A
DISSERTATION
ON

# “A comparative study of CSA & GA by multi-modal optimization ”

*Submitted under the partial fulfillment of the requirement*
*for the awardof the degree of*

**MASTER OF TECHNOLOGY**
in
Software Engineering

**Submitted by:**
**Vipin Kumar Sharma**
Roll no.: 19/SE/09
Registration no.: 11/MT/SE/FT

**Under the guidance of:**
**Mr. Shailender Kumar**
Asst. Professor,
Deptt. of Computer Engineering
Delhi Technological University
Bawana Road, Delhi – 110042


**DEPARTMENT OF COMPUTER ENGINEERING**
**DELHI TECHNOLOGICAL UNIVERSITY**
**DELHI, 2011**

# CERTIFICATE


DELHI TECHNOLOGICAL UNIVERSITY
BAWANA ROAD, DELHI – 110042

This is to certify that the work contained in this dissertation entitled **"A comparative study of CSA & GA by multi-modal optimization"** submitted by **Vipin Kumar Sharma,** Roll no. 19/SE/09 in partial fulfillment of the requirement for the award of the degree of Master of Technology in software Engineering at Delhi Technological, Delhi is a record of the candidate's own work carried out by him under my guidance and supervision in the academic year 2010-2011.

**Mr. Shailender Kumar**
Asst. Professor
Project Guide
Delhi Technological University
Delhi

# ACKNOWLEDGEMENT

It is a great pleasure to have the opportunity to extend my heartiest felt gratitude to everybody who helped me throughout the course of this project.

I take this opportunity to express my deep sense of gratitude and indebtedness to my learned supervisor **Mr. Shailender Kumar, Asst. Professor, Dept. of COE, DTU,** for his invaluable guidance, encouragement and patient reviews. With his continuous inspiration only, it has become possible to complete this dissertation. He helped in pointing out places in several drafts of the thesis where clarity could be improved and claims made more precise.

I also extend my gratitude towards **Dr. Daya Gupta, Head, Dept. of COE, DTU** who has always been cooperative throughout the whole coursework and gave us valuable inputs. I am also thankful to all other faculty members and staff of the Department of Computer Engineering at Delhi Technological University for sharing their knowledge and experiences with me as well as for their kind support.

I also like to thank my batch mates at Delhi Technological University for sharing their ideas and opinions on several topics that were important for my work. I also owe gratitude towards my parents for their patience and support. They have been always around to cheer me up in the odd times of this work.

**Vipin Kumar Sharma**

# ABSTRACT

In the last few years we perceived a great increase in interest in studying biologically inspired systems. Among these, we can emphasize *artificial neural networks*, *evolutionary computation*, *DNA computation*, and now *artificial immune systems*. This work discusses immune algorithm and genetic algorithm, the two classes of algorithms at the forefront of artificial systems inspired by human body mechanism. We then move on to compare these two classes of algorithms on the parameters of population, generation and clone sizes. This comparison will help in the analysis of feasibility of these algorithms for specific purposes.

The immune system is a complex of cells, molecules and organs which has proven to be capable of performing several tasks, like pattern recognition, learning, memory acquisition, generation of diversity, noise tolerance, generalization, distributed detection and optimization. Based on immunological principles, new computational techniques are being developed, aiming not only at a better understanding of the system, but also at solving engineering problems. We discuss the main strategy used by the immune system to problem solving, and introduce the concept of *immune engineering*. The *immune engineering* makes use of immunological concepts in order to create tools for solving demanding machine-learning problems using information extracted from the problems themselves.

A genetic algorithm (GA) is a search heuristic that mimics the process of natural evolution. This heuristic is routinely used to generate useful solutions to optimization and search problems. Genetic algorithms belong to the larger class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover. A genetic algorithm (GA) is a search heuristic that mimics the process of natural evolution. This heuristic is routinely used to generate useful solutions to optimization and search problems. Genetic algorithms belong to the larger class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.

# List of Figures

List of tables

# Table of contents

# CHAPTER 1: INTRODUCTION

## 1.1 Introduction

Brains inspired computational techniques are known as neural networks. Immune systems have also (much more recently) inspired computational techniques. Before we can look at artificial immune systems and genetic algorithms we need some knowledge of how our immune systems and genetic algorithms work.

### 1.1.1 Immune algorithms

**Immune algorithms and genetic algorithms** are two important technique for function optimization which is based on as our human body work .

The **Immune System** is a complex of cells, molecules and organs that represent an identification mechanisms capable of perceiving and combating dysfunction from our own cells(infectious self) and the action of exogenous infectious microorganisms(infectious no self) [1]. The interaction among the Immune System and several other systems and organs allows the regulation of the body, guaranteeing its stable functioning. The system is capable of **"remembering"** each infection, so that a second exposure to the same pathogen is dealt with more efficiently [2] [5].

The **Immune Engineering** is a meta-synthesis process that uses the information contained in the problem itself to define the solution tool to a given problem, and then apply it to obtain the problem solution. Nature provides many examples of systems with simple basic components, in which a collective competition and cooperation turns out to an extremely complex overall behavior, e.g., insects colonies (like ants), the immune system, etc. One of the most striking characteristics of such systems is their robustness, expressed as a high tolerance to small perturbations to individual components. This **robustness** underlies the principles of **distribution**, where small pieces are put together as an ensemble of individuals (or agents), very complex behaviors can emerge. Conventional engineering techniques usually require detailed

specification of the precise behavior of each of the components of the systems. On the other hand, the engineering paradigm(immune engineering) demands only general, or approximate, specifications of some aspects of the overall behavior of the system, like a performance measure or a fitness function[18,19].

A new field of research called **Artificial Immune System** has arisen; the properties of the immune system are of great interest for computer scientist and engineers:

- Imperfect Detection: An absolute recognition of the pathogens is not required, hence the system is flexible.
- Reinforcement learning and memory: The system can 'learn' the structures of pathogens, so that future responses to the same pathogens are faster and stronger.
- Anomaly Detection: The immune system can detect and react to pathogens that the body has never encountered before.
- Uniqueness: Each individual possesses its own immune system, with its particular vulnerabilities and capabilities[18]

The topics involved in the definition and development of the artificial immune system covers mainly

- Hybrid structures and algorithm that take into account immune-like mechanisms.
- Computational algorithms based on immunological principles, like distributed processing, clonal selection algorithm, and immune network theory.
- Immunity-based optimization, learning, self organization, artificial life, cognitive models, multi agent systems, design and scheduling, pattern recognition and anomaly detection.
- Immune engineering tools.

### 1.1.2 Terminology in immune algorithms

- ***Pathogen*:** a foreign invader such as a virus, bacterium, fungus, or parasite.
- ***B cell, helper T cell, killer T cell, macrophage, memory cell, plasma cell:*** the main cells in our adaptive immune system (also known as white blood cells).
- ***Stem cells*:** general purpose cells in our bone marrow that make all the cells in our blood.
- ***Self cells:*** all the normal cells that make up 'self' (you).
- ***Lymphatic network:*** the collection of lymph vessels throughout our bodies that collect "leakage" of blood from the capillaries.
- ***Lymph nodes*:** small organs under our arms, chins, chest and groin that are used as "security stations".
- ***Thymus:*** an organ in which B cells that produce antibodies harmful to self cells are removed.
- ***Antibody*:** a protein made by B cells to mark pathogens as harmful.
- ***Antigen:*** a protein on the surface of pathogens that is used by other cells for identification.
- ***Gene library:*** evolved fragments of DNA within each B cell used as building blocks to produce its antibody[8] [9].

### 1.1.3 Genetic algorithms

**Genetic algorithms** are a part of **evolutionary computing**, which is a rapidly growing area of artificial intelligence. As you can guess, genetic algorithms are inspired by Darwin's theory about evolution. Simply said, solution to a problem solved by genetic algorithms is evolved.

Algorithm is started with a **set of solutions** (represented by **chromosomes**) called **population**. Solutions from one population are taken and used to form a new population. This is motivated by

a hope, that the new population will be better than the old one. Solutions which are selected to form new solutions (**offspring**) are selected according to their fitness - the more suitable they are the more chances they have to reproduce.

Genetic Algorithms are good at taking larger, potentially huge, search spaces and navigating them looking for optimal combinations of things and solution which we might not find in lifetime.

The most important aspects of using GA are:

1. definition of Objective function
2. definition and implementation of genetic representation
3. definition and implementation of genetic operators.

### 1.1.3 Terminology in genetic algorithm

- ***Individuals*** *:* An *individual* is any point to which you can apply the fitness function. The value of the fitness function for an individual is its score.
- ***Populations and Generations****:* A *population* is an array of individuals. For example, if the size of the population is 100 and the number of variables in the fitness function is 3, you represent the population by a 100-by-3 matrix.
- ***Diversity :*** *Diversity* refers to the average distance between individuals in a population. A population has high diversity if the average distance is large; otherwise it has low diversity
- ***Fitness Values and Best Fitness Values :*** The *fitness value* of an individual is the value of the fitness function for that individual. Because the toolbox finds the minimum of the fitness function, the *best* fitness value for a population is the smallest fitness value for any individual in the population.

- ***Parents and Children :*** To create the next generation, the genetic algorithm selects certain individuals in the current population, called *parents*, and uses them to create individuals in the next generation, called *children.*
- ***Fitness function :*** The *fitness function* is the function you want to optimize. For standard optimization algorithms, this is known as the objective function.[10]

## 1.2 Problem statement

In this work, we aim to compare and analyze the results of immune algorithms and genetic algorithms, according to population size and generation size. This will help us evaluate and ascertain the better algorithm based on these parameters. This work aims to establish which algorithm gives better optimization and batter stability with same parameters .

We have developed a tool in C#.Net on windows platform that implements these algorithms and makes a comparison of these two using Zed-Graph. This tool will therefore help people compare the results and they may use it in their work.

In immune algorithms we implement the Colonial Selection Algorithm that describes the basic features of an immune response to an antigenic stimulus. It establishes the idea that only those cells that recognize the antigens proliferate, thus being selected against those which do not. The selected cells are subject to an affinity maturation process, which improves their affinity to the selective antigens. This Clonal Selection Principle describes the adaptive immune system which recognizes and responds to the stimuli[6] [7]. It takes into account the affinity maturation of the immune response. The algorithm is shown to be capable of solving complex machine-learning tasks, like pattern recognition and multi-modal optimization.

In genetic algorithm we implement simple algorithms approach. In this work we take data set do genetic operation over those such as crossover, mutation, and  than apply selection algorithms according to their fitness the element which have fitness above a threshold is selected form the next generation.

## 1.3 Motivation

From last two decade human nature based algorithms becoming very popular there are many (thousands) of algorithms proposed from than but some algorithms have their own advantage according to situation and parameter so the people who will use these algorithms should knows pros and con of the algorithms. In this work we will compare the two algorithms that will help the people to evaluate the performance of these algorithms and give a batter choice to select the algorithms according to their need.

## 1.4 Organization of thesis

The thesis organized as follows: In first chapter we provide introduction and motivation work. In chapter 2, we present literature survey and some basic background of the algorithms . in chapter 3, we explore the optimization technique which is used in algorithms. In chapter 4, we present the CSA &GA algorithms and their implementation. In chapter 5 , we depicted results and their comparison. In chapter 6 and 7, we present conclusion and future scope of algorithms.

# CHAPTER 2: LITERATURE REVIEW

## 2.1 Immune algorithm

Physics, biology, economy or sociology often has to deal with the classical problem of optimization. Economy particularly has become specialist of that field. Generally speaking a large part of mathematical development dealt with this topic. Purely analytical methods widely proved their efficiency. They nevertheless suffer from a insurmountable weakness.

At the beginning of a run of a **genetic algorithm** a large population of chromosomes is created. Each one when decode will represent a different solution to the problem at hand. Let's say there are N chromosomes in the initial population. Then the following steps are repeated until a solution is found[18].

- Test each chromosome to see how good it is at solving the problem at hand and assign a **fitness** score accordingly. The fitness score is a measure of how good that chromosome is at solving the problem to hand.
- Select two members from the current population. The chance of being selected is proportional to the **chromosomes fitness.** Roulette wheel selection is the commonly used method
- Dependent on the **cross over rate**, cross over the bits from each chosen chromosome at a randomly chosen point.
- Step through the chosen chromosomes bits and flip dependent on the **mutation rate**.
- Repeat step 2, 3, 4 until a new population of N member has been created.

The above algorithm is user to compare the result of Clonal Selection Algorithm, it help us to find out which one gives better result over the generations. It help us to explore which algorithm gets the population polarize the whole population of individuals towards the best one. The clonal selection is discussed below, the algorithm of clonal selection is

discussed below in general, it should be noted that the algorithm related to multi model optimization would be discussed in the following section not in this section.

The clonal principle is used by the immune system by the immune system to describe the basic features of an immune response to an antigenic stimulus. It establishes the idea that only those cells that recognize the antigen proliferate, thus being selected against those which do not.

The algorithm works as follows:

- Generate a set (P) of candidates solutions, composed of the subset of memory cells (M) added to the remaining (Pr) population (P = Pr+M).
- Determine the n best individuals Pn of the population P, based on an affinity measure.
- Clone(reproduce) these n best individuals of the population, giving rise to a temporary population of clones (C). The clone size is a increasing function of the affinity measure of the antigen.
- Submit the population of clones to a hyper mutation is proportional to the affinity of the antibody. A matured antibody population is generated (C*).
- Re-select the improved individuals from C* to compose the memory set. Some members of the P set can be replaced by other improved members if C*.
- Replace d low affinity antibodies of the population, maintaining its diversity[1,7].

In Negative Selection Algorithm , the algorithm try to find out the maximum number of antigens with the help of antibodies.

The algorithm as follows:

- A robust system should detect any foreign activity rather than looking for specific known patterns of intrusion.
- No prior knowledge of anomaly (non-self) is required.
- The size of the detectors set does not necessarily increase with the number of strings being protected

In the Immune System learning involves raising the population size and affinity of those lymphocytes that have proven them to be valuable by having recognized some antigen. Because the total number of lymphocytes in the immune system is regulated, increase in the sizes of some clones may have to decrease in size. However, the total number of lymphocytes is not kept absolutely constant. If the immune systems learns only by increasing the population sizes of specific lymphocytes, it must either "forget" previously learned antigens, increasing in size, or constantly decrease the portion of its repertoire that is generated at random and responsible for responding to novel antigens. It is important to remark that under an engineering perspective, the cells with highest affinity must be preserved somehow as high quality candidate solutions, and shall only be replaced by improved candidates, based on statistical evidences.

Immune learning and memory are acquired through

- Repeated exposure to a pathogen.
- Affinity maturation of the receptor molecules.
- Low grade chronic infection.
- Cross-reactive to endogenous and exogenous pathogens.
- Idiotypic networks.

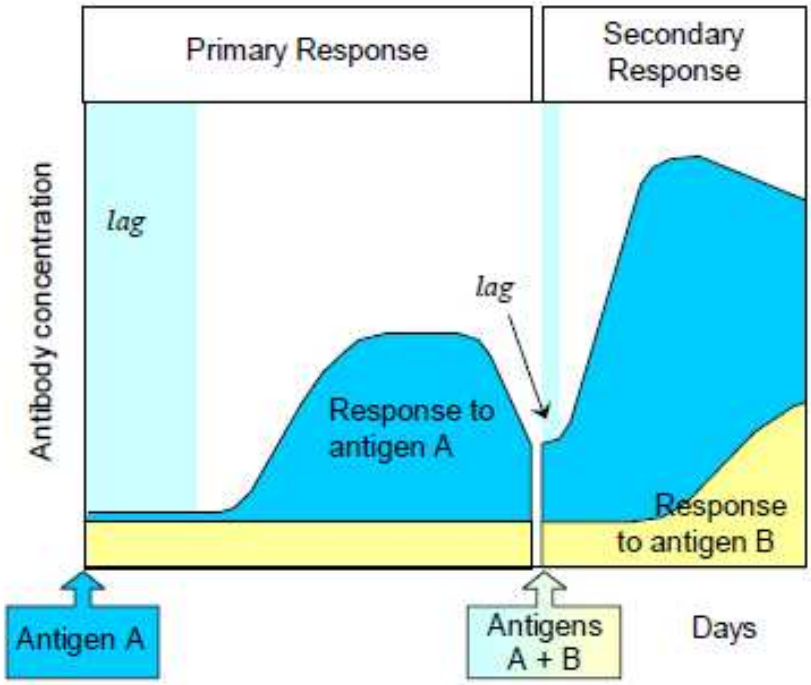



**Figure 2.1:** Primary, secondary and cross-reactive immune responses

## 2.1.1 CLONAL SELECTION PRINCIPLE

By considering the above theory, **Clonal Selection Algorithm** has been implemented in to computer science terms which would help in computing the multi model optimization.
The Clonal Sectional Algorithm is as follows.

- The new cells are copies of their parents(clone) subjected to a mutation mechanism with high rates(somatic hyper mutation)
- Elimination of newly differentiated lymphocytes carrying self-reactive receptors
- Proliferation and differentiation on contact mature cells with antigens
- The persistence of forbidden clones, resistant to early elimination by self-antigens, as the basis of autoimmune disease.

The Clonal Selection Algorithm is compared with **Genetic Algorithm** to find out which one gives better results.

For each cellular component in the lymphoid system we can consider three classes of repertoire

- The potential repertoire, determined by the number, structure and mechanisms of expression of germ-line collection of genes.
- The available repertoire defined as the set of diverse molecules that are used as the lymphocytes receptors.
- The actual repertoire, that set of antibodies and receptors produced by effectors lymphocytes activated in the internal environment.

The main factors that result in the repertoire completeness are its diversity (obtained by mutation, editing and gene rearrangement) it’s cross reactivity and its multi-specificity.
By considering the above affinity maturation Negative Selection Algorithm has been implemented which would help in binary character recognition .

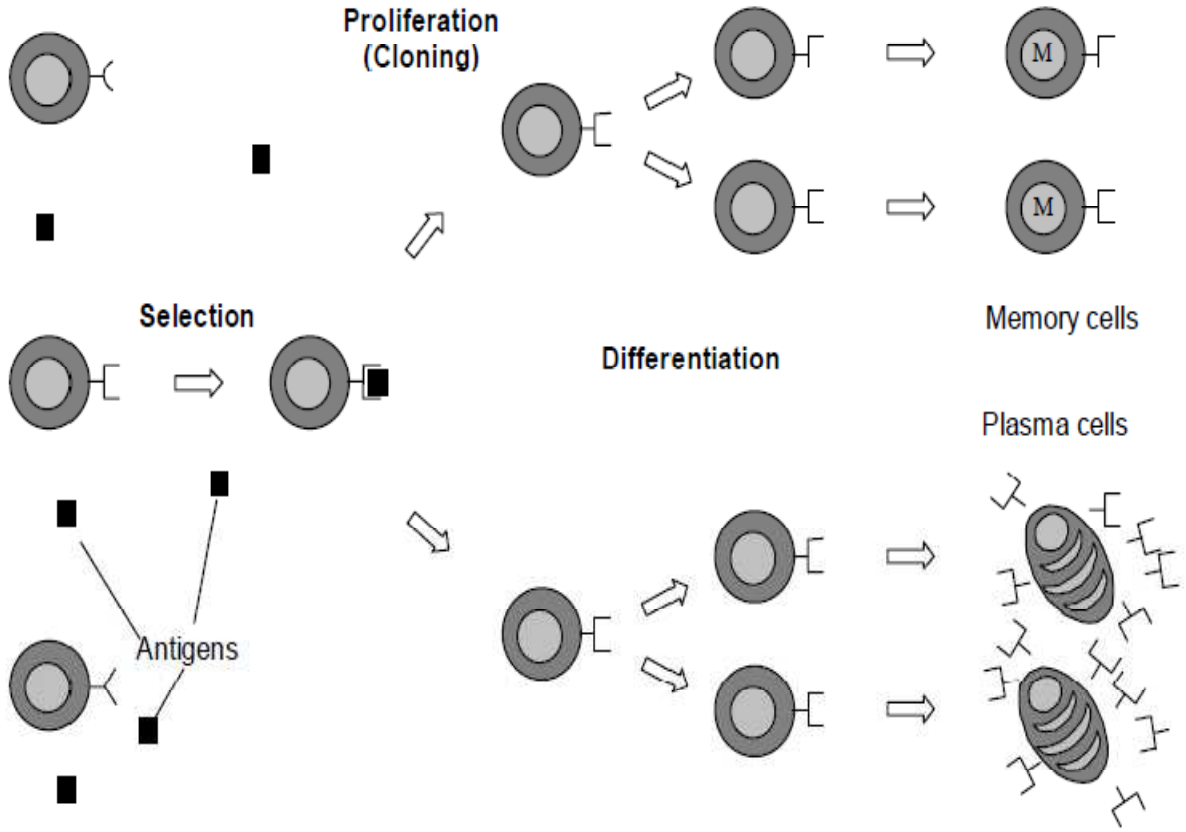



**Figure 2.2:** The clonal selection principle. Small resting B cells created in the bone marrow each carry a different receptor type defined by their VH and VL regions. Those cells carrying receptors specific for the antigen, proliferate and differentiate into plasma and memory cells.

## 2.2 Genetic algorithm

All living organisms consist of cells. In each cell there is the same set of **chromosomes**. Chromosomes are strings of DNA and serves as a model for the whole organism. A chromosome consists of **genes**, blocks of DNA. Each gene encodes a particular protein. Basically it can be said, that each gene encodes a **trait**, for example color of eyes. Possible settings for a trait (e.g. blue, brown) are called **alleles**. Each gene has its own position in trestle chromosome. This position is called **locus**.

Complete set of genetic material (all chromosomes) is called **genome**. Particular set of genes in genome is called **genotype**. The genotype is with later development after birth base for the organism's **phenotype**, its physical and mental characteristics, such as eye color, intelligence etc[22].

### 2.2.1 Reproduction

During reproduction, first occurs **recombination** (or **crossover**). Genes from parents form in some way the whole new chromosome. The new created offspring can then be mutated. **Mutation** means, that the elements of DNA are a bit changed. This changes are mainly caused by errors in copying genes from parents. The **fitness** of an organism is measured by success of the organism in its life[22].

### 2.2.2 Search Space

If we are solving some problem, we are usually looking for some solution, which will be the best among others. The space of all feasible solutions (it means objects among those the desired solution is) is called **search space** (also state space). Each point in the search space represents one feasible solution. Each feasible solution can be "marked" by its value or fitness for the problem. We are looking for our solution, which is one point (or more) among feasible solutions - that is one point in the search space.

The looking for a solution is then equal to a looking for some extreme (minimum or maximum) in the search space. The search space can be whole known by the time of solving a problem, but usually we know only a few points from it and we are generating other points as the process of finding solution continues.

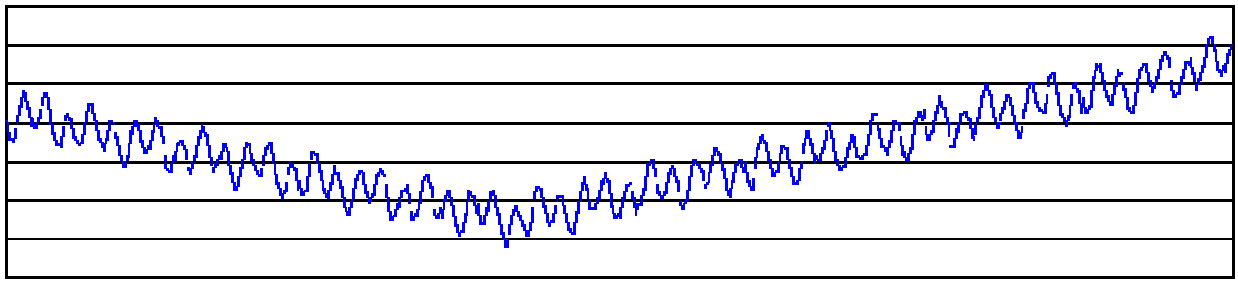



**Figure 2.3:** *Example of a search space*

The problem is that the search can be very complicated. One does not know where to look for the solution and where to start. There are many methods, how to find some **suitable solution** (i.e. not necessarily the **best solution**), for example **hill climbing**, **tabu search**, **simulated annealing** and **genetic algorithm**. The solution found by this methods is often considered as a good solution, because it is not often possible to prove what is the real optimum[22].

### 2.2.3 Operators of GA

## Overview

As you can see from the genetic algorithm outline, the crossover and mutation are the most important part of the genetic algorithm. The performance is influenced mainly by these two operators. Before we can explain more about crossover and mutation, some information about chromosomes will be given.

## Encoding of a Chromosome

The chromosome should in some way contain information about solution which it represents. The most used way of encoding is a binary string. The chromosome then could look like this:

| | |
|---|---|
| Chromosome 1 | 1101100100110110 |
| Chromosome 2 | 1101111000011110 |

Each chromosome has one binary string. Each bit in this string can represent some characteristic of the solution. Or the whole string can represent a number.

Of course, there are many other ways of encoding. This depends mainly on the solved problem. For example, one can encode directly integer or real numbers, sometimes it is useful to encode some permutations and so on.

## Crossover

After we have decided what encoding we will use, we can make a step to crossover. Crossover selects genes from parent chromosomes and creates a new offspring. The simplest way how to do this is to choose randomly some crossover point and everything before this point copy from a first parent and then everything after a crossover point copy from the second parent.

Crossover can then look like this ( | is the crossover point):

| | |
|---|---|
| Chromosome 1 | 11011 \| 00100110110 |
| Chromosome 2 | 11011 \| 11000011110 |
| Offspring 1 | 11011 \| 11000011110 |

| | |
|---|---|
| Offspring 2 | 11011 \| 00100110110 |

There are other ways how to make crossover, for example we can choose more crossover points. Crossover can be rather complicated and very depends on encoding of the encoding of chromosome. Specific crossover made for a specific problem can improve performance of the genetic algorithm[22].

## Mutation

After a crossover is performed, mutation takes 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 can then be following:

| | |
|---|---|
| Original offspring 1 | 1101111000011110 |
| Original offspring 2 | 1101100100110110 |
| Mutated offspring 1 | 1100111000011110 |
| Mutated offspring 2 | 1101101100110110 |

The mutation depends on the encoding as well as the crossover. For example when we are encoding permutations, mutation could be exchanging two genes[22].

### 2.2.4 Parameters of GA

## Crossover and Mutation Probability

There are two basic parameters of GA - crossover probability and mutation probability.

**Crossover probability** says how often will be crossover performed. If there is no crossover, offspring is exact copy of parents. If there is a crossover, offspring is made from parts of parents' chromosome. If crossover probability is **100%**, then all offspring is made by crossover. If it is **0%**, whole new generation is made from exact copies of chromosomes from old population (but this does not mean that the new generation is the same!). Crossover is made in hope that new chromosomes will have good parts of old chromosomes and maybe the new chromosomes will be better. However it is good to leave some part of population survive to next generation[22].

**Mutation probability** says how often will be parts of chromosome mutated. If there is no mutation, offspring is taken after crossover (or copy) without any change. If mutation is performed, part of chromosome is changed. If mutation probability is **100%**, whole chromosome is changed, if it is **0%**, nothing is changed. Mutation is made to prevent falling GA into local extreme, but it should not occur very often, because then GA will in fact change to **random search**.

### 2.2.5 Other Parameters

There are also some other parameters of GA. One also important parameter is population size.

**Population size** says how many chromosomes are in population (in one generation). If there are too few chromosomes, GA has a few possibilities to perform crossover and only a small part of search space is explored. On the other hand, if there are too many chromosomes, GA slows down. Research shows that after some limit (which depends mainly on encoding and the problem) it is not useful to increase population size, because it does not make solving the problem faster.

### 2.2.6 Selection

## Introduction

As you already know from the GA outline, chromosomes are selected from the population to be parents to crossover. The problem is how to select these chromosomes. According to Darwin's evolution theory the best ones should survive and create new offspring. There are many methods how to select the best chromosomes, for example roulette wheel selection, Boltzman selection, tournament selection, rank selection, steady state selection and some others.

Some of them will be described in this chapter.

## Roulette Wheel Selection

Parents are selected according to their fitness. The better the chromosomes are, the more chances to be selected they have. Imagine a **roulette wheel** where are placed all chromosomes in the population, everyone has its place big accordingly to its fitness function, like on the following picture.

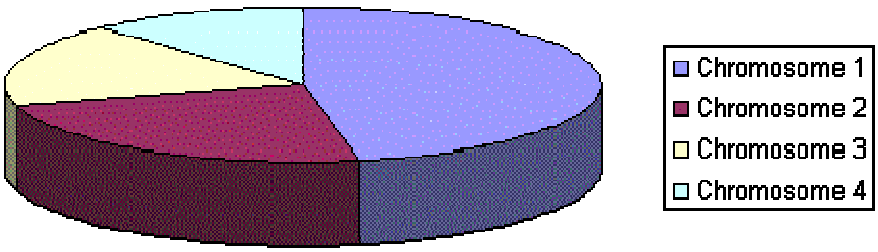



**Figure 2.4: Roulette Wheel Selection**

Then a marble is thrown there and selects the chromosome. Chromosome with bigger fitness will be selected more times[22].

This can be simulated by following algorithm.

1. **[Sum]** Calculate sum of all chromosome fitnesses in population - sum ***S***.
2. **[Select]** Generate random number from interval ***(0,S)*** - ***r***.
3. **[Loop]** Go through the population and sum fitnesses from ***0*** - sum ***s***. When the sum ***s*** is greater then ***r***, stop and return the chromosome where you are.

Of course, step **1** is performed only once for each population.

## Rank Selection

The previous selection will have problems when the fitnesses differ very much. For example, if the best chromosome fitness is 90% of the entire roulette wheel then the other chromosomes will have very few chances to be selected.

Rank selection first ranks the population and then every chromosome receives fitness from this ranking. The worst will have fitness ***1***, second worst ***2*** etc. and the best will have fitness ***N*** (number of chromosomes in population).

You can see in following picture, how the situation changes after changing fitness to order number.

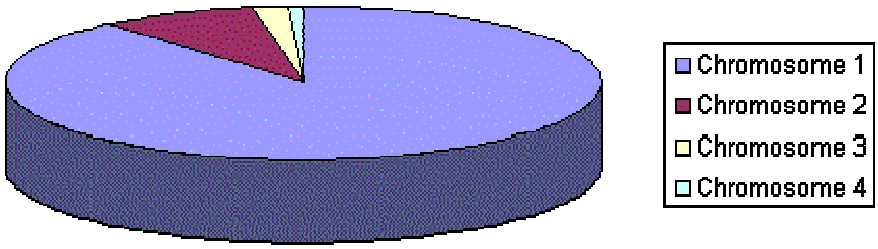



**Figure 2.5:** ***Situation before ranking (graph of fitnesses)***

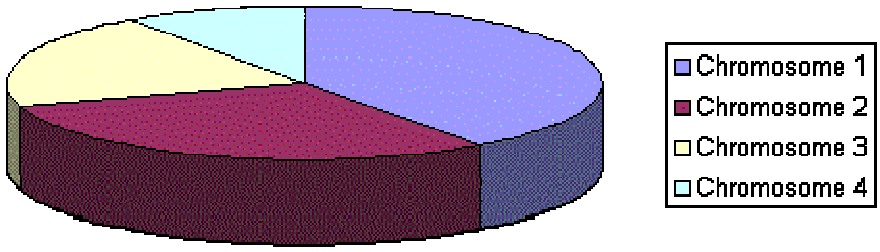



**Figure 2.6:** *Situation after ranking (graph of order numbers)*

After this all the chromosomes have a chance to be selected. But this method can lead to slower convergence, because the best chromosomes do not differ so much from other ones.

## Steady-State Selection

This is not particular method of selecting parents. Main idea of this selection is that big part of chromosomes should survive to next generation.

GA then works in a following way. Every generation is selected a few (good - with high fitness) chromosomes for creating a new offspring. Then some (bad - with low fitness) chromosomes are removed and the new offspring is placed in their place. The rest of population survives to new generation[22].

## Elitism

Idea of elitism has been already introduced. When creating new population by crossover and mutation, we have a big chance, that we will loose the best chromosome.

Elitism is name of method, which first copies the best chromosome (or a few best chromosomes) to new population. The rest is done in classical way. Elitism can very rapidly increase performance of GA, because it prevents losing the best found solution.

## 2.2.7 Encoding

## Introduction

Encoding of chromosomes is one of the problems, when you are starting to solve problem with GA. Encoding very depends on the problem.

In this chapter will be introduced some encodings, which have been already used with some success

## Binary Encoding

Binary encoding is the most common, mainly because first works about GA used this type of encoding.

In **binary encoding** every chromosome is a string of **bits**, **0** or **1**.

| | |
|---|---|
| Chromosome A | 10110010110010101110010 1 |
| Chromosome B | 11111110000011000001111 1 |

*Example of chromosomes with binary encoding*

Binary encoding gives many possible chromosomes even with a small number of alleles. On the other hand, this encoding is often not natural for many problems and sometimes corrections must be made after crossover and/or mutation[22,23].

**Example of Problem:** Knapsack problem
**The problem:** There are things with given value and size. The knapsack has given capacity. Select things to maximize the value of things in knapsack, but do not extend knapsack capacity.
**Encoding:** Each bit says, if the corresponding thing is in knapsack.

## Permutation Encoding

Permutation encoding can be used in ordering problems, such as traveling salesman problem or task ordering problem.

In **permutation encoding**, every chromosome is a string of numbers, which represents number in a **sequence**.

| | |
|---|---|
| Chromosome A | 1 5 3 2 6 4 7 9 8 |
| Chromosome B | 8 5 6 7 2 3 1 4 9 |

*Example of chromosomes with permutation encoding*

Permutation encoding is only useful for ordering problems. Even for this problems for some types of crossover and mutation corrections must be made to leave the chromosome consistent .

**Example of Problem:** Traveling salesman problem (TSP)
**The problem:** There are cities and given distances between them. Traveling salesman has to visit all of them, but he does not to travel very much. Find a sequence of cities to minimize traveled distance.
**Encoding:** Chromosome says order of cities, in which salesman will visit them.

## Value Encoding

Direct value encoding can be used in problems, where some complicated value, such as real numbers, is used. Use of binary encoding for this type of problems would be very difficult.

In **value encoding**, every chromosome is a string of some values. Values can be anything connected to problem, form numbers, real numbers or chars to some complicated objects.

| | |
|---|---|
| Chromosome A | 1.2324 5.3243 0.4556 2.3293 2.4545 |
| Chromosome B | ABDJEIFJDHDIERJFDLDFLFEGT |
| Chromosome C | (back), (back), (right), (forward), (left) |

*Example of chromosomes with value encoding*

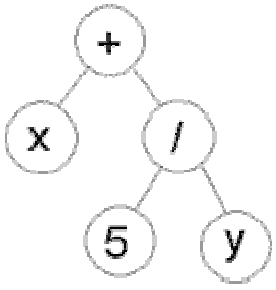
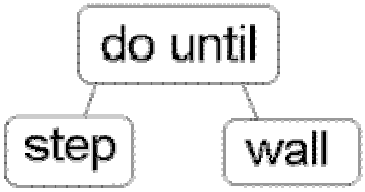
Value encoding is very good for some special problems. On the other hand, for this encoding is often necessary to develop some new crossover and mutation specific for the problem[22].

**Example of Problem:** Finding weights for neural network
**The problem:** There is some neural network with given architecture. Find weights for inputs of neurons to train the network for wanted output.
**Encoding:** Real values in chromosomes represent corresponding weights for inputs.

## Tree Encoding

Tree encoding is used mainly for evolving programs or expressions, for **genetic programming**.

In **tree encoding** every chromosome is a tree of some objects, such as functions or commands in programming language.

| Chromosome A | Chromosome B |
|---|---|
|  |  |

| | |
|---|---|
| ( + x ( / 5 y ) ) | ( do_until step wall ) |

*Example of chromosomes with tree encoding*

Tree encoding is good for evolving programs. Programming language LISP is often used to this, because programs in it are represented in this form and can be easily parsed as a tree, so the crossover and mutation can be done relatively easily.

**Example of Problem:** Finding a function from given values
**The problem:** Some input and output values are given. Task is to find a function, which will give the best (closest to wanted) output to all inputs.
**Encoding:** Chromosome is functions represented in a tree.

## 2.2.8 Crossover and Mutation

### Introduction

Crossover and mutation are two basic operators of GA. Performance of GA very depends on them. Type and implementation of operators depends on encoding and also on a problem.

There are many ways how to do crossover and mutation.

### Crossover

Single point crossover - one crossover point is selected, binary string from beginning of chromosome to the crossover point is copied from one parent, the rest is copied from the second parent

**11001**011+11011**111** = **11001111**

**Two point crossover** - two crossover point are selected, binary string from beginning of chromosome to the first crossover point is copied from one parent, the part from the first to the second crossover point is copied from the second parent and the rest is copied from the first parent


**11**001**011** + 11**011**111 = **11011111**

Uniform crossover - bits are randomly copied from the first or from the second parent

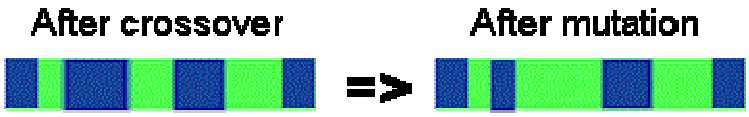

1**10**01**011** + **1**10**111**01 = 11011111

Arithmetic crossover - some arithmetic operation is performed to make a new offspring


11001011 + 11011111 = 11001001 (AND)

## Mutation

**Bit inversion** - selected bits are inverted




1**1**001001 => 1**0**001001

## Permutation Encoding

### Crossover

**Single point crossover** - one crossover point is selected, till this point the permutation is copied from the first parent, then the second parent is scanned and if the number is not yet in the offspring it is added

*Note: there are more ways how to produce the rest after crossover point*

(**1 2 3 4 5** 6 7 8 9) + (**4 5 3 6 8** 9 7 2 1) = (**1 2 3 4 5** 6 8 9 7)

## Mutation

**Order changing** - two numbers are selected and exchanged

(1 **2** 3 4 5 6 **8** 9 7) => (1 **8** 3 4 5 6 **2** 9 7)

## Value Encoding

### Crossover

All crossovers from **binary encoding** can be used

### Mutation

**Adding** a small number (for real value encoding) - to selected values is added (or subtracted) a small number

(1.29 5.68 **2.86** **4.11** 5.55) => (1.29 5.68 **2.73** **4.22** 5.55)

## Tree Encoding

### Crossover

**Tree crossover** - in both parent one crossover point is selected, parents are divided in that point and exchange part below crossover point to produce new offspring

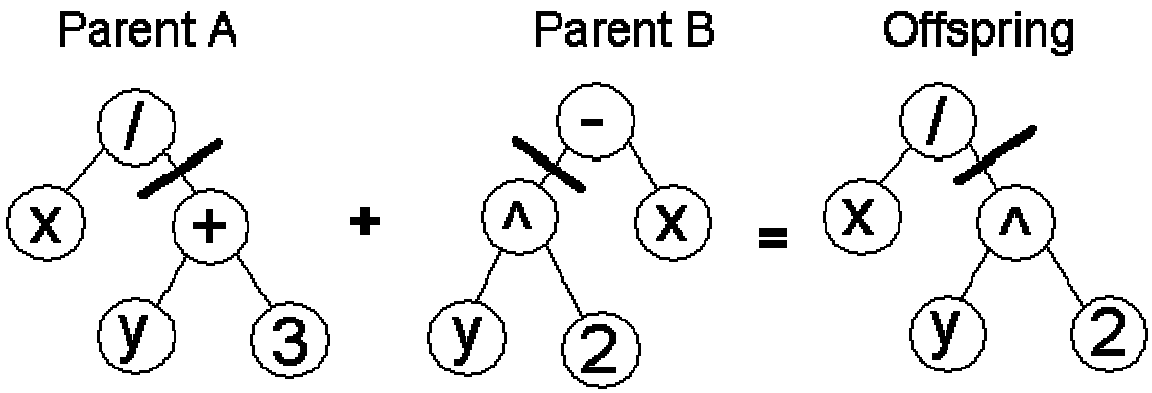



**Figure 2.7:** Tree Encoding

### Mutation

**Changing operator, number** - selected nodes are changed

# CHAPTER 3: MULTI-MODAL OPTIMIZATION

## 3.1 Introduction

Immune system can provide new ways of solving problems. The function optimizing problem involves finding the best solution (either the peak or trough) to a function bounded by constraints.

In the code, a test function has been used and is plotted. This test function uses the sine function to produce the hilly plot, and so finding the highest peak is a challenged. The equation is given by:

**fitness = x*Math.Sin(4*Math.PI*x)-y*Math.Sin(4*Math.PI*y+Math.PI)+1;**

The optimum is at f (1.63, 1.63) =4.25 which is the highest peak at the center of the plot.

By considering an antibody as a potential solution (i.e. a cell object which has x-y value) and the fitness function as the antigen, then the degree of fit or binding represents the quality of the solution. If we start with an initial population of antibody solutions and test them against the fitness function (antigen), then those with the highest affinity (i.e. best fit) are allowed to clone and mutate in the hope of finding a better solution[18,19].

## 3.2 *Objective function1*

- f(x,y) = x * sin(4.PI.x) –y * sin(4.PI.y + PI) + 1

This function has several local maxima and minima and the optimal solution for them is at f (1.63, 1.63) = 4.25 for maximization problem. The graph below shows the function between [-1, 2].

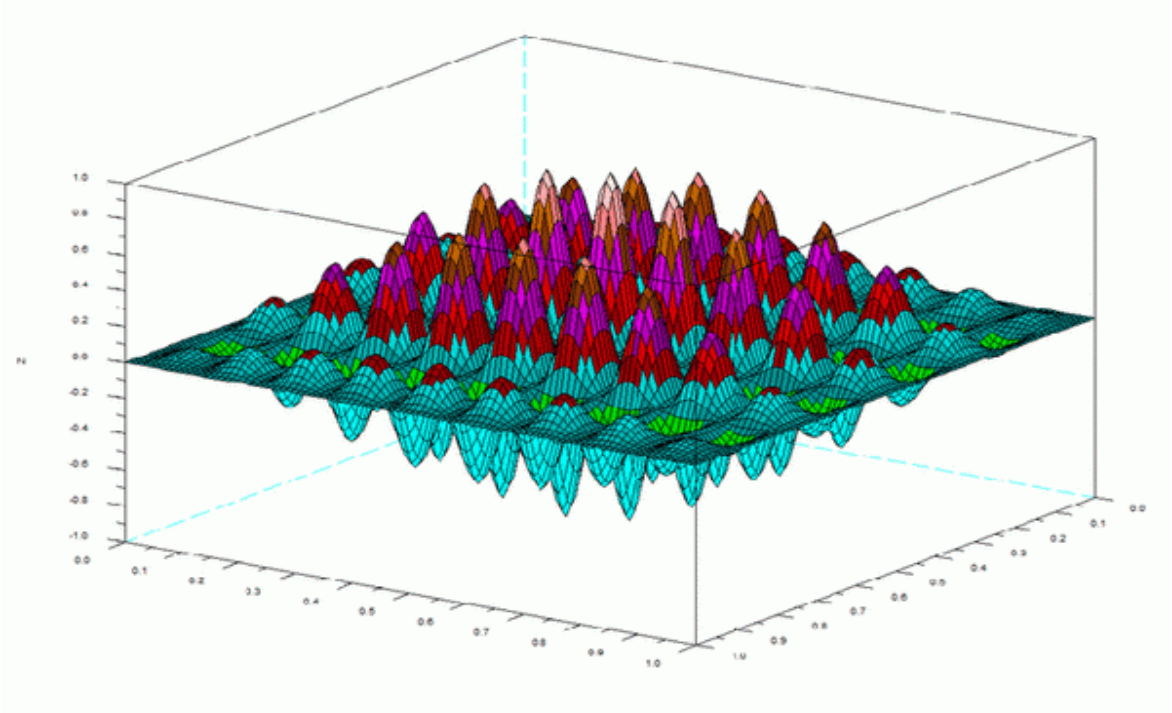


**Figure 3.1:** ***Objective function1***

Consequently, an immune algorithm can be devised as follows:

1. Generate an initial population of antibodies
2. Perform clonal selection to generate high affinity clones and mutate
3. Remove antibodies whose affinity with the antigen is less than a predefined threshold
4. Calculate affinity interactions between all antibodies in the system
5. Remove antibodies whose affinity with other antibodies is below a predefined threshold

Introduce randomly generated antibodies into population (diversity)

Repeat steps 2 to 6 until the stopping criterion is met.

The best fit clones would be plotted with the help of Zed graph.

## 3.3 Algorithm of Multi Model Optimization:

The code contains five essential classes. These are:

- the main form class which provides a simple GUI
- the immune algorithm controller class
- the antibody class
- the fitness function class
- the Zed graph class

The code has been commented and so should be straightforward to follow:

The fitness function class has a static method called evaluateFunction(), which returns the fitness value given x and y input values. The antibody class attempts to model a biological antibody cell. It has methods for cloning itself, finding the affinity with another antibody and an affinity based mutation. Each antibody represents a candidate solution, which in this example is simply an x-y value. The immune algorithm controller class allows parameters to be defined, and has a method called GoOptimise() which creates an initial population of antibodies and iterates a solution until the stopping criteria (a maximum number of generations) is reached.

## 3.4 ***Objective function2***

- $f(x,y) = -2[\ 2.\sin(x) + 2.\sin(y)] - 4\cos(x/2) - 4\cos(y/2) - 4\cos(x)$

  This function is of potential energy of a two bar pendulum with data

  weight W1=2, W2=2, Lengths L1= 2, L2=2, and

  p(weight of lowest pendulum) =2

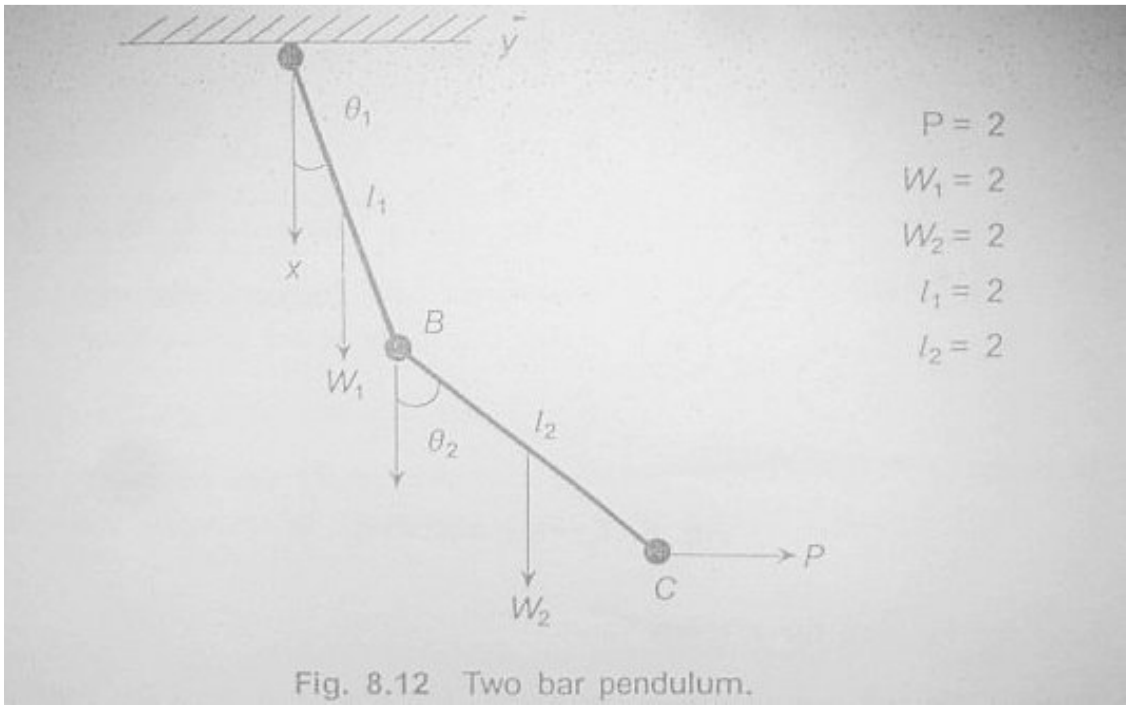



**Figure 3.2:** ***Objective function2***

total potential energy is given by

P.E= - P [ (l1.sin(x) + l2.sin(y) ) ] - W1.L1.cos(x)/2 - W2 [ l2.cos(y)/2 + l1.cos(x)]

Where x = first angle, and

Y= second angle

Lower bound = 0 (in degrees)

Upper bound = 90 (in degrees)

# CHAPTER 4: CSA & GA EVALUTION

## 4.1 Introduction of CSA

we discussed the clonal selection principle and the affinity maturation process, which will be used as the fundamental basis for the development of the clonal selection algorithm (CSA).

The main immune aspects taken into account were:

- maintenance of the memory cells functionally disconnected from the repertoire;
- selection and cloning of the most stimulated individuals;
- death of non-stimulated cells;
- affinity maturation and re-selection of the higher affinity clones;
- generation and maintenance of diversity; and
- hypermutation proportional to the cell affinity.

## 4.2 Steps of CSA

The algorithm works as follows (see Figure ):

(1) Generate a set (P) of candidate solutions, composed of the subset of memory cells (M) added to the remaining (P*r*) population (P = P*r* + M);

(2) Determine the *n* best individuals P*n* of the population P, based on an affinity measure;

(3) Clone (reproduce) these *n* best individuals of the population, giving rise to a temporary population of clones (*C*). The clone size is an increasing function of the affinity measure of the antigen;

(4) Submit the population of clones to a hypermutation scheme, where the hypermutation is proportional to the affinity of the antibody. A maturated antibody population is generated (*C**);

(5) Re-select the improved individuals from $C^*$ to compose the memory set. Some members of the P set can be replaced by other improved members of $C^*$;

(6) Replace $d$ low affinity antibodies of the population, maintaining its diversity.

For each problem to be presented, the coding and affinity measure adopted will be discussed separately[18,19].

Steps 2 and 3 are crucial in this algorithm. If we choose $n = N$ in Step 2, i.e. the number of highest affinity individuals equals the number of candidates, each member of the population will constitute a potential candidate solution locally, implying a local exploitation of the shape-space, characterizing a greedy search. In addition, if all the individuals are accounted locally, their clones (Step 3) will have the same size. In all the example applications, steps 2 and 3 were taken as discussed in this paragraph.

The clonal selection algorithm reproduces those individuals with higher affinities and selects their improved maturated progenies. This strategy suggests that the algorithm performs a greedy search, where single members will be locally optimized (exploitation of the surrounding space), and the newcomers yield a broader exploration of the search-space. This characteristic makes the CSA very suitable for solving multi-modal optimization tasks and, as an illustration, consider the case of maximizing the function $f(x,y) = x.sen(4px)-y.sen(4py+p)+1$, depicted in Figure 37, in the compact region [-1,2] ´ [-1,2] Notice that this function is composed of many local optima and a single global optimum at $f(1.63,1.63) = 4.25$.

We employed the Hamming shape-space, with binary strings representing real values for the variables $x$ and $y$. The chosen bitstring length was $L = 22$, corresponding to a precision of six 80 decimal places. The variables $x$ and $y$ are defined over the range [-1, 2], and the mapping from a binary string $m = <mL,..., m2, m1>$ into a real number $x$ or $y$ is completed in two steps:

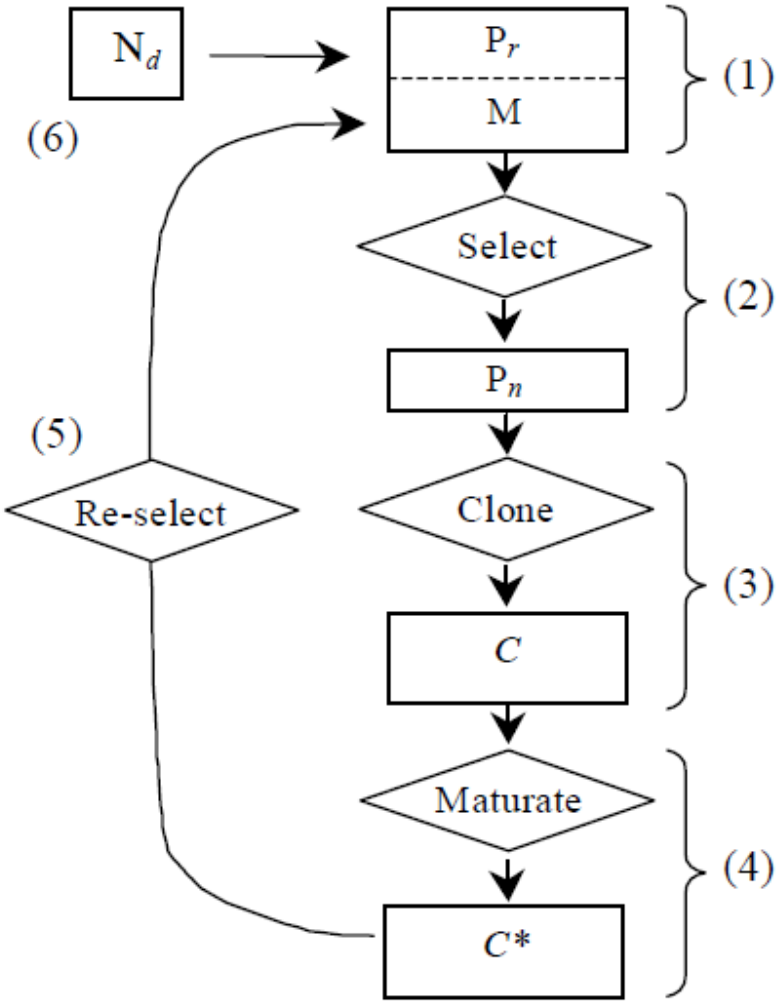



**Figure 4.1**: Block diagram for algorithm implemented

- convert the binary string *m* = <*mL*,..., *m*2, *m*1> from base 2 to base 10:

$$(< m_L, \dots m_2, m_1 >)_2 = (\sum_{i=0}^{21} m_i . 2^i)_{10} = x'$$

- find the corresponding value for x ,$x = -1 + x' \frac{3}{2^{22}-1}$ where -1 is left boundary of domain ,and 3 is it's length

The affinity measure corresponds to the evaluation of the function *f*(*x*,*y*) after decoding *x* and *y*, as described above.

Figure (a) and (b) presents the evolved populations, after 100 generations, by the standard genetic algorithm (see Section 10.3.3 for a brief description of the standard genetic algorithm GA) and the clonal selection algorithm (CSA), respectively. Notice that the genetic algorithm guided the whole population towards the global optimum of the function, while the CSA generated a diverse set of local optima, including the global optimum.

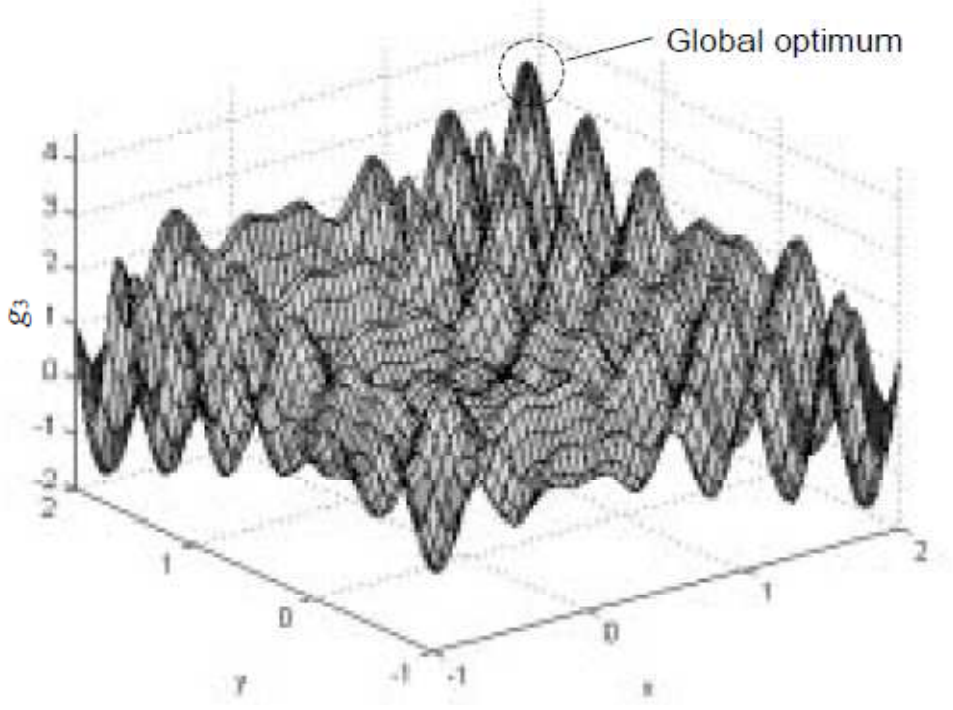



Figure 4.2: **Function *f*(*x*,*y*) = *x*.*sen*(4p*x*)-*y*.*sen*(4p*y*+p)+1 to be optimized by the CSA and standard GA.**

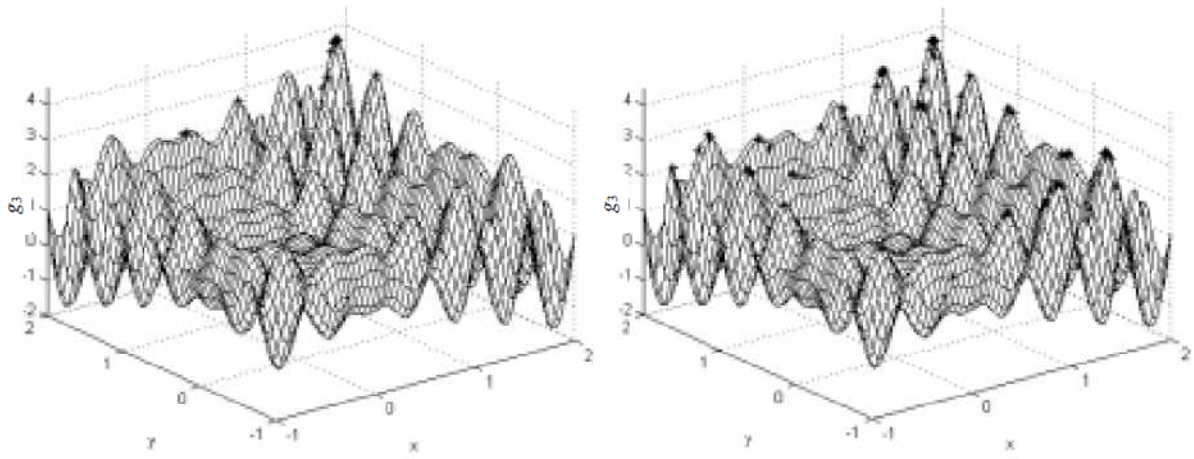



Figure 4.3: Function *x.sen*(4p*x*)-*y.sen*(4p*y*+p)+1 optimized (100 generations) by the GA (a) and CSA (b).

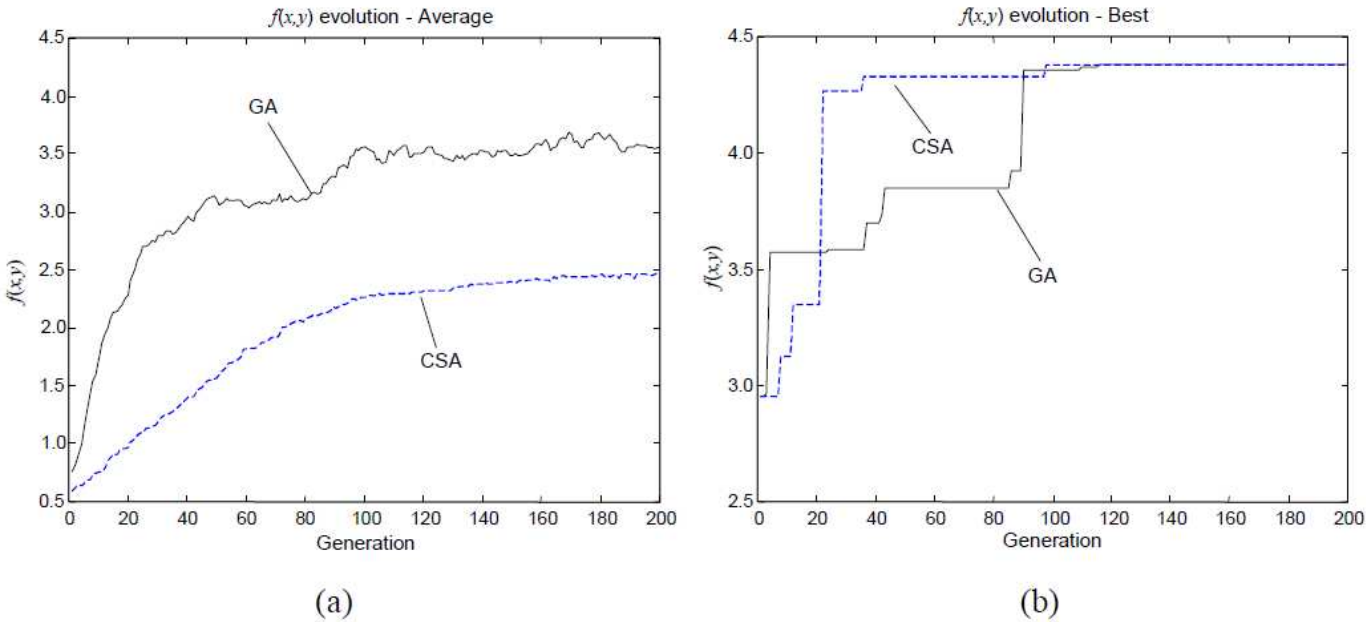



Figure 4.4: Evolutionary behavior of the decoded average value of *f*(*x*,*y*) (a) and the maximum value (b), for the genetic and clonal selection algorithms.

Figure (a) compares the decoded average value of the function *f*(*x*,*y*), for the whole population, evolved by the GA and the CSA algorithms. Figure 39(b) depicts the best individuals (candidates with higher values for *f*(*x*,*y*)) of the populations for each algorithm. The GA approach presented a greater average value, indicating a less diverse set of individuals. Both strategies successfully determined the global optimum.

## 4.3 Code description

The antibodies (candidate solutions to the function optimization problem) that are generated by the immune algorithm are displayed in a text box. The final value i.e. the

best fit would be displayed with the help of Zed Graph. The best antibody is displayed

first, and should be a good match to the required solution for this problem which, as stated above, is 4.25 at x=1.63 and y=1.63. An immune

Algorithm is a non-deterministic algorithm, meaning that it gives different results on different runs.

It is necessary to set threshold values for removing (suppressing) antibodies from the population pool (clonalSelectionThreshold, removeThreshold). The settings used for these threshold values were derived by a process of trial and error. The parameter called antibodyNumber determines the initial number of antibodies used to

solve the problem or, in biological terms, neutralise the antigen.. The cloneNumber parameter sets the number of clones generated during clonal selection. Affinity based mutation is set using the mutationFactor parameter. Constraints for the x and y values

are imposed using the lowerBoundary and upperBoundary parameters.

## 4.4 GA(genetic algorithm)

There is potential for further investigation. It would be interesting to look into the discrimination between antibodies destined to be deleted and those not, and new types of operators for cloning and mutation. Many other sine functions are available for calculating the best fit value.

The Genetic Algorithms (GAs) constitute stochastic evolutionary techniques whose search methods model some natural phenomena: genetic inheritance and Darwinian strife for survival. GAs perform a search through a space of potential solutions, which are distinguished by the definition of an evaluation (fitness) function, which plays the role of an environment feedback.

A genetic algorithm (or any evolutionary program) for a particular problem, must have the

following five components (Michalewicz, 1996):

- a genetic representation for potential candidate solutions;
- a way to create an initial population of potential solutions;
- an evaluation (fitness) function;
- genetic operators that alter the composition of an offspring;
- values for the various parameters used by the algorithm: population sizes, genetic operators probabilities, etc.

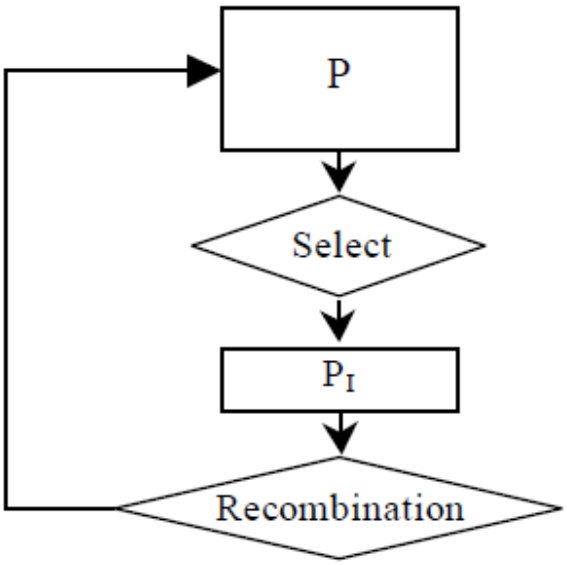



Figure 4.5: Block diagram for the standard genetic algorithm (GA), where PI is the intermediate population

There are number of ways to implement genetic algorithm (GA) as it is more of domain specific and mutation and crossover functions vary greatly according to the problem.

We will implement GA in these ***objective*** functions

*Objective function*

- f(x,y) = x * sin(4.PI.x) –y * sin(4.PI.y + PI) + 1

  This function has several local maxima and minima and the optimal solution for them is at f (1.63, 1.63) = 4.25 for maximization problem. The graph below shows the function between [-1, 2].

# CHAPTER 5: RESULTS AND DISCUSSION

We have implemented immune and genetic algorithms both now we will check the result. and check and compare which algorithm is better than other based on generation size. We will use graph to compare the result in which one direction shows the affinity of antibody or clone or gene. Both of the algorithms not guarantying the optimum result. And the result may be different in every time program is run because it use random variable for mutation so it will be different maximum time . this randomness provide the dynamic approach every time

We will use the same objective function to compare the result . and we also make the table of performance. Both algorithms gave the different result every time because we are using random variables in mutation . it provides a dynamic approach that have low probability that algorithms stuck in local maxima or minima.

**Objective function**

$f(x,y) = x * \sin(4.PI.x) – y * \sin(4.PI.y + PI) + 1$

Optimal solution found using following parameters for genetic algorithm:

- Number of variables in the objective function = 2
- Genome size = 2 * chromosome size = 24
- Lower bound for x = -1
- Lower bound for y = -1
- Upper bound for x = 2
- Upper bound for y = 2
- Crossover rate = 80 %
- Mutation rate = 3 %
- Population size = 5,10,20
- Generation size =10,100.500,1000(It will act as a terminator for program)

***Real values***

**x = 1.63, y = 1.63**

**Value for objective function = 0.95,2.18,3.25,4.25**

**Objective function**

f(x,y) = x * sin(4.PI.x) –y * sin(4.PI.y + PI) + 1

Optimal solution found using following parameters for clonal selection algorithms:

No. of antibody =2

No of clones =5,10,20,30

Generation size =10,100,500,1000

Lower bound = -1

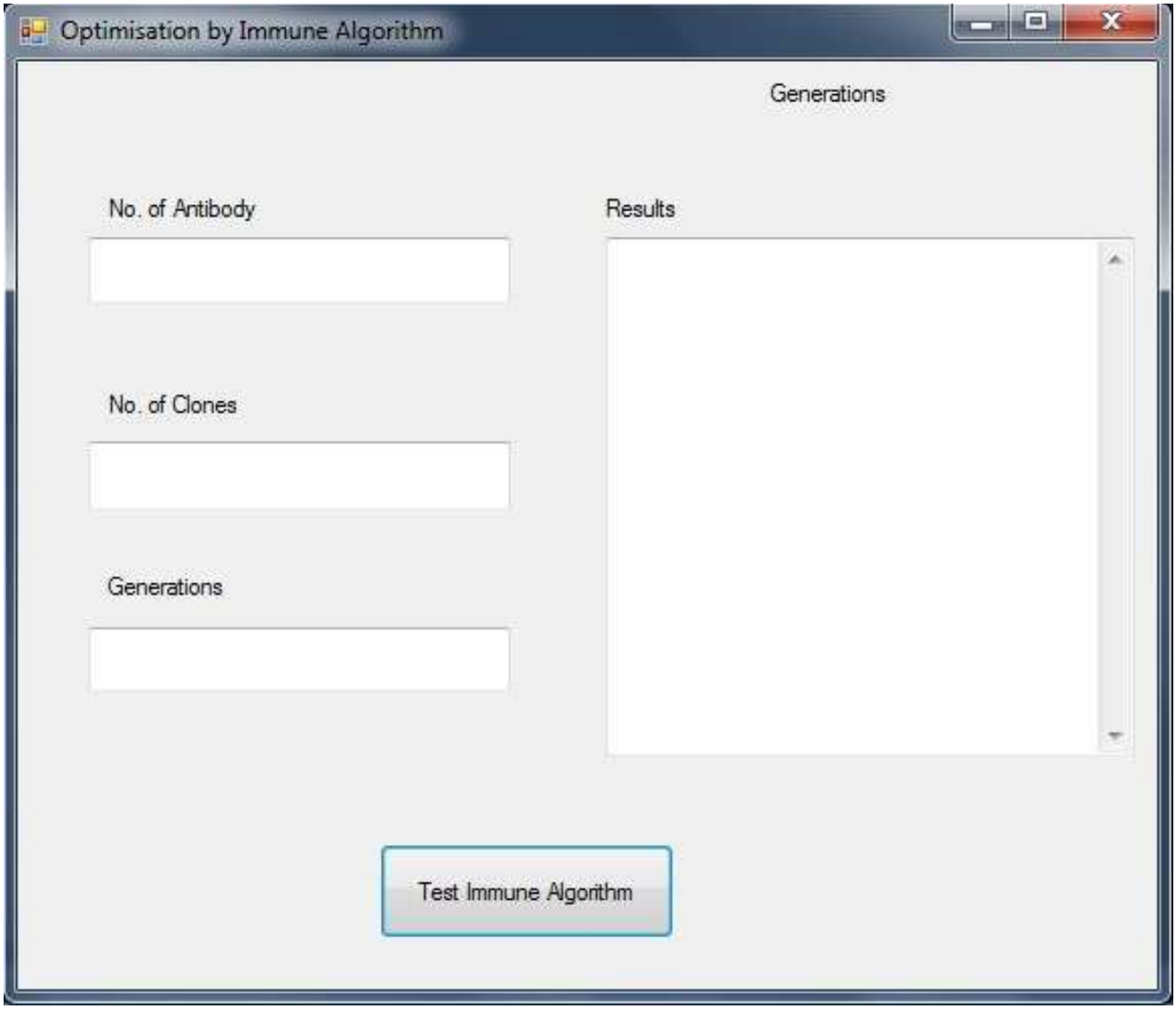
Upper bound = 2

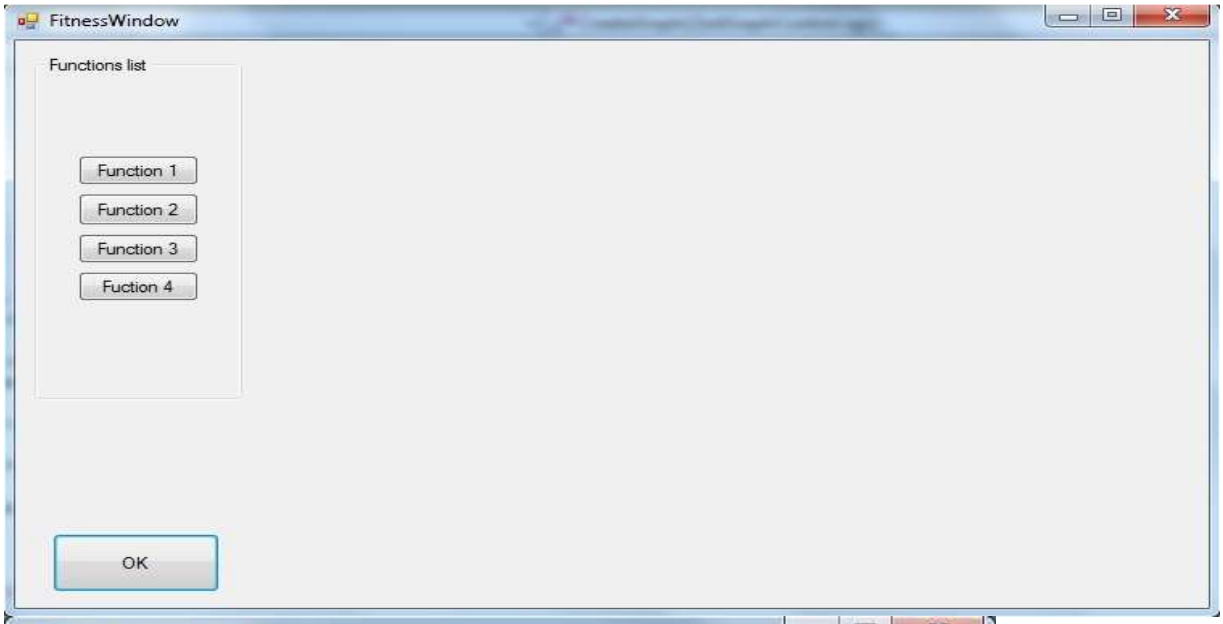
Out put = 3.25,4.25,4.25,4.25

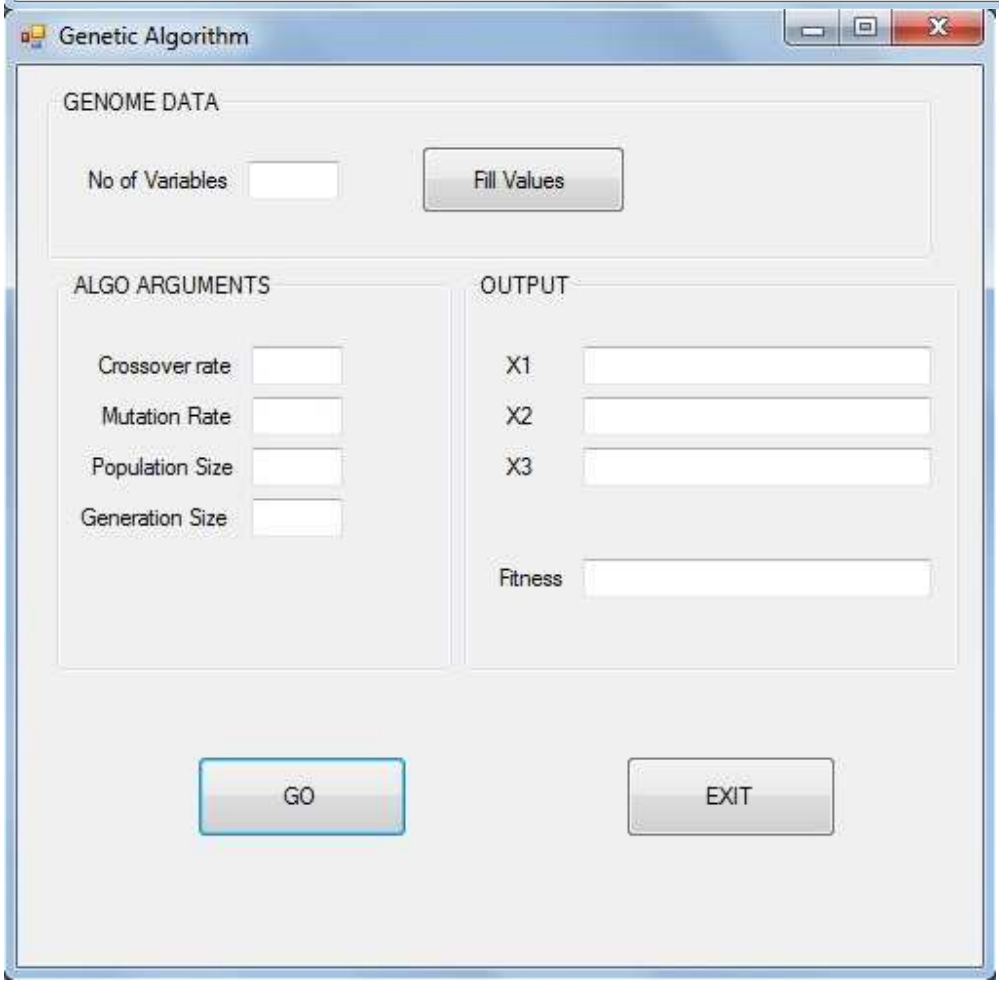
The table below show that performance of immune algorithm gave batter result when population size is low and generation size also low and genetic algorithm gave very poor performance as compare to CSA. as we increasing in the parameter that affect the results we find GA performance increase gradually and finally both gave the same result as we expecting. And it is also very close to real values of function.

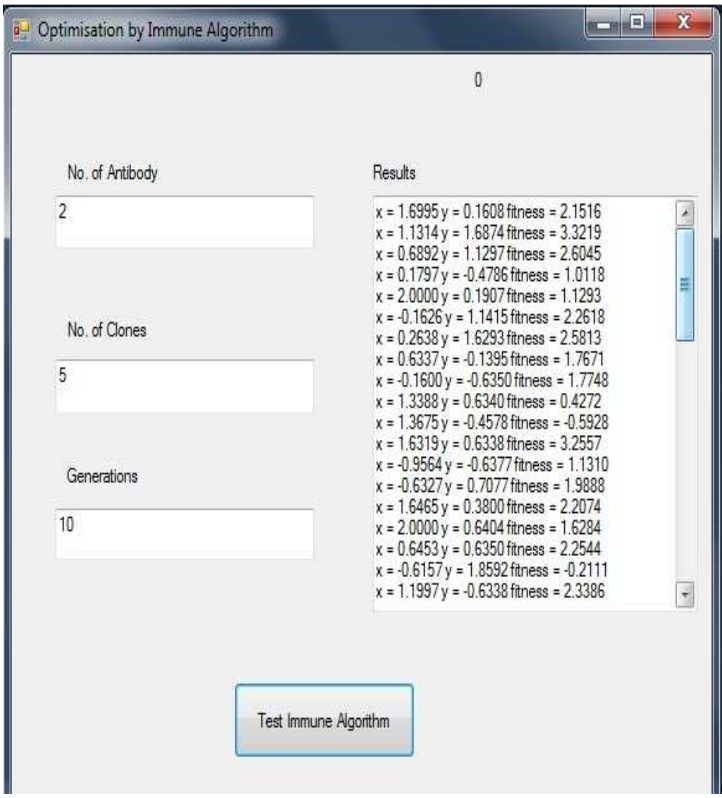
| S.No. | No. of population/clones | No. of generation | CSA | | | GA | | |
|---|---|---|---|---|---|---|---|---|
| | | | X | Y | affinity | X | Y | fitness |
| 1. | 5 | 10 | 1.134 | 1.688 | 3.32 | 0.28 | -0.513 | 0.957 |
| 2. | 5 | 2000 | 1.624 | 1.627 | 4.25 | 1.618 | 1.625 | 4.24 |
| 3. | 10 | 100 | 1.628 | 1.627 | 4.25 | 1.725 | 1.053 | 2.18 |
| 4. | 20 | 200 | 1.627 | 1.628 | 4.25 | 1.623 | -0.623 | 3.25 |
| 5. | 20 | 500 | 1.628 | 1.627 | 4.25 | 1.628 | 1.628 | 4.25 |
| 6. | 30 | 1000 | 1.628 | 1.628 | 4.25 | 1.628 | 1.627 | 4.25 |
| 7. | 40 | 2000 | 1.629 | 1.628 | 4.25 | 1.628 | 1.624 | 4.25 |

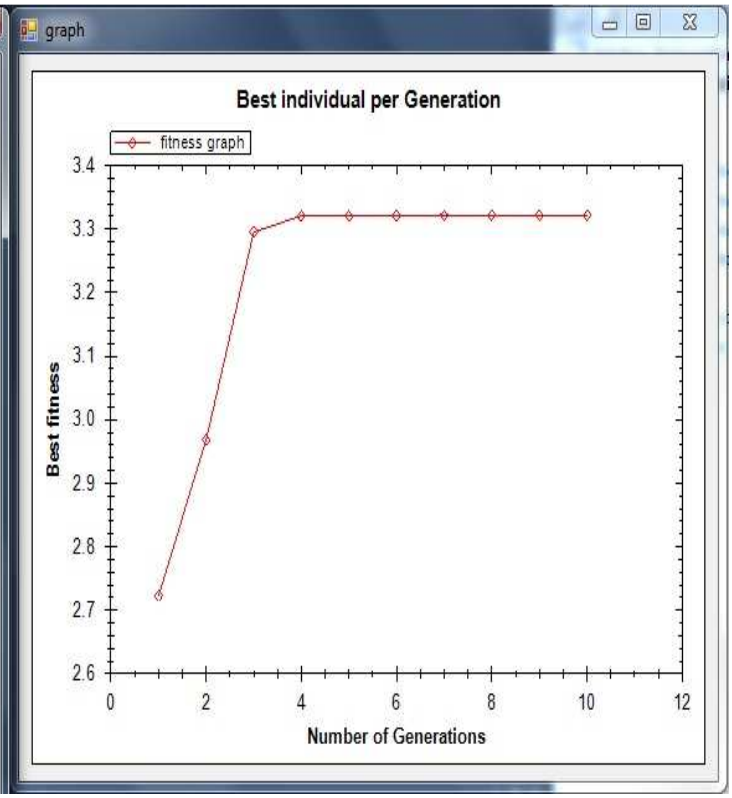
Table 1 : **Result of CSA & GA on same parameter**

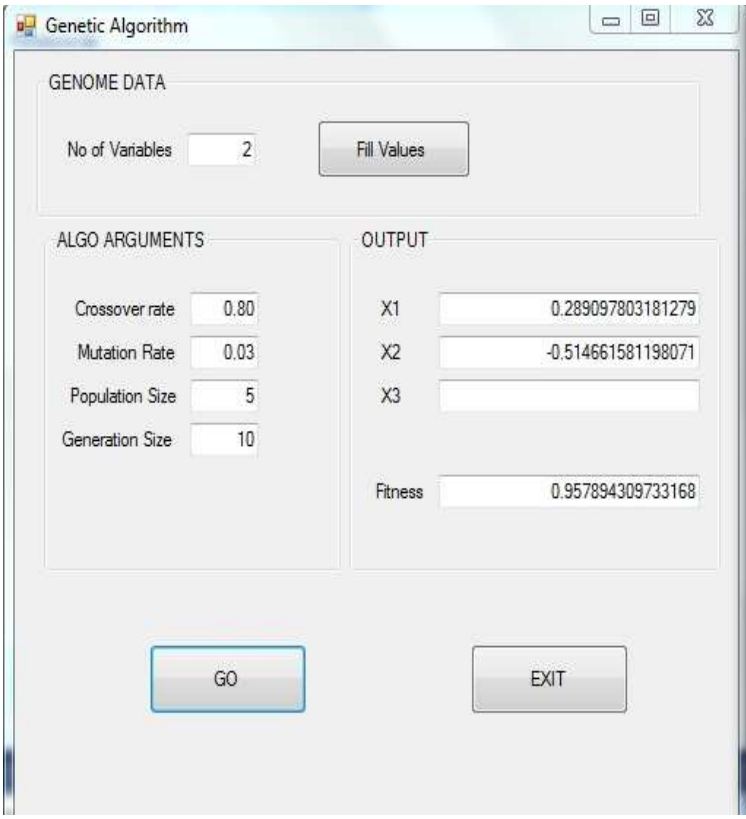

Optimisation by Immune Algorithm
Generations
No. of Antibody
Results
No. of Clones
Generations
Test Immune Algorithm

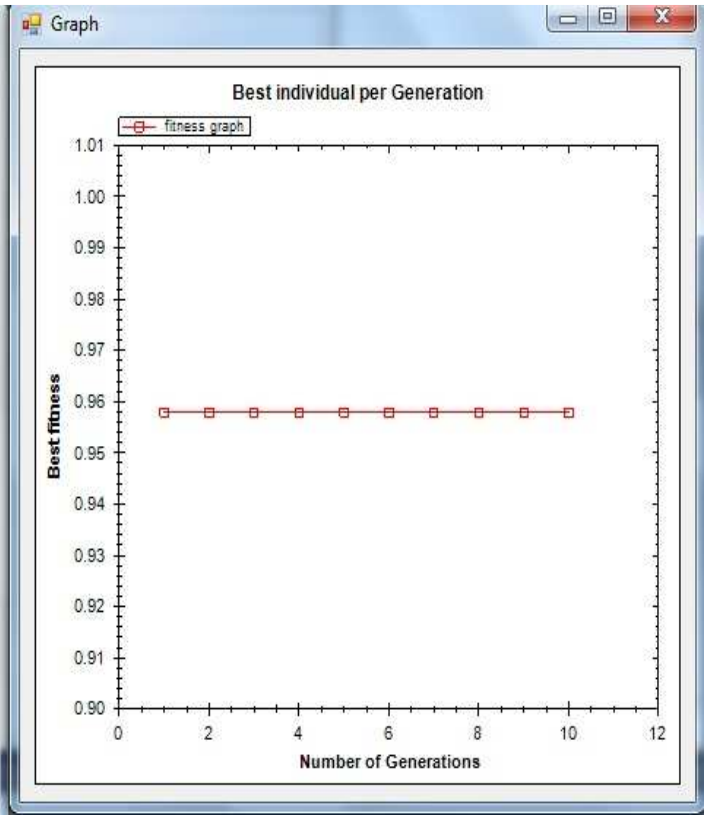

FitnessWindow
Functions list
Function 1
Function 2
Function 3
Fuction 4
OK

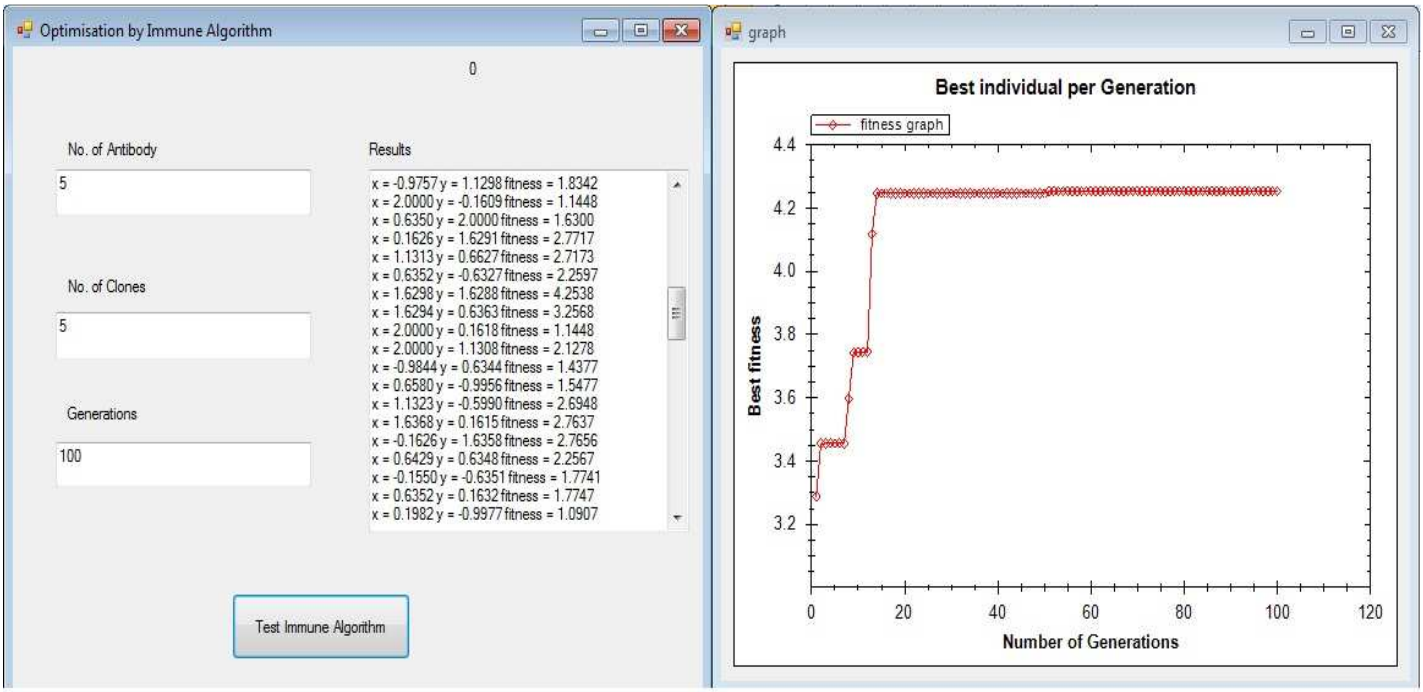

Genetic Algorithm
GENOME DATA
No of Variables
Fill Values
ALGO ARGUMENTS
Crossover rate
Mutation Rate
Population Size
Generation Size
OUTPUT
X1
X2
X3
Fitness
GO
EXIT

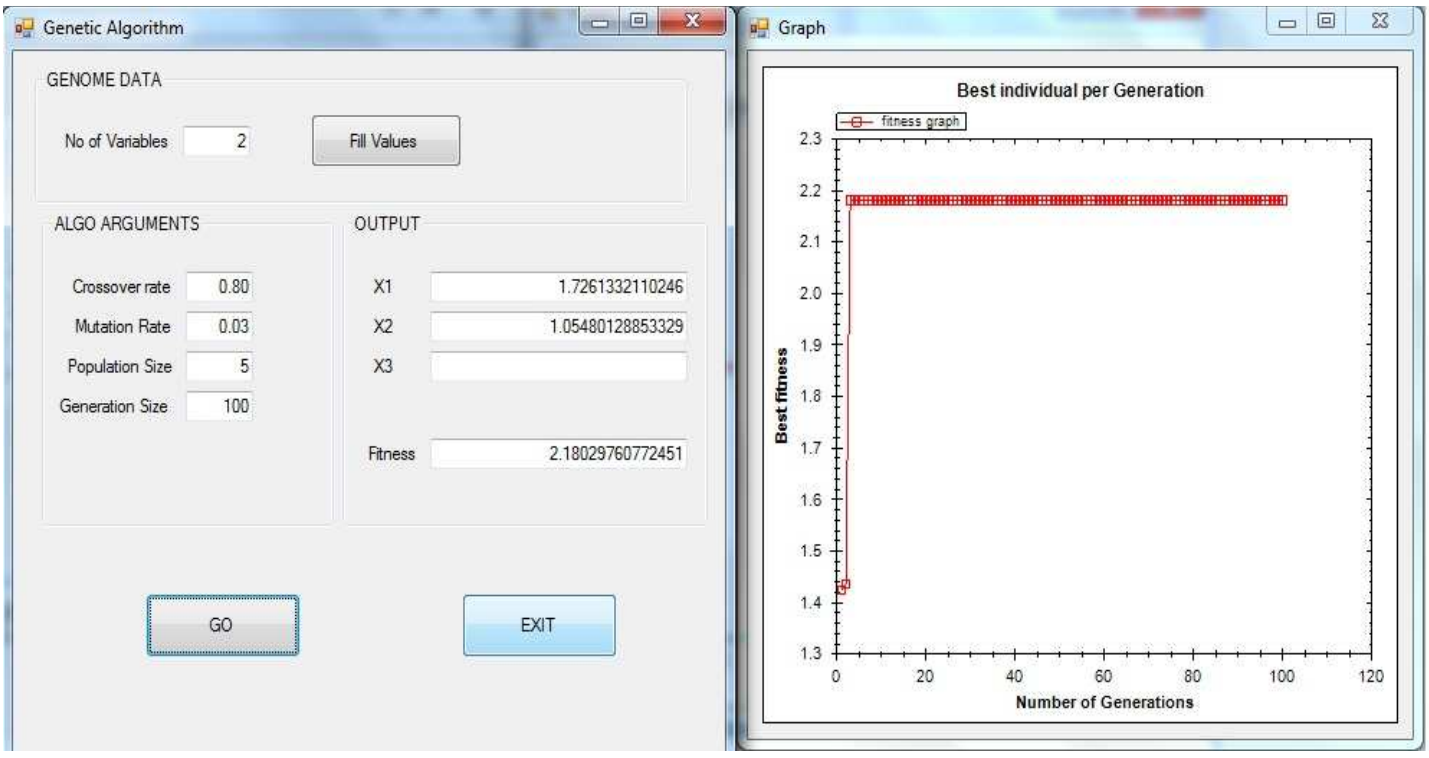

Optimisation by Immune Algorithm
0
No. of Antibody
2
No. of Clones
5
Generations
10
Results
x = 1.6995 y = 0.1608 fitness = 2.1516
x = 1.1314 y = 1.6874 fitness = 3.3219
x = 0.6892 y = 1.1297 fitness = 2.6045
x = 0.1797 y = -0.4786 fitness = 1.0118
x = 2.0000 y = 0.1907 fitness = 1.1293
x = -0.1626 y = 1.1415 fitness = 2.2618
x = 0.2638 y = 1.6293 fitness = 2.5813
x = 0.6337 y = -0.1395 fitness = 1.7671
x = -0.1600 y = -0.6350 fitness = 1.7748
x = 1.3388 y = 0.6340 fitness = 0.4272
x = 1.3675 y = -0.4578 fitness = -0.5928
x = 1.6319 y = 0.6338 fitness = 3.2557
x = -0.9564 y = -0.6377 fitness = 1.1310
x = -0.6327 y = 0.7077 fitness = 1.9888
x = 1.6465 y = 0.3800 fitness = 2.2074
x = 2.0000 y = 0.6404 fitness = 1.6284
x = 0.6453 y = 0.6350 fitness = 2.2544
x = -0.6157 y = 1.8592 fitness = -0.2111
x = 1.1997 y = -0.6338 fitness = 2.3386
Test Immune Algorithm


graph
Best individual per Generation
fitness graph
Best fitness
Number of Generations
3.4
3.3
3.2
3.1
3.0
2.9
2.8
2.7
2.6
0
2
4
6
8
10
12

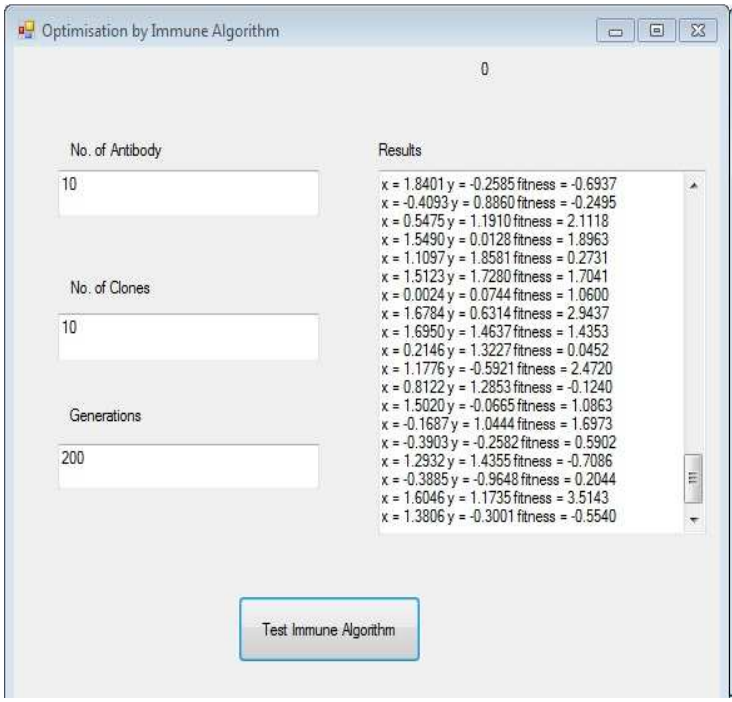

Genetic Algorithm
GENOME DATA
No of Variables
2
Fill Values
ALGO ARGUMENTS
Crossover rate
0.80
Mutation Rate
0.03
Population Size
5
Generation Size
10
OUTPUT
X1
0.289097803181279
X2
-0.514661581198071
X3
Fitness
0.957894309733168
GO
EXIT

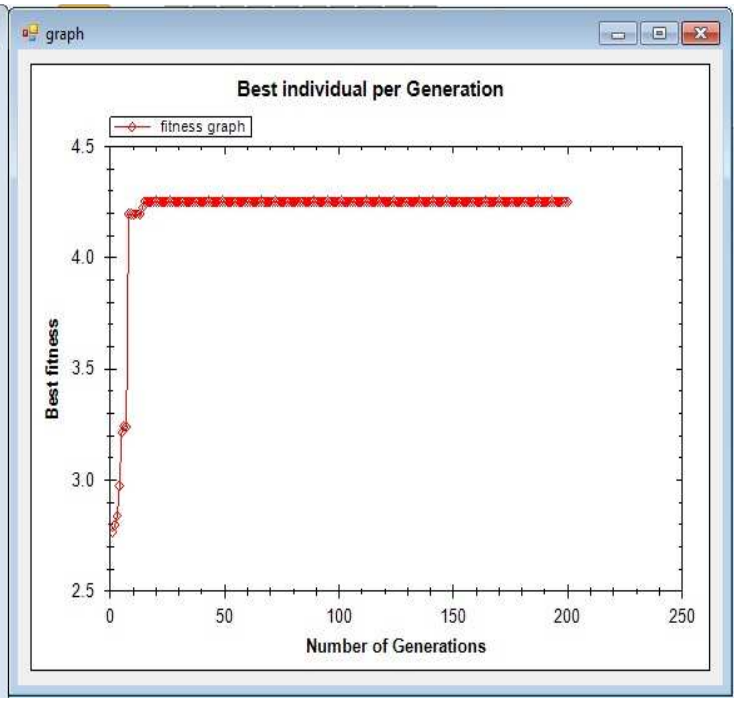

Graph
Best individual per Generation
fitness graph
Best fitness
Number of Generations
1.01
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0.91
0.90
0
2
4
6
8
10
12

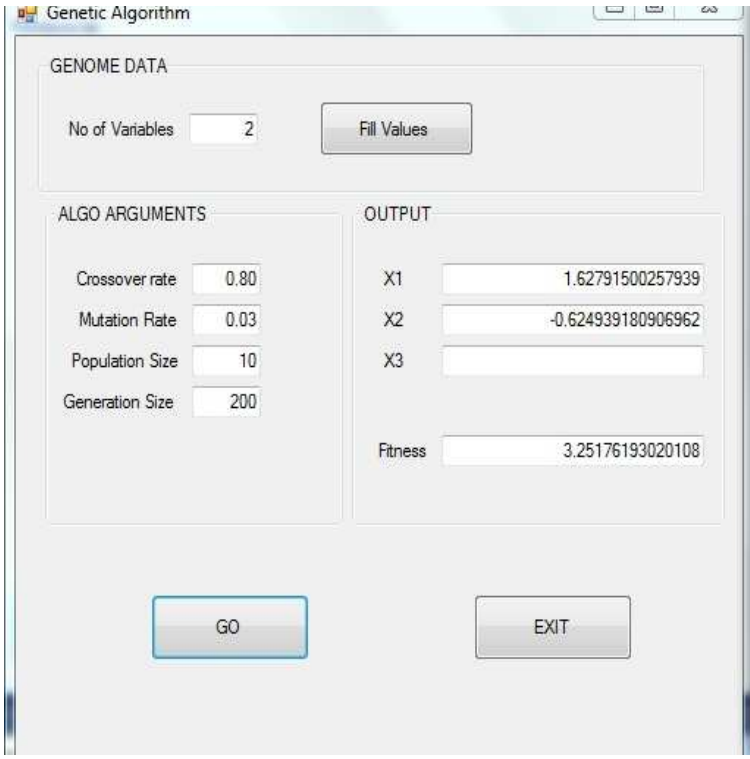


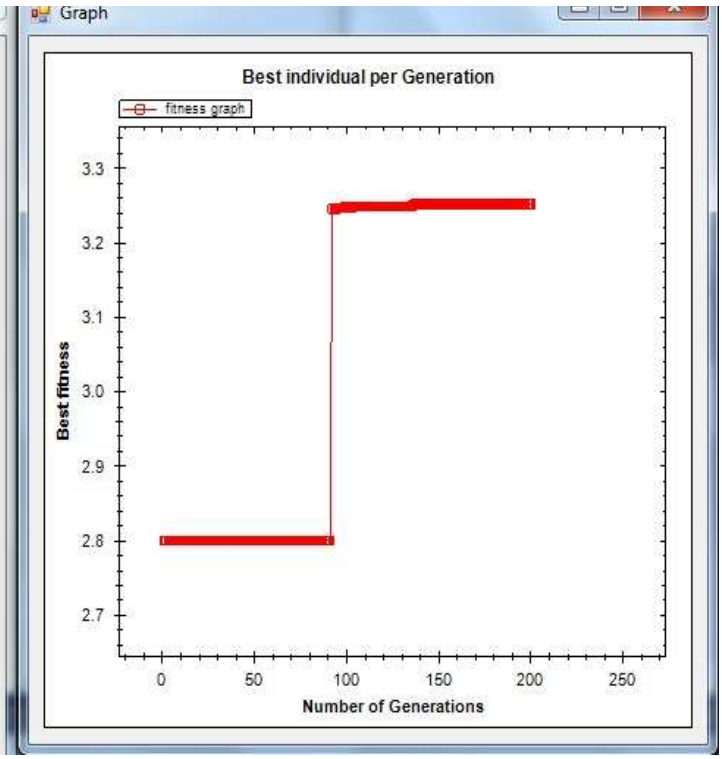




**Result shows that if No. of population size, clone size and generation size is same(5,5,100) than CSA perform batter than genetic algorithms.**

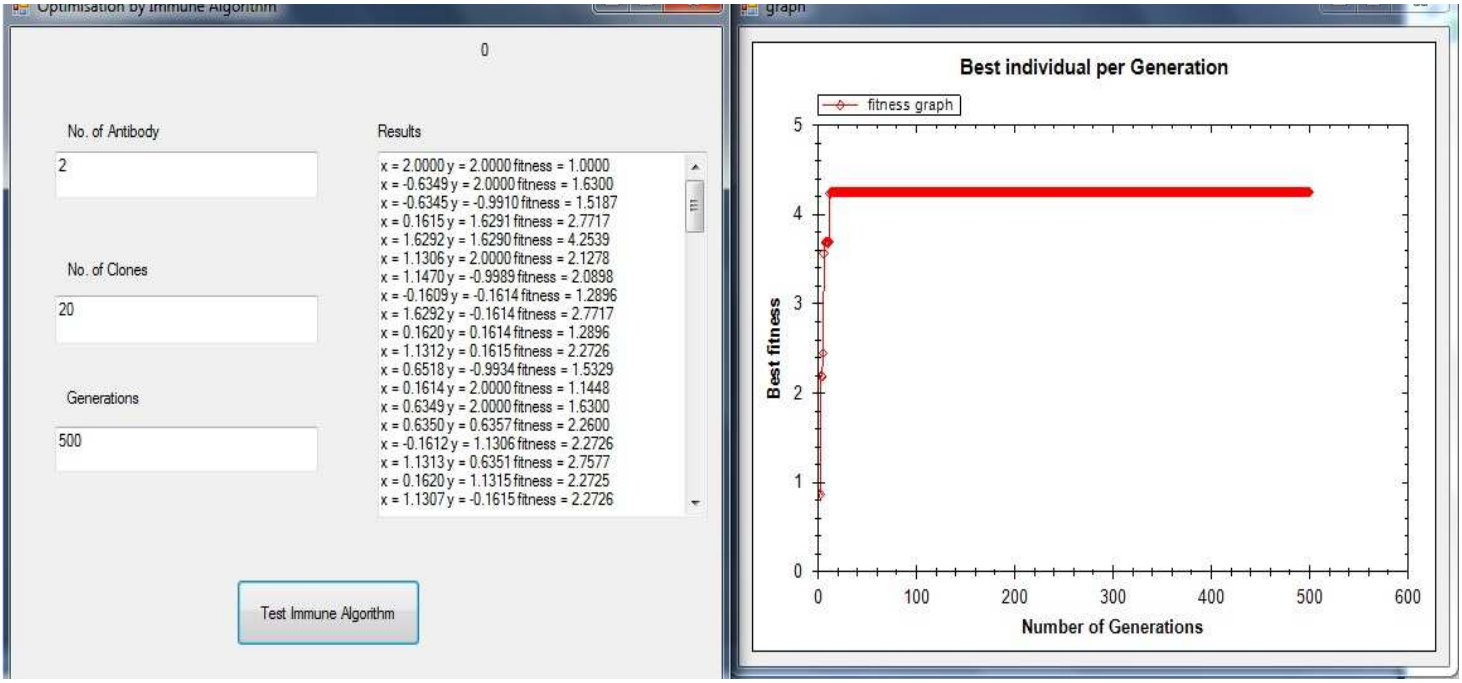


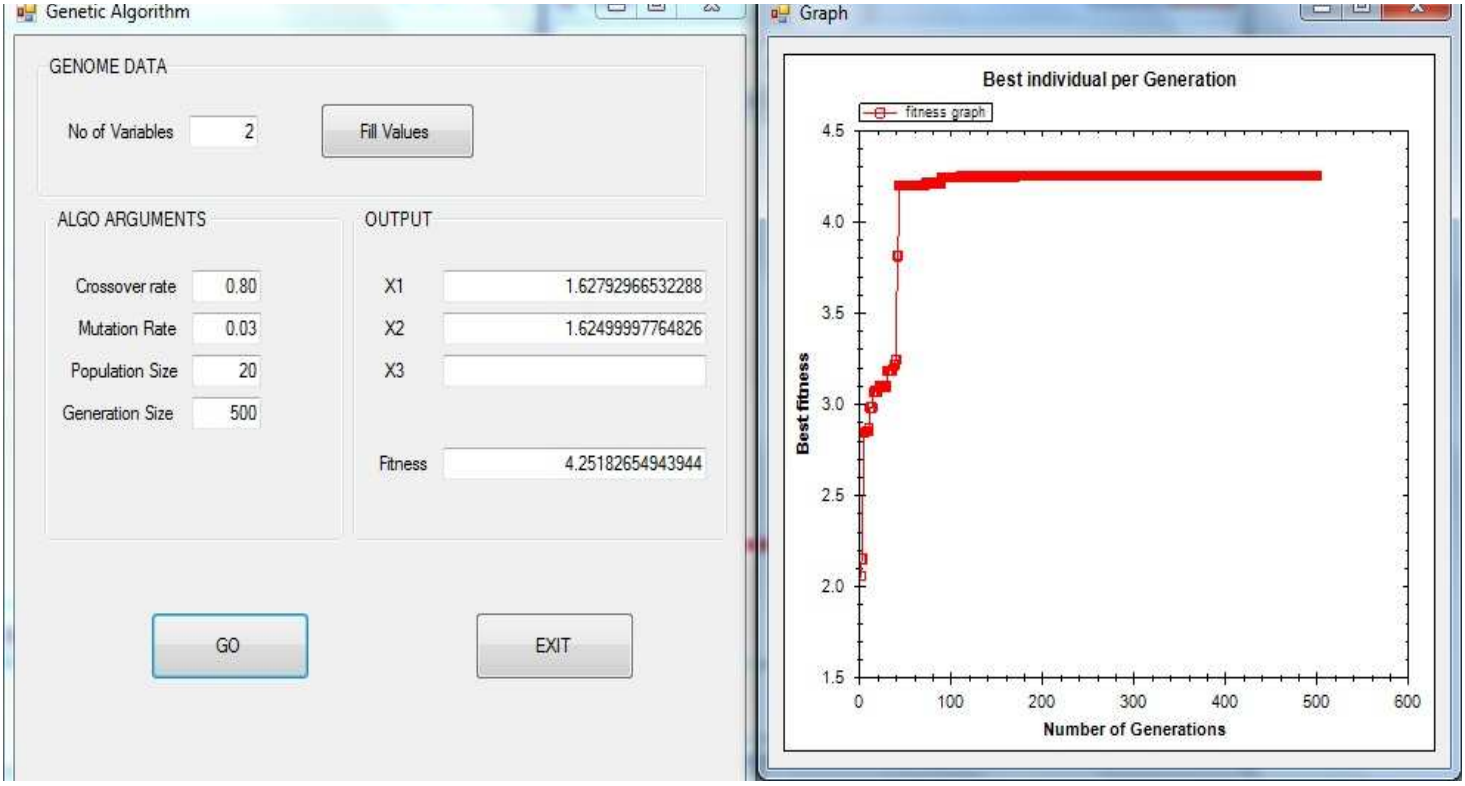



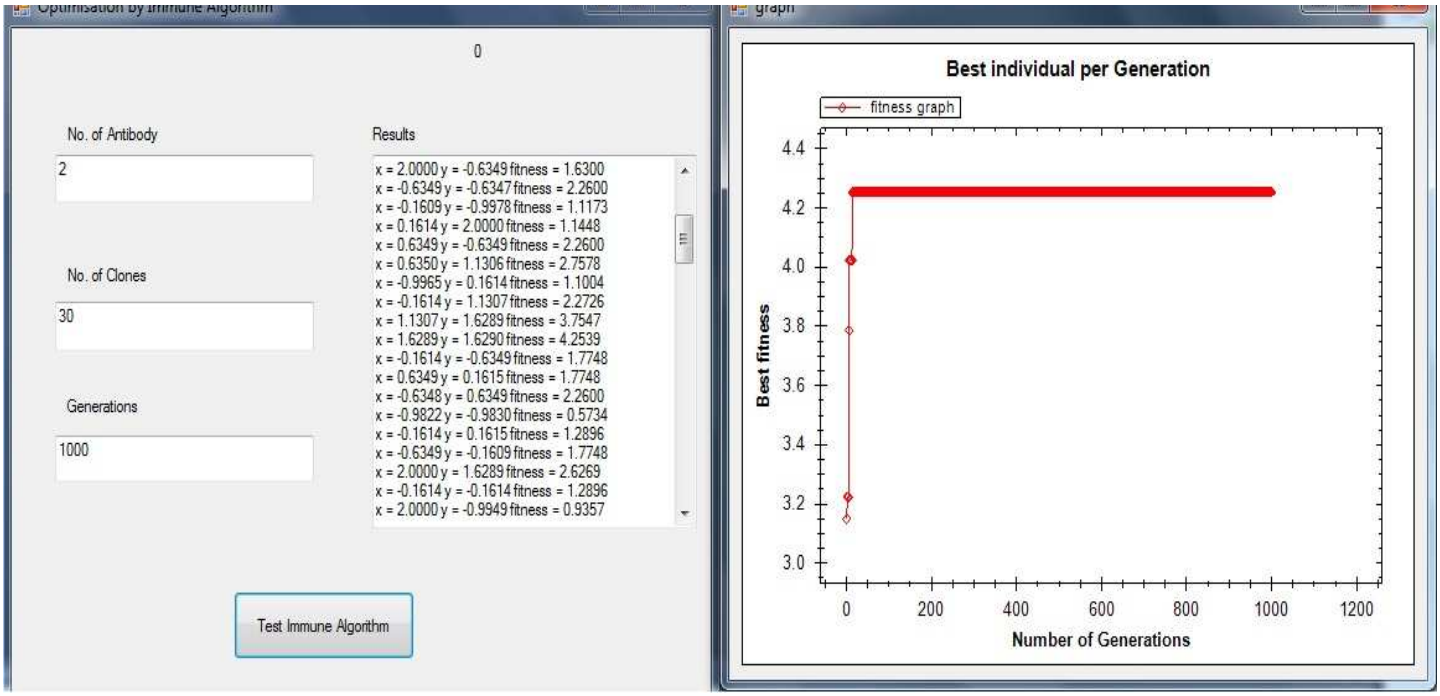



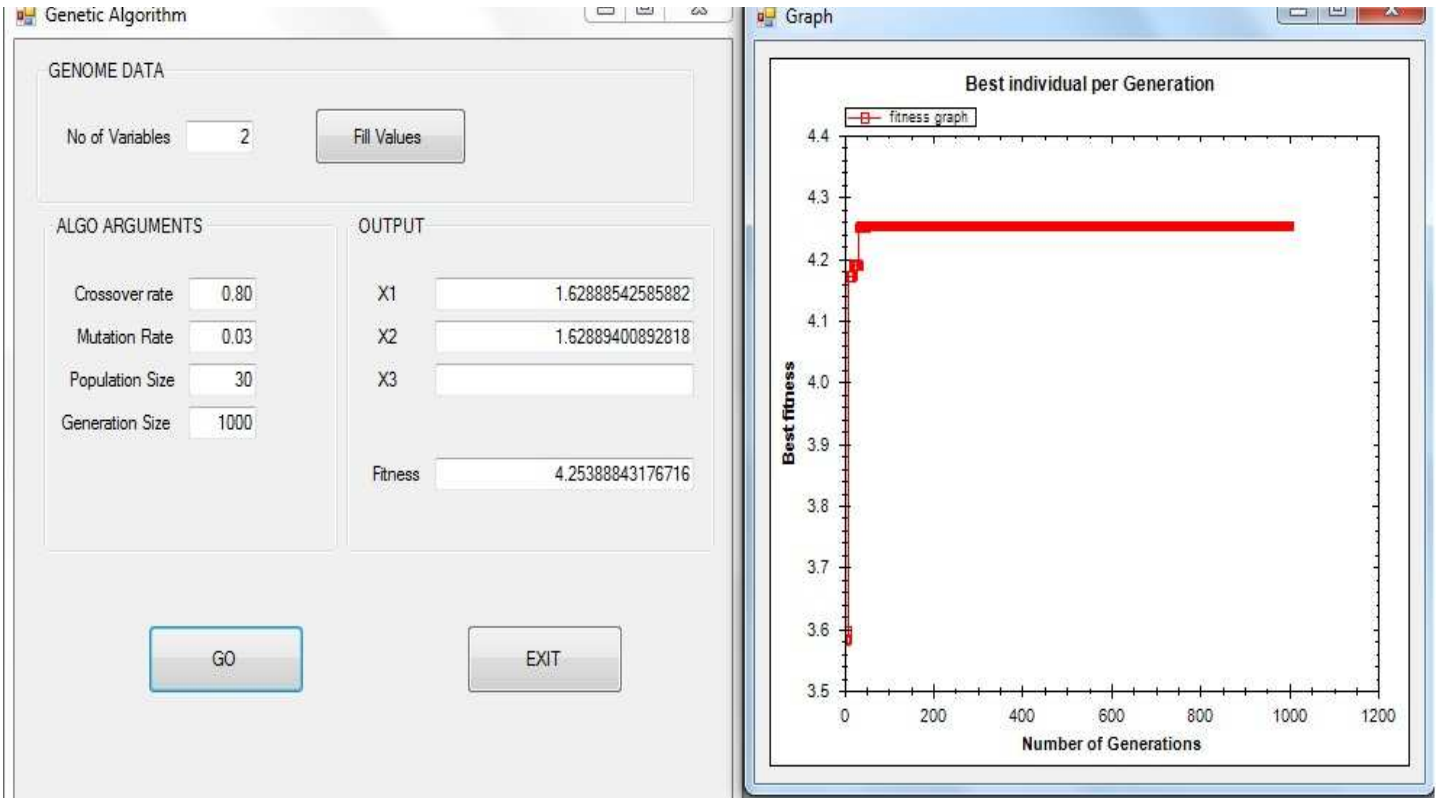

**Result shows that if No. of population size, clone size and generation size is same(10,10,200) than CSA perform batter than genetic algorithms.**




**Result shows that if No. of population size, clone size and generation size is same(20,20,500) than CSA perform batter than genetic algorithms.**




**Result shows that if No. of population size, clone size and generation size is same(30,30,1000) than CSA perform batter than genetic algorithms.**

# CHAPTER 6: CONCLUSION

The Clonal Selection Algorithm performs its search, through the mechanisms of somatic mutation and receptor editing, balancing of the best solutions with the exploration of the search-space. We have seen that the result improves with the number of generations.

The research was done to compare the result of Clonal Selection Algorithm with Genetic Algorithm. During the research it was found that Clonal Selection Algorithm maintains a diverse set of local optimum solutions, while the Genetic Algorithm tends to polarize the whole population towards the best one. The result depends upon the error generated. The system is developed to find the non-self of the body.

The tool was developed on C#.Net on Windows platform and has compared the two algorithms using Zed-Graph. It was found that when the size of the generation is small, both the algorithms give nearly equivalent results. But as we increase the size of the generation, immune algorithm tends to perform more and more better. This makes a decision in the favor of immune algorithm.

# CHAPTER 7: FUTURE WORK

During the last three decades there has been a growing interest in algorithms which rely on analogies to natural phenomena such as evolution, heredity, immunity and so on. The emergence of massively parallel computers made these algorithms of practical interest. All of these research above provide the possibility for the emergence of evolutionary algorithms, DNA computation techniques, artificial intelligence methods, etc.

Immune algorithms becoming very popular from last decade . So here is lot of work has to be done. It produces very efficient tool for computing large problems. There are many modification may be possible in clonal selection algorithms.

Genetic algorithm is a powerful tool to solve variety of NP-hard problems. But, it is also very domain dependent that is we cannot apply same algorithm for all known type of problems. We have to change Genome structure of GA according to the problem within hand. This is not an easy task because it needs a variety of encoding and decoding techniques and also mutation and crossover function need to be changed. Therefore, a lot can be done in this area.

To get an idea about problems solved by GA, here is a short list of some applications:

- Nonlinear dynamical systems - predicting, data analysis
- Designing neural networks, both architecture and weights
- Robot trajectory
- Evolving LISP programs (genetic programming)
- Strategy planning
- Finding shape of protein molecules
- TSP and sequence scheduling
- Functions for creating images

As far as this project is concerned, all the three functions have not tested the capability of this project. This algo is designed for taking any number of parameters x1, x2, x3 and so on. We have tested it only on two variables objective function. Therefore, it is not limited to only three objective functions given. As long as bit string encoding is good for any optimization problem like in case of real numbers we can implement this GA for that problem.

Genetic algorithm is a probabilistic solving optimization problem which is modeled on a genetic evaluations process in biology and is focused as an effective algorithm to find a global optimum solution for many types of problem. It has been shown that the genetic algorithm perform better in finding areas of interest even in a complex, real-world scene. Genetic Algorithms are adaptive to their environments, and as such this type of method is appealing to the vision community who must often work in a changing environment. However, several improvements must be made in order that GAs could be more generally applicable. Grey coding the field would greatly improve the mutation operation while combing segmentation with recognition so that the interested object could be evaluated at once. Finally, timing improvement could be done by utilizing the implicit parallelization of multiple independent generations evolving at the same time.

There are many limitations of GA which needs to be worked upon. It includes the following.

- The Genetic Algorithm requires that population considered for the evolution should be moderate or suitable one for the problem (normally 20-30 or 50-100)
- It is also necessary that crossover rate should be 80%-95% for better results.
- Mutation rate should be low i.e. 0.5%-1% for genetic algorithm to work properly.

We also require developing new crossover methods for more convergence in single step. Also there should be new mutation methods for reducing divergence due to excess mutation and a function to decide the initial number of population.

# CHAPTER 8: REFERENCES AND BIBLIOGRAPHY


[1] **Jerne, N. K. (1973)**, "The Immune System", *Scientific American*, **229**(1), pp. 52-60.

[2] **Jerne, N. K. (1974)**, "Towards a Network Theory of the Immune System", *Ann. Immunol.* (Inst. Pasteur) **125C**, pp. 373-389.

[3] **Janeway Jr., C. A. (1992)**, "The Immune System Evolved to Discriminate Infectious Nonself from Noninfectious Self", *Imm. Today*, **13**(1), pp. 11-16.

[4] **Janeway Jr, C. A. & P. Travers (1997)**, "Immunobiology The Immune System in Health and Disease", Artes Médicas (in Portuguese), 2nd Ed.

[5] **Mannie, M. D. (1999)**, "Immunological Self/Nonself Discrimination", *Immunologic Research*, **19**(1), pp. 65-87.

[6] Learning and Optimization Using the Clonal Selection Principle 2002

[7] L. N. de Castro and J. Timmis, *Artificial Immune Systems: A New Computational Intelligence Approach*. London, U.K.: Springer-Verlag, 1996

[8] An interdisciplinary perspective on artificial immune systems J. Timmis, P. Andrews, N. Owens, E. Clark (2008)

[9] **www.cs.ucl.ac.uk/staff/p.bentley/teaching/L9_AIS.pdf**

[10] **http://www.kxcad.net/cae_MATLAB/toolbox/gads/f10173.html**

[11] http://www.msci.memphis.edu/~dasgupta

[12] http://www.cs.princeton.edu/~stevenk/

[13] http://www.zpr.Uni-Koeln.DE/~filippo/

[14] http://www.cs.ucl.ac.uk/staff/J.Kim/

[15] Mr. Kashif Waqas1 Dr. Rauf Baig1and Dr. Shahid Ali2,Feature Subset Selection Using Multi-Objective Genetic Algorithms

[16] Deb, K. "Multi-Objective Optimization Using Evolutionary Algorithms". Reading, John Wiley & Sons, Ltd, Reprinted April 2002.

[17] Srinivas, N., Deb, K. “Multi-Objective Optimization Using Non-Dominated Sorting In Genetic Algorithms”. Evolutionary Computation, Vol. 2. No. 3, pp 221–248, 1994.

[18] Artificial Immune System: Part-1, By Leandro Nunes de Castro

[19] Artificial Immune System: Part-2, By Leandro Nunes de Castro

[20] **Dasgupta, D., (1997)**, “Artificial Neural Networks and Artificial Immune Systems: Similarities and Differences”, *Proc. of the IEEE SMC*, **1**, pp. 873-878.

[21] **Leandro Nunes de Castro(2002),** comparing immune and neural network .

[22] **S. Rajasekaran, G A Vijaylakshmi pai “**neural network, fuzzy logic ana genetic Algorithms” prentice hall india 2006

[23] **Yang, y, and C.K Soh (2000)** fuzzy logic integrated genetic programming for optimization and design ASCE

# APPENDIX

## ZED GRAPH

Zed Graph is a class library, Windows Forms User Control, and ASP web-accessible control for creating 2D line bar and pie graphs of arbitrary datasets. The classes provide a high degree of flexibility- almost every aspect of the graph can be user modified. At the same time, usage of the classes provides high degree of flexibility- almost every aspect of the graph can be user modified. Zed Graph is compatible with .Net 2.0 and VS.Net 2005.

## Using Zed Graph as a User Control:

Zed Graph is accessible as a control from the control toolbox in Visual Studio .Net.
To access Zed Graph first launch Visual Studio .Net, and create a new Windows Application (Forms) project. Open the form design so that it appears in the current window. View the toolbox and right click inside the “General” or “Components” sub panel of the tool box and select “Choose item” option. Click browse and navigate to the zed graph.dll file. Once this file is added, you should see a Zed Graph Control option in the tool box.

1. Select add reference from project menu and use the browse button to find Zed Graph.dll and click ok.
2. Add a using Zed Graph entry to your main form code.
3. Drag the Zed Graph Control from the tool box over to the form.
4. All the Zed Graph functionality is accessible through the ZedGraphControl. MasterPane property.