IMPLEMENTATION OF A CLONAL SELECTION ALGORITHM

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

By

Srikanth Valluru

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

December 2014

Fargo, North Dakota

North Dakota State University

Graduate School

Title

IMPLEMENTATION OF A CLONAL SELECTION ALGORITHM

By

Srikanth Valluru

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

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Simone Ludwig
Advisor

Dr. Xiangqing Wang Tangpong

Dr. Gursimran Walia

Approved by Department Chair:

12/17/2014	Dr. Brian Slator
Date	Signature

ABSTRACT

Some of the best optimization solutions were inspired by nature. Clonal selection algorithm is a technique that was inspired from genetic behavior of the immune system, and is widely implemented in the scientific and engineering fields. The clonal selection algorithm is a population-based search algorithm describing the immune response to antibodies by generating more cells that identify these antibodies, increasing its affinity when subjected to some internal process. In this paper, we have implemented the artificial immune network using the clonal selection principle within the optimal lab system. The basic working of the algorithm is to consider the individuals of the populations in a network cell and to calculate fitness of each cell, i.e., to evaluate all the cells against the optimization function then cloning cells accordingly. The algorithm is evaluated to check the efficiency and performance using few standard benchmark functions such as Alpine, Ackley, Rastrigin, Schaffer, and Sphere.

ACKNOWLEDGEMENTS

I would like to take the opportunity to thank my advisor Dr. Simone Ludwig for her continues support and valuable advice with continuous mentoring helped me in completing my research. I would also like to thank my committee members Dr. Xiangqing Tangpong, Dr. Gursimran Walia for being a part of my research and for their consistent support. Finally I would like to thank my parents and specially my brother Srinivas Valluru, without his support I wouldn’t have come this far and others who have been a great support throughout my career and always had belief in me that I could do it.

DEDICATION

Dedicated to,

My parents and my brother Srinivas Valluru

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Optimization is a widely used process to get optimal results for given conditions. The optimization can be used in different fields like designing, construction, maintenance and engineering in order to allow to take dynamic decisions. In every field, the main aim of optimization is to maximize or minimize a given objective. Mainly the optimization has three elements of problems decisions, constraints, and objective. To develop a mathematical model of an optimization problem, it requires mathematical terms or symbols to represent those three elements.

1.1. Decisions

In an optimization problem the decisions are symbolized in mathematical representation [2] as $A_1, A2, A3, \ldots\ldots\ldots\ldots\ldots\ldots, An.$

Here, $A_1, A2, A3, \ldots., A_n$ represents the decision variables.

1.2. Constraints

In an optimization problem, the constraints are expressed in several mathematical representations. The constraint relationships in an optimization problem are expressed in different ways. Those are

$f(A_1,A_2,A_3,\ldots.,A_n) \leq$ b less than or equal to constraint

$f(A_1,A_2,A_3,\ldots.,A_n) \geq$ b greater than or equal to constraint

$f(A_1,A_2,A_3,\ldots.,A_n) =$ b equal to constraint.

The constraint is representing one function with the decision variables. The function should be less than or equal to, greater than or equal to, or equal to some specific value. The function "$f(A_1,A_2,A_3,\ldots.,A_n)$" is represented by the left hand side (LHS) of the constraint and the value "b" is represented by the right hand side (RHS) of the constraint. The given optimization problem occurs in different type of situations and problems. To solve those problems one can use a number of constraints [2].

1.3. Objective

In an optimization problem, the objective of a mathematical representation is in the form of an objective function.

The general form of objective function is

$$\text{MAX (or MIN): } f(A_1, A2, A3, \ldots, A_n) \tag{1}$$

Using the objective function the decision maker takes decisions on the function in terms of decisions variables to maximize or minimize.

An optimization problem can be described as a general mathematical formulation problem.

$$\text{MAX (or MIN): } f(A_1, A2, A3, \ldots, An)$$

$$f(A_1, A2, A3, \ldots, A_n) \leq b \tag{2}$$

$$f(A_1, A2, A3, \ldots, An) \geq b \tag{3}$$

$$f(A_1, A2, A3, \ldots, A_n) = b \tag{4}$$

Equation 1 represents the objective function that will be maximized or minimized when the constraints satisfied by Equations 2-4. The main aim of the optimization is to find the decision variable values to maximize or minimize the objective function without violating any of the constraints [2].

The optimization categorized into many different fields that exist today and nature provides some of the most efficient ways to solve problems. Certain algorithms that are inspired by nature, which resembles its features and imitate a certain process from nature, are nature inspired algorithms. The nature inspired algorithms further split into categories of evolutionary computation and swarm intelligence.

Evolutionary algorithms are inspired by selection, the mechanism that closely mimics biological evolution. Evolutionary algorithms follow the percept of natural selection from a given population based on the survival of the fittest. The basic principle the evolutionary algorithm follows would be selection and mutation to generate candidates for the next generation and apply the fitness function. The fitness in the present generation is now used to seed the next generation and to generate offspring for the current generation by mutation until a good candidate solution is reached.

Evolutionary algorithm implementing genetic algorithm uses a population of individuals, where an individual is referred to as a chromosome. A chromosome defines the characteristics of the individuals in the population. The survival strength of the chromosome is measured based on the function under evaluation and the fitness value of each individual chromosome is determined. The best chromosomes are selected based on their fitness value to reproduce the next generation of chromosomes by the crossover process. Each individual chromosome in the population can also undergo the mutation process. The existing population is now replaced with the new population and the survival strength individual chromosome uses the fitness function, which reflects the objective and constraint of the problem to be solved, and the culling steps are repeated until a predetermined value is reached [3].

Swarm intelligence is originated from the study of colonies and it is collated behavior of disintegrated systems. The design of very efficient optimization and clustering algorithms is prompted from studies of the social behavior of swarms. The swarm intelligence is mostly motivated from insects, bees, birds, fish and insects that follow a pattern when moving in a swarm or flock. Simulation on swarm intelligence lead to the study of Particle Swarm Optimization, which is based on the stochastic optimization technique developed by Dr. Eberhart and Dr. Kennedy in 1995, where the algorithm uses the swarming behavior of fish or birds [2].

The Ant Colony optimization is a population-based algorithm developed for combinatorial problems and uses the interaction of social insects like ants. The Firefly algorithm is proposed by Xin She Yang in 2007, it is a metaheuristic algorithm which uses flashing behavior of swarming fireflies to attract each other during the mating process [4].

In this paper, we describe the details about the clonal selection algorithm theory and takes this as an inspiration for the development of Artificial Immune Systems that perform computational optimization to obtain the best fitness. In order to test the fitness of the clonal selection algorithm we have implemented 5 different benchmark functions, by varying number of iterations and dimensions in optimization suite.

The arrangement of the chapters is as follows: Chapter 2 briefly introduces the artificial immune system architecture. Chapter 3 describes the basic immune inspired algorithms. Chapter 4 explains about the implementation of the clonal selection algorithm details. Chapter 5 describes the different benchmark functions, and the clonal selection algorithm implemented within the optimization suite. Chapter 6 shows the sample outputs of the results. Chapter 7 lists the conclusions and future work.

2. ARTIFICIAL IMMUNE SYSTEM

Artificial immune system (AIS) is a broad area of research in immunology as well as in engineering, which is intended to form a bridge between immunology and engineering. Artificial immune systems use ideas and the techniques such as mathematical and computational modeling gleaned from immunology, in order to design and implement algorithms to develop an adaptive system capable of performing a wide range of tasks in engineering and various areas of research. AIS has emerged in the early 1990s as a branch of computational intelligence. The early work by Hugues Bersini [5] on crossing the divide between computing and immunology has laid the foundation for immunology describing immune network theory, how a immune system maintains its memory, and how one develops models and algorithms from its property. The early work by Stephanie Forest in the field of computer security raised attention to the ability of the immune systems to recognize the distinction between antigens and antibodies, which helped in using the immune inspired techniques for the development of applications in computer security.

2.1. Artificial Immune System

The artificial immune system is mainly used to map the structure and functions of the immune system to the computational system investigating mathematics, engineering and information technology problems [1]. Furthermore, the artificial immune system is very useful to solve bioinformatics problems and modeling the biological process by applying immune algorithms [1].

The main aim of the biological immune system is to defend the body from distant molecules called the antigens. Finding the difference between foreign cells and native cells is the ability of pattern reorganization of biological immune systems. The immune system has different characteristics; those are uniqueness, autonomous, recognition of foreigner's cells, distribution detection and noise tolerance.

The architecture of the artificial immune system is mapped to biological immune system. All the mechanisms of artificial immune systems are inspired by the biological immune system model.

2.1.1. Problem Definition

An artificial immune system has two types of classification errors. When a self-string is classified as irregular it is known as a false positive error. When a non-self-string is classified as normal that is called false negative. The immune system tries to minimize the errors because in one body these both kinds of errors are very dangerous.

2.1.2. Detectors

The immune system we take as one distributed environment. For example, let us consider a graph G, which consists of vertex and edges (V,E). Here, each vertex acts as a detector, and each detector can move from one vertex to another vertex via the edges. At the vertex, the detector can communicate with another detector [3].

Here, we show a matching rule using the hamming distance. We adopt a more immunologically plausible rule called *r*-contiguous bits. The *r*-contiguous bits are common when two strings match. Here *r* is one threshold value that is determined by the detector.

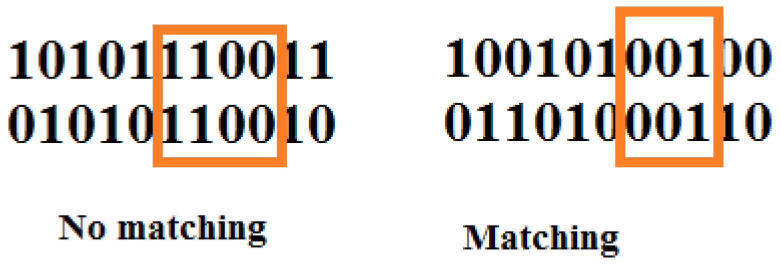


Figure 1: Matching under the contiguous bits match rule [3]

For example, if *r=l* that means every single string will match itself. If *r=0* the detector will match every single string of length *l*. Please note, that *r* is the threshold conformed by detector. In the above example, the string is matching *r=3* contiguous bits.

A detector should match at least T strings within a given period of time. At any time step the ℓ_{match} probability of the match will be reduced by one. The match count will be reset to zero when a detector is activated.

2.1.3. Memory

When multiple detectors are activated at a node using the non-self-strings *s*, a detector to become a memory detector enters into the competition. The detectors that have close matching bits with *s* will be selected to become memory detectors. The memory detectors make duplicate copies them-selves, which then spread out to neighboring nodes. This will be spread throughout the entire graph G [3]. In the future, any string matching string *s* will be detected very fast because every node have existing *s* matching bits. So, memory detectors have a lowered activation threshold value that means, they activated very fast re-occurrence of previously encountered non-self-strings.

2.1.4. Sensitivity

The sensitivity of an artificial immune system is very high. The detection of non-self-strings is very fast and detected within a short period of time.

2.2. Basic Immune Inspired Algorithms

Following are the few basic immune inspired algorithms discussed in this paper.

1. Negative Selection Algorithm
2. Immune Network Theory
3. Dendritic Cells

4. Clonal Selection

2.2.1. Negative Selection Algorithm

For a biological point of view, the purpose of the immune system is to identify all the molecules within the internal structure of the body and place them in groups as self-reactive cells and non-self-reactive cells. These non-self-reactive molecules are carried out to induce a defensive mechanism and are further divided into groups. For self-cells in the immune system, the native selection provides tolerance in order to have the ability to detect the unknown virus while not reacting to self-cells. For T-cells this mainly occurs in the thymus**,** where it undergoes a censoring process called native selection and thus providing an environment rich in antigens presenting cells that present self-cells. The T-cells that do not bind to self-cells are allowed to leave the thymus, and thus to protect the body against the virus or bacteria the matured T-cells move all around the body performing immunological tolerance [6].

The negative selection algorithm is one of the earliest artificial immune system algorithms, which was used in various real world applications. According to Forrest et al. [5], the first negative selection algorithm proposed in 1994 was to detect the data manipulation caused by virus in a computer system. The negative selection algorithm was based on the mechanism followed in the thymus that produces a set of matured T-cells capable of binding only non-self-antigens. The algorithm is to produce a set of detectors. The initial stage of the algorithm is to produce a set of self-string *S*, which defines the normal state of the system. In the generation stage, a set of detectors *D*, are generated to match self-cells with some process and censoring. The self-cells, which matches are removed and the rest are considered as detectors. In the detection stage, the available detectors set are used to verify whether the incoming data is self-cells or non-self cells. This process is repeated, highlighting the fact that data has been manipulated.

```
input   : $S_{seen}$ = set of seen known self elements
output  : D = set of generated detectors

begin

  repeat
    Randomly generate potential detectors and place them in a set $P$
    Determine the affinity of each member of $P$ with each member of the self-set $S_{seen}$
    If at least one element in $S$ recognizes a detector in $P$ according to a recognition threshold,
      then the detector is rejected, otherwise it is added to the set of available detectors $D$
  until  Stopping criteria has been met

end
```

Figure 2: Negative selection algorithm

2.2.2. Immune Network Theory

In the mid-seventies the Immune Network Theory was proposed. In recognition of antigens, the immune system maintains the interconnected B cells of the idiotypic network. It was first suggest that idiotypic network interactions are symmetrical. Depending on the symmetrical stimulatory, inhibitory and killing interactions, the detailed immune network theory is developed. It provides a skeleton to understand the large number of immunological phenomena dependent on small number of postulates. The immunological theory involves roles of different type of cells. For example, B cells to make antibodies, T cells to regulate the production of antibodies by B cells, and A cells (non-specific accessory cells).

The immune network theory proposed by Jerne in 1974, it explains the emergent properties of the immune system about recognizing the input pattern for learning the network theory and memory detectors to classify the patterns [5]. In this model, a set of patters *S* is recognized as the initial population and cloned population. In the initial stage, a random set of network antibodies *N* is considered and the affinity of each antibody in network *N* is determined. Then, the cells are cloned and mutated - these mutation yields a diverse set of antibodies, which can used in the classification procedure.

```
input   : S = set of patterns to be recognized, nt  network affinity threshold, ct  clonal pool
threshold, h  number of highest affinity clones, a  number of new antibodies to introduce
output  : N = set of memory detectors capable of classifying unseen patterns
begin
   Create an initial random set of network antibodies, N
   repeat
      forall  patterns in  S do
         Determine the affinity with each antibody in N
         Generate clones of a subset of the antibodies in N with the highest affinity. The number
of clones for an antibody is proportional to its affinity
         Mutate attributes of these clones to the set A , a and place h  number of the highest
affinity clones into a clonal memory set, C
         Eliminate all elements of C whose affinity with the antigen is less than a predefined
threshold ct
         Determine the affinity amongst all the antibodies in C and eliminate those antibodies
whose affinity with each other is less than the threshold ct
         Incorporate the remaining clones of C into N
       end
      Determine the affinity between each pair of antibodies in N and eliminate all antibodies
whose affinity is less than the threshold nt
      Introduce a random number of randomly generated antibodies and place into N
   end until a stopping criteria has been met
end
```

Figure 3: Immune network algorithm

Elimination of the elements of clonal memory set *C* whose affinity with antibodies is less than the predefined threshold *ct*. After eliminating the elements from the clonal memory set *C*, the affinities among all the antibodies in *C* are determined, and those antibodies whose affinities with each other is less than the threshold *ct* are eliminated. Until the binding is successful, the antibodies are incorporated into the network.

2.2.3. Dendritic Cell

In the regulation of the adaptive immune response, the dendritic cells play a critical role. Dendritic cells are antigen-presenting cells. The main function of dendritic cells is presenting antigens. Only dendritic cells have this capability compared to rest of the *T* lymphocytes. Dendritic Cells have the ability of capturing antigens, processing antigens, and presenting them on the cell surface along with appropriate co-stimulation molecules [10]. The main function of dendritic cells is divided into three categories, each of which are involved in antigen presentation.

1. T Cells Activation.
2. Immune Tolerance
3. B Cells Stimulation

The dendritic cell algorithm proposed by Greensmith et al. [5] is inspired from biological immune inspired algorithm based on the theory called Danger theory. The Danger theory was proposed by Matzinger [5] in the year 1994, and has become very popular in recent years with the classification from the self and non-self-viewpoint amongst immunologists introducing the concept of notion of danger signal to the immune system, which is produced by the ordinary cells of the body that have been injured due to an attack by pathogens at a given time. This context is integrated via a process inspired by the role of dendritic cells. This removes the need to define what self is, but adds the necessity to define the danger, safe and PAMP (Pathogen-associated Molecular Patterns) signals.

2.2.4. Clonal Selection Theory

The Clonal selection theory explains how the body has the essential ability to generate immense diversity of antibodies and it describes the basic features of an immune response to an antigenic stimulus. The basic principle of clonal selection theory is to consider those cells that can recognize the antibodies that reproduce rapidly. In 1659, Burnet [14] proposed the clonal selection theory based on the lifetime of an individual the immune system regularly undergoes a selection mechanism. The clonal selection theory states that when the immune system is binding with

```
input   : S  = set of data items to be labeled safe or dangerous
output  : D = set of data items labeled safe or dangerous
begin
Create an initial population of dendritic cells (DCs), D
Create a set to contain migrated DCs, M
forall  data items in S do
  Create a set of DCs randomly selected from D , P
    forall  DCs in P do
      Add data item to DCs collected list
      Update danger, PAMP and safe signal concentrations
      Update concentrations of output cytokines
      Migrate the DC from D to M and create a new DC in D if concentration of co-
stimulatory molecules is above a threshold
    end
   end
  forall  DCs in M   do   Set DC to be semi-mature if output concentration of semi-mature
cytokines is greater than mature cytokines, otherwise set as mature
  end
  forall data items in S   do
    Calculate number of times data item is presented by a mature DC and a semi-mature DC
    Label data item a safe if presented by more than semi-mature DCs than mature DCs,
otherwise label as dangerous
    Add data item to labeled set M
  end
end
```

Figure 4: Dendritic algorithm

suitable antibodies, the lymphocytes get activated. Once the cells are activated, the cells are cloned and new cells are the copies of their parents and forced to undergo a mutation mechanism with higher rates to ensure that the lymphocytes specific to activating the antibodies are produced in a large number. The newly differentiated lymphocytes that have the self-reactive receptors are eliminated, and thus, to draw forth a destructive adaptive immune response only antibodies form a microorganism such as bacterium allows the lymphocytes to have clonal expansion [9]. Here, the immune system will increase the number of antibodies and its differentiation into self and non-self, stating the non-self to be assumed to be from microorganism like bacterium that needs to be destroyed.

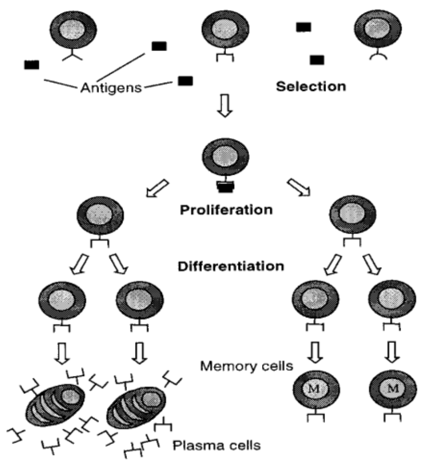


Figure 5: Clonal selection principle [9]

Here, the clonal selection theory explains how the immunological memory allows a rapid reaction upon a second exposure to an antigen. The immunological memory is the basis of natural immunity and artificial immunity. Each B cell has a specific antibody as a cell surface receptor. The agreement and creation of antibody genes occurs prior to any exposure to antigen when a soluble antigen is presented, it binds to the antibody on the surface of B cells that have the correct specificity. These B cell clones develop into antibody-producing plasma cells or memory cells. Only B cells, which are antigen-specific, are able of secreting antibodies. The early exposure to antigen, the plasma cells stop creating antibody and die. Memory cells remain in greater numbers than the initial B cells, allocating the body to quickly respond to a second exposure of the antigen [9].

The body can build low levels of soluble antibody about one week after exposure to the antigen. However, a second exposure to antigen produces a much faster response, and several orders of magnitude higher levels of antibody. The capability of the antibody to bind antigen also increases in the secondary response. The memory of antigen and stimulated response is the basis for vaccination. A secondary immune response is not only quicker, but also generates antibody with up to 10,000 fold increase in binding similarity. This higher similarity comes from a method that alters the variable regions of light and heavy chains of the memory cells by specific somatic mutation. This is a random process that by chance can improve antigen binding.

The system makes a mutation for each two cell divisions. Re-exposure to antigen is most likely to cause clonal expansion of memory cells, which produce the highest affinity antibody. The mutations, which lead to increased affinity, will be clonally selected by antigen similarly to the primary response. Cells with inactive antibody will die by apoptosis from a lack of T cells signaling.

The implementation of the clonal selection theory in artificial immune systems is applied to optimization problems. There are many different implementation approaches of the clonal selection theory in the computational field [11], and the features are: 1) the population size is dynamically adjustable, 2) exploitation and exploration of search space, 3) location of multiple optima, 4) capability of multiple solutions, 5) defining stopping criterion, and also specifically the antibody driven affinity maturation process of B-cells and its associated hyper mutation mechanism gave inspiration towards implementation of clonal selection theory in artificial immune system.

According to Leandro N. de Castro and Timmis [16], considering these features they have highlighted the importance of affinity maturation of B-cells in computation field stating that the higher the affinity that binds during the proliferation B-cells, the more clones are produced. The mutations suffered from B-cells are inversely proportional to the affinity of the antigen it binds. De Castro et al. developed an immune algorithm, named CLONALG that was developed to perform pattern recognition and optimization tasks. This algorithm is capable of learning a set of input patterns by selecting, reproducing and mutating a set of artificial immune cells.

The following steps implement the clonal selection theory when applied to Pattern matching [5].

De Castro and Jerne implemented the CLONALG, which was used in several data analysis and clustering applications. A subsequent development leads to the implementation of the optimization version of aiNet [19].

```
Input   : S = set of patterns to be recognized, n the number of worst elements to select for
removal
 output  : M = set of memory detectors capable of classifying unseen patterns
begin
  Create an initial random set of antibodies, A
  forall  patterns in S   do
    Determine the affinity with each antibody in A
    Generate clones of a subset of the antibodies in A with the highest affinity.
      The number of clones for an antibody is proportional to its affinity
    Mutate attributes of these clones to the set A , and place a copy of the highest
      affinity antibodies in A  into the memory set, M
    Replace the n lowest affinity antibodies in A with new randomly generated antibodies
  end
end
```

Figure 6: Clonal selection algorithm

3. OPTIMIZATION OF IMMUNE NETWORK USING CLONAL SELECTION ALGORITHM

The following describes the basic terminology considered to present the optimization version of the artificial immune network using the clonal selection theory [16].

Network cell: The number of individuals of the population in a network cell, where each cell is a real-valued vector in a Euclidean search space;

Fitness: Fitness of the cell in relation to the objective function to be optimized to either minimum or maximum. The value of the function when evaluated for a given cell;

Affinity: Euclidean distance between two cells;

Clone: The offspring cells that are identical copies of their parent cell with further somatic mutation so that they become variation of their parent.

The summary of optimization version of artificial immune network using clonal selection theory is as follows:

1. Initialize the population of cells randomly.
2. The fitness of each network cell is determined.
3. Evaluate all cells against the optimization function then clone accordingly.
4. Perform clonal selection and network cell interaction and generate number of clones for each network cell.
5. All cells in the network undergo cloning, mutation and selection. Only the clones of the cell are mutated, with the parent remaining unchanged.
6. A set of clones is produced depending on the number of clone’s parameter. All clones are mutated proportional to the fitness evaluated against the optimization function.

7. If the best clone produced has a better fitness than its parent cell then the clone replaces the parent in the network. If not, then the parent remains in the network.
8. The network interactions do take place between the network cells. Affinities are calculated between all network cells and if two cells have the affinity below a pre-defined threshold, then the cell with the lowest fitness are deleted from the network.
9. Introduce a percentage d% of randomly generated cells and return to the process of determining the fitness.
10. The minimum best fitness at each iteration is plotted on the graphs, plotting the fitness values on to y-axis and iterations on x-axis.

The behavior of the algorithm is explained in the above steps. At each iteration a population of cells is optimized locally through the affinity proportion. Here, we can infer that no parent cell has the selective advantage over the other contribution. In this loop, once the population reaches the stable state, the cells interact with each other in the network. In order to avoid redundancy in the network the similar cells are removed from the network form. The current population is added with the randomly generated cells from the network and the process of optimization starts again. The affinity proportion mutation is calculated by the following equations [16]:

$$\begin{aligned} \boldsymbol{c'} &= c + \boldsymbol{a}\, N(0,l), \\ \boldsymbol{a} &= (1/P)\exp(-f^{*}\, 1), \end{aligned} \tag{5}$$

Where *c'* is mutated cell *c*, *N(0,1)* is the Gaussian random variable,

P is a parameter that controls the decay of the inverse exponential function,

*f** is fitness of an individual.

Here, the algorithm runs until a stopping criterion is met. The stopping criteria used in this algorithm are based on the size of the population and the number of clones for each cell in the network. As only some cells are going to remain in the standard state after the network suppression, if we do not see any variation from one suppression to another suppression then the network remains to be stable, which implies that the remaining cells are memory cells resembling solutions to the problem.

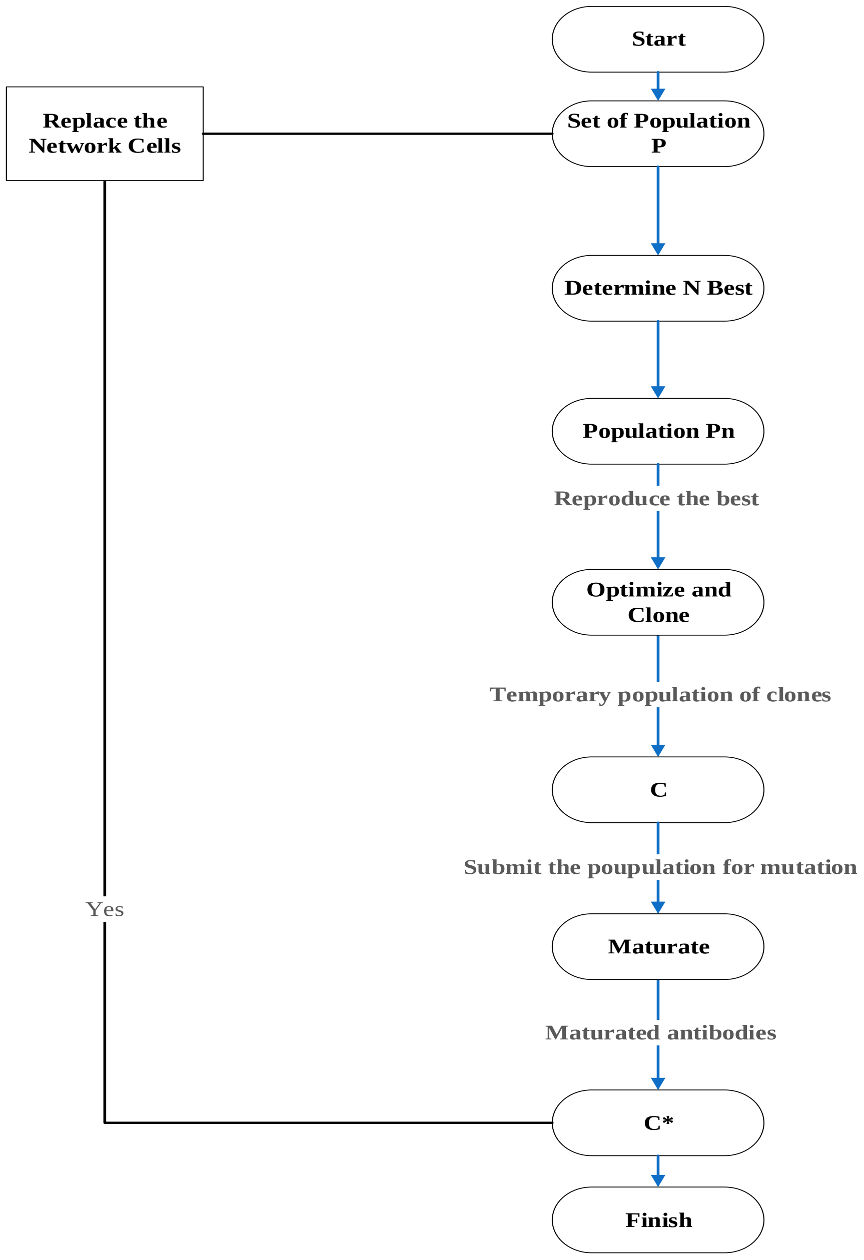


Figure 7: Simplified flow chart of the clonal selection algorithm

4. IMPLEMENTATION DETAILS

As part of this project a C# implementation of an optimization suite was provided wherein the artificial immune network using clonal selection was to be implemented. We briefly discuss classes, methods used to implement this algorithm in C# in the further sections. The inputs required for this application are number of initial cells, number of clones for each network, number of algorithm iterations, suppression threshold for network cell, clonal selection average error threshold, and affinity proportionate mutation parameter, number of dimensions for optimization, lower bound and upper bound for each dimension.

4.1. Running Optimal Lab System

Here the C# optimization suite runs by loading all the plugins and dll's of all the algorithms integrated. Initially it calls public MainForm() and loads the splash screen and loads the libraries required to open the main window to select the optimization task. Figure 10 shows the screen showing the C# optimization suite's main window.

```
SplashScreen splash = new SplashScreen();
splash.ShowDialog();
splash.Update();
InitializeComponent();
```

Figure 8: Code for splash screen

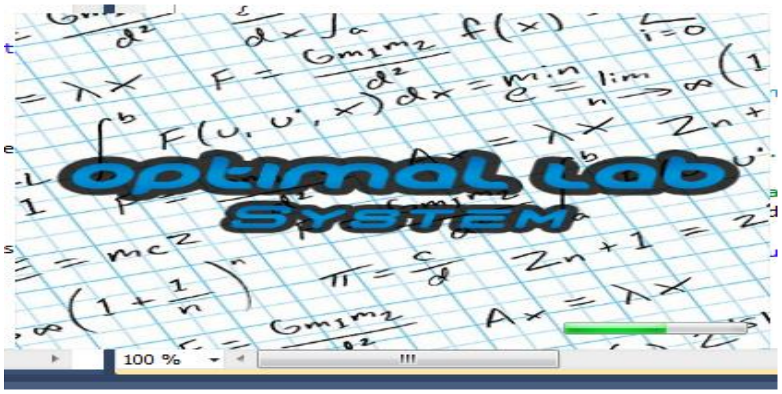


Figure 9: Shows loading splash screen

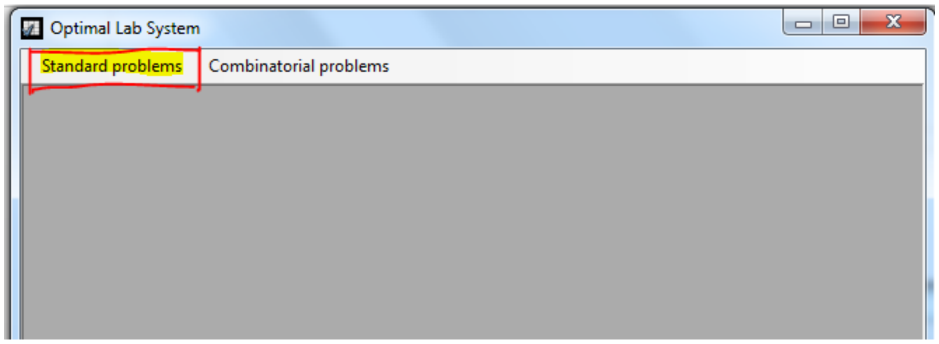


Figure 10: Program main window

Class Libs is used to get the list of library names and load the libs as per path specified by the user. It also takes the plugin category from the list to select the optimization categories in the new optimization window.

```
public class Libs
{
public string libName { get; set; }
public string libPath { get; set; }
public Plugin.PluginCategory libCategory { get; set; }
public List<Plugin.ParameterDesc> Parameters { get; set; }
public List<string> Authors { get; set; }

}
```

Figure 11: Code showing library files

The figure 12 displays selecting the “Artificial Intelligence” Tab in the “New Optimization” form displaying a panel having different types of algorithms added under the “Artificial Intelligence” Tab using the base class.

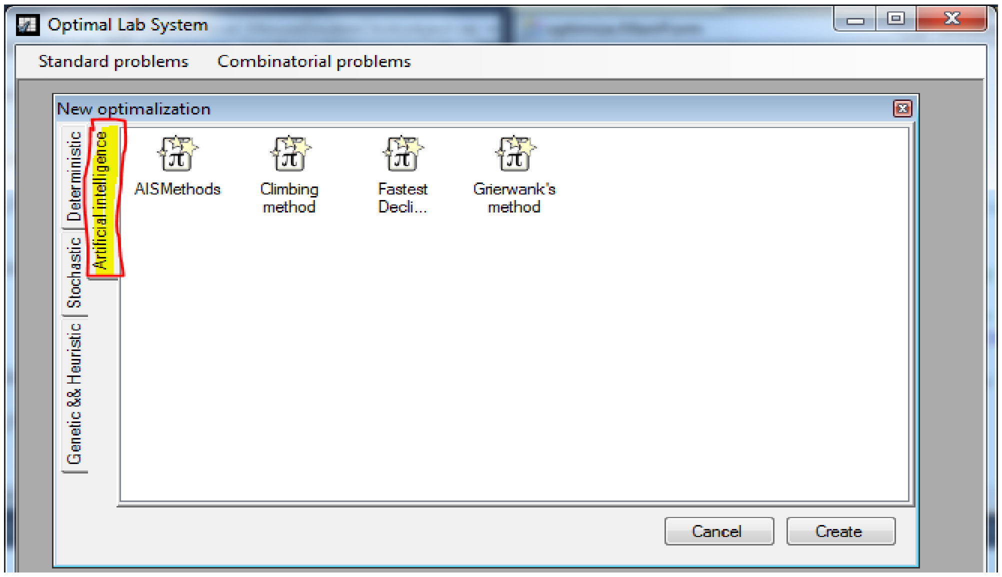


Figure 12: Showing the optimization categories

4.1.1. Controls Used in the Forms

The controllers used in the optimal labs system are added in the following method representing the basic code used to load the controls dynamically like Buttons, Labels, textbox, checkbox, Panels, Tabs and text fields.

```
public void controlsadd()

foreach (var item in editableparameters)
{
TextBox txt;
Label lbl;
Button btn;
switch (item.Type)
{
case Plugin.InputType.Restriction:{…}
            case Plugin.InputType.Integer:{…}
            case Plugin.InputType.Double:{…}
            case Plugin.InputType.Text:{…}
            }
}
```

Figure 13: Code showing controllers added

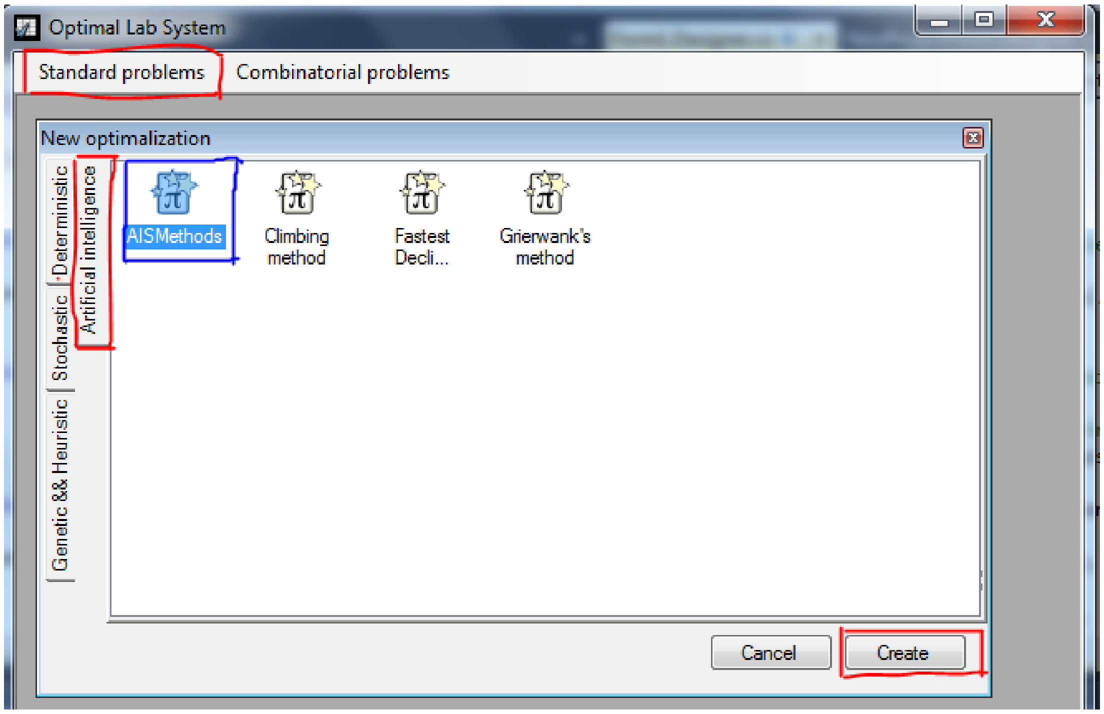


Figure 14: Shows select artificial immune system method from the categories listed

4.1.2. Benchmark Functions

The following are the benchmark functions added to the code in order to evaluate the algorithm by calculating the fitness value of each network cell of artificial immune network using clonal selection.

```
BenchmarkList.Add(new Custom());
BenchmarkList.Add(new Alpine());
BenchmarkList.Add(new DeJong());
BenchmarkList.Add(new Rastrigin());
BenchmarkList.Add(new Schweffer());
BenchmarkList.Add(new Ackley());
BenchmarkList.Add(new Sphere());
```

Figure 15: Code showing benchmark functions

BenchmarkList.Add() is used to create an object in the code for each benchmark function, which calls to “Receive Formula”. Depending on the benchmark function selected it will create the object with the necessary parameters. The figure 16 displaying the user to select the benchmark functions from the list of the AISMethod.

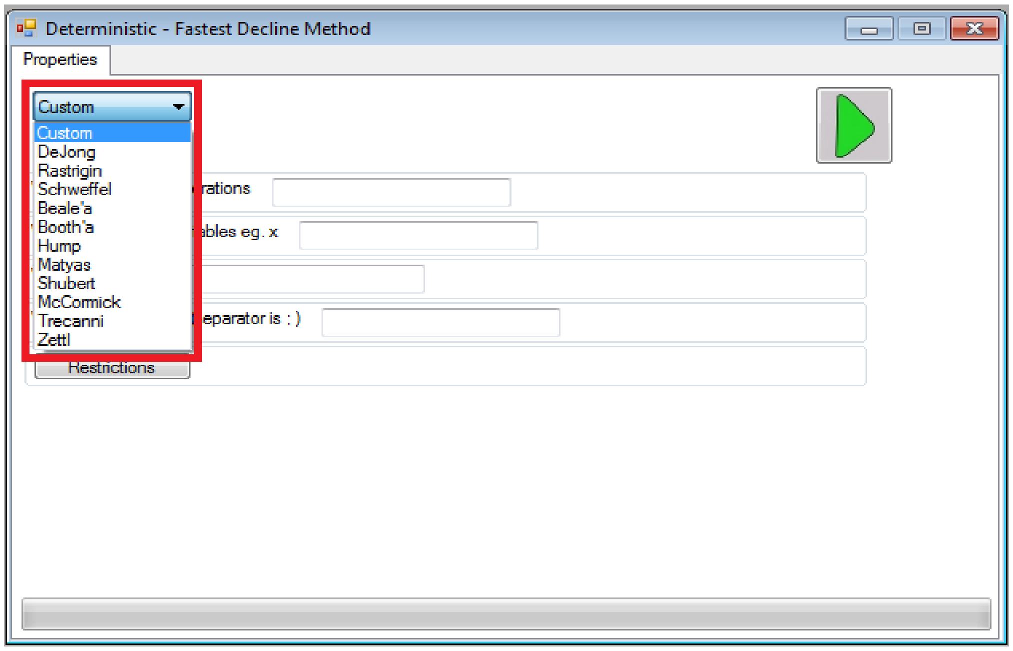


Figure 16: Showing selecting the benchmark function from the list

4.1.3. Input Parameters

```
public void defaultparameters(int id, int lwymiarow)
{
defaultitems.Clear();
//List<ParameterItem> p = new List<ParameterItem>();
MainForm.BenchmarkList[id].ReciveFormula(lwymiarow);
defaultitems.Add(new ParameterItem { Name = "Formula", Value =
MainForm.BenchmarkList[id].Formula });
defaultitems.Add(new ParameterItem { Name = "VariableNames", Value
= MainForm.BenchmarkList[id].variableNames });
defaultitems.Add(new ParameterItem { Name = "VariableRestrictions",
Value = MainForm.BenchmarkList[id].Restrictions });
}
```

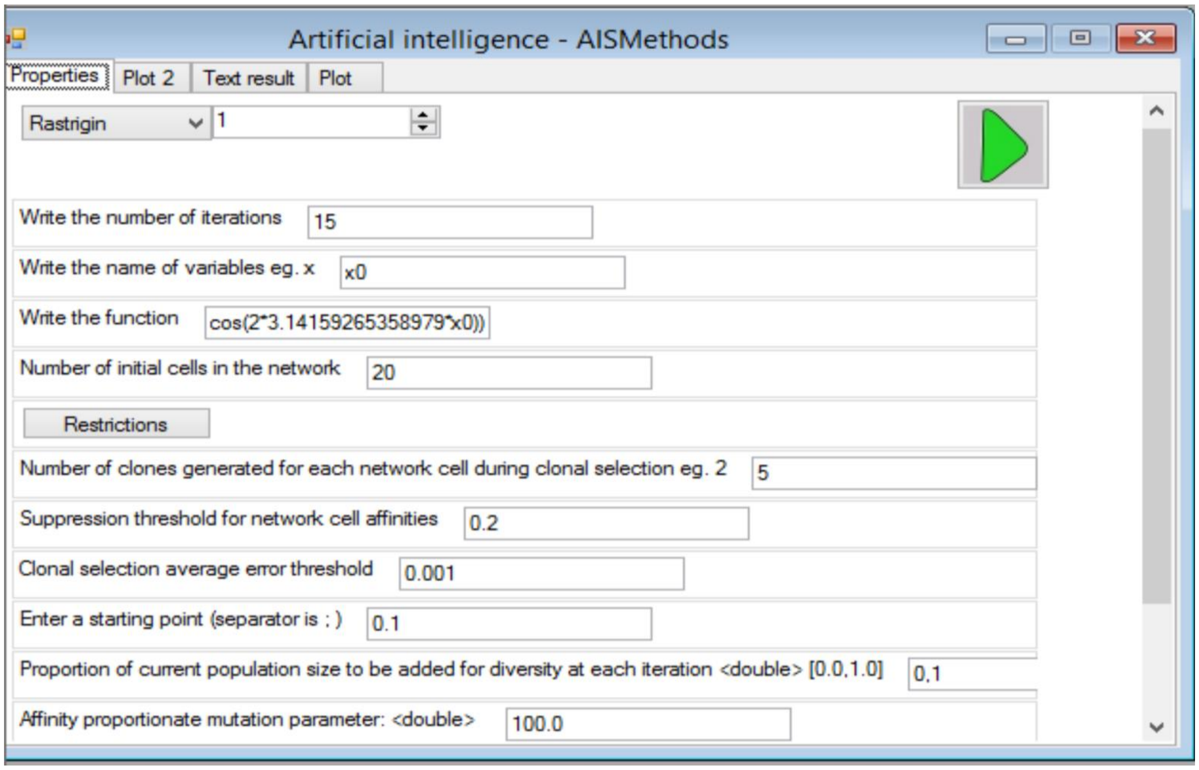
Figure 17: Code showing default parameters

This method is used to initialize the formula for the selected benchmark function, and to initialize the default variables for the fitness function providing the variable restriction for the selected benchmark function, which loads automatically once the benchmark function is selected from the list.

SetDefaultValues(): This method is used to set the default values in the controls for functions, name of variables, and restrictions taken from default parameters.

Type pluginType = pluginAssembly.GetExportedTypes()[0];

This method is used to load the assembly dynamically and we can select assembly so that the assemble parameters are loaded to set the initial values. Figure 18 displays the input parameter fields to enter for the AISMethod after integrating the algorithm within the optimization suite.



Rastrigin	1
Write the number of iterations	15
Write the name of variables eg. x	x0
Write the function	cos(2*3.14159265358979*x0))
Number of initial cells in the network	20
Number of clones generated for each network cell during clonal selection eg. 2	5
Suppression threshold for network cell affinities	0.2
Clonal selection average error threshold	0.001
Enter a starting point (separator is ;)	0.1
Proportion of current population size to be added for diversity at each iteration <double> [0.0,1.0]	0.1
Affinity proportionate mutation parameter: <double>	100.0

Figure 18: Showing input parameters passed fields in optimization suite

4.2. Methods and Classes Used to Implement Opt Artificial Immune Algorithm

4.2.1. Run AISMethod Class

This class is the main class that loads the Iplugin of the optimization suite and all the classes of the AIS category implementation. This class has methods to call Authors List, PluginCategory, Default parameters used, Resume and Stop states, and this class has variables which hold input parameter lists for number of iterations, initial cells in the network, number of clones to be generated for each network cell, suppression threshold, affinity proportion, number of dimensions, boundary points. All the methods used in this project are called from this class. The main functionality of this class is to run the optimization task to find the best fitness at each iteration by varying population size and number of clones.

4.2.2. NetworkCell Class

This class represents the network cells of the optimization of AIS algorithm. Each cell has a real-valued vector representing a possible solution to the function being optimized. For each value in this vector lower bound and upper bounds exist. Each cell also holds its fitness value and provides methods to mutate dimension values, clone the cell, compare affinity with another cell, and to evaluate the cell against the optimization function.

Table 1: Methods of AIS Main class

Return Type	**Method**	**Purpose**
Void	Main()	This method is used to load the initial Iplugin in the optimization suite.
Void	List<ParameterDesc>	This method is used to input parameters including the default parameters from the optimization suite entered.
Void	Start(),Resume(),Stop()	These methods are used to start the optimization suite to run on AIS Category to find the best fitness at each iteration and to plot the result.

4.2.3. OptAInet Class

This class is used to represent the main functionality of the optimization of the Artificial Immune network algorithm, which creates the new instance for optimization settings for the given input parameters in the optimization suite. Table 3 shows the methods and variables used to represent this class.

Table 2: Methods of NetworkCell class

Return Type	**Method**	**Purpose**
Void	Mutate()	This method is used to mutate each dimension in the network cell.
Double	getAffinity()	This method is used to get the affinity value between the network cells.
Object	Clone()	This method is used to clone each network cell and make a copy.
Double	getFitnessNorm()	This method is used to get the normalized fitness of the network cell.
Void	Evaluate()	This method is used to evaluate the network cell against the optimization function.

Table 3: Methods of OptAInet class

Return Type	**Methods**	**Purpose**
Void	Optimize()	This method provides the control of the loop for the optimization of the artificial immune network algorithm and runs until a stopping condition is met.
Void	Addcells()	This method is used to add a specified number of cells to the network.
Double	clonalSelection()	This method is used to have main control loop for the affinity maturation stage of the optimization of Artificial immune network algorithm.
Void	evaluateCells()	This method is used to evaluate each network cell against the optimization function and sets the fitness for each network cell.
Void	cloneCells()	This method is used to generate the number of clones for each network cell producing a clonal pool.
Void	network Interactions()	This method is used to carry out the interaction between the network cells.

4.2.4. OptFunction Class

This class is used to represent the optimization problem that is to be solved by this algorithm, which contains just one method that evaluates an array of real valued numbers that represent the dimension values of a network cell. To evaluate an array of real value numbers that represent the dimensions of a network cell, returning a single real number value that represents the fitness of the provided input in the optimization suite. Table 4 shows the benchmark functions used to find the fitness value for each network cell provided.

Table 4: Methods of OptFunction class

Return Type	**Method**	**Purpose**
double	OptFunction()	This method is used to calculate the fitness value of each cell and it is called depending on the function selected in the optimization suite for AISMethod.

5. BENCHMARK FUNCTIONS AND SETTINGS

In the real world, optimization problems are discrete and non-discrete. In order to validate the performance and reliability of an optimization problem there is the need of various test functions, which can compare and validate the optimization algorithm. The test function involving the optimization is commonly used in the evolutionary computation area in order to compare algorithms on large test sets. According to the *no free lunch theorem* [20], if two algorithms are tested on all possible functions then they would perform the same way on average that is the overall performance of both algorithms on average would be the same. As, each algorithm would be having its own forte of problem that attempts to develop a best test. In large test sets to obtain best results, many researchers follow the process of comparing different algorithms whenever there is the involvement of optimization technique. This helps in identifying and categorizing the problem for which the algorithm suites best. The following are the different categories of different kinds of benchmark functions:

1. Unimodal, convex, multidimensional functions
2. Multimodal and two dimensional functions
3. Separable and non-separable functions

In this paper, we have used four different benchmark functions to evaluate the performance of the algorithm. The test functions used in this paper are explained below with their mathematical formula.

5.1. Ackley Function

Ackley is a multimodal non-separable function. It is widely used for validation and testing purposes of algorithms.

$$f(x) = -a.\exp\left(-0.02.\sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}\right) - \exp\left(\frac{\sum_{i=1}^{n} \cos(cx_i)}{n}\right) + a + \exp(1) \tag{6}$$

Where, a = 20, n is the dimension of the problem and the test area lies between $-32 \leq x_i \leq 32$, i=1,2,…n. Its global minimum f(x) = 0 is obtainable for x_i=0, i= 1,…,n.

5.2. Alpine Function

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

$$f(x) = \sum_{i=1}^{n} |x_i sin(x_i) + 0.1x_i| \quad (7)$$

Where, n ≥2 is the dimension and x = (x_1, x_2, …, x_d) is a n-dimensional row vector (i.e., a 1×n matrix). Test area is usually restricted to -10 ≤ x_i ≤ 10, i=1,…,n. Its global minimum F(x) = 0 is obtainable for x_i, i=1,…,n.

5.3. Rastrigin's Function

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

$$f(x) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10cos(2\pi x_i)\right) \quad (8)$$

Where, n is the dimension and x = (x_1, x_2, … , x_d) is a n-dimensional row vector (i.e., a 1×n matrix). Similarly, to make it non separable, an orthogonal matrix is also used for coordinate rotation. Rastrigin's function is a classical multimodal problem. It is difficult since the number of local optima grows exponentially with the increase of dimensionality. The test area is usually restricted to a hypercube$-5.12 \leq x_i \leq 5.12$, i=1,…,n. Its global minimum f(x)=0 is obtainable for x_i=0, i= 1,…,n.

5.4. Schwefel Function

Schwefel function is deceptive in that the global minimum is geometrically distant, over the parameter space, from the next best local minima Schaffer function is also naturally non separable. Schaffer is defined as follows:

Function definition:

$$f_7(x) = \sum_{i=1}^{n} - x_i \cdot \sin\left(\sqrt{|x_i|}\right) \qquad -500 \le x_i \le 500 \tag{9}$$

Table 5: Benchmark problems

Function Name	Search Space
Ackley	-32 to +32
Alpine	-10 to +10
Rastrigin	-5.12 to +5.12
Schwefel	-500 to +500
Sphere	-5.12 to +5.12

5.5. Sphere Function

It is a simple and a continuous benchmark problem and belongs to the unimodal, multi-dimensional benchmark problems. It is mathematically defined as:

$$f(x) = \sum_{i=1}^{n} x_i^2 \tag{10}$$

Where n is the dimension of the problem and the test area lies between $-5.12 \le x_i \le 5.12$, i=1,2,…n. Its global minimum f(x)=0 is obtainable for x_i=0, i= 1,…,n.

6. EVALUATION AND TEST RESULTS

6.1. Experimental Evaluation

As discussed in the previous sections about implementation of optimization of artificial immune network using clonal selection algorithm is evaluated and tested using some well know benchmark functions such as Ackley function, Alpine function, Rastrigin function, Schwefel function and sphere function in optimization suite. The algorithm has been evaluated by varying the number of iterations and dimensions using these each benchmark function. Also, the algorithm is evaluated in different configurations varying population size (network cells), number of clones to be generated at each cell, number of dimensions with lower bounds and upper bounds calculating the best fitness at each iteration. The fitness values obtained for each benchmark function using the artificial immune network algorithm. The minimum best fitness for each clone is plotted along the y-axis and the iterations are plotted along the x-axis. The efficiency of the algorithm is tested and evaluated by running the algorithm for several number of runs.

6.2. Results

The algorithm has been tested by passing some constant parameters as input in the optimization suite along with some default parameters as input for each benchmark function:

- Suppression threshold for network cell affinities = 0.01
- Clonal selection average error threshold = 0.00000000000001
- Proportion of current population size to be added for diversity at each iteration = 0.01
- Affinity proportionate mutation parameter = 1.0

Here, we have considered the test cases in which we intent to plot the minimum best fitness at each iteration for each individual benchmark function using the AIS algorithm integrated within the optimization suite.

6.2.1. Test Case 1

Experiments are performed by changing the benchmark function for each run with population size (initial cells) = 60, number of clones = 2, number dimension = 2, number of iterations = 100, lower and upper bounds on each dimension of the optimization problem (-3.0, 3.0) and (-4.0, 4.0).

The figure 19 shows the variation of the fitness values obtained versus the number of iterations for the Alpine function and showing the iterations varying between 0 and 100. The best fitness value obtained with Alpine in this graph is 0.014755684688552595.

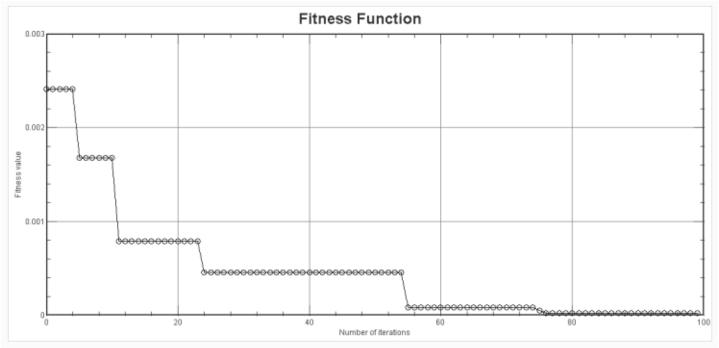


Figure 19: Shows the variation of best fitness with number of iterations for Alpine

The figure 20 shows the variation of the fitness values obtained versus the number of iterations for Ackley function and showing the iterations varying between 0 and 100. The best fitness value obtained with Ackley function in this graph is 8.306761852945772E-5.

The figure 21 shows the variation of the fitness values obtained versus the number of iterations for Rastrigin function and showing the iterations varying between 0 and 100. The best fitness value obtained with Rastrigin function in this graph is 0.00866944618321952.

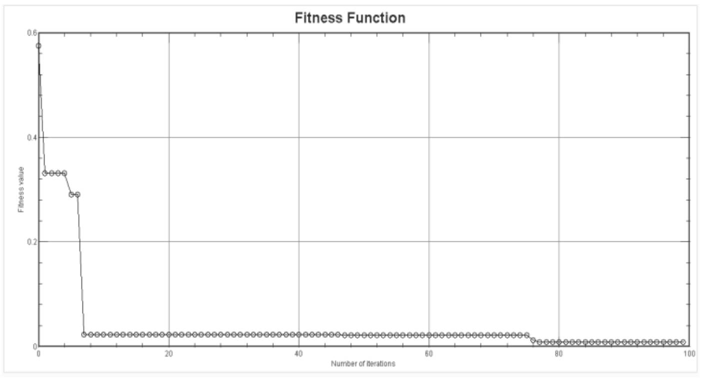


Figure 20: Shows the variation of best fitness with number of iterations for Ackley

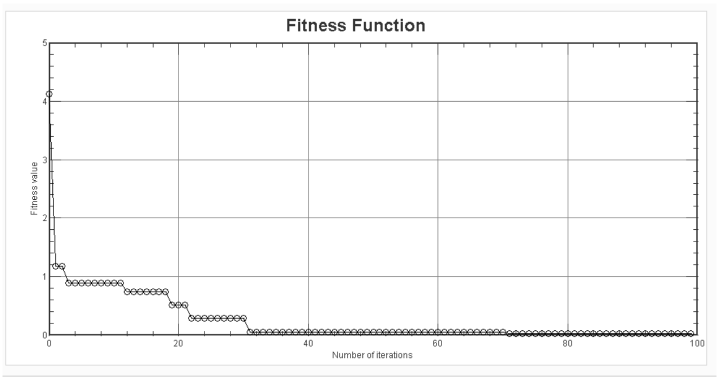


Figure 21: Shows the variation of best fitness with number of iterations for Rastrigin

The figure 22 shows the variation of the fitness values obtained versus the number of iterations for Schwefel function and showing the iterations varying between 0 and 100. The best fitness value obtained with Schwefel function in this graph is 2.9940657407023785E-5.

The figure 23 shows the variation of the fitness values obtained versus the number of iterations for Sphere function and showing the iterations varying between 0 and 100. The best fitness value obtained with Sphere function in this graph is 2.95175843059767E-6. The plots in test case 1 clearly shows that the best values decrease, which implies the performance of the algorithm increases with the increase in the number of iterations.

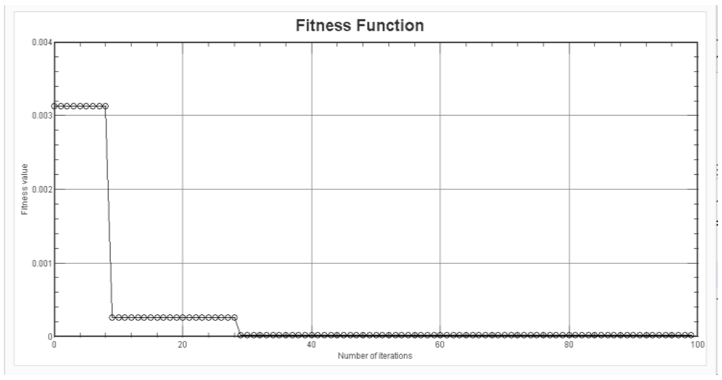


Figure 22: Shows the variation of best fitness with number of iterations for Schwefel

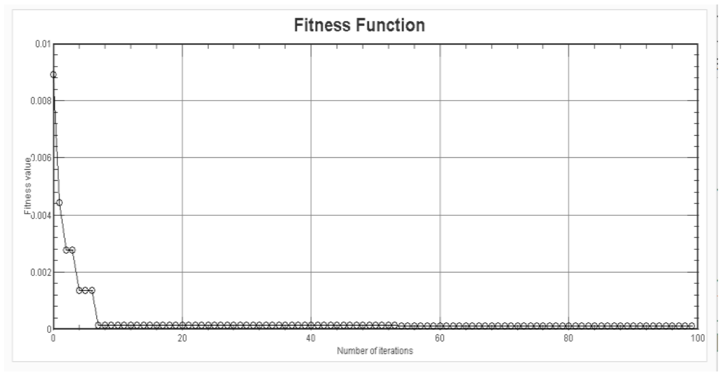


Figure 23: Shows the Variation of best fitness with number of iterations for Sphere

6.2.2. Test Case 2

Experiments are performed by changing the benchmark function for each run with population size (initial cells) = 80, number of clones = 2, number dimension = 2, number of iterations = 100, lower and upper bounds on each dimension of the optimization problem (-3.0, 3.0) and (-4.0, 4.0).

The figure 24 shows the variation of the fitness values obtained versus the number of iterations for Alpine function and showing the iterations varying between 0 and 100. The best fitness value obtained with alpine function in this graph is 1.4334052051569071E-5.

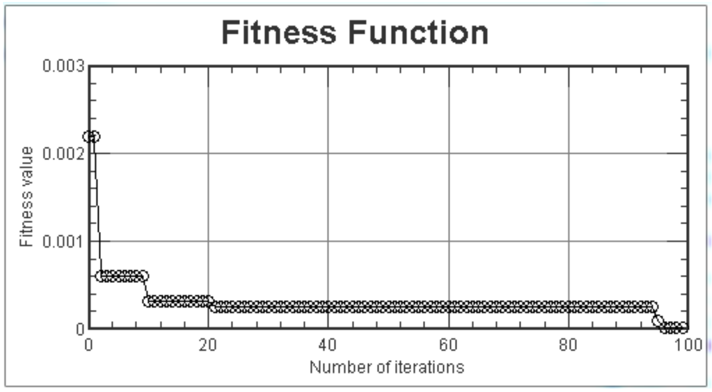


Figure 24: Shows the variation of best fitness with number of iterations for Alpine

The figure 25 shows the variation of the fitness values obtained versus the number of iterations for Ackley function and showing the iterations varying between 0 and 100. The best fitness value obtained with Ackley function in this graph is 0.0033833460755521294.

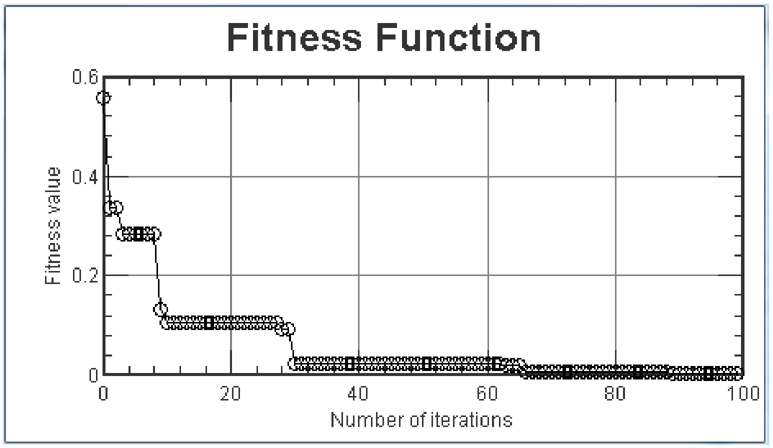


Figure 25: Shows the variation of best fitness with number of iterations for Ackley

The figure 26 shows the variation of the fitness values obtained versus the number of iterations for Rastrigin function and showing the iterations varying between 0 and 100. The best fitness value obtained with Rastrigin function in this graph is 0.04343244733718521.

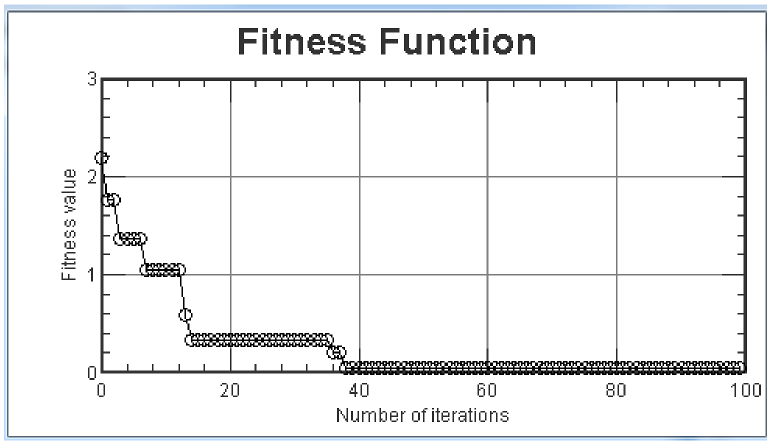


Figure 26: Shows the variation of best fitness with number of iterations for Rastrigin

The figure 27 shows the variation of the fitness values obtained versus the number of iterations for Schwefel function and showing the iterations varying between 0 and 100. The best fitness value obtained with Schwefel function in this graph is7.430789700265672E-4.

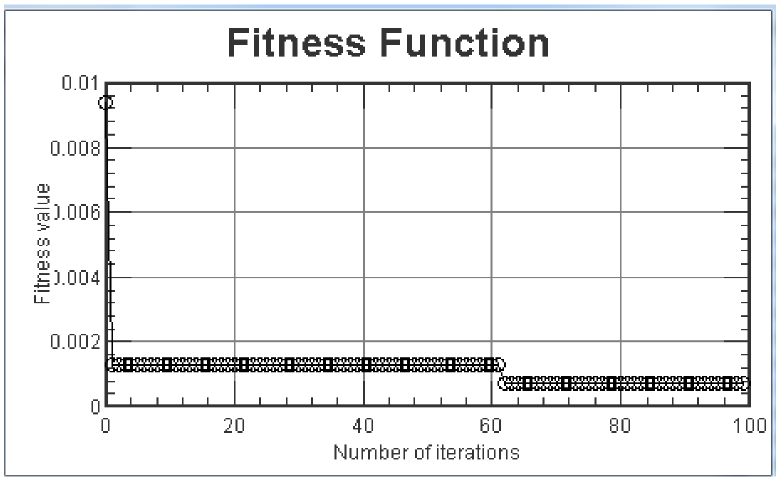


Figure 27: Shows the variation of best fitness with number of iterations for Schwefel

The figure 28 shows the variation of the fitness values obtained versus the number of iterations for Sphere function and showing the iterations varying between 0 and 100. The best fitness value obtained with Sphere function in this graph is 3.694299528858042E-5.

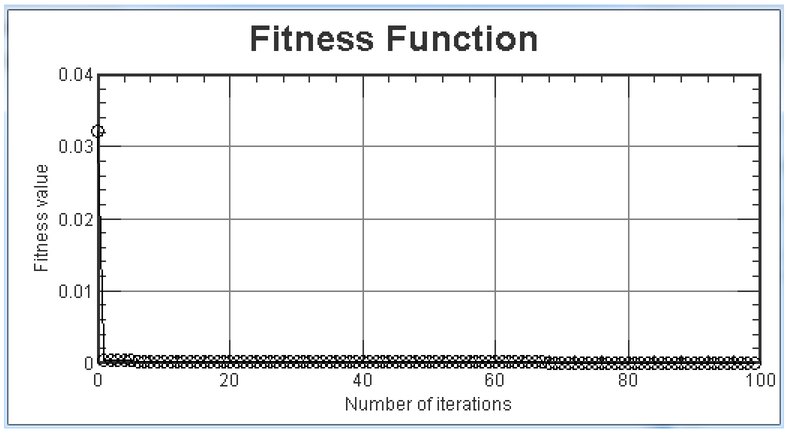


Figure 28: Shows the variation of best fitness with number of iterations for Sphere

In Test case 2, there is an improvement in performance of the algorithm with the increase in number of population size from 60 to 80. The plots obtained for in this test case for all the different functions clearly shows that the best values are decreasing, resembling the improvement in the performance of the algorithm.

6.2.3. Test Case 3

Experiments are performed by changing the benchmark function for each run with population size (initial cells) = 60, number of clones =5, number dimension = 2, number of iterations = 100, lower and upper bounds on each dimension of the optimization problem (-3.0, 3.0) and (-4.0, 4.0).

The figure 29 shows the variation of the fitness values obtained versus the number of iterations for Alpine function and showing the iterations varying between 0 and 100. The best fitness value obtained with alpine function in this graph is 6.153802332628348E-6.

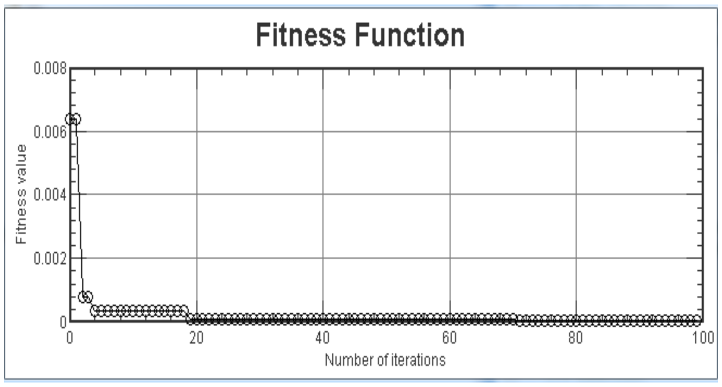


Figure 29: Shows the variation of best fitness with number of iterations for Alpine

The figure 30 shows the variation of the fitness values obtained versus the number of iterations for Ackley function and showing the iterations varying between 0 and 100. The best fitness value obtained with Ackley function in this graph is 0.001013476507683908.

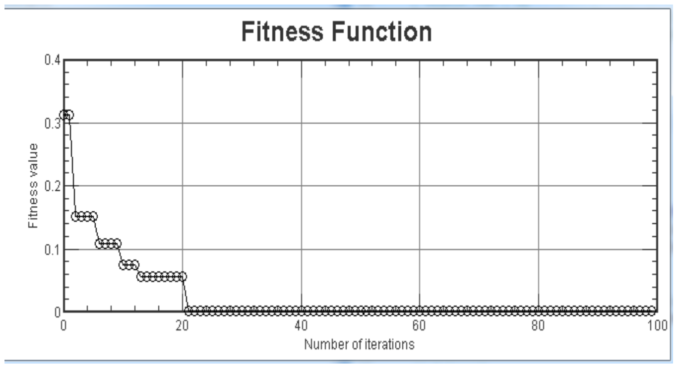


Figure 30: Shows the variation of best fitness with number of iterations for Ackley

The figure 31 shows the variation of the fitness values obtained versus the number of iterations for Rastrigin function and showing the iterations varying between 0 and 100. The best fitness value obtained with Rastrigin function in this graph is 0.002035692041573611З.

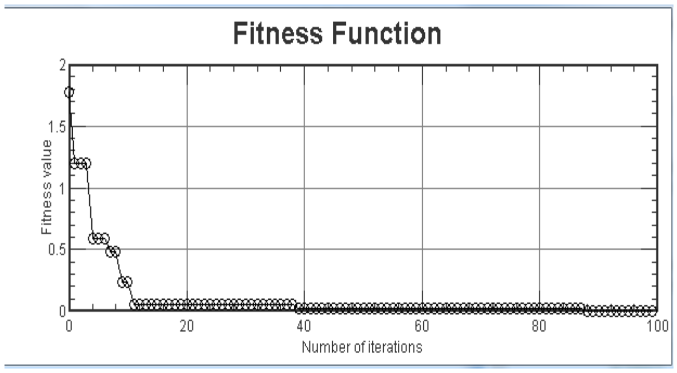


Figure 31: Shows the variation of best fitness with number of iterations for Rastrigin

The figure 32 shows the variation of the fitness values obtained versus the number of iterations for Schwefel function and showing the iterations varying between 0 and 100. The best fitness value obtained with Schwefel function in this graph is 2.99983257057157E-5.

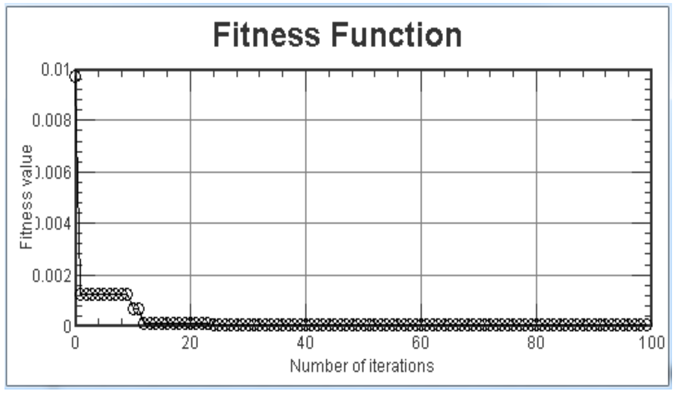


Figure 32: Shows the variation of best fitness with number of iterations for Schwefel

The figure 33 shows the variation of the fitness values obtained versus the number of iterations for Sphere function and showing the iterations varying between 0 and 100. The best fitness value obtained with Sphere function in this graph is 6.36161361768458E-5.

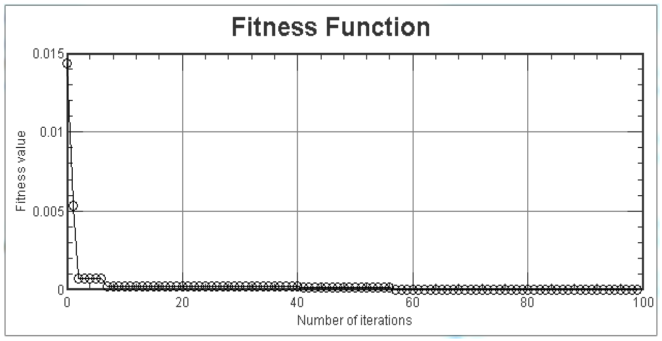


Figure 33: Shows the variation of best fitness with number of iterations for Sphere

With the increase in number of clones to 5 for population size 60 the plots in the above graphs for test case 1 clearly shows that the best values decreases very fast compared to test case 1, the performance of the algorithm increases with the increase in the number of iterations.

6.2.4. Test Case 4

Experiments are performed by changing the benchmark function for each run with population size (initial cells) = 80, number of clones =5, number dimension = 2, number of iterations = 100, lower and upper bounds on each dimension of the optimization problem (-3.0, 3.0) and (-4.0, 4.0).

The figure 34 shows the variation of the fitness values obtained versus the number of iterations for Alpine function and showing the iterations varying between 0 and 100. The best fitness value obtained with Alpine function in this graph is 1.3700604787677574E-5.

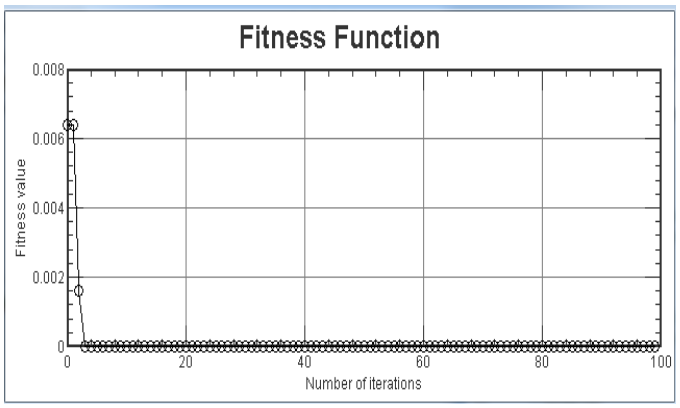


Figure 34: Shows the variation of best fitness with number of iterations for Alpine

The figure 35 shows the variation of the fitness values obtained versus the number of iterations for Ackley function and showing the iterations varying between 0 and 100. The best fitness value obtained with Ackley function in this graph is 0.002517348264799768.

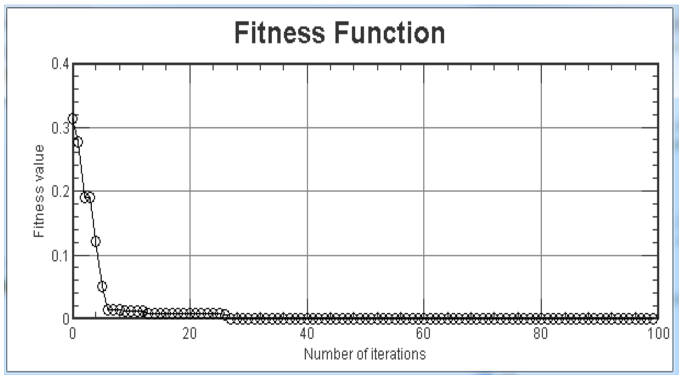


Figure 35: Shows the variation of best fitness with number of iterations for Ackley

The figure 36 shows the variation of the fitness values obtained versus the number of iterations for Rastrigin function and showing the iterations varying between 0 and 100.The best fitness value obtained with Rastrigin function in this graph is 0.0014972326888056386.

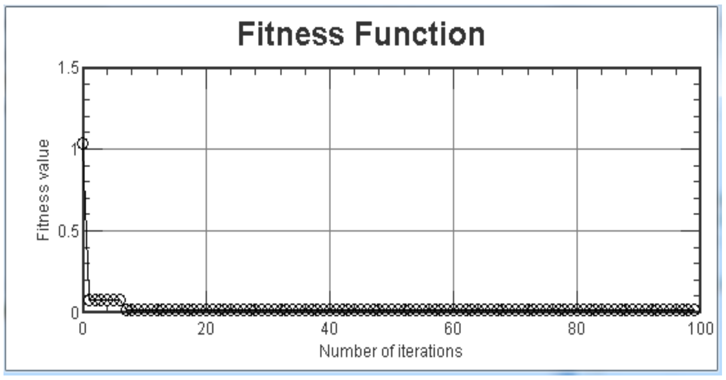


Figure 36: Shows the variation of best fitness with number of iterations for Rastrigin

The figure 37 shows the variation of the fitness values obtained versus the number of iteration for Schwefel function and showing the iterations varying between 0 and 100.The best fitness value obtained with Schwefel function in this graph is 5.2972738104062334E-5.

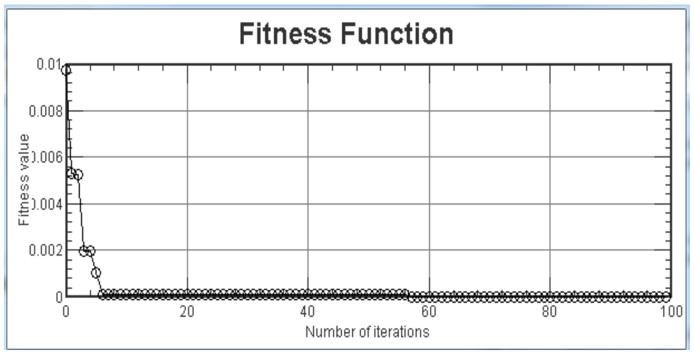


Figure 37: Shows the variation of best fitness with number of iterations for Schwefel

With increase in number of clones to 5 for population size 80 the plots in the above graphs for test case 2 clearly shows that the best values decreases very fast compared to test case 1, the performance of the algorithm increases with the increase in the number of iterations. The increase is profoundly seen in a few functions where as it is non-significant for few functions.

All the above test cases shows that the algorithm functions as expected.

7. CONCLUSION AND FUTURE WORK

This paper presents the implementation of artificial immune network using clonal selection algorithm and the integration within an optimization suite called optimal lab system. The optimization suite provides different categories of algorithm like Genetic, Stochastic, and Deterministic. We implemented the artificial immune network using clonal selection algorithm that has shown to perform well on different optimization tasks. Based on the results obtained we can conclude that the optimization of artificial immune network using clonal selection algorithm works effectively with regards to standard benchmark functions tested.

For future work, the performance of the algorithm might be improved by implementing a dynamic search for the optimum population size in the network. Furthermore, future work could also expand and implement additional optimization algorithms within the optimization suite. It would also be interesting to apply the AIS algorithm to solve various real world optimization problems.

8. REFERENCES

1. Angeline, Peter J. "Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences." *Evolutionary Programming VII*. Springer Berlin Heidelberg, 1998.
2. Kennedy, James. "Particle swarm optimization." *Encyclopedia of Machine Learning*. Springer US, 2010. 760-766.
3. Hofmeyr, Steven A., and Stephanie Forrest. "Architecture for an artificial immune system." *Evolutionary computation* 8.4 (2000): 443-473.
4. Yang, Xin-She. *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.
5. http://www.artificial-immune-systems.org/
6. Dasgupta, Dipankar. "Advances in artificial immune systems." *Computational Intelligence Magazine, IEEE* 1.4 (2006): 40-49.
7. http://en.wikipedia.org/wiki/Immune_network_theory/
8. http://www.rcsed.ac.uk/RCSEDBackIssues/journal/vol46_1/4610003.htm#MIGRATION
9. De Castro, Leandro N., and Fernando J. Von Zuben. "Learning and optimization using the clonal selection principle." *Evolutionary Computation, IEEE Transactions on* 6.3 (2002): 239-251.
10. http://www.biology.arizona.edu/immunology/tutorials/immunology/09t.html
11. De Castro, Leandro Nunes, and Jonathan Timmis. *Artificial immune systems: a new computational intelligence approach*. Springer, 2002.
12. De Castro, Leandro Nunes, and Fernando José Von Zuben. "Artificial immune systems: Part I–basic theory and applications." *Universidade Estadual de Campinas, Dezembro de, Tech. Rep* 210 (1999).

13. De Castro, L. Nunes, and Fernando J. Von Zuben. "The clonal selection algorithm with engineering applications." *Proceedings of GECCO*. Vol. 2000. 2000.

14. Burnet, Macfarlane. "The clonal selection theory of acquired immunity." *The Glonal Selection Theory of Acquired Immunity.* (1959).

15. Satyanarayana Daggubati. COMPARISON OF PARTICLE SWARM OPTIMIZATION VARIANTS. Diss. North Dakota State University, 2012

16. de Castro, L. Nunes, and Jon Timmis. "An artificial immune network for multimodal function optimization." *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*. Vol. 1. IEEE, 2002.

17. Farmer, J. Doyne, Norman H. Packard, and Alan S. Perelson. "The immune system, adaptation, and machine learning." *Physica D: Nonlinear Phenomena*22.1 (1986): 187-204.

18. Timmis, Jon. *Artificial immune systems: a novel data analysis technique inspired by the immune network theory*. Diss. Department of Computer Science, 2000.

19. de Castro, Leandro Nunes, and Fernando J. Von Zuben. "aiNet: An artificial immune network for data analysis." *Data mining: a heuristic approach* 1 (2001): 231-259.

20. Wolpert, David H., and William G. Macready. *No free lunch theorems for search*. Vol. 10. Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.