constrained surface. The optimization problem, now-a-days, is represented as an intelligent search problem, where one or more agents are employed to determine the optima on a search landscape, representing the constrained surface for the optimization problem.

PSO is based on two fundamental disciplines: social science and computer science. In addition, PSO uses the swarm intelligence concept, which is the property of a system, whereby the collective behaviors of unsophisticated agents that are interacting
locally with their environment create coherent global functional patterns. Therefore, the cornerstones of PSO can be described as follows.

Social Concept :

It is known that "*human intelligence results from social interaction*." Evaluation, comparison, and imitation of others, as well as learning from experience allow humans to adapt to the environment and determine optimal patterns of behavior, attitudes, and suchlike. In addition, a second fundamental social concept indicates that "*culture and cognition are inseparable consequences of human sociality*."
Culture is generated when individuals become more similar due to mutual social learning. The sweep of culture allows individuals to move towards more adaptive patterns of behavior.

PSO has the following attributes :

1. Individual particles (cells) are updated in parallel.
2. Each new value depends only on the previous value of the particle (cell) and its neighbors.
3. All updates are performed according to the same rule

The goal of the algorithm is to have all the particles locate the optima in a multi-dimensional hyper-volume. This is achieved by assigning initially random positions to all particles in the space and small initial random velocities. The algorithm is executed like a simulation, advancing the position of each particle in turn based on its velocity, the best known global position in the problem space and the best position known to a particle. The objective function is sampled after each position update. Over time, through a combination of exploration and exploitation of known good positions in the search space, the particles cluster or converge together around an optima, or several optima.

3.2 USES OF PSO

The goal of the algorithm is to have all the particles locate the optima in a multi-dimensional hyper-volume. This is achieved by assigning initially random positions to all particles in the space and small initial random velocities. The algorithm is executed like a simulation, advancing the position of each particle in turn based on its velocity, the best known global position in the problem space and the best position known to a particle. The objective function is sampled after each position update. Over time, through a combination of exploration and exploitation of known good positions in the search space, the particles cluster or converge together around an optima, or several optima.

PSO is easy to implement and there are few parameters to adjust. PSO has been successfully applied in many areas: function optimization, artificial neural network training, fuzzy system control, and other areas where GA can be applied. PSO simulates the behaviors of bird flocking. Suppose the following scenario: a group of birds are randomly searching food in an area. There is only one piece of food in the area being searched. All the birds do not know where the food is. But they know how far the food is in each iteration. So what's the best strategy to find the food? The effective one is to follow the bird which is nearest to the food.

PSO learned from the scenario and used it to solve the optimization problems. In PSO, each single solution is a "bird" in the search space. We call it "particle". All of particles have fitness values which are evaluated by the fitness function to be optimized, and have velocities which direct the flying of the particles. The particles fly through the problem space by following the current optimum particles.

There are several papers reported using PSO to replace the back-propagation learning algorithm in ANN in the past several years. It showed PSO is a promising method to train ANN. It is faster and gets better results in most cases. It also avoids some of the problems GA met. 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.

3.3 ALGORITHM AND FLOW CHART

PSO algorithm works by having a population (called a swarm) of candidate solutions (called particles). These particles are moved around in the search-space according to a few simple formulae. The movements of the particles are guided by their own best known position in the search-space as well as the entire swarm's best known position. When improved positions are being discovered these will then come to guide the movements of the swarm. The process is repeated and by doing so it is hoped, but not guaranteed, that a satisfactory solution will eventually be discovered.

PSO is initialized with a group of random particles (solutions) and then searches for optima by updating generations. In every iteration, each particle is updated by following two "best" values. The first one is the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called "pbest". Another "best" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the population. This best value is a global best and called "gbest". When a particle takes part of the population as its topological neighbors, the best value is a local best and is called "lbest".

For each particle
  Initialize particle with feasible random number
End

Do
  For each particle
    Calculate the fitness value
    If the fitness value is better than the best fitness value (pbest) in history
      Set current value as the new pbest
End

Choose the particle with the best fitness value of all the particles as the gbest
  For each particle
    Calculate particle velocity according to velocity update equation
    Update particle position according to position update equation
  End
While maximum iterations or minimum error criteria is not attained

Velocity update equation :

$$v_{n+1} = v_n + c_1 rand1() * (p_{best,n} - \text{CurrentPosition}_n) + c_2 rand2() * (g_{best,n} - \text{CurrentPosition}_n)$$

Where,
vn+1 : Velocity of particle at n+1 th iteration
Vn : Velocity of particle at nth iteration
c1 : acceleration factor related to gbest
c2 : acceleration factor related to lbest
rand1() : random number between 0 and

rand2( ): random number between 0 and 1
gbest : gbest position of swarm
pbest : pbest position of particle

Current Position[*n+1*] = Current Position [*n*] + *v*[*n+1*]

Where,
current position[*n+1*] : position of particle at *n+1th* iteration
current position[*n*] : position of particle at *nth* iteration
v[*n+1*] : particle velocity at *n+1th* iteration

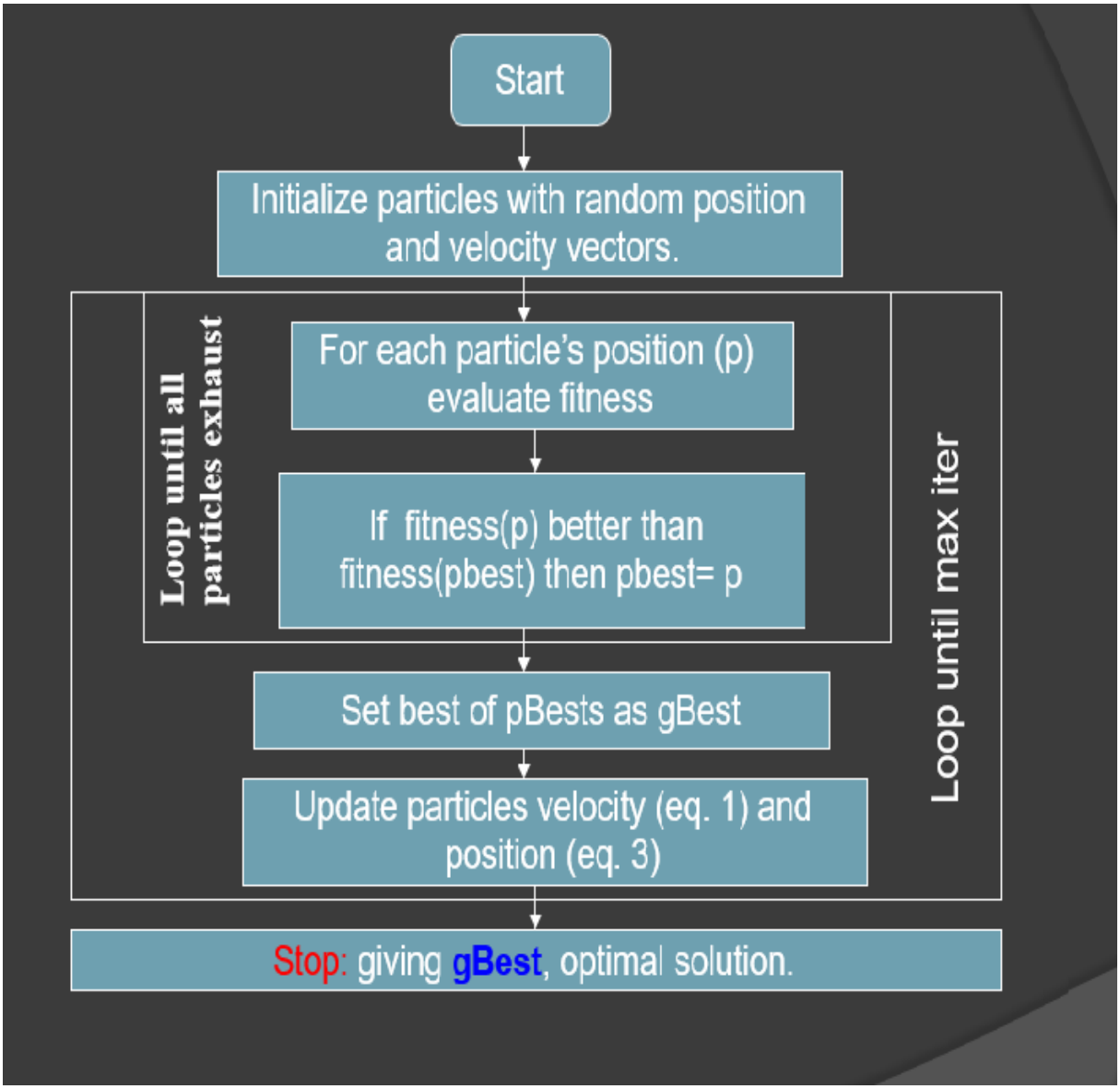

Figure 3.2: Flow Chart of PSO

3.4 COMPARISION OF PSO WITH GA

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

We can see 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 optimium 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.

PSO does not have a crossover operation. However, the concept of crossover is represented in PSO because each particle is stochastically accelerated toward its own previous best position, as well as toward the global best position or the neighborhood (local) best position, depending on the version of PSO being used. The crossover concept is also apparent in the behavior of particles that appear approximately midway between "swarms" of particles that are clustering around local best positions, or, occasionally, between successive global best positions. These particles seem to be exploring (for a short time, anyway) a region that represents the geometric mean between two promising regions.

GA is having problem of less exploitation in work space. but in PSO can exploit more than GA. Because GA is solving binary variable so in conversion binary into number, can skip the optimal point. for handling this problem require big dimension of binary bit and that will take more time in conversion than PSO

3.5 ADVANTAGES OF PSO

- PSO is based on the intelligence. It can be applied into both scientific research and engineering use.
- PSO have no overlapping and mutation calculation. The search can be carried out by the speed of the particle. During the development of several generations, only the most optimist particle can transmit information onto the other particles, and the speed of the researching is very fast.
- The calculation in PSO is very simple. Compared with the other developing calculations, it occupies the bigger optimization ability and it can be completed easily.
- PSO adopts the real number code, and it is decided directly by the solution. The number of the dimension is equal to the constant of the solution.
- It has a greater diversity and exploration over a single population.
- The momentum effects on particle movement can allow faster convergence
- Particles are semi-autonomous agent which see each other the status of each other and decide to change their status toward the best observed particle in their locality.

3.6 DISADVANTAGES OF PSO

- The method easily suffers from the partial optimism, which causes the less exact at the regulation of its speed and the direction.
- The method cannot work out the problems of non-coordinate system, such as the solution to the energy field and the moving rules of the particles in the energy field
- It is easy to fall into local optimum in high-dimensional space and has a low convergence rate in the iterative process.

3.7 APPLICATIONS

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

- Individualized Modeling : Individualized Modeling is a computational technique that models a problem by trying to achieve an optimal solution with regard to a specified input data
- Optimization of Non-Linear Objective Functions : Many of the optimization problems in science and engineering involve nonlinear objective functions. Particle swarm optimization has been applied for optimization of such problem

3.8 CONCLUSIONS

Particle Swarm Optimization (PSO) is a biologically inspired computational search and optimization method developed based on the social behaviors of birds flocking or fish schooling. A number of basic variations have been developed due to improve speed of convergence and quality of solution found by the PSO. On the other hand, basic PSO is more appropriate to process static, simple optimization problem. the flocks achieve their best condition simultaneously through communication among members who already have a better situation.

Animal which has a better condition will inform it to its flocks and the others will move simultaneously to that place. This would happen repeatedly until the best conditions or a food source discovered. The process of PSO algorithm in finding optimal values follows the work of this animal society. Particle swarm optimization consists of a swarm of particles, where particle represent a potential solution.

3.9 REFERENCES

1. Link : https://en.wikipedia.org/wiki/Particle_swarm_optimization

2. Chih-Cheng Kao, *Applications of Particle Swarm Optimization*, Kao Yuan University

3. Jaco F. Schutte, *The Particle Swarm Optimization Algorithm*

4. Video Lecture Link: https://www.youtube.com/watch?v=F3RKF3LE2Wo, *Particle Swarm Optimization*

5. Riccardo Poli , James Kennedy , Tim Blackwell, *Particle Swarm Optimization*, LIACS Natural Computing Group Leiden University

CHAPTER 4

CHAPTER 4

BENCHMARK FUNCTIONS

4.1 INTRODUCTION TO BENCHMARK FUNCTIONS

Test functions are important to validate and compare the performance of optimization algorithms. There have been many test or benchmark functions reported in the literature; however, there is no standard list or set of benchmark functions. Ideally, test functions should have diverse properties so that can be truly useful to test new algorithms in an unbiased way. For this purpose, we have reviewed and compiled a rich set of 175 benchmark functions for unconstrained optimization problems with diverse properties in terms of modality, separability , and valley landscape. This is by far the most complete set of functions so far in the literature, and can be expected this complete set of functions can be used for validation of new optimization in the future.

The test of reliability, efficiency and validation of optimization algorithms is frequently carried out by using a chosen set of common standard benchmarks or test functions from the literature. The number of test functions in most papers varied from a few to about two dozen. Ideally, the test functions used should be diverse and unbiased, however, there is no agreed set of test functions in the literature. Therefore, the major aim of this paper is to review and compile the most complete set of test functions that we can find from all the available literature so that they can be used for future validation and comparison of optimization algorithms.

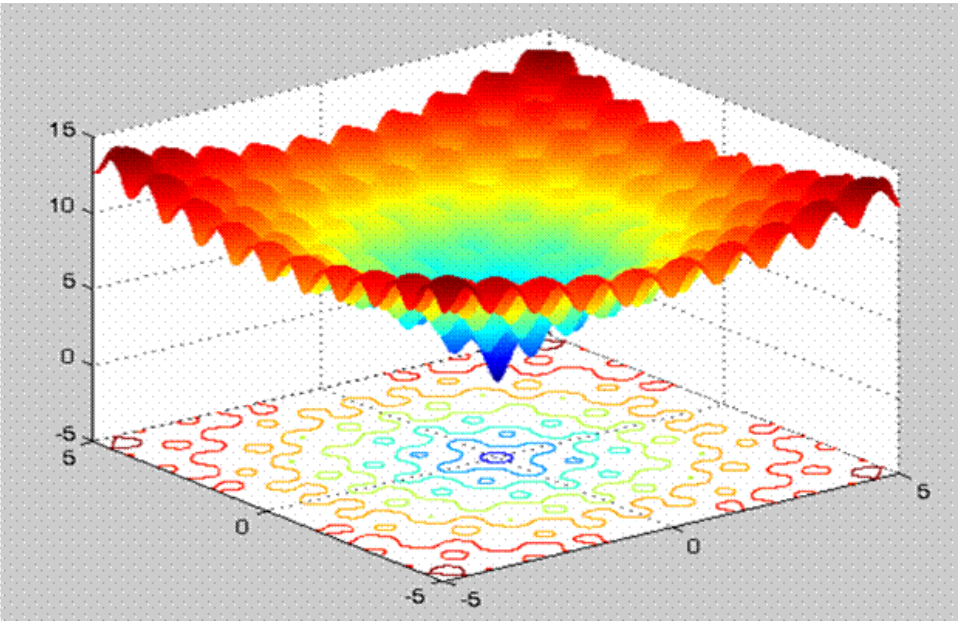

Figure 4.1: An example of a Benchmark Function

For any new optimization, it is essential to validate its performance and compare with other existing algorithms over a good set of test functions. A common practice followed by many researches is to compare different algorithms on a large test set, especially, when the test involves function optimization (Gordon 1993, Whitley 1996). However, it must be noted that effectiveness of one algorithm against others simply cannot be measured by the problems that it solves if the the set of problems are too specialized and without diverse properties. Therefore, in order to evaluate an algorithm, one must identify the kind of problems where it performs better compared to others. This helps in characterizing the type of problems for which an algorithm is suitable. This is only possible if the test suite is large enough to include a wide variety of problems, such as unimodal, multimodal, regular, irregular, separable, non-separable and multi-dimensional problems.

Many test functions may be scattered in different textbooks, in individual research articles or at different web sites. Therefore, searching for a single source of test function with a wide variety of characteristics is a cumbersome and tedious task. The most notable attempts to assemble global optimization test problems can be found in Online collections of test problems collection of test functions a collection of continuous global optimization test problems COCONUT [61] and a subset of commonly used test functions . This motivates us to carry out a thorough analysis and compile a comprehensive collection of unconstrained optimization test problems. In general, unconstrained problems can be classified into two categories: test functions and real-world problems. Test functions are artificial problems, and can be used to evaluate the behavior of an algorithm in sometimes diverse and difficult situations.

Artificial problems may include single global minimum, single or multiple global minima in the presence of many local minima, long narrow valleys, null-space effects and flat surfaces. These problems can be easily manipulated and modified to test the algorithms in diverse scenarios. On the other hand, real-world problems originate from different fields such as physics, chemistry, engineering, mathematics etc. These problems are hard to manipulate and may contain complicated algebraic or differential expressions and may require a significant amount of data to compile. A collection of real-world unstrained optimization problems can be found. In this present work, we will focus on the test function benchmarks and their diverse properties such as modality and separability.

A function with more than one local optimum is called **multimodal**. These functions are used to test the ability of an algorithm to escape from any local minimum. If the exploration process of an algorithm is poorly designed, then it cannot search the function landscape effectively. This, in turn, leads to an algorithm getting stuck at a local minimum. Multi-modal functions with many local minima are among the most difficult class of problems for many algorithms. Functions with flat surfaces pose a difficulty for the algorithms, since the flatness of the function does not give the algorithm any information to direct the search process towards the minima.

According to , the dimensionality of the search space is an important issue with the problem. In some functions, the area that contains that global minima are very small, when compared to the

whole search space. For problem such as Perm the global minimum is located very close to the local minima. If the algorithm cannot keep up the direction changes in the functions with a narrow curved valley, in case of functions like Beale, Colville, or cannot explore the search space effectively, in case of function like Pen Holder, Test tube-Holder having multiple global minima, the algorithm will fail for these kinds of problems.

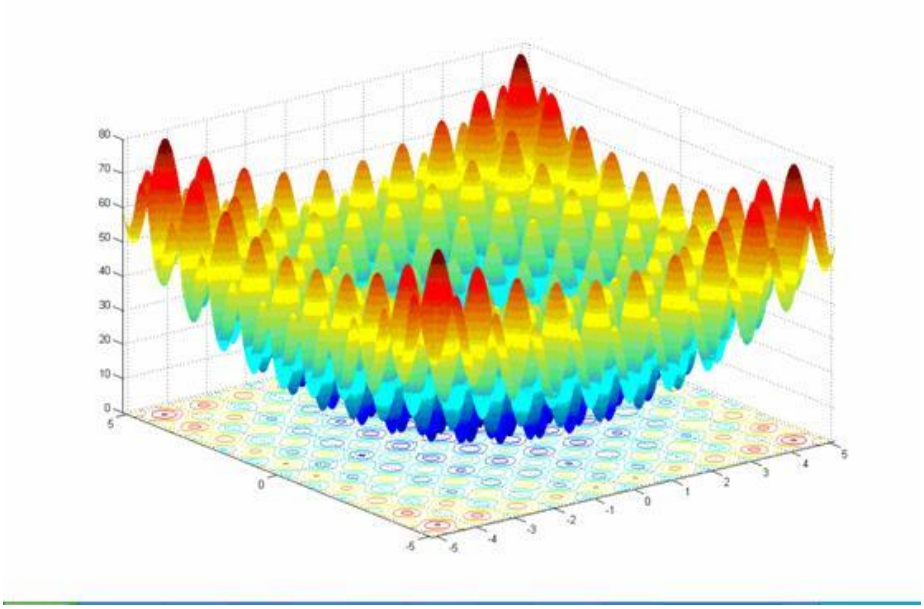

Figure 4.2: Benchmark Function PERM

4.2 BENCHMARK PROBLEM

Fully separable functions : In a fully-separable problem there is no interaction between any pair of variables. Therefore, no particular decomposition is imposed due to variable interaction. It may appear that the best decomposition for such cases is to break the problem into a set of 1-dimensional problems. The other extreme decomposition is to group all variables in one subcomponent. This decomposition uses the least amount of resources in each cycle but makes the search space extremely large.

The base functions that are used to form the separable and non-separable subcomponents are: Sphere, Elliptic, Rastrigin's, Ackley's, and Rosenbrock's functions. These functions which are classical examples of benchmark functions in many continuous optimization test suites are mathematically defined in next section.

4.3 CHARACTERSTICS OF BENCHMARK FUNCTIONS

1. Modality : The number of ambiguous speaks in the functional and scope corresponds to the modality of a function. If algorithms encounters the speaks during a search process, the reisatendency .That the algorithm may be trapped in one of such peaks This will have a negative impact on the search process, as this can direct the search away from the true optimal solutions.
2. Basins : A relatively steep decline surrounding a large area is called a basin. Optimization algorithms can be easily attracted to such regions. Once in these regions, the search process of an algorithm is severely hampered. This is due to lack of information to direct the search process towards the minimum. According to ,a basin corresponds to the plateau for a maximization problem, and a problem can have multiple plateaus.
3. Valleys : A valley occurs when a narrow area of little change is surrounded by regions of speed descent .As with the basins , minimize initially attracted to this region .

The four bench mark function that we are using for optimization purpose are

- Ackley
- Cigar
- Ellipse
- Sphere

4.4 ACKLEY FUNCTION

The Ackley function is widely used for testing optimization algorithms. In its two dimensional form, as shown in the plot above, it is characterized by a nearly flat outer region, and a large hole at the centre. The function poses a risk for optimization algorithms, particularly hill climbing algorithms, to be trapped in one of its many local minima.

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

Never the less, this function type Potentially has relevance for real world applications since e.g. the free energy hyper surface Of proteins is considered to be of similar, yet less symmetric, shape.

PLOTS

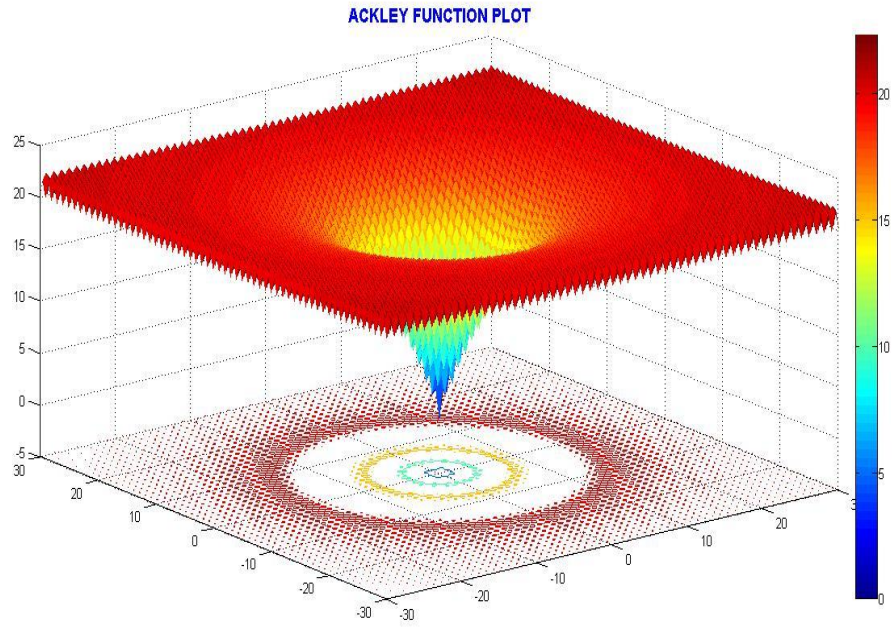

Figure 4.3(a): ACKLEY Function Plot

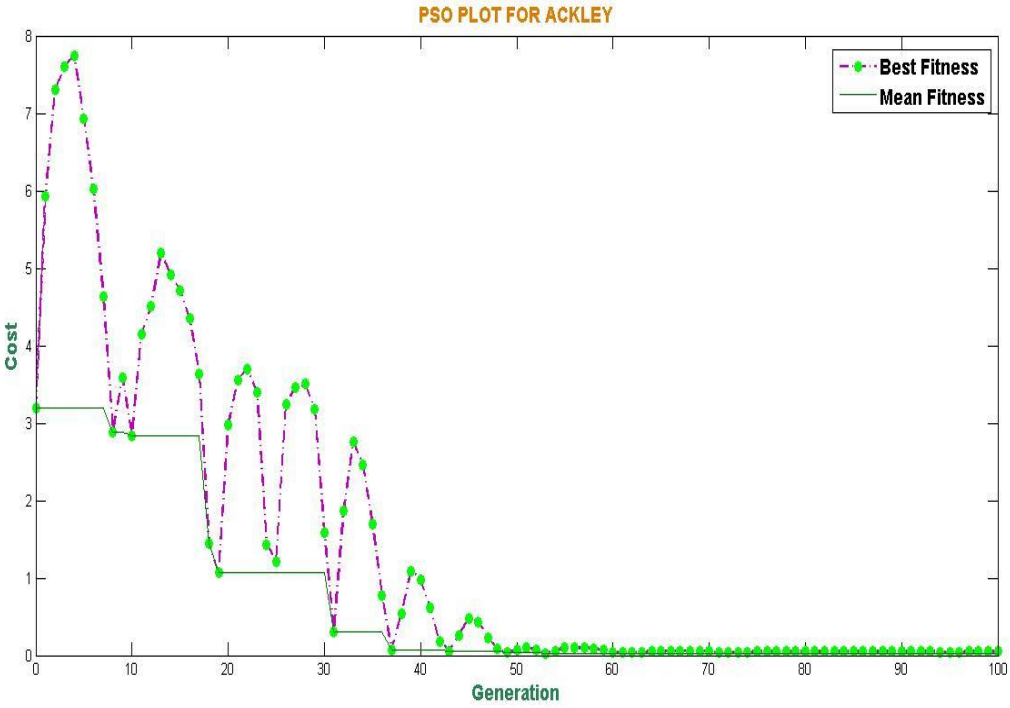

Figure 4.3(b): ACKLEY Plot Using PSO

4.5 SPHERE FUNCTION

The first function of De Jong's or Sphere function is one of the most simple test functions available in the specialized literature. This continuous, convex, unimodal and additively separable test function can be scaled up to any number of variables. It belongs to a family of functions called quadratic functions and only has one optimum in the point o=(0,...,0). The search range commonly used for the Sphere function is [-100, 100] for each decision variable. It is defined by:

$$f(x) = (x\text{^}2);$$

PLOTS

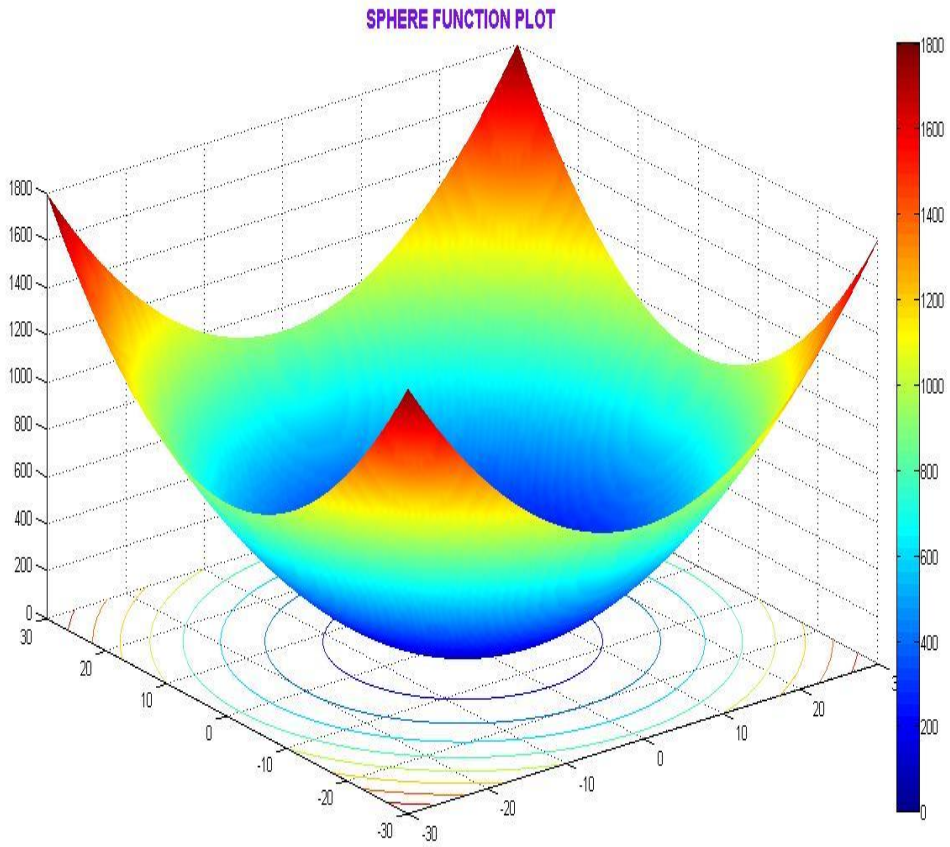

Figure 4.4(a): SPHERE Function Plot

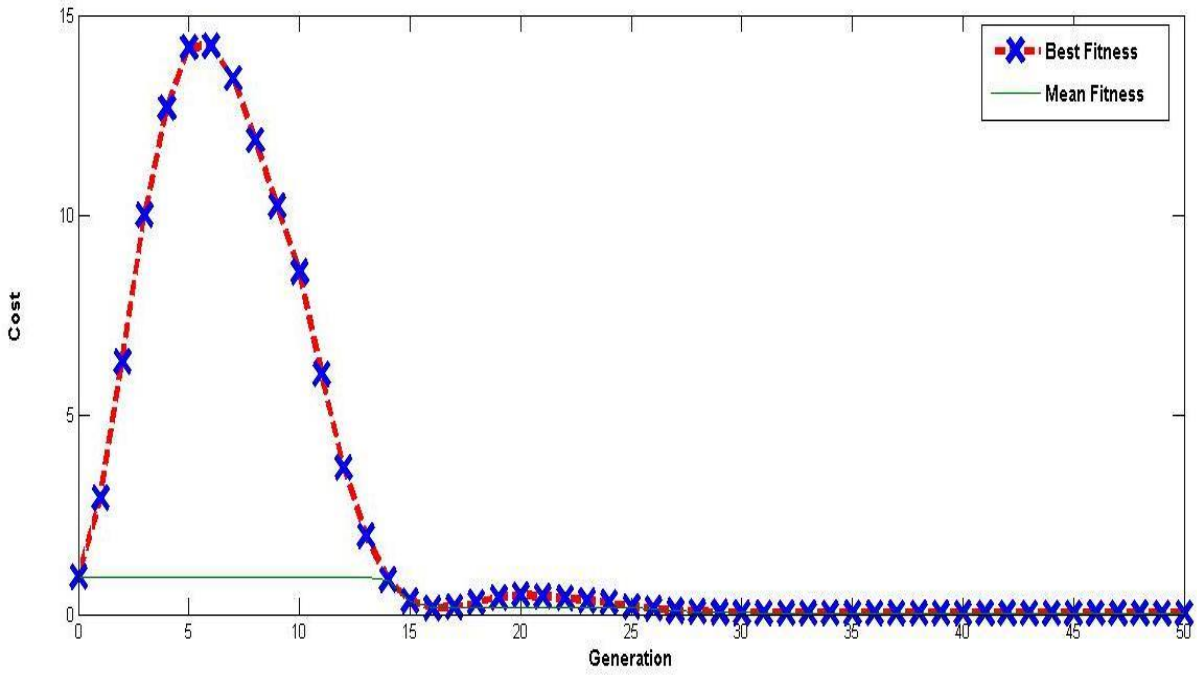

Figure 4.4(b): SPHERE Plot Using PSO

4.6 CIGAR FUNCTION

The properties of cigar functions are unimodal , non-separable, optimized located in a smooth but narrow valley. The cigar function is defined by

$$f(x) = f(x) + (10\text{^}4) * x(i)\text{^}2$$

PLOTS

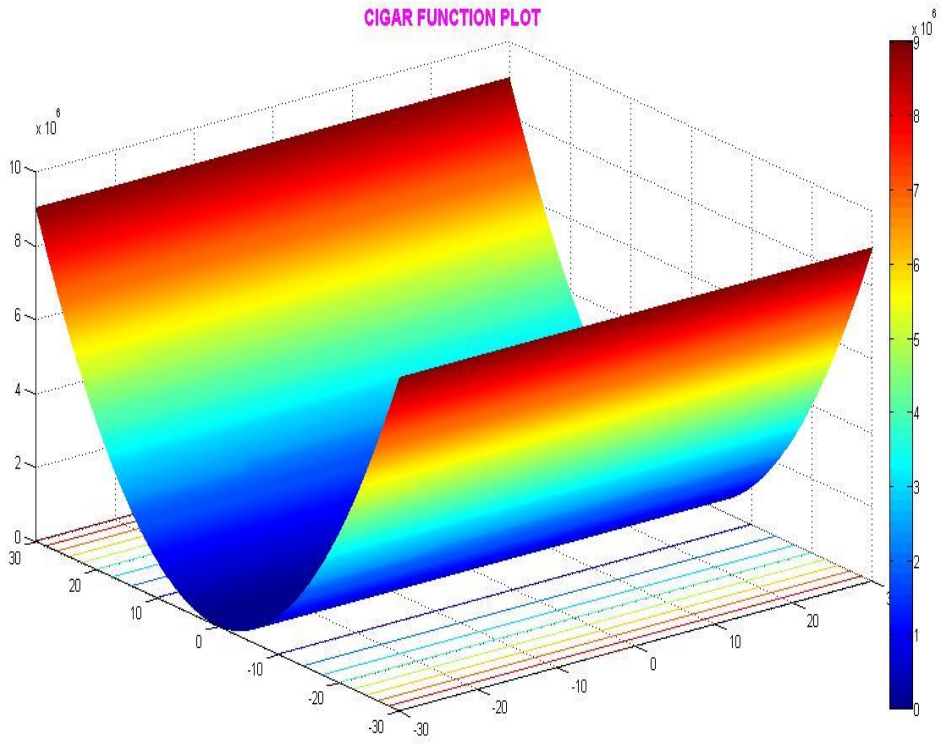

Figure 4.5(a): CIGAR Function Plot

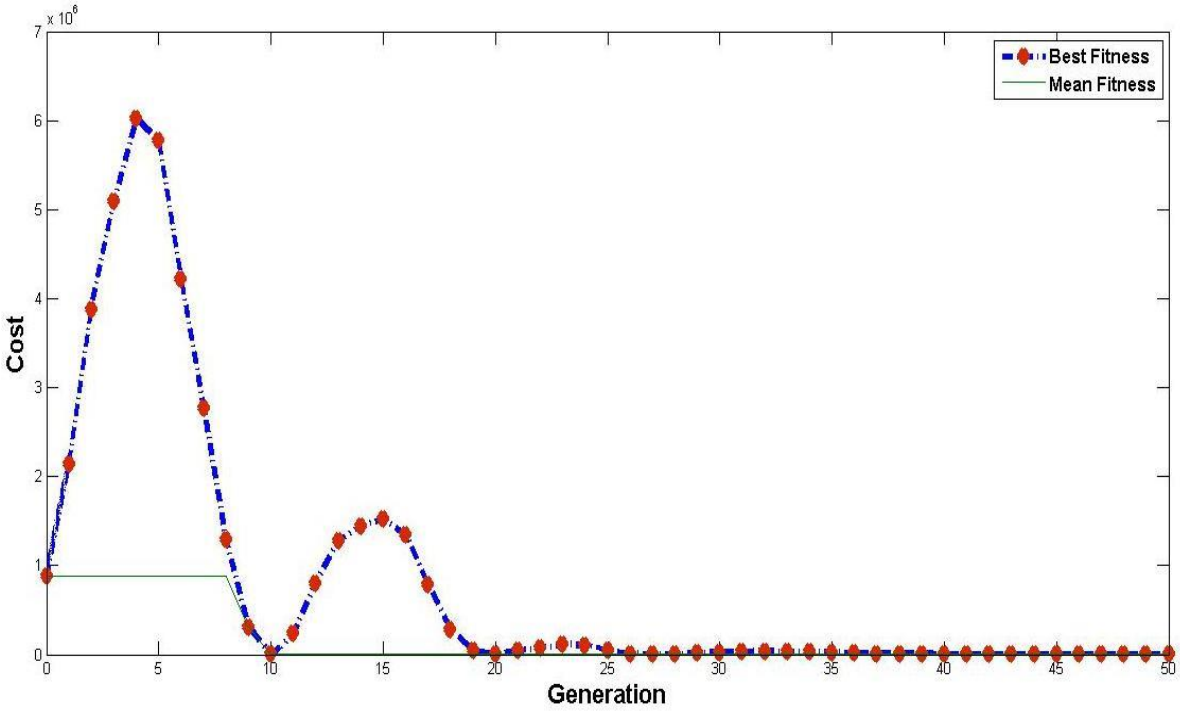

Figure 4.5(b): CIGAR Plot Using PSO

4.7 ELLIPSE FUNCTION

The properties of ellipse functions are unimodal, non- separable ,quadratic hill condition and smooth local irregularities . The center of the ellipse is the center of the specified bounding rectangle. The ellipse function is defined by

$$f(x) = f(x) + (10\text{^}(4 * (i-1)/(n-1)\ )\ ) * x(i)\text{^}2$$

PLOTS

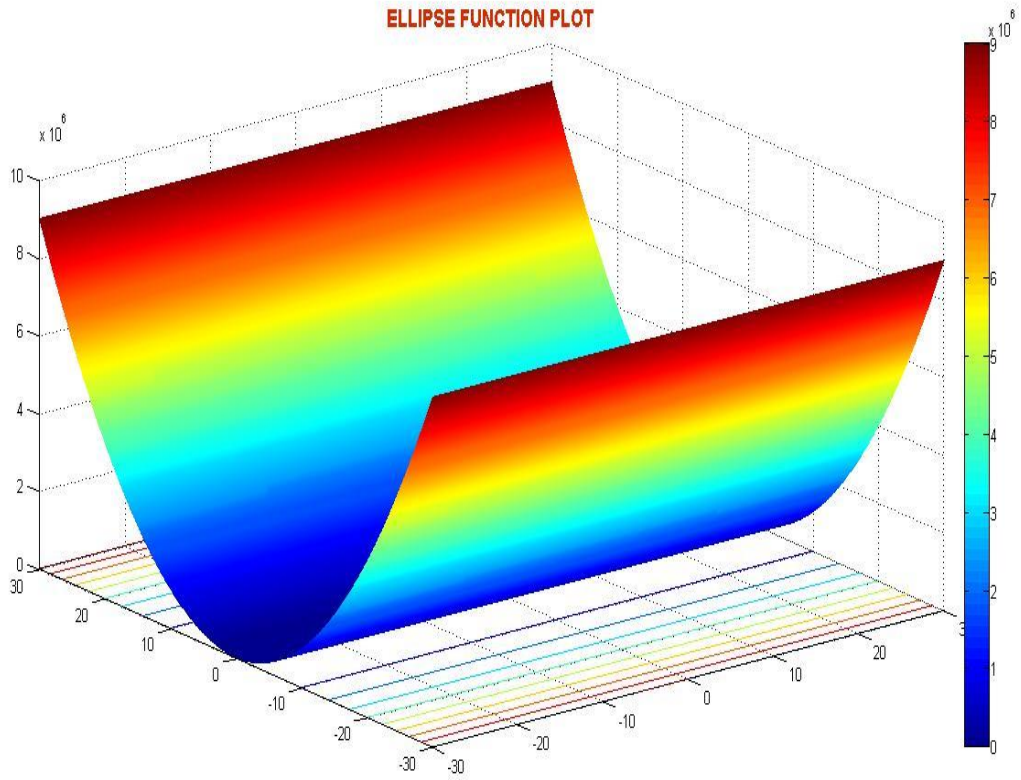

Figure 4.6(a): ELLIPSE Function Plot

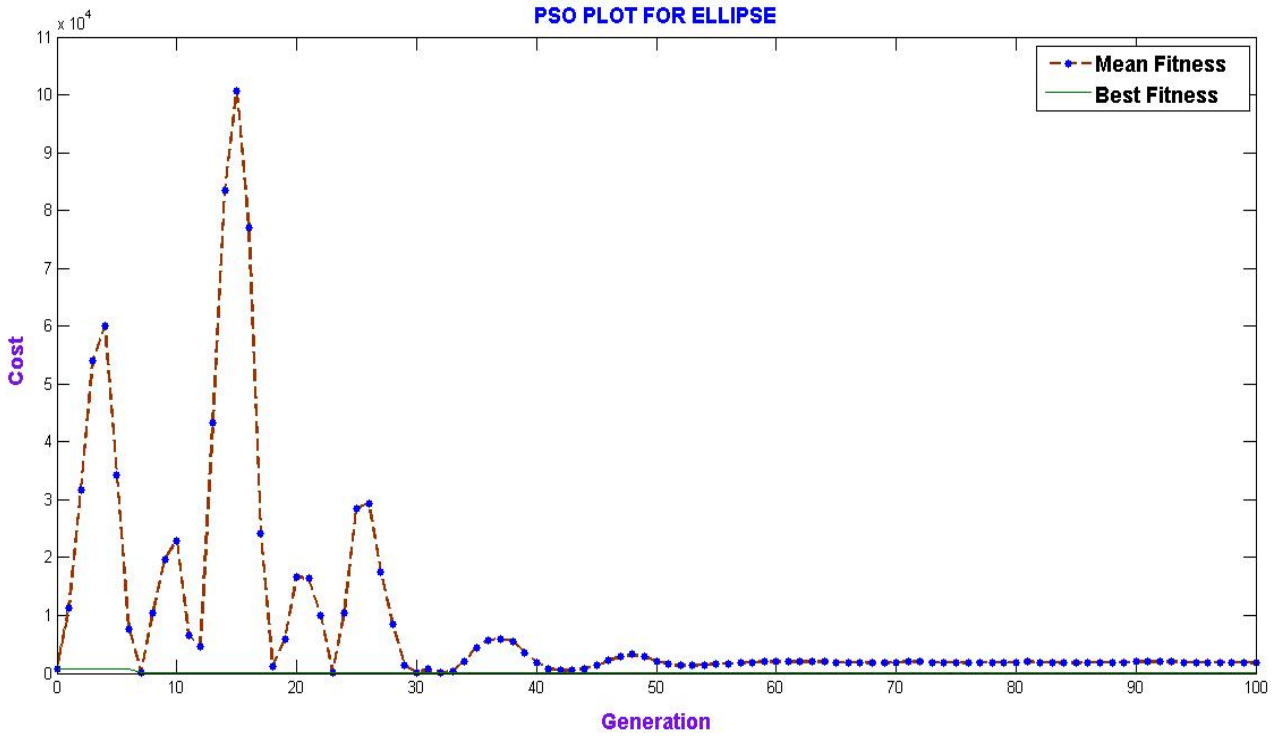

Figure 4.6(b): ELLIPSE Plot Using PSO

4.8 CONCLUSION

It is common to compare different algorithms using a large test set, especially when the test involves function optimization. However, the effectiveness of an algorithm against another algorithm cannot be measured by the number of problems that it solves better. If we compare two searching algorithms with all possible 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 fruitless task.

That is the reason why, when an algorithm is evaluated, we must look for the kind of problems where its performance is good, in order to characterize the type of problems for which the algorithm is suitable. In this way, we have made a previous study of the functions to be optimized for constructing a test set with fewer functions and a better selection. This allows us to obtain conclusions of the performance of the algorithm depending on the type of function.

Hence from the previous study we can conclude that the algorithm correctly optimizes the given objective function that is in this case the Ackley, Cigar, Sphere and Ellipse. By comparing the two figures in the respective sections, we can see that, the optima lies on (0,0) and this is proved by using the same function in the PSO algorithm.

CHAPTER 5

CHAPTER 5

OPTIMIZATION TOOL BOX IN MATLAB

5.1 INTRODUCTION

Optimization Toolbox provides functions for finding parameters that minimize or maximize objectives while satisfying constraints. The toolbox includes solvers for linear programming, mixed-integer linear programming, quadratic programming, nonlinear optimization, and nonlinear least squares. You can use these solvers to find optimal solutions to continuous and discrete problems, perform tradeoff analyses, and incorporate optimization methods into algorithms and applications

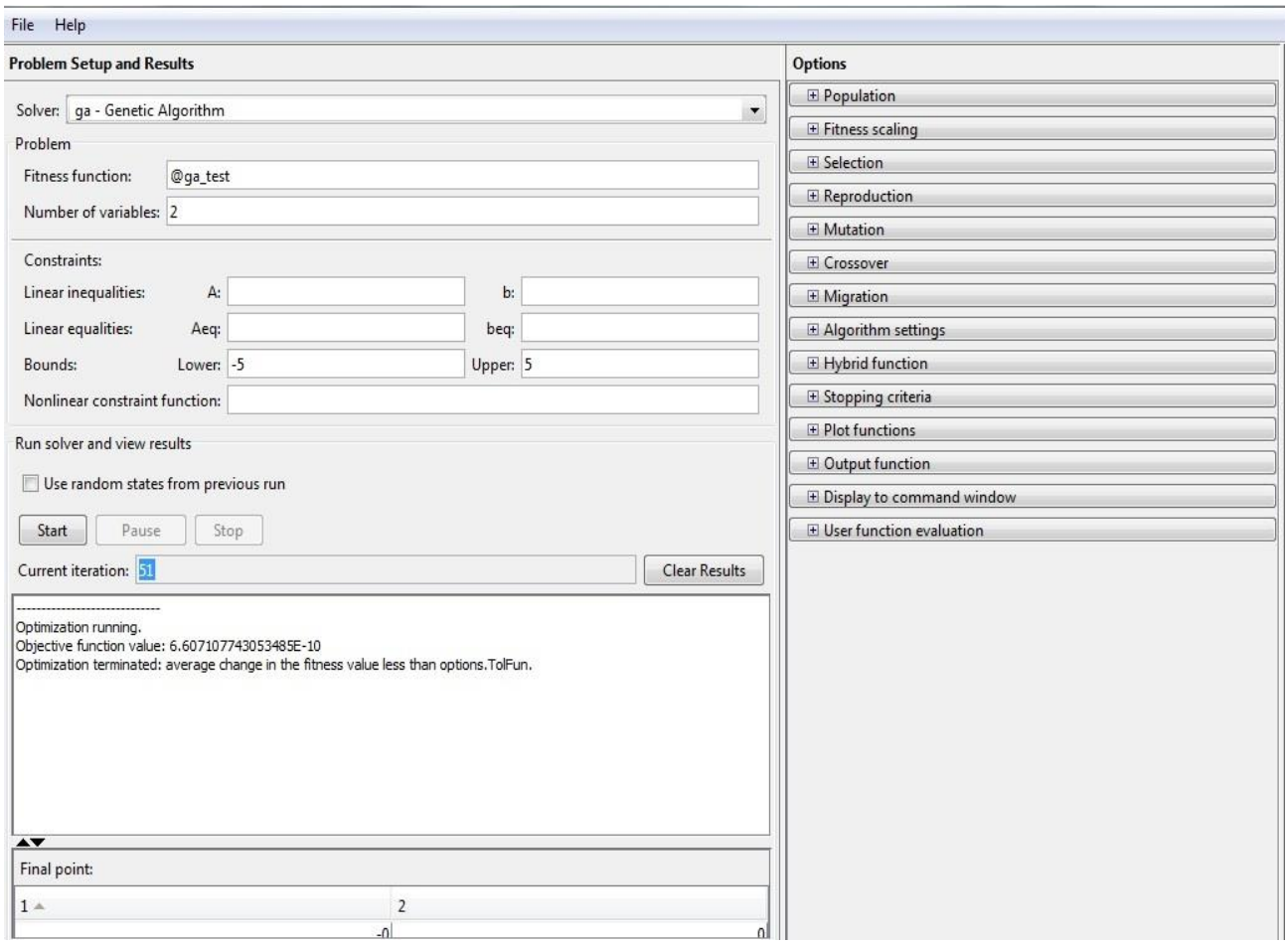

Figure 5.1: Optimization Toolbox

5.2 DIFFERENT FIELDS IN OPTIMIZATION TOOLBOX

To represent your optimization problem for solution, you generally follow these steps:

• Choose an optimization solver.

• Create an objective function, typically the function you want to minimize.

• Create constraints, if any.

• Set options, or use the default options.

• Call the appropriate solver.

Choosing a Solver

Optimization Toolbox contains different solvers for different types of objectives and constraints. The Optimization Decision Table helps you choose the best solver for your problem.

In our project we use the solver "ga - Genetic Algorithm".

Fitness Function

Many Optimization Toolbox solvers minimize a scalar function of a multidimensional vector. The objective function is the function the solvers attempt to minimize. Several solvers accept vector-valued objective functions, and some solvers use objective functions you specify by vectors or matrices.

The Fitness Function is always represented using an '@' symbol followed by the function name. The function is written in matlab and saved using the same function name. An example is "@ga_test", here "ga_test" is the name of the function and the function definition is x^2 + y^2.

The function is written as follows:

```
function f = ga_test(x)

    f =x(1).^2 + x(2).^2;

end
```

Here, x(1) and x(2) represents 2 variables.

Number of Variables

This field indicates the number of variables used in the function. In the above example the number of variables used is 2 hence this field contains 2. Likewise we can have any number of variables.

Constraints

Optimization Toolbox solvers have special forms for constraints:

- Bound Constraints — Lower and upper bounds on individual components: $x \geq l$ and $x \leq u$.
- Linear Inequality Constraints — $A \cdot x \leq b$. A is an m-by-n matrix, which represents m constraints for an n-dimensional vector x. b is m-dimensional.
- Linear Equality Constraints — $Aeq \cdot x = beq$. Equality constraints have the same form as inequality constraints.
- Nonlinear Constraints — $c(x) \leq 0$ and $ceq(x) = 0$. Both c and ceq are scalars or vectors representing several constraints.

We use only Bound Constraints that is set the lower and the upper limit to the function. In our example we have set the lower and upper limit to -5 and 5 respectively. Hence the function has to converge in between the two ranges.

Start

It start the optimization, since here we are using genetic algorithm as our solver we are finding the minima of a function. The "**number of iterations**" field indicates the number of times it has run before attaining the minima.

In our example, the optimizer has run for "**51"** times before it has attained the minima which has a value "**6.607107743053485E-10**".

This indicates that for the function x^2 + y^2 the minima lies at 0,0.

Options

On the right hand side of the optimization toolbox we have the options field this has various other fields which are used to improve the optimization result.

Some of the options which we use here are:

1. Population : specifies options for the population of the genetic algorithm.

- Population type : specifies the type of the input to the fitness function and we use **"Bit String".**
- Population size : specifies how many individuals there are in each generation. We can use default of 20 or specify. In our case we are setting it to 100.
- Creation function : specifies the function that creates the initial population. In our case we use "**Use constraint dependent default**"
- Initial population : It enables you to specify an initial population for the genetic algorithm.
- Initial scores : It enables you to specify scores for the initial population.
- Initial range : specifies lower and upper bounds for the entries of the vectors in the initial population.

2. Fitness Scaling : The scaling function converts raw fitness scores returned by the fitness function to values in a range that is suitable for the selection function.
    - Scaling Function : specifies the function that performs the scaling.
      We use "**Rank**" scaling function. Rank scales the raw scores based on the rank of each individual, rather than its score. The rank of an individual is its position in the sorted scores. The rank of the fittest individual is 1, the next fittest is 2, and so on.

3. Selection : The selection function chooses parents for the next generation based on their scaled values from the fitness scaling function.
    - Selection function : specifies the function that performs the selection.
      We use "**Roulette**" selection function. Roulette simulates a roulette wheel with the area of each segment proportional to its expectation. The algorithm then uses a random number to select one of the sections with a probability equal to its area.

4. Reproduction : Reproduction options determine how the genetic algorithm creates children at each new generation.
    - Elite count : specifies the number of individuals that are guaranteed to survive to the next generation. We set it to default value in our case.
    - Crossover fraction : specifies the fraction of the next generation that crossover produces. Mutation produces the remaining individuals in the next generation.

5. Mutation : Mutation functions make small random changes in the individuals in the population, which provide genetic diversity and enable the genetic algorithm to search a broader space.
    - Mutation function : We Use constraint dependent default chooses Gaussian if there are no constraints and Adaptive feasible otherwise.
      Gaussian adds a random number to each vector entry of an individual. This random number is taken from a Gaussian distribution centered on zero.
      Adaptive feasible randomly generates directions that are adaptive with respect to the last successful or unsuccessful generation.

6. Crossover : It combines two individuals, or parents, to form a new individual, or child, for the next generation

- Crossover function : We use "**single point crossover"**.
  Single point chooses a random integer n between 1 and Number of variables, and selects the vector entries numbered less than or equal to n from the first parent, selects genes numbered greater than n from the second parent, and concatenates these entries to form the child.

7. Plot functions : Plot functions enable you to plot various aspects of the genetic algorithm as it is executing. Each one draws in a separate axis on the display window.
   - We use "**best fitness**".
     Best fitness plots the best function value in each generation versus iteration number.

8. Display to Command Window : specifies the amount of information displayed in the MATLAB Command Window when you run the algorithm.
   - We use "**iterative**".
     Display information at each iteration of the algorithm.

5.3 PLOTS FOR VARIOUS BENCHMARK FUNCTIONS USING TOOLBOX

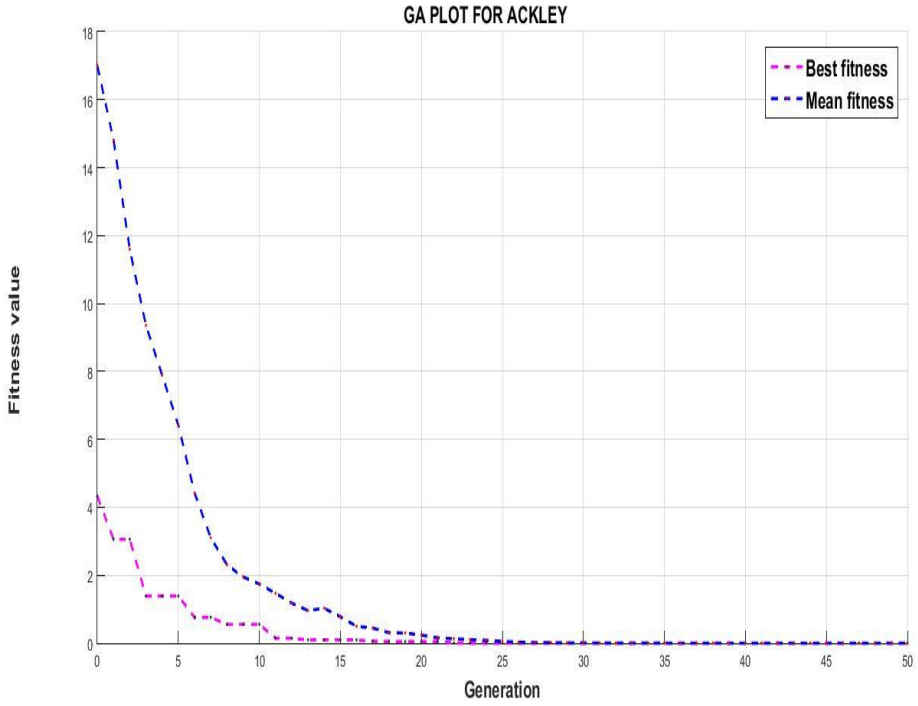

Figure 5.2: ACKLEY Plot Using GA

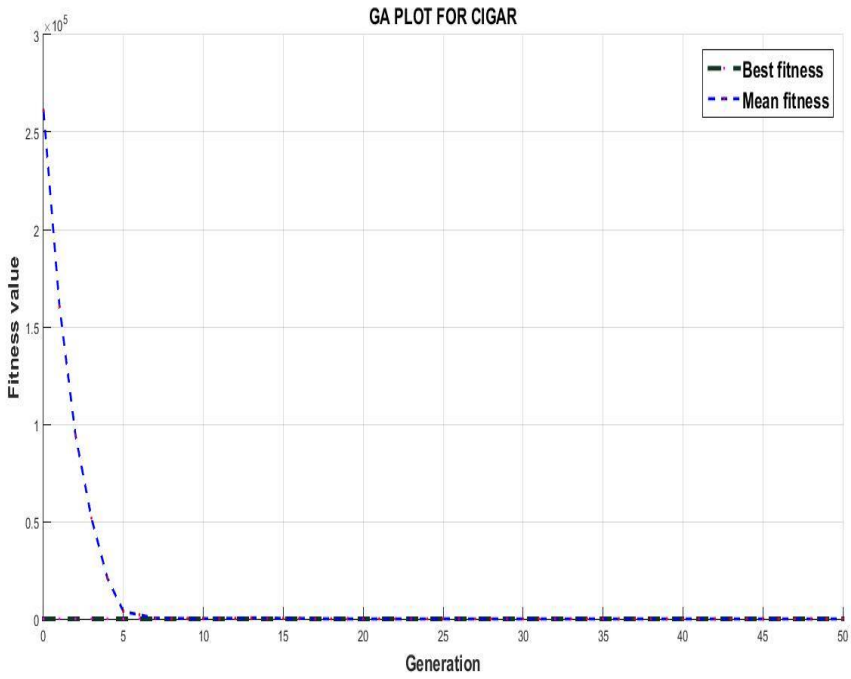

Figure 5.3: CIGAR Plot Using GA

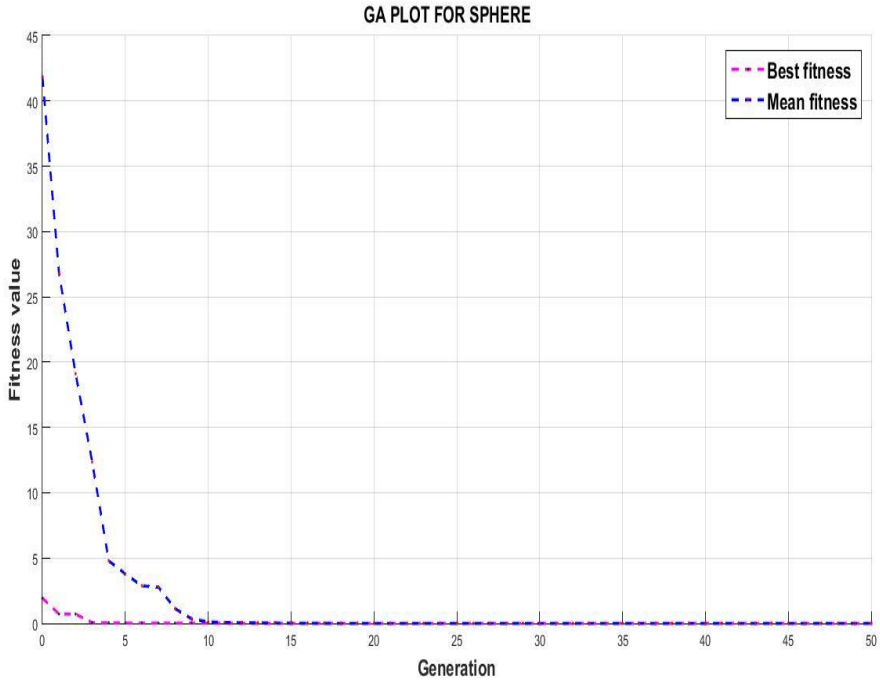

Figure 5.4: SPHERE Plot Using GA

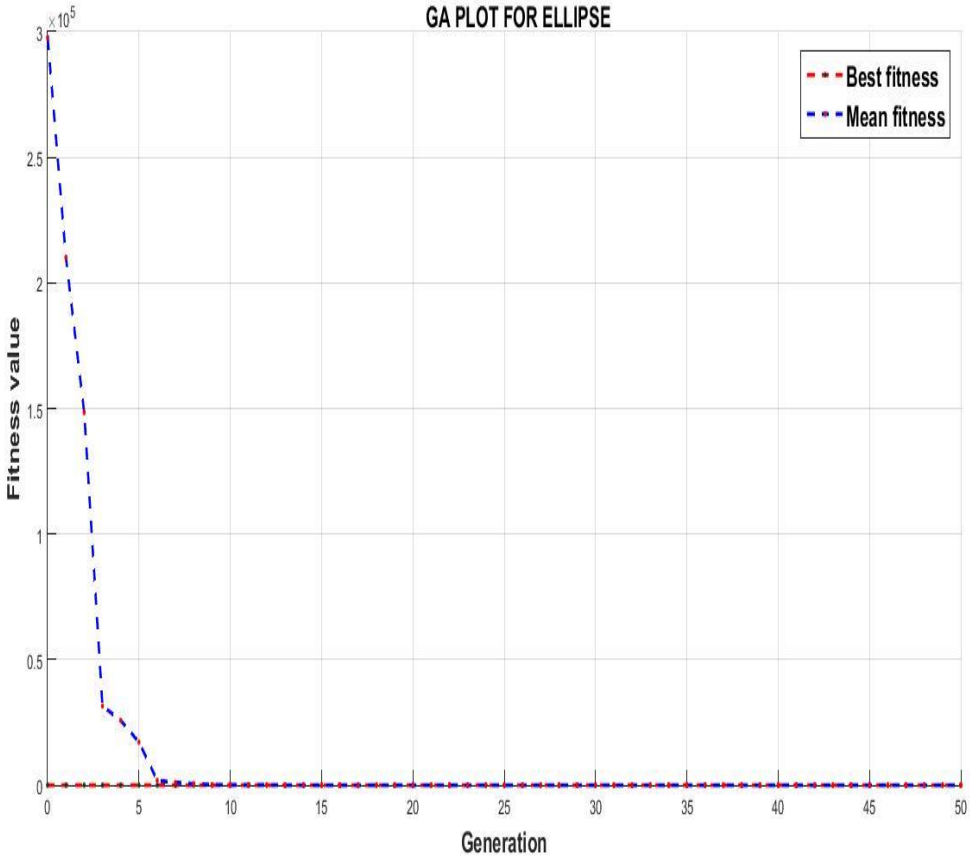

Figure 5.5: ELLIPSE Plot Using GA

CHAPTER 6

CHAPTER 6

PARAMETER VARIATION IN PSO AND GA

6.1 PARAMETER VARIATION IN PSO

Different parameters play a vital role in the performance of the algorithm. A slight variation in the parameter can lead to a large difference in the final solution. The most important parameters which define this in the Particle Swarm optimization are population size and the number of iterations these are discussed in the next section.

6.2 POPULATION SIZE

Population size : This is the number of particles that are present in the population. A rite amount of population size is required for the outcome to be acceptable.
Example: If we use too less population the particles may not converge as shown in the figure 6.1

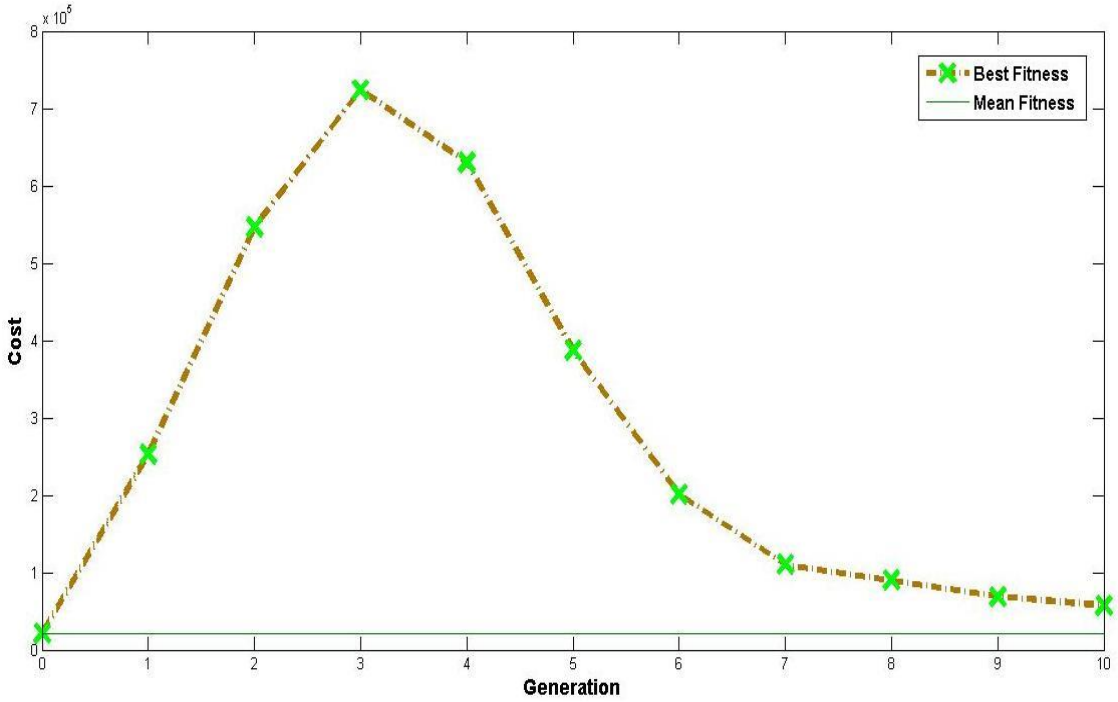

Figure 6.1: Population size reduced

Now, if we use a large population size, the amount of time required to converge to the minima is more hence the delay is increased.
Hence to produce the result in optimum time and also to produce the correct result that is converge at the minima we need to use a population size in between 200-500.

6.3 NUMBER OF ITERATIONS

Number of Iterations : This is another important parameter which decides the convergence of particles at the local minima.
Example: If we use a small number of iterations we encounter the same error caused when we used less number of populations. This is shown in figure 6.2.

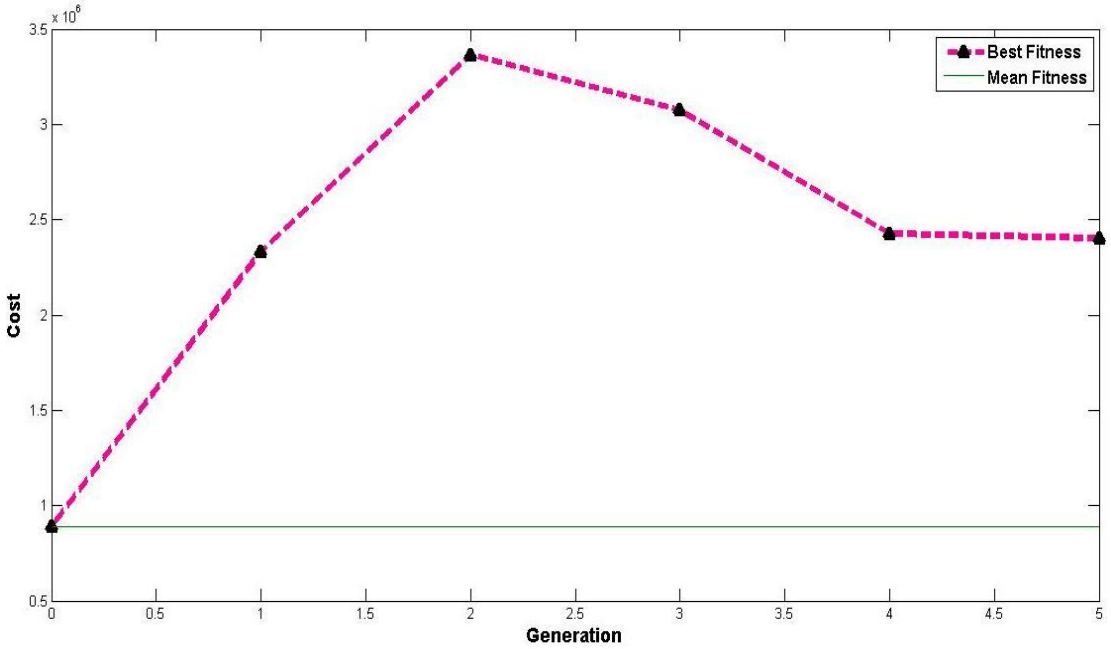

Figure 6.2: Number of iterations reduced

To avoid these problems we use an optimal number of iterations in between 100-200.

6.4 COEFFICIENTS IN THE VELOCITY UPDATE EQUATION

The coefficients that are present in the velocity update equation are c1 , c2 and C.
c1 : the value of c1 should be very small when compared to the value of c2 and c. It should be in between 0.01 - 0.5. If the value of greater than 0.5 there is large amount of distortion present in the output and the particles do not converge at the local minima.

c2 : the value of c2 should be in between 0.1 - 0.9. A larger value indicates larger multiplicative factor hence the distortion is more.

C : the value of C is maintained at 1. If this value is below or above 1 the system behaves erratically.

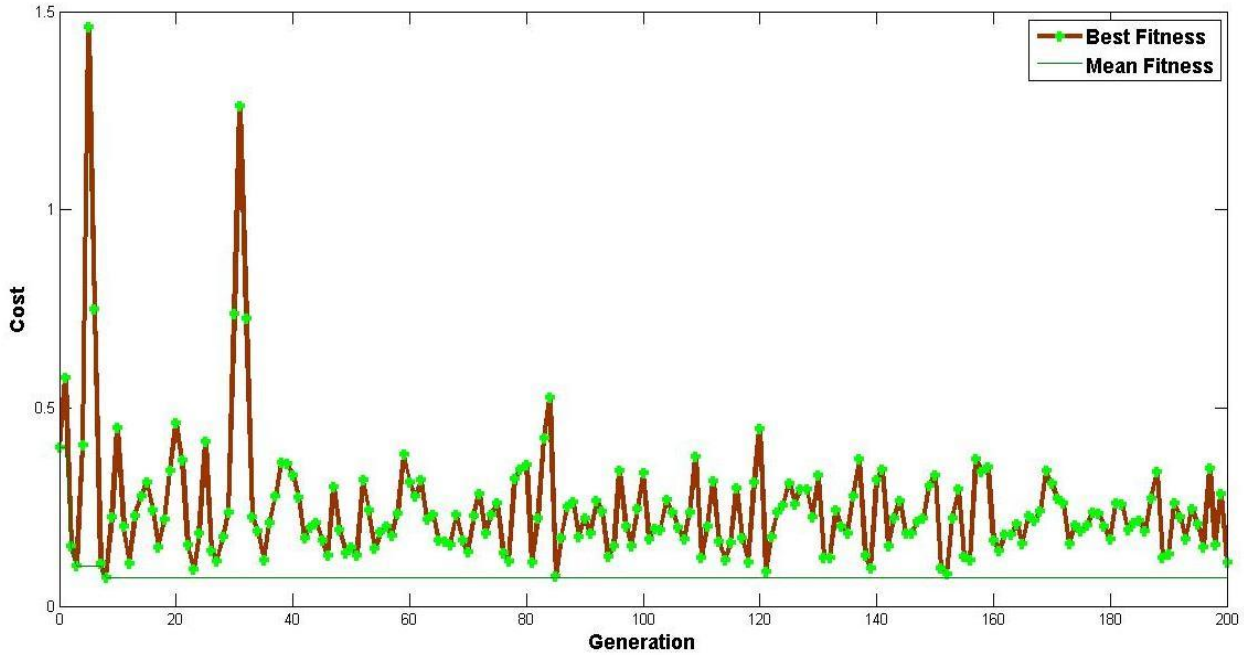

Figure 6.3: c1=0.9, c2=1.5, C=2

6.5 PARAMETER VARIATION IN GA

Crossover Fraction

Crossover fraction specifies the fraction of the next generation that crossover produces. Mutation produces the remaining individuals in the next generation. Set Crossover fraction to be a fraction between 0 and 1.
If this value is greater than 0.8 then the result varies. This can be seen in the figure 6.5

Stopping Criteria

Stopping criteria determines what causes the algorithm to terminate.
Generations : specifies the maximum number of iterations the genetic algorithm performs. If we set the Generations field to be less than 10 then the result will not be very accurate.

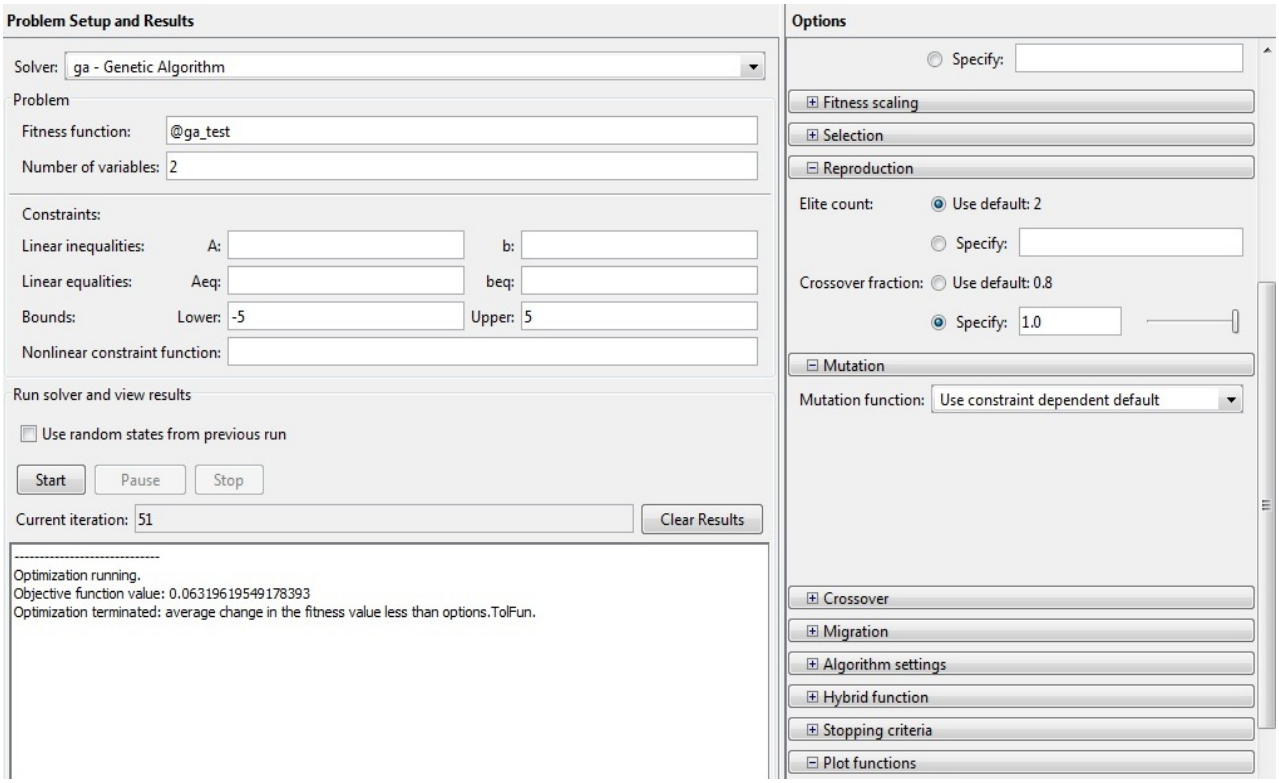

Figure 6.4: Crossover Fraction = 1

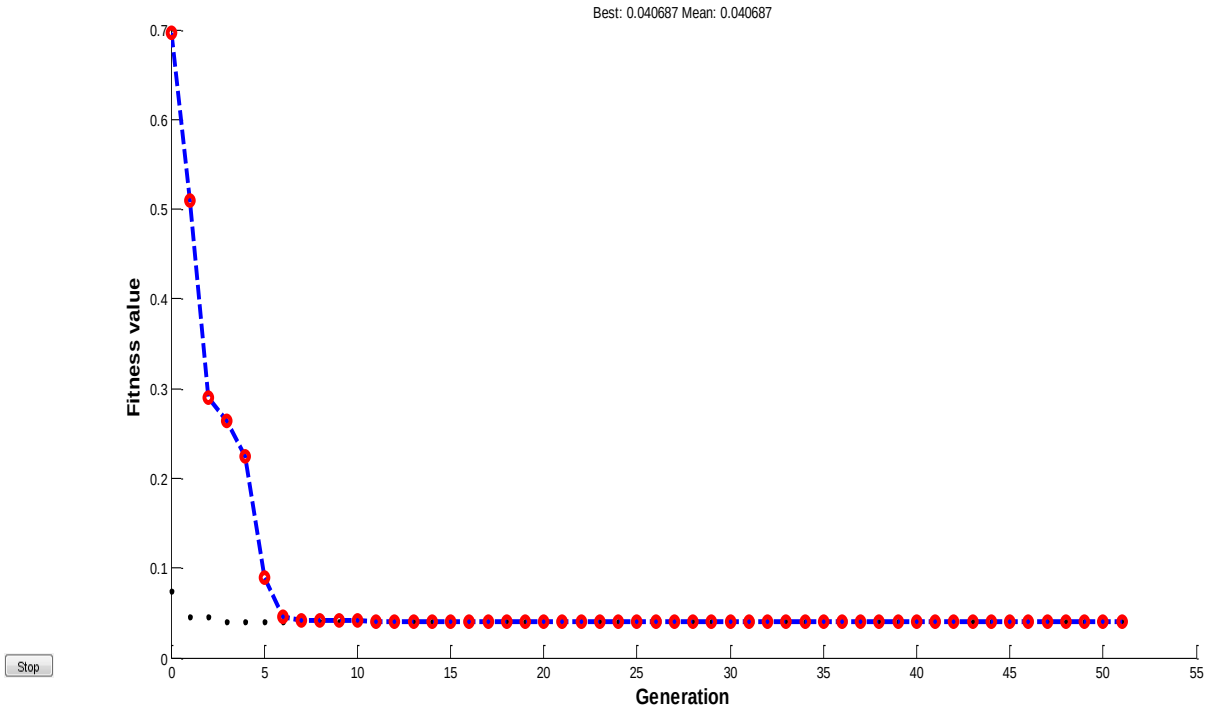

Figure 6.5: Plot of Best fitness for crossover fraction = 1

CHAPTER 7

CHAPTER 7

PROJECT PLANNING

7.1 GANTT CHART

A **Gantt chart** is a type of bar chart, adapted by Karol Adamiecki in 1896 and independently by Henry Gantt in the 1910s, that illustrates a project schedule. Gantt charts illustrate the start and finish dates of the terminal elements and summary elements of a project. Terminal elements and summary elements comprise the work breakdown structure of the project.

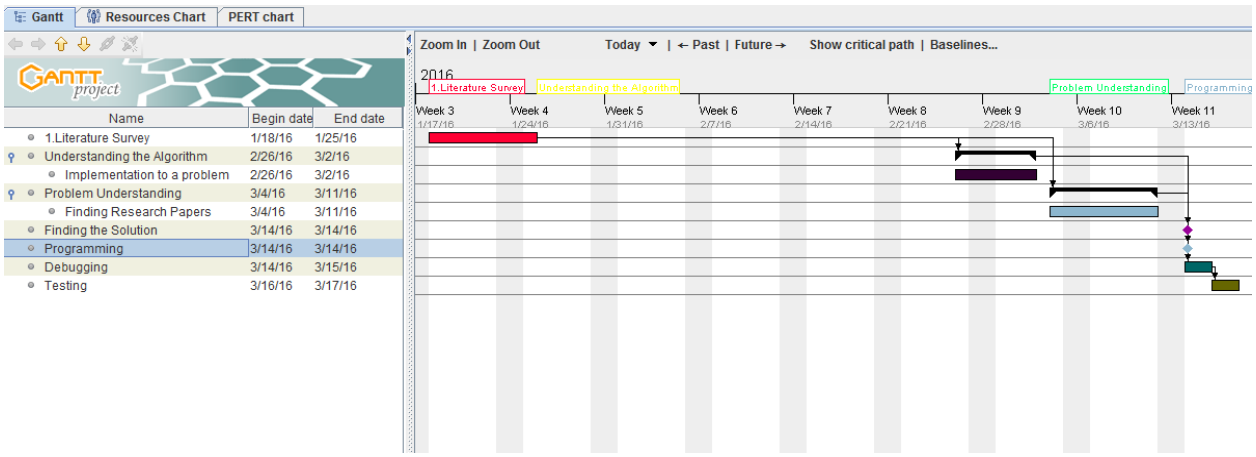

Figure 7.1: Gantt Chart

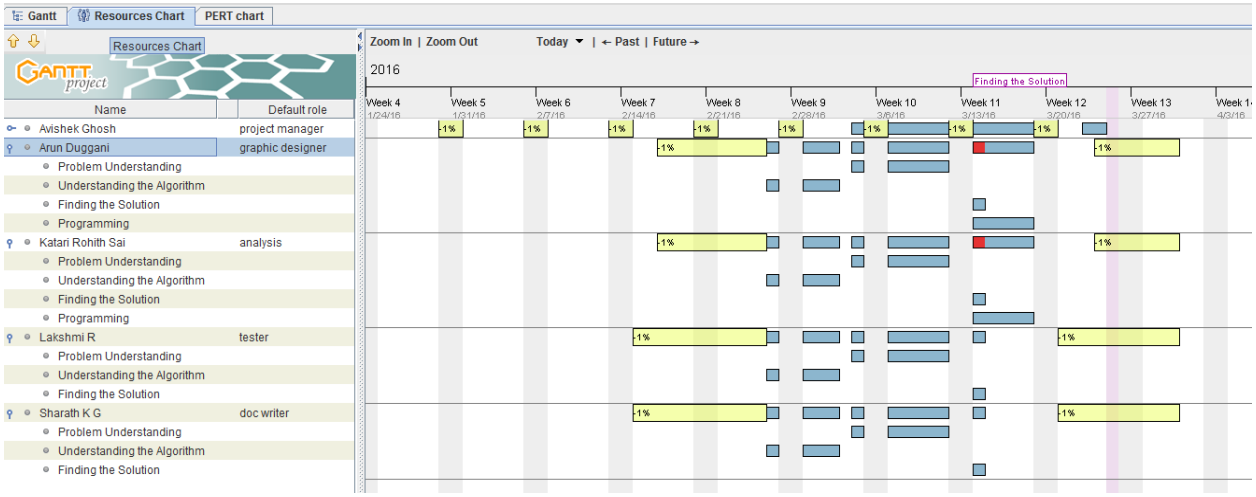

Figure 7.2 Resources Chart

1. Literature Survey

The Literature survey :

- uses a descriptive writing approach
- describes the existing and established theory and research in your report area by providing a context for your work.
- can show where you are filling a perceived gap in the existing theory or knowledge
- can propose something that goes against or is controversial to existing ideas.
- accurately references all sources mentioned in the survey and gives a full citation in the Reference List.

2. Understanding the Algorithm

This includes a thorough knowledge about working of Particle Swarm optimization and Genetic Algorithm.

3. Understanding the Problem and Finding the solution

This includes given any objective function how to go about solving it. If it is a complex function then it cannot be solved theoretically, hence it has to be solved using any of the two algorithms.
Hence the optimum solution for the objective function is found out

4. Programming and Debugging

During this part code for the Particle Swarm Optimization is written and the working of the Optimization Toolbox for Genetic Algorithm is understood and written.

5. Testing

This is done using the Benchmark Functions to test the algorithm and the results are compared with the function plot.

7.2 ADVANTAGES

1. **Clarity** : One of the biggest benefits of a Gantt chart is the tool's ability to boil down multiple tasks and timelines into a single document. Stakeholders throughout an organization can easily understand where teams are in a process while grasping the ways in which independent elements come together toward project completion.
2. **Communication** : Teams can use Gantt charts to replace meetings and enhance other status updates. Simply clarifying chart positions offers an easy, visual method to help team members understand task progress.
3. **Motivation** : Some teams or team members become more effective when faced with a form of external motivation. Gantt charts offer teams the ability to focus work at the front of a task timeline, or at the tail end of a chart segment. Both types of team members can

4. find Gantt charts meaningful as they plug their own work habits into the overall project schedule.
5. **Coordination** : For project managers and resource schedulers, the benefits of a Gantt chart include the ability to sequence events and reduce the potential for overburdening team members. Some project managers even use combinations of charts to break down projects into more manageable sets of tasks.
6. **Time Management** : Most managers regard scheduling as one of the major benefits of Gantt charts in a creative environment. Helping teams understand the overall impact of project delays can foster stronger collaboration while encouraging better task organization.
7. **Efficiency** : Another one of the benefits of Gantt charts is the ability for teams members to leverage each other's deadlines for maximum efficiency. For instance, while one team member waits on the outcome of three other tasks before starting a crucial piece of the assignment, he or she can perform other project tasks. Visualizing resource usage during projects allows managers to make better use of people, places, and things.

CHAPTER 8

CHAPTER 8

CONCLUSION

The main aim of the project was to understand and implement the Genetic Algorithm using the Matlab toolbox and the Particle Swarm Optimization Algorithm to optimize a give objective function. It also included to implement the benchmark functions using both Genetic Algorithm and Particle Swarm Optimization and find out the minima of these functions, hence comparing them with the actual results of the algorithms' output.

The above mentioned has been successfully implemented and the given functions haven been optimized, that is, the minima for the respective benchmark functions have been found. It is also compared with the algorithms' plots, which confirms that the algorithm correctly implemented and is ready to optimize any function.

We learn from this project that, Nature has its own way of solving any given problem and we need to incorporate these attributes of the nature in solving our real world problems. The Particle Swarm Optimization derived from flock of birds indicates that for a given objective function the particles converge to the local minima just like the birds converge to the bird's position which is closest to the food source.

In Genetic Algorithm we introduce the three parameters crossover, mutation and selection which is the basis of genetics to produce new offspring. These are adaptive to the new environment which produces the optimized solution to a given problem.

CHAPTER 9

CHAPTER 9

REFERENCES

1. Prof. de Weck and Prof. Willcox ,*Multidisciplinary System Design Optimization*, Massachusetts Institute of Technology
2. Goldberg DE, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, New York, 1989
3. Dr. Rajib Kumar Bhattacharjya , *Introduction To Genetic Algorithms,* IIT Guwahati.
4. Mir Asif Iquebal , *Genetic Algorithms And their Applications* , I.A.S.R.I. Library Avenue, New Delhi.
5. A.E Eiben and J.E. Smith , *Introduction to Evolutionary Computing genetic Algorithms*.
6. Darrell Whitley, *A Genetic Algorithm Tutorial*, Colorado State University
7. Link : https://en.wikipedia.org/wiki/Genetic_algorithm
8. Erik D. Goodman, *Introduction to Genetic Algorithms*, Michigan State University
9. Iztok Fister Jr., Xin-She Yang, Iztok Fister, Janez Brest and Dusan Fister , *A Brief Review of Nature-Inspired Algorithms for Optimization*, University of Maribor and University of Middlesex
10. Chih-Cheng Kao, *Applications of Particle Swarm Optimization*, Kao Yuan University
11. Link : https://en.wikipedia.org/wiki/Particle_swarm_optimization
12. Jaco F. Schutte, *The Particle Swarm Optimization Algorithm*
13. Video Lecture Link: https://www.youtube.com/watch?v=F3RKF3LE2Wo, *Particle Swarm Optimization*
14. Video Lecture Link : https://www.youtube.com/watch?v=cCfHiqup3NU, *Genetic Algorithm Matlab Toolbox*

CHAPTER 10

CHAPTER 10

APPENDIX

9.1 USER MANUAL

```
clear all
clc

ff = 'ga_test';
dim = 2;
c1 = 0.9;
c2 = 1.5;
C = 1;
maxiter = 200;
popsize = 500;
pop = rand(popsize,dim);
vel = rand(popsize,dim);
cost = feval(ff,pop);
pbcost = cost;
pb = pop;
z = min(cost);
globalmin = min(cost);
[pgcost,indx] = min(cost);
pg = pop(indx,:);

i = 0;

while i < maxiter
    i = i+1;
    w=(maxiter-i)/maxiter;
    r1 = rand(popsize,dim);
    r2 = rand(popsize,dim);
    vel = C*w*vel + c1 *r1.*(pb-pop) + c2*r2.*(ones(popsize,1)*pg-pop);
    pop = pop + vel;
    overlimit=pop<=5;
    underlimit=pop>=-5;
    pop=pop.*overlimit+not(overlimit);
    pop=pop.*underlimit;

    cost = feval(ff,pop);
    bettercost = cost < pbcost;
    pbcost = pbcost.*not(bettercost) + cost.*bettercost;
    pb(bettercost,:) = pop(bettercost,:);
    [temp, t] = min(pbcost);

    if temp<pgcost
        pg=pop(t,:);
        indx=t;
        pgcost=temp;
    end

    [i pg pgcost]
```

```
    z(i+1)= min(cost)

globalmin(i+1)=pgcost;

end

disp(globalmin)
i=0:length(z)-1;
plot(i,z,i,globalmin);
xlabel('generation');ylabel('cost');
```