A Novel Clonal Selection Algorithm Approach with Modified Replacement: Cell Rescue

Aleksandra Talar

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Bachelor of Science
of the
University of Aberdeen.

Department of Computing Science

2019

Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed:

Date: 2019

Abstract

Nature has already served as an inspiration for computer scientists many times being a foundation for genetic algorithms, neural networks, tree data structures and etc. It has also been a thing which allowed to create the first version of Clonal Selection Algorithm (CLONALG): algorithm which mimics the behavior of human immune system. In particularly, the algorithm follows the Clonal Selection Theory which explains how the organism reacts when it is exposed to an infection. Thanks to the recognition of non-self, cloning and mutation of immune cells (called B cells) an organism is capable of protecting against the viruses. The following research looks closely at the nature again to find inspiration for improving the existing Clonal Selection Algorithm and, as a result, a new version of CLONALG - Cell Rescue - is proposed which performs reshuffling of low-affinity antibodies instead of disregarding them (as done in the standard version). 4 versions of reshuffling are implemented and tested against standard CLONALG in pattern recognition and function optimisation tasks using real-world data. Conducted experiments and results analysis have allowed to state that 3 methods (out of 4) increase the efficiency of the algorithm for pattern recognition and that the effectiveness is increased for selected functions only but decreased or the same for others.

Acknowledgements

I would like to thank Professor George M. Coghill for his support, guidance and supervision during the development process of this project.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

Machine learning is a fast growing area of computing science, more precisely, a subset of artificial intelligence. It is a field where computers can learn from data without being programmed again and make appropriate predictions. This research concentrates on Artificial Immune Systems (AIS) - subfield of Biologically Inspired Computing. Computers have successfully been used in immunology to help solve problems and maintain good health. But the experience has shown that the opposite is also possible: immunology can be used to solve problems in computing science. Biology has already helped scientists to gain inspiration for constructing and designing incredible building, creating planes and many more. In computing the nature has helped create many concepts as well. They are artificial neural networks [13], swarm systems [17], genetic algorithms and programming [18] to name a few.

Intuitive and natural metaphor suggests that usage of mechanism of the Immune System can improve safety of systems (increase robustness). It appears that the application of AIS is broader. Many immune algorithms exist using immunology concepts like Negative Selection, Clonal Selection, Immune Networks, Dendritic cell allowing for anomaly detection, data filtering, classification, pattern recognition and optimisation [2]. Main interest of this research is in Clonal Selection Algorithm (CLONALG). So far it has found applications in sciences like engineering performing binary characters recognition task (admitting ability to learn and acquire memory as described in [5]). In the same paper it was also shown that CLONALG is also suitable for multi-modal and combinatorial optimization. This research aims to improve Clonal Selection Algorithm by finding appropriate inspiration back in the nature and reformulating it to computing needs. More precisely the concept follows from the receptor editing process during the immune response described by Andrew J.T. George and David Gray [9].

This report presents a comparative case study of two AIS' algorithms: CLONALG and Cell Rescue version. The main goal of this research is to answer the question: *If the Cell Rescue version is better than the standard CLONALG?* To address this issue and establish the relationship between treatment of low affinity antibodies and effectiveness

of Clonal Selection Algorithm I am going to implement two versions of it for two applications: patter recognition and function optimisation. This will provide a wide picture for the problem. More precisely, the null hypothesis for the problem is that there is no difference between CLONALG and Cell Rescue version. Others say that the Cell Rescue version will classify successfully more patterns than the original CLONALG using different methods of reshuffling and Cell Rescue version will find the minimum quicker than the original CLONALG using one method of reshuffling. Proper experiment is designed to test above hypotheses (in order to compare both algorithms).

This is an interesting correlation to investigate as the results may be unexpected. Both negative and positive results will be very informative. Such an evidence of the impact of performance could serve for optimisation of the algorithm and, hence, significantly increase its effectiveness. Furthermore, positive results would mean that the nature-inspired concepts could be successfully translated to computing and would open a wide source of new possibilities.

The paper is structured in following order: Chapter 2 describes the theoretical aspects of the problem including biological context. Proposed solutions follow from concepts presented in the background section and are presented and analysed in chapter 3. Chapter 4 outlines the details of experiments conducted to test stated hypotheses. All results are presented in chapter 5 and discussed in chapter 6. Suggestions for future improvements and personal reflection on the research are provided in chapter 7.

Chapter 2

Background

The presented research follows completely from the theoretical form of human immune systems (especially clonal selection theory), thus, it is fundamental to review and understand the biology aspects of the problem. The following chapter looks closely at both human and artificial immune systems, describes all corresponding processes (of these systems) and their application in computing together with a sketch of new concept used for improvement and the review of current researches done so far in the field.

2.1 Immune Systems

The immune system is a host defense system which aims to protect the body against diseases and changes by identification and elimination of pathogens and cancers. The immune system collaborates with other systems (neural system and endocrine system) to maintain the well-being of the organism and allow all processes to be carried out within the host. The main tasks of the immune system are: *differentiation* between self and non-self and *elimination* the non-self i.e. the pathogens which are viruses, bacteria etc.

The immune system is made of white blood cells, called lymphocytes, which are divided into two categories: B cells and T cells. Every such cell has a "marker" (antigen) on its surface to allow for differentiation between self and non-self. For example, when one has the blood type A then the blood cells have the antigen A on their surface and will be recognised as selves. When cells with antigen B will be encountered (incorrect blood transfusion), antibodies (type of protein produced by cells) will recognise them as foreign (non-self) and these cells will be destroyed. The task of detection of non-self is complicated as pathogens have evolved and managed to adapt to different conditions by changing their behaviour. However, through the evolution process, the defensive mechanisms have changed as well to effectively counteract pathogens.

When studying the structure of the human immune system, scientists realized that it is an attractive source of inspiration and resembles a well-designed computer system. Within the book "In Silico Immunology", Jon Timmis and Darren Flower highlighted

features of immune systems which served as a motivation for exploring the artificial immunity and are still up to data guidelines on what to look for [19]. First thing to notice is that immune systems do not have a central controller: immune agents are not told what to do in different situations, their behaviour is a result of local interactions. It is computationally attractive to have an algorithm capable of self-organisation. It can be also captured that the mechanism of producing B cells and the whole adaptive system is able to learn itself based on the repeated exposure to an infection (details will be provided later on). Additionally, immune system perform something what computer scientists would call "classification" - it differentiates between self and non-self - this follows from the definition of immune system quite intuitively.

2.2 Artificial Immune Systems

All the above observations were applied to create an Artificial Immune System which mimics the behaviour of the above functions and features of a natural immune system to solve a given (computational) problem [3]. The simple framework of the AIS is shown in the Figure 2.1. When considering a particular problem (within the specific domain) one need to choose suitable representation of data (like binary string in pattern recognition or set of integers in function optimisation). The affinity calculation method partially follows from the representation choice and the domain and must be chosen with care as it can significantly influence the performance of the algorithm and thus influence the results [19]. Next, the choice of an immune algorithm determines the behaviour of the whole system and also follows from the domain specificity. Each of the immune algorithm is inspired from the immune theories like clonal selection, negative selection, etc. Execution of every layer leads to the solution of the given problem.

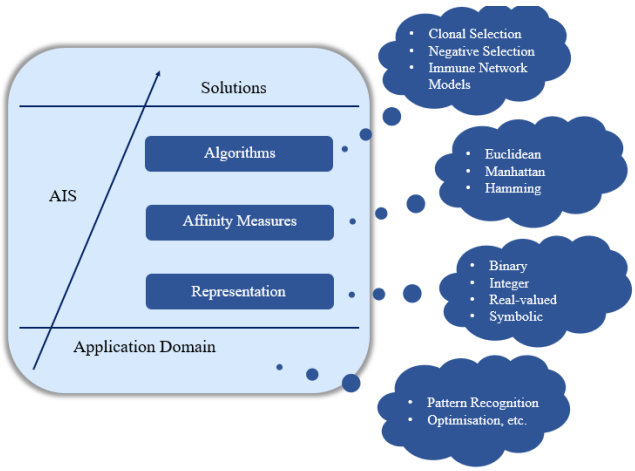


Figure 2.1: The AIS Framework

2.3 Clonal Selection Theory

Clonal selection theory describes the behaviour of the adaptive immune system when exposure to a pathogen (intruder: virus, bacteria, etc.), in other words, this theory explains how the immune system responds to an infection. Described in 1957 by Frank Macfarlane Burnet and his colleagues was a milestone in a development of immunology (especially molecular immunology in adaptive immunity) [14].

To understand the concept of immunology response one should look at the following theories. Assume that every lymphocyte has a unique receptor which can only recognise a specific antigen. When a pathogen would invade a body, only a minor number of lymphocytes would be able to recognise the intruder. Before the few lymphocytes would produce sufficient antibodies to fight the pathogen, the organism would die. In the other hand, if all the lymphocytes were active, the organism would not be able to perform fundamental processes and would die as well not able to control billions of activated cells (lymphocytes). Here is where the sense of clonal selection theory can be seen.

It explains that only the lymphocytes with the ability to recognise a particular antigen are activated and selected to proliferate. In other words, when the B cells' antibodies encounter (and bind with) a "foreign" antigen, they start producing new B cell clones. The binding is possible due to the affinity of antibodies, where affinity (according to Oxford Dictionary) is *"a degree to which a substance tends to combine with another"*. So the bigger the affinity of an antibody, the more successful is the binding and hence recognition of the antigen. Later in the process, antibody genes in B lymphocytes undergo many rounds of mutations and subsequent selections to best bind the specific antigen [1] - this process is called affinity maturation. It is important to notice that after the proliferation phase there will be a large number of clones capable of recognising a particular antigen [3]. After fighting the intruder (pathogen) the cloned cells will be absorb to plasma cells (and later destroyed by the organism) or turned into long-lived B "memory cells".

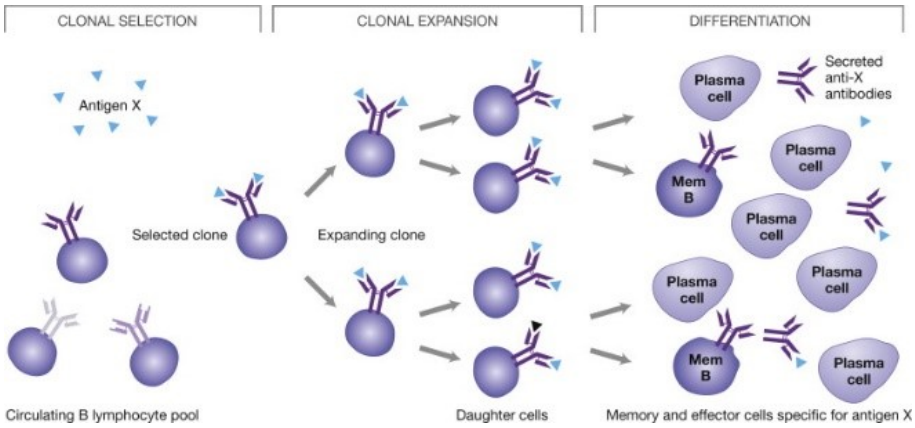


Figure 2.2: The Clonal Selection Mechanism (picture imported from [12])

First exposure to the intruder's antigen is handled by the small number of B cells with antibodies of different affinities. After that, memory cells are stored. Thanks to the bigger number of initial B cells than before the response (elimination of the pathogen) and the higher affinity of antibodies, the next response is quicker and more effective [19]. Memory cells are related to the well-known concept of "immunity". In the usual evolution of the immunology system, an organism is expected to repeatedly encounter an infection (exposure to the antigen) during its lifetime. Thanks to that immune reaction is stronger after each infection. Consequently, after many infections, an organism thoroughly knows how to fight a particular infection - what to do when a particular antigen ("marker" on the antibody of that infection) is encountered. With every infection the organism do not start from the beginning - it *has learnt* how to tackle the problem.

The clonal selection theory also states that a lymphocyte with ability to bind organism's own molecules must be deleted during its development process. It ensures that only a pathogen can activate the clones expansion and that the destructive immune response will not occur (autoimmunity) [19]. Summarizing, the clonal selection mechanism is as follows:

I. CLONAL SELECTION

   (a) B cells antigen receptors (antibodies) detect the non-self antigen

II. CLONAL EXPANSION

   (b) B cells are activated and begin to proliferate

   (c) New B cells (clones of the parent) mutate

III. CLONAL DIFFERENTIATION

   (d) All B cells (together with clones) turn into either plasma cells or memory cells

   (e) Plasma cells are destroyed

   (f) Memory cells are high affinity cells and are stored to initially produce high affinity clones during the next exposure to an infection

2.4 CLONALG

Figure 2.3 presents the flowchart of modelling paradigm: it illustrates the creation process of the Clonal Selection Algorithm - from nature/immune system observations through model to clonal selection algorithm. The following observations from the computational viewpoint can be done from the Clonal Selection Theory:

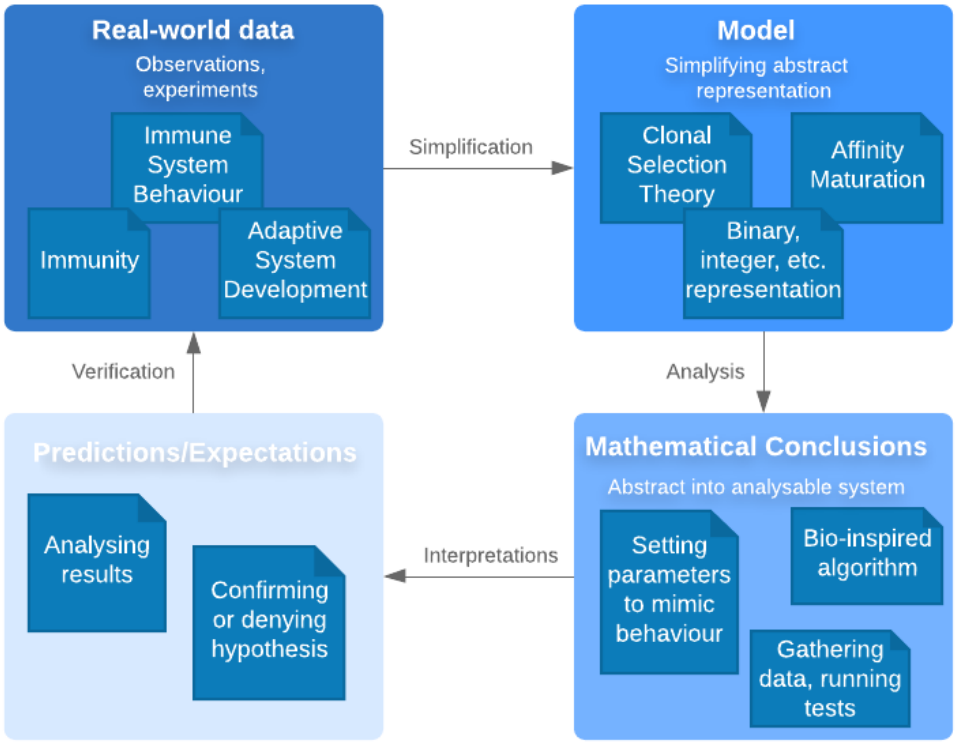


Figure 2.3: Model Paradigm

- The immune system does not learn with the use of single antibody (agent) but with collection of antibodies (multiple agents).
- The system is able to change itself in order to adapt to the environment
- The number of clones, which will be produced, is proportional to the affinity of the antibody that binds it [3]
- Mutations are inversely proportional to the affinity of the antigen it binds [3]
- Immunology System can serve as a classifier (it distinguishes between self and non-self antibodies)

All of the above features served de Castro and Von Zuben [4] as an inspiration to create an AIS capable of performing pattern recognition, function optimization, and combinatorial optimization tasks - Clonal Selection Algorithm (CLONALG) presented in the Figure 2.4. For simplification of the computing part, the affinity can be understood as a "degree of matching". Details of the algorithm in Chapter 3.

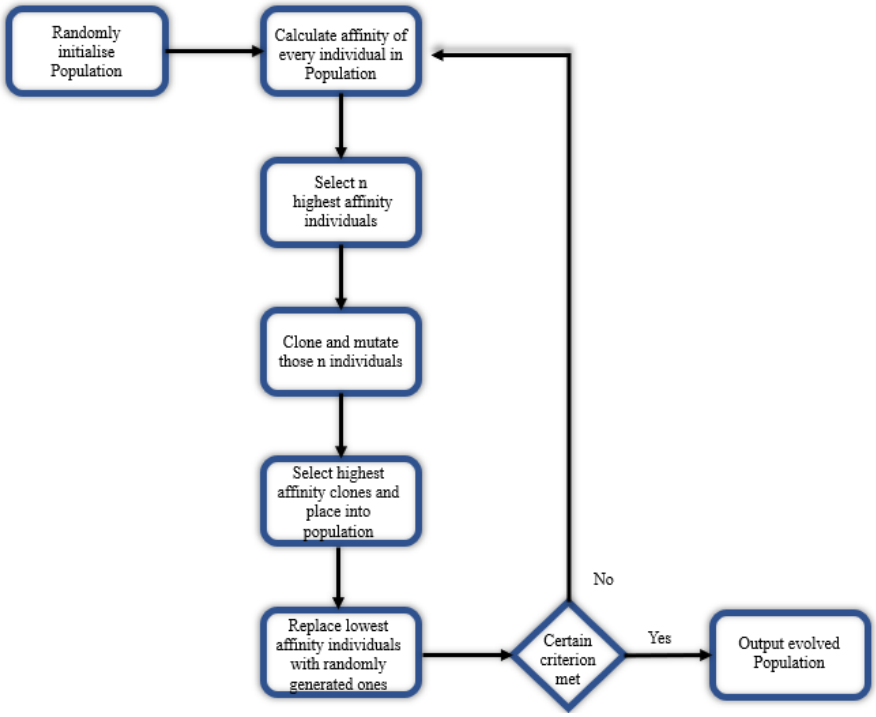


Figure 2.4: Simplified CLONALG FLOWCHART

2.5 Receptor Editing for "Cell Rescue"

The theory of clonal selection (process of affinity maturation during the immunology response) was deeply studied since it was first introduced [3]. Careful investigation of the development of the biology field allows one to improve the current version of the algorithm. The following paragraphs describe ideas expressed in the immunology field and examine how they can be "translated" to the computing science needs.

In 1999, Andrew J.T. George and David Gray introduced a new idea to improve the affinity maturation of antibodies to the biology community [9]. They argued that the standard point mutation (used in a process of affinity maturation) does only allow to explore local regions of the affinity landscape (see Figure 2.5, imported from [9]). In the case of algorithmic thinking, it means that the individual (antibody) after point mutation is only "slightly" changed with each iteration and will not necessarily achieve the desired (highest) affinity measure. Figure 2.5 presents a simplified affinity landscape and it can be seen that the antibody "explores" the landscape in such a way that the point mutations are performed (red lines) and whenever the affinity is higher the antibody climbs the hill. As a result the antibody reaches the local maximum and cannot go any further as all the lower affinities are lost. The issue arises when a local maximum (which was reached by the antibody) is very low relatively to the global maximum. To optimise the process of affinity maturation, receptor editing was proposed as an ability which allows the antibody to escape from the local maximum (green) and, hence, (possibly) reach the higher affinity. The authors emphasised the fact that both point mutation together with receptor editing

can improve affinity maturation of antibodies.

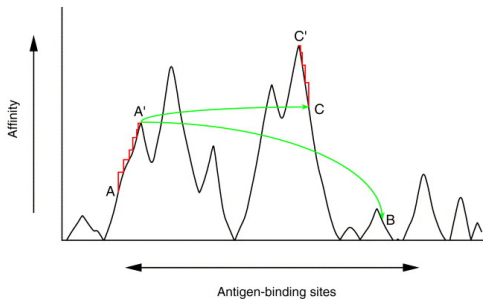


Figure 2.5: Affinity Landscape

In his answer to this article, Matthew Cook questioned the receptor editing idea saying that it may be unnecessary in affinity maturation process [7]. He used other existing models (like NK model of Kauffman) to prove his claims and argued that even without performing receptor editing an antibody is able to attain many local optima with only the use of point mutation (in 831 trials 148 optima are reached). George and Gray explained that the care must be taken when applying the model, as the model's parameters must be optimise for a specific scheme [10]. They also pointed out the evidence (*in vitro* affinity maturation with phage-display libraries) showing that a process like receptor editing is used in order to increase affinity. The method is called "chain shuffling" and is similar to the receptor editing. Both strategies mutation and chain shuffling were said to be complementary in the process of affinity maturation.

The CLONALG mechanism is based on the affinity maturation process so above study can be used to further optimaise the algorithm. The mentioned "receptor editing" is a process that changes the structure of an antigen's receptor in order to rescue the cell from cell death [15] (as described earlier, according to clonal selection theory some of the B cell clones change into memory set, the rest is removed). A simplified picture of the antibody is shown in the Figure 2.6. Each antibody consist of four "polypeptides"- two heavy chains and two light chains. The process of receptor editing refers to "reshuffling" these polypetides. From here the idea of CLONALG optimisation follows: instead of replacing the lowest affinity individuals with randomly generated ones, do the reshuffling of each cell (for example, for a binary string - reshuffle the bits). Figure 2.7 presents the situation: when affinity is high, more clones is produced, when affinity is low more mutation is performed, but also antibodies below certain threshold would be normally disregarded and replaced by new. These are exactly the cells to which the new concept of cell rescue can be applied. This strategy allows to rescue the cells from cell death and suggests the area of investigation for CLONALG improvements.

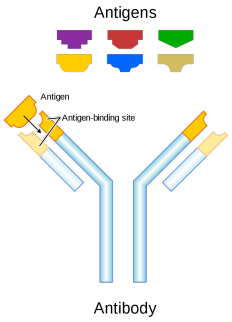


Figure 2.6: Antibody

Certainly, when applying the biological idea to the natural computing, the caution

must be taken. Jon Timmis and Darren Flower point out the importance of careful investigation as not every desirable feature of immune systems guarantees the success of the algorithm and does not mean that the computing problem can be solved with it [19].

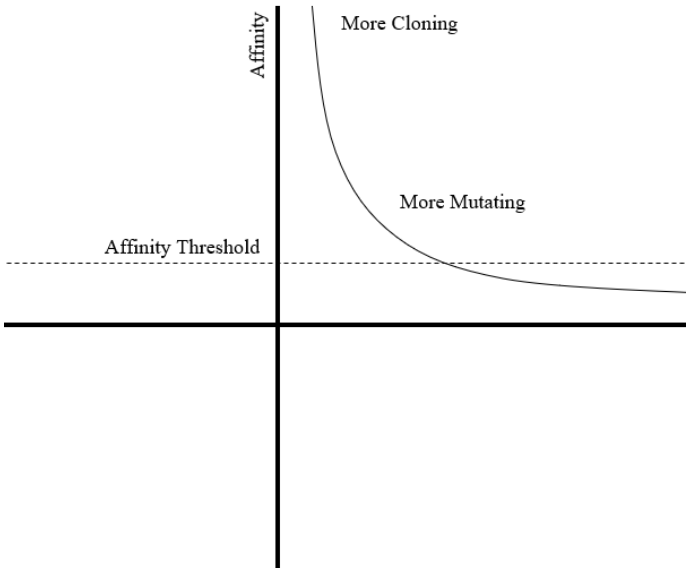


Figure 2.7: Affinity of B-cells

2.6 Related Work

Many scientists have suggested changes to the Clonal Selection Algorithm in order to increase its efficiency. However, every research focuses on specific part of the algorithm. For example, authors of "Improved Clonal Selection Algorithm (ICLONALG)" emphasized the role of selection and reproduction [16]. They investigated the result of increasing the affinity value of selected cells and maximizing the optimized result. After selection, cloning and mutation a suggestion was made to clone the highest affinity individuals once again, mutate and select the best members. It was noticed that proposed algorithm reaches a diverse set of local optima solutions in comparison to standard CLONAG. Meanwhile, J. A. White, and S. M. Garrett focused on improving the CLONALG for pattern recognition [20]. They investigated and tested variations of hamming distance method to create the CLONALG for classification called CLONCLAS. They managed to show improvement in performance of existing algorithm. Another approach was showed in the "Novel Hybrid Clonal Selection Algorithm with Combinatorial Recombination and Modified Hypermutation Operators for Global Optimization" [21]. Authors argued that the standard Clonal Selection Algorithm (named CSA here) presents better performance with recombination inspired by the biological combinatorial recombination (RCSA) and recombination together with redefined hypermutation (RHCSA). The idea was successfully tested on a set of benchmark functions. Furthermore, authors of CLONALG, Castro and Zuben, mentioned the receptor editing in the biological context when first introducing the Clonal

Selection Algorithm [6]. In their design of algorithm, the antibody repertoire of size N is composed of m memory cells and r remaining cells where d of them are the lowest affinity antibodies which will be replaced by d new ones. However, in every implementation of the algorithm they set $d = 0$ and did not investigate the matter further.

Chapter 3

Analysis and Design of Algorithms

This chapter provides details of CLONALG algorithm and explains exactly the structure of a new one - Cell Rescue. Both algorithms are evaluated in case of time complexity and the justification for all design decisions is provided.

3.1 CLONALG

Algorithm 1 presents the pseudocode for the standard CLONALG. For the research and experiment purposes, the CLONALG is implemented for 2 applications: pattern recognition and function optimisation.

$Pattern_{Learn}$ is an element of set of antigens **Ag**, these are the patterns to be learned by the algorithm. They are encoded by binary strings (which may correspond to a simple bitmaps for a given pattern or lists of real numbers- details in chapter 4) or a vector of real numbers.

Population is the set of antibodies **Ab**. It consists of a set of binary strings or real vectors, where length of each string/vector is stored in an $Individual_{Size}$ variable. Population is of size N such that $N = m + r$, where m is $Memory_{Size}$ and $r = N - m$ is the size of the remaining Population repertoire.

$Clone_{Rate}$ is a multiplying factor.

$Mutation_{Rate}$ is the probability with which the mutation is performed.

$Degree_{Generation}$ is a stopping condition which informs how many generations of Population are desired.

Data: $Pattern_{Learn}$, $Population_{Size}$, $Individual_{Size}$, $Selection_{Size}$, $Mutation_{Rate}$, $Clone_{Rate}$, $Degree_{Generation}$, $Memory_{Size}$
Result: Population

```
Population = GenerateRandom(Population_Size, Individual_Size)
Count = 0
while Count < Degree_Generation do
    for Pattern_Learn ∈ Antigenic_Patterns do
        for p_i ∈ Population do
            AffinityCalculate(p_i)
        end
        Selected_Individuals = Select(Population, Selection_Size)
        Clones_Population = ∅
        for p_i ∈ Selected_Individuals do
            Clones_Population += Clone(p_i, Clone_Rate)
        end
        for p_i ∈ Clones_Population do
            Mutated_Clones += Mutate(p_i, Mutation_Rate)
        end
        Population = Select(Mutated_Clones, Memory_Size) + Random_Individuals
    end
    Count += 1
end
```

Algorithm 1: Pseudocode for CLONALG

3.1.1 Pattern Recognition

Pattern recognition version of CLONALG manipulates randomly generated individuals in order to achieve a closest picture of a given pattern. For this application the variables in the Algorithm 1 are as follows:

The set of binary strings is generated and stored as population (**Ab**). Then the affinity of each member is calculated with respect to the antigenic pattern $\mathbf{Ag_i}$. In the pattern recognition binary version, it is done by calculating the hamming distance, so the affinity is expressed as the number of 1's in the outcome of the exclusive OR of a pattern and an individual. For example, for the strings [1, 0, 1] and [0,1,1] the hamming distance is 2 ([1,0,1] XOR [0,1,1] = [1,1,0]).

Next, the n highest affinity antibodies are selected and cloned. The number of clones of each individual is proportional to the affinity, meaning that the higher the affinity, the bigger number of clones. Precisely, the number of clones N_c is expressed with the formula

[6]:

$$N_c = round(\frac{\beta N}{i}),$$

where, β is a certain multiplying factor, called also $Clonal_{rate}$, N is the size of the whole Population and i corresponds to the $i-th$ selected individual. Each clone is then mutated: $k-th$ entry is changed with a certain probability ($Mutation_{Rate}$). The $Mutation_{Rate}$ is defined by the formula [2]:

$$Mutation_{Rate} = \exp(-\rho f),$$

where ρ is a control parameter and f is the affinity of a particular individual. The mutation is inversely proportional to the affinity meaning that the higher the affinity the less likely the mutation. The affinity of each mutated clone is determined again with relation to the antigenic pattern $\mathbf{Ag_i}$. The best one is selected and becomes a candidate to enter the memory set. If the affinity of the corresponding memory cell is lower than the high affinity clone then it is replaced by the mutated clone. Low-affinity individuals (antibodies with the affinities below the $Affinity_{Threshold}$) of the Population are disregarded and replaced by new random individuals. One generation is complete when all steps are completed for every antigenic pattern $\mathbf{Ag_i}$. The whole procedure is repeated until the desired degree of generation is reached. The resulting Memory set is the solution to the problem.

JUSTIFICATION OF PARAMETERS SELECTION

The choice of the affinity measure function needs to take into the account representation of data and the problem domain. The hamming distance is widely used to transform a random string towards a correct result, especially with the binary representation and was used in other implementations of CLONALG [5], [16]. Hence, the choice of affinity function in binary pattern recognition follows. Additionally, it is important to remember that the $Mutation_{Rate}$ is the probability with which the cells are mutated so it must be in the interval $[0,1]$ Therefore,

$$0 \leq e^{(-\rho f)} \leq 1$$

$$\Rightarrow \rho = 0.03$$

is optimal for $f \in (0,228)$ preserving the inverse proportion as well.

Additionally, rest of the parameters were chosen similarly to the Castro and Zuben implementation for pattern recognition to mimic proposed conditions: $\beta = 10$, $Memory_{Size} = 42$, $N = 52$ [6]. Figure 3.1 illustrates an example of matured memory set

(after 1,5,10,20,50 generations) after executing CLONALG with given training patterns (antigens **Ag**) and initial memory set. Each image is represented as a binary string of length 120 bits.

CLASSIFICATION

The additional classification steps are added to both algorithms (CLONALG and Cell Rescue) to compare between obtained solution to a problem (i.e. the memory set) and the initial training set of patterns (**Ag** set) as well as to test the memory set with new unseen patterns:

1. Choose a cell from the memory set.

2. Compare it with each antigenic pattern.

3. Calculate the affinity accuracy: $\frac{affinity}{max} * 100$.

4. If percentage is above the threshold and it is greater than the current highest, make it the new highest.

5. If percentage is below the threshold, the classification cannot be made, i.e. the pattern class is unknown.

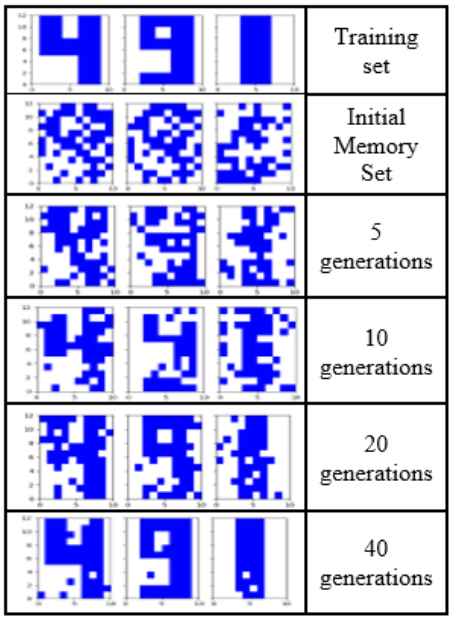
6. Add the result (class) to the set of classifications.

7. Repeat for every memory cell.

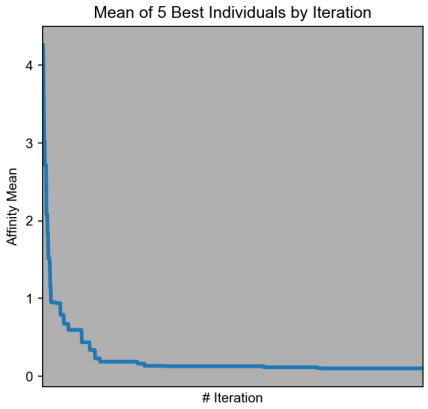
The accuracy threshold is set to 80%. Later those steps will be used as well to test the algorithm. In other words, every cell of the memory set will be presented with a new set of patterns to be classified.

3.1.2 Function Optimisation

Application of CLONALG in function optimisation looks for the minimum of a given function. The mechanism of an algorithm is as described above but a few changes must be done to apply it to optimisation task. Population should consist of N vectors of length k, where each entry of the vector is a real number x such that $x_i \in (a,b)$ for all $i = 1,...,k$ and some $a,b \in \mathbb{R}$. There is no particular antigenic pattern to recognise, but a function f to be optimised. Hence, the affinity is calculated by evaluating the value of the function at x i.e $f(x)$. As there are no patterns to "remember" the Population composes now the whole $Memory_{Set}$ so there is no need to keep the separate $Memory_{Set}$ of size m any longer. Therefore, the $Memory_{Size} = Population_{Size} = N$. Figure 3.1 presents the CLONALG output for continuous function optimization task that seeks min f where $f = \sum_{i=1}^{n} x_i^2$, $-5 \leq x_i \leq 5$ and $k = 3$.



(a) Memory Cells of CLONALG (pattern recognition)



(b) Memory Cells of CLONALG (function optimisation)

Figure 3.1: CLONALG Example

3.2 Cell Rescue

Algorithm 2 presents the pseudocode for Cell Rescue. The Cell Rescue algorithm is the same as CLONALG up to the last step. In CLONALG, the lowest affinity antibodies from the remaining repertoire are replaced by new random individuals. Instead of that, the Cell Rescue version reshuffles low-affinity antibodies with one of the following method:

Method 1: The individual is divided into 2 parts, then the parts are reshuffled.

Method 2: The individual is divided into 3 parts, then the parts are shuffled randomly.

Method 3: The individual is divided into 4 parts, then the parts are shuffled randomly.

Method 4: Every element of an individual is shuffled randomly.

Data: $Pattern_{Learn}$, $Population_{Size}$, $Individual_{Size}$, $Selection_{Size}$, $Mutation_{Rate}$, $Clone_{Rate}$, $Degree_{Generation}$, $Memory_{Size}$, $Affinity_{Threshold}$
Result: Population

```
Population = GenerateRandom(Population_Size, Individual_Size)
Count = 0
while Count < Degree_Generation do
    for Pattern_Learn ∈ Antigenic_Patterns do
        for p_i ∈ Population do
            AffinityCalculate(p_i)
        end
        Selected_Individuals = Select(Population, Selection_Size)
        Clones_Population = ∅
        for p_i ∈ Selected_Individuals do
            Clones_Population += Clone(p_i, Clone_Rate)
        end
        for p_i ∈ Clones_Population do
            Mutated_Clones += Mutate(p_i, Mutation_Rate)
        end
        Population = Select(Mutated_Clones, Memory_Size) +
          reshuffle(RemainingRep,Affinity_Threshold)
    end
    Count+ = 1
end
```

Algorithm 2: Pseudocode for Cell Rescue

Precisely, the low-affinity antibody is every antibody from the remaining repertoire whose affinity is below certain $Affinity_{Threshold}$ (see chapter 4 for details). Antibodies above the threshold are left unchanged. Each method is applicable for pattern recognition, while only method 4 is used in the function optimisation task.

TIME COMPLEXITY OF ALGORITHMS

Both algorithms consist of composition of for loops which select n individuals, iterate through the members of population, iterate through the individuals etc. (see pseudocode Algorithm 1 and 2). Every such operation can be performed in $O(n)$, $O(CL)$ or $O(N)$ time, where:
n is the number of selected individuals.
N is the Population size.
C is the overall number of all created clones, i.e. $\sum_{i=1}^{N} \frac{\beta * N}{i}$.

L is the length of individual (antibody).
M is the number of patterns to recognise.
Indeed, Castro and Zuben [6] admit the time complexity of CLONALG to be: $O(M(N + N_cL)$ for pattern recognition and $O(N + N_cL)$ for function optimisation. However, they do not consider changes to particular members of remaining repertoire (as mentioned earlier $d = 0$) and antibodies' replacement is based on the selection size n. In the above design of CLONALG, antibodies above certain threshold are changed so it adds dL operations for every pattern and every generation. The revised complexity for both CLONALG and Cell Rescue is then:

Pattern recognition	$O(M(N + N_cL + dL)$
Function optimisation	$O(N + N_cL + dL)$

RESEARCH QUESTION

Given the problem context (Chapter 1), background (described in Chapter 2) and the algorithms presented above the research question arises: *Is the Cell Rescue version better than the standard CLONALG?* An approach to address this question is to state precise hypotheses and conduct two kinds of experiment each adequate to the application with careful assess of domain suitability.

Chapter 4

Experiment Details

The chapter 4 states clearly all hypotheses and points out all experiment details providing specific values for all parameters, explains training and testing data, function selection, number of trials and general practise of experiments.

4.1 Hypotheses

Following hypotheses follow from the research question:

H0 There is no difference between CLONALG and Cell Rescue version.

H1 Cell Rescue version will classify successfully more patterns than the original CLONALG using Method 1 of reshuffling.

H2 Cell Rescue version will classify successfully more patterns than the original CLONALG using Method 2 of reshuffling.

H3 Cell Rescue version will classify successfully more patterns than the original CLONALG using Method 3 of reshuffling.

H4 Cell Rescue version will classify successfully more patterns than the original CLONALG using Method 4 of reshuffling.

Cell Rescue version will find the minimum quicker than the original CLONALG using Method 4 of reshuffling.

4.2 Experiment Design

To appropriately test all hypotheses two versions of experiments were conducted: pattern recognition and function optimisation. This section is divided into two parts: one part details the pattern recognition experiment, second section details function optimisation experiment.

4.2.1 Pattern Recognition

In order to establish the difference between CLONALG and Cell Rescue, the same set of patterns was introduced to all versions of algorithm: CLONALG and 4 versions of Cell Rescue, called Cell Rescue 1, Cell Rescue 2, Cell Rescue 3, Cell Rescue 4 respectively for the method described above. The output (solution) memory set was classified against training patterns after every generation and also classified against 171 new previously unseen patterns. The test was repeated 5 independent times using 5 different populations (1 per trial) creating 100 generations. Different populations must be used to establish central tendency and ensure that the algorithm can achieve better results regardless the initial population. Then the mean of the 5 trials was taken and, based on the mean, accuracy was calculated for every generation, such that:

$$Accuracy = \frac{\text{Number of correctly classified patterns}}{\text{All patterns}}.$$

To allow for clear comparison and ensure noise elimination, the same randomly generated initial population was used for both algorithms (within one trial). Without that it could appear that one algorithm was run with more "convenient" initial population while the second started from the worse case. Additionally, the standard deviation was calculated for all trials to see the amplitude between trials. The experiment was conducted with use of real-world data set imported from the online UCI Machine Learning Repository [8]. The chosen data set is a Glass Identification Data Set for classification task. The sample set consists of 6 types of glass defined in terms of their oxide content. Every pattern in the set is a vector with 11 attributes:
- Identification number
- Refractive index
- Sodium measurement
- Magnesium measurement
- Aluminum measurement
- Silicon measurement
- Potassium measurement
- Calcium measurement
- Barium measurement
- Iron measurement
- Type of glass: class attribute

Attributes 2-10 were used as a pattern. Due to the specification of the algorithm, all patterns were encoded from real to binary form such that every entry is now a 16 bits long binary number (8 bits for decimal part and 8 for integer part). This way every pattern is a 228 bits list. All input data (test populations, patterns in real and binary versions - training and test sets) can be found within the folder ..... All parameters introduced above were

optimised for individual length and training set size and were constant for all algorithms and all tests:

Population size	**52**
Individual length	**288**
Memory size	**42**
Selection size	**32**
Affinity threshold	**140**
Control parameter	**0.03**
Clone rate	**10**
Number of generations	**101**

To sum up, all algorithms were trained on 7 different examples of each class and were first tested using the same set of examples and later using a new set of 171 previously unseen patterns.

4.2.2 Function Optimisation

CLONALG and Cell Rescue 4 for function optimisation were tested with 7 functions - looking for the minimum of a function. Every test was repeated 5 times for every function, similarly as above with 5 different populations. (The same as above it is a crucial case to check algorithms against different initial populations). Again, the standard deviation was calculated to see if the variance between trials is significant. This time, however, the memory set composes the whole population (as outlined in chapter 3) so only the best affinity individual is considered as a solution after each generation. The final results are the mean of the 5 tests. Then the difference between f(x,y) Cell rescue and f(x,y) CLONALG was computed to establish with which algorithm function attains smaller minimum after every generation. The benchmark functions for testing are:

Function	**Formula**	**Domain**	**Min**	**Affinity threshold**
Sphere	$f(x,y) = x^2 + y^2$	$(-4.5, 4.5)$	$f(0,0) = 0$	25
Matyas	$f(x,y) = 0.26(x^2 + y^2) - 0.48xy$	$(-10, 10)$	$f(0,0) = 0$	22
Booth	$f(x,y) = (x + 2y - 7)^2 + (2x + y - 5)^2$	$(-10, 10)$	$f(1,3) = 0$	1200
Beale	$f(x,y) = (1.5 - x + xy)^2$ $+(2.25 - x + xy^2)^2 + (2.625 - x + xy^3))^2$	$(-4.5, 4.5)$	$f(3, 0.5) = 0$	90000
Levi N.13	$f(x,y) = (x-1)^2(1 + sin^2(3\pi y))$ $+sin^2(3\pi x)) + (y-1)^2(1 + sin^2(2\pi y))$	$(-10, 10)$	$f(1,1) = 0$	121
Ackley	$f(x,y) = -20exp(-0.2\sqrt{0.5(x^2 + y^2)} -$ $exp(0.5(cos(2\pi x) + cos(2\pi y))) + e + 20$	$(-4.5, 4.5)$	$f(0,0) = 0$	7
Three-hump camel	$f(x,y) = 2x^2 - 1.05x^4 + \frac{x^6}{6}$ $+xy + y^2$	$(-4.5, 4.5)$	$f(0,0) = 0$	21000000

IMPLEMENTATION DETAILS

Every algorithm was implemented in Python 3.6. The libraries used are: NumPy, math. The NumPy library is a fundamental package for scientific usage and allowed for easy manipulation of the antibody's lists. I.e. computing the exclusive OR for Hamming method, performing reshuffling of list elements, generating random numbers, zipping two attributes to sort antibodies according to affinity. Additionally, this library simplifies the syntax of the code - creating loops is intuitive, additional parameters can be introduced, like sorting, which is important due to the nature of algorithm (selecting best affinity antibodies occurs many times). What is more, NumPy admits efficiency in handling data - arbitrary data-types can be defined [11]. Besides, standard Numerical and Mathematical module (math) was imported to allow for implementing described functions for optimisation what required using cos(), sin(), powers functions, etc.

Chapter 5

Results

Similarly, the chapter is split into two sub-sections to present results for every application independently. Detailed results are attached together with the used Populations and patterns to allow for experiment replication (see *Results & Data* folder).

PATTERN RECOGNITION RESULTS

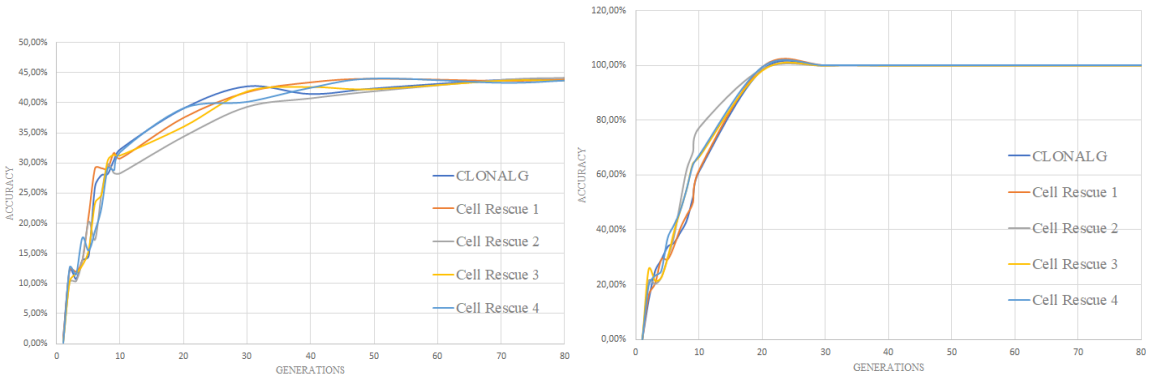
Figures 5.1 present the overall number of patterns classified as: correct, incorrect and unknown where all algorithms were trained on 7 different examples of each class and tested using a set of 171 previously unseen patterns. Figure 5.2 (a) shows the corresponding accuracy of algorithms. In addition to that Figure 5.2 (b) presents the accuracy when training the algorithms, i.e. it shows accuracy when algorithms were trained on 7 different examples of each class and tested using the same set of patterns. Also, the standard deviation test was conducted and its outputs are shown in the Appendix.

The results shows that biggest accuracy was achieved by Cell Rescue 2, the smallest by Cell Rescue 3, while CLONALG was between them:

- CLONALG max accuracy: 43.74%, classifying correctly 74.8 patterns out of 171
- Cell Rescue 1 max accuracy: 43,98%, classifying correctly 75.2 patterns out of 171
- Cell Rescue 2 max accuracy: 44.09%, classifying correctly 75.4 patterns out of 171
- Cell Rescue 3 max accuracy: 43.63%, classifying correctly 74.6 patterns out of 171
- Cell Rescue 4 max accuracy: 43.98%, classifying correctly 75.2 patterns out of 171

It can be concluded that Cell Rescue 1,2 and 4 managed to classify correctly more patterns than CLONALG, while Cell Rescue 3 classified correctly less patterns than CLONALG. Though, the difference is small. Training, however, was achieved quicker with Cell Rescue 2,3,4 (see figure 5.2(b)). What is more, a low standard deviation (in particular for the more matured patterns - after 50 generations) indicates that all trials are close to the mean, so the results are reliable.

Figure 5.1: Output: CLONALG and Cell Rescue 1, 2, 3, 4 pattern recognition



(a) Accuracy of testing set

(b) Accuracy of training set

Figure 5.2: Accuracy through generations

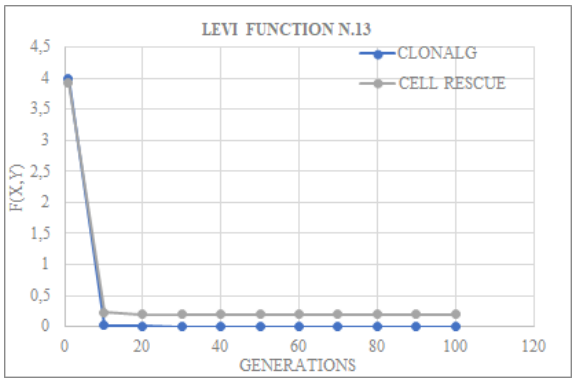


Figure 5.3: Levi function N.13

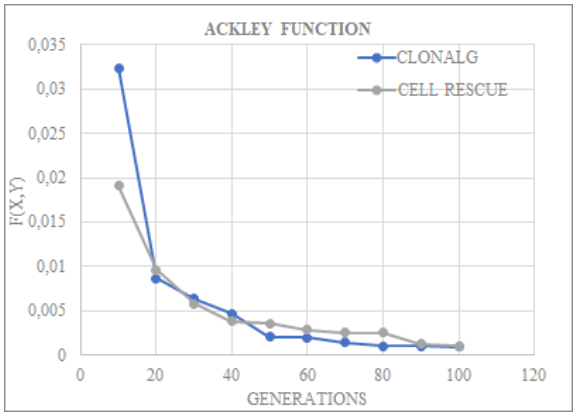
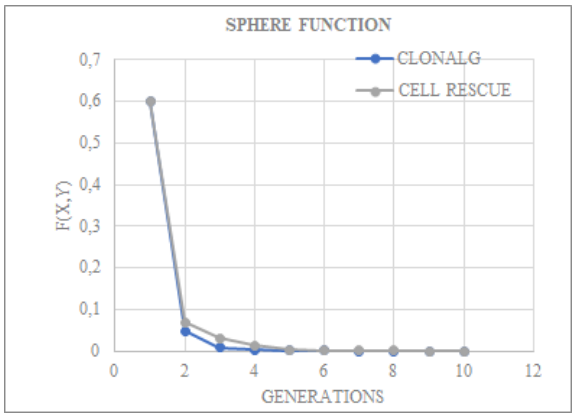


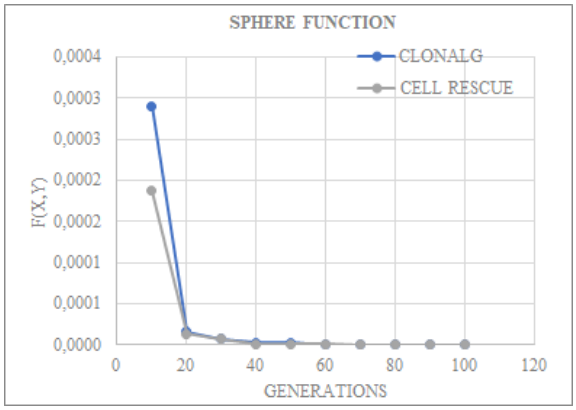
Figure 5.4: Ackley function

FUNCTION OPTIMISATION RESULTS

Figures 5.3-5.9 illustrate results obtained from executing above algorithms 5 times and calculating the mean of every trial. The value of f(x,y) is plotted for every generation for every tested function and both algorithms (appropriate line is added as well as the legend to differentiation between algorithms). In some cases (Sphere, Matyas, Booth, Beale) the graphs are split in two pictures - one presenting generations 1-10, the other one 10-100 - due to big amplitude of values, when it is still important two preserve both. Additionally, the difference between values of f(x,y) in Cell Rescue and CLONALG is computed for every generation: Difference = $CellRescue_{f(x,y)}$ - $CLONALG_{f(x,y)}$, meaning that when Difference > 0 then CLONALG is closer to the expected minimum, but when Difference < 0 Cell Rescue is closer to the minimum (see "FunctionsOutput" files). Standard deviation (see Appendix) shows that there are no big differences between tests.

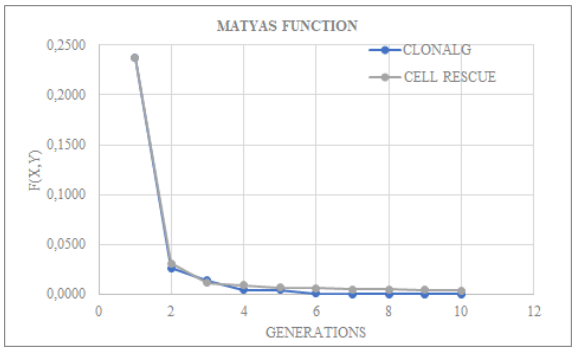


(a) Generations 1-10

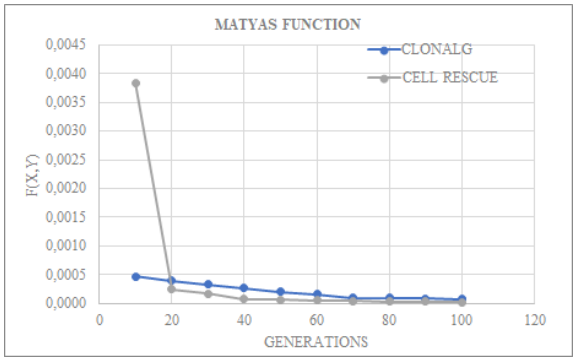


(b) Generations 20-100

Figure 5.5: Sphere function

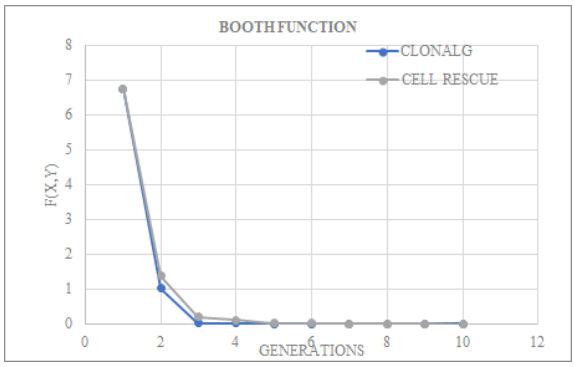


(a) Generations 1-10

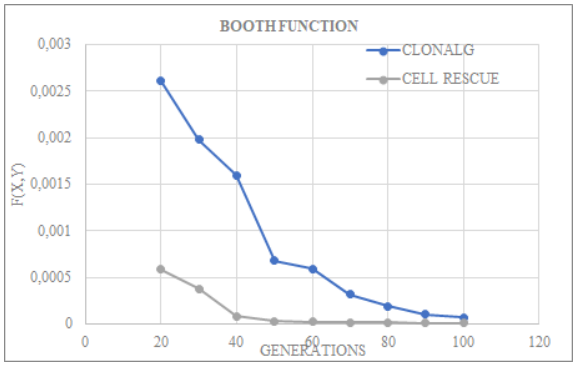


(b) Generations 20-100

Figure 5.6: Matyas function

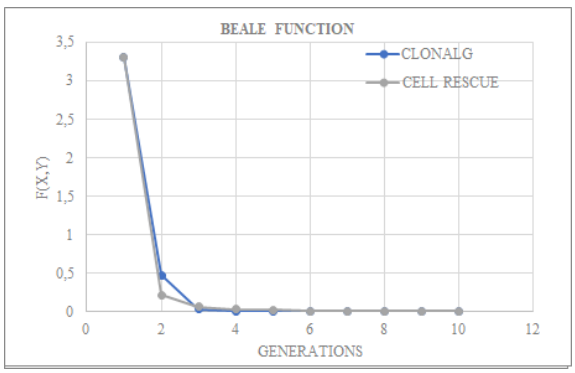


(a) Generations 1-6

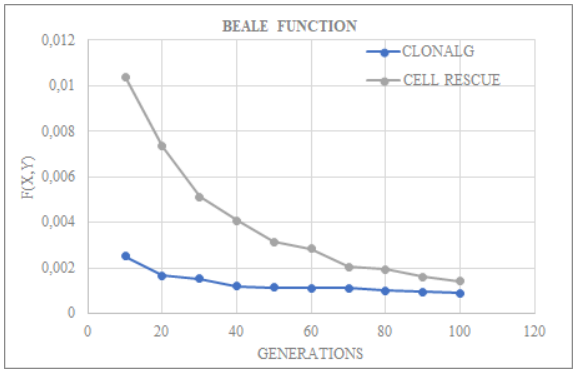


(b) Generations 20-100

Figure 5.7: Booth function



(a) Generations 1-10



(b) Generations 20-100

Figure 5.8: Beale function

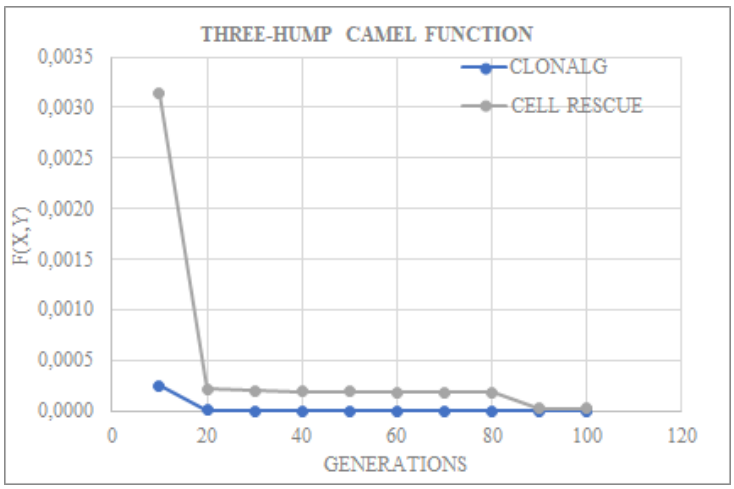


Figure 5.9: Three-hump camel function

Chapter 6

Discussion and Conclusion

This chapter gives qualitative analysis of results and explains why such results were obtained. Last section concludes the research.

6.1 Discussion

The results suggest that there is a difference between CLONALG and Cell Rescue pattern recognition. 3 out of 4 hypotheses can be accepted, however, the differences are smaller than expected. Intresting case is outlined in the training accuracy as then Cell Rescue improves the pattern recognition process significantly. The aim of the research was to establish the difference between the algorithms, so the overall performance of all algorithms is not the priority. Therefore, test accuracy at the level of 43-44% is not an issue. What is more, similar results (in case of accuracy) were obtained by White and Garrett [20] when testing the binary character pattern recognition with patterns the same as in testing and with 80 examples of previously unseen patterns.

The authors of "Receptor editing during affinity maturation" claimed that the reshuffling can *occasionally* improve the performance [9]. The graph of accuracy (Figure 5.2 (a)) seems to prove this claim: CLONALG line is quite smooth, while all Cell Rescue methods take unexpected "jumps".

The other purpose of the experiment was to see if Cell Rescue would allow to minimise any function. From the results it can be observed that not necessarily. With given functions and trial number it is impossible to accept the general hypotheses that the minimum will be achieved quicker with every function using Cell Rescue. The experiment has shown that further dependencies need to be establish like function complexity (see future work) not predicted earlier. For now, it can be concluded that the Cell Rescue is better for very specific functions like Matyas, Booth functions and worse for Three-hump camel and Levi function N.13. The null hypotheses can be accepted for sphere and Ackley functions - there is no significant difference between both algorithms, they both attains similar minima through the generations. It is also important to notice that the performance of CLONALG and Cell Rescue changes. In the case of Booth and Matyas functions Cell

Rescue appears to obtain worse results at the beginning but better later (after 10 generations). In opposite to Beale function where the situation is reversed. This observation may be important as it suggests which algorithm allows for better results with small number of generations and which is more optimal in general (possibly big number of generations). Another conclusion is that the functions which manage to attain smaller minima with Cell Rescue were the ones which required smaller number of operations (like multiplication addition, etc.). For example, Matyas has 7 operations, Booth - 9, while Levi around 18.

6.2 Conclusion

For the 3 months I have been working on the project reviewing the existing implementation of CLONALG and its improvements as well as the theoretical aspect of the project. As a result I proposed, analysed and implemented improved version of CLONALG: Cell Rescue for pattern recognition and function optimisation. Provided study has shown that Cell Rescue indeed influences the efficiency. CLONALG efficiency is shown to be increased for 3 methods in pattern recognition and selected functions in optimisation task but, in the same time, decreased for others. Throughout the experiment it has become clear that more more dependencies need to be established and that the stated research question cannot be answered in general.

Chapter 7

Future Work and Evaluation

Obtained results have allowed to confirm or deny stated hypotheses and the presented research fulfilled all designed goals. However, both versions of experiment have showed that still a lot can be done to achieve more consistent results and apply the outcomes to bigger domain. This chapter describes future actions that can be undertaken to further improve the research. At the end, the overall evaluation of the project is provided.

7.1 Future work

First thing to consider is developing an algorithm capable of handling the real representation (in pattern recognition) not only binary. This way the noises caused by the real to binary conversion could be eliminated. Moreover, the research could be extended by repeating the experiment with different sets of patterns not just one. Later additional noises could be introduced to the individuals to test the proposed algorithms against robustness as well.

Function optimisaiton version of the experiment has shown that there are many ways in which the Cell Rescue can be improved. Firstly, establishing correlations between function complexity and Cell Rescue effectiveness may appear crucial. For that definition of "function complexity" based on the number of operations should be made. Another thing to notice is that neither of the algorithms has managed to achieve really close approximation of x and y (see "FunctionsOutput" file). Finding the appropriate library to handle better approximation of real numbers with long decimal extension (e.g. "decimals" library) would allow to further optimise the algorithm.

For both experiment versions, it also seems useful to increase the number of tests per population and general number of populations for testing. The aim of the research was to see the difference between CLONALG and Cell Rescue - this is way all parameters were left the same for both algorithms. However, further optimisation of parameters can be done for Cell Rescue version itself. Especially choosing the affinity threshold which determines cells to reshuffling. Setting the threshold lower would mean more cells for reshuffling. Also, it would be important to consider the parameters choice to prevent over

fitting. Not only increasing the number of test cases is essential, but also the length of individuals. In particular, the pattern recognition version achieved similar results what could be possibly change by significantly increasing the length of patterns (twice or even more). For now, the research managed to show the general tendency.

What is more, another research on specifically light and heavy chains of antibody reshuffling in the immune system would let to gain inspiration for creating perfect reshuffling method in the future.

Last thing for future evaluation is to check the effectiveness of Cell Rescue for other application like Combinatorial Optimization. Conducted research showed that there are differences in effectiveness of Cell Rescue depending on the application. Hence, the conclusion that the Cell Rescue is better in general for every application cannot be made and opens the new possibilities of testing. Castro and Zuben successfully applied CLONALG to the travelling salesman problem (TSP) [6], where a salesman must visit each of 30 cities exactly once and return to the starting city with the minimal tour coast. The trial could be repeated now using reshuffling of low affinity antibodies instead of generating new ones.

7.2 Evaluation

Overall, the conducted research provided satisfactory results and was accomplished in reasonable amount of time thanks to the good time schedule. The development process followed the agile approach: firstly simple implementation of all algorithms were tested with 12 bits binary string encoding simple bitmap of character. Later, the other application of the algorithm was developed - function optimisation. Next step wast extending the memory size of CLONALG. After creating the algorithm capable of recognising many patterns, I considered options to implement it for real representation (using Euclidean distance method). However, this method were not able to differentiate between small differences i.e. numbers like 5.4783 and 5.4130 and had really low effectiveness. Therefore, instead of improving new version, I decided to change the representation of my data set to binary what prevented relatively good effectiveness but probably produced small noises caused by conversion. Before this decision I also considered many different data sets available on the UCI Machine Learning Repository and additionally tried animal classification set which was originally encoded in binary form. The time needed to executed all 5 algorithms was smaller in comparison to the glass identification set when one algorithm takes 1 hour to execute (time constraints were the reason of doing only 5 trials in proper experiment). First trials seemed to prove a few of stated hypotheses, however after consideration, the set was changed to glass identification as the length of an individual (in animal set) was only 19 bits what was too small to see the difference clearly. Before that, I also prepared the set of binary string encoding the mentioned bitmaps (now of length 120) but it was essential to test all algorithms with real-world data. From this part of the

project the personal reflection is that after implementing the algorithm with simple case I should have focused more on extending it to more complicated cases as this caused a delay later (difficulties at this stage were not predicted), when moving to testing, the code was not entirely ready to deal with real life examples. It appeared that many issues were simplified and needed to be fixed. Another reflection is that the programming should have been started earlier to go "deeper" into the problem (see the problem not only from the theoretical but also practical side). The effectiveness of all algorithms is not a strength but as the task is to compare between algorithm the true difference, which is the purpose of the research, is constant and is not influenced.

Bibliography

[1] Berek, C. and Ziegner, M. (1993). The maturation of the immune response. *Immunology Today*, 14.

[2] Brownlee, J. (2011). *Clever Algorithms: Nature-Inspired Programming Recipes*. Lulu.com, 1st edition.

[3] Castro, L. N. and Timmis, J. (2002). *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer, Great Britain.

[4] Castro, L. N. and Zuben, F. J. V. (1999). Artificial immune systems: Part 1 - basic theory and applications. Technical Report TR DCA 01/99, 01/99, Department of Computer Engineering and Industrial Automation, School of Electrical and Computer Engineering, State University of Campinas, Brazil.

[5] Castro, L. N. and Zuben, F. J. V. (2000). The clonal selection algorithm with engineering applications 1. In *Workshop on GECCO*.

[6] Castro, L. N. and Zuben, F. J. V. (2002). Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation*, 6(3):239–251.

[7] Cook, M. (2000). Receptor editing (and the evolution of sex). *Immunology Today*, 21(1):55 – 56.

[8] Dua, D. and Graff, C. (2017). UCI machine learning repository.

[9] George, A. J. and Gray, D. (1999). Receptor editing during affinity maturation. *Immunology Today*, 20(4):196.

[10] George, A. J. and Gray, D. (2000). Jumping or walking: which is better? *Immunology Today*, 21(1):55.

[11] Jones, E., Oliphant, T., Peterson, P., et al. (2001–). SciPy: Open source scientific tools for Python. [Online; accessed <today>].

[12] Mak, T. W., Saunders, M. E., and Jett, B. D. (2014). *Primer to the Immune Response, 2nd Edition*. Elsevier, US.

[13] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

[14] Medzhitov, R. (2013). Pattern recognition theory and the launch of modern innate immunity. *The Journal of Immunology*, 191(9).

[15] Meffre, E. and Wardemann, H. (2008). B-cell tolerance checkpoints in health and autoimmunity. *Current Opinion in Immunology*, 20(6):632 – 638. Autoimmunity /

Allergy and Hypersensitivity.

[16] Rai, N. and Singh, A. (2015). Improved clonal selection algorithm (iclonalg). *International Journal of Current Engineering and Technology*, 5(4).

[17] Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21(4):25–34.

[18] Sadeghi, J., Sadeghi, S., and Niaki, S. T. A. (2014). Optimizing a hybrid vendor-managed inventory and transportation problem with fuzzy demand: An improved particle swarm optimization algorithm. *Information Sciences*, 272:126 – 144.

[19] Timmis, J. and Flower, D. (2007). *In Silico Immunology*. Springer Science, Spring Street, New York.

[20] White, J. and Garrett, S. (2003). Improved pattern recognition with artificial clonal selection? In Timmis, J., Bentley, P., and Hart, E., editors, *Artificial Immune Systems*, volume 0 of *Lecture Notes in Computer Science*, pages 181–193, Netherlands. Springer.

[21] Zhang, W., Lin, J., Jing, H., and Zhang, Q. (2016). A novel hybrid clonal selection algorithm with combinatorial recombination and modified hypermutation operators for global optimization. *Computational Intelligence and Neuroscience*, 2016:1–14.

Appendix

Standard Deviation Tests

Gen No	CLONALG			Cell Rescue 1			Cell Rescue 2			Cell Rescue 3			Cell Rescue 4		
	Correct	Incorrect	Unknown	Correct	Incorrect	Unknown	Correct	Incorrect	Unknown	Correct	Incorrect	Unknown	Correct	Incorrect	Unknown
1	5.73	55.29	60.18	4.38	41.00	44.71	2.19	27.14	28.76	1.79	12.03	12.77	0.89	29.41	29.15
2	2.00	2.68	1.79	2.00	3.03	4.15	9.65	8.67	12.17	12.54	11.22	14.74	2.28	3.29	3.63
3	11.54	11.54	0.00	9.65	9.65	0.00	4.56	4.98	1.79	8.29	8.29	0.00	5.40	5.40	0.00
4	12.84	12.84	0.00	18.74	18.74	0.00	9.21	9.21	0.00	11.52	11.52	0.00	11.66	11.66	0.00
5	9.70	9.70	0.00	18.63	18.63	0.00	17.41	17.41	0.00	14.32	14.32	0.00	15.72	15.72	0.00
6	15.01	15.01	0.00	12.60	12.60	0.00	15.66	15.66	0.00	12.60	12.60	0.00	19.39	19.39	0.00
7	22.73	22.73	0.00	9.94	9.94	0.00	16.77	16.77	0.00	17.85	17.85	0.00	18.42	18.42	0.00
8	9.53	9.53	0.00	18.35	18.35	0.00	11.61	11.61	0.00	20.40	20.40	0.00	14.72	14.72	0.00
9	10.33	10.33	0.00	8.88	8.88	0.00	11.63	11.63	0.00	18.90	18.90	0.00	13.52	13.52	0.00
10	9.42	9.42	0.00	14.11	14.11	0.00	6.57	6.57	0.00	10.92	10.92	0.00	16.89	16.89	0.00
20	10.81	10.81	0.00	9.74	9.74	0.00	17.05	17.05	0.00	18.90	18.90	0.00	7.13	7.13	0.00
30	7.75	7.75	0.00	10.06	10.06	0.00	12.60	12.60	0.00	12.85	12.85	0.00	6.42	6.42	0.00
40	9.53	9.53	0.00	3.29	3.29	0.00	6.42	6.42	0.00	9.21	9.21	0.00	5.59	5.59	0.00
50	1.79	1.79	0.00	5.73	5.73	0.00	4.15	4.15	0.00	8.29	8.29	0.00	6.39	6.39	0.00
70	3.29	3.29	0.00	2.28	2.28	0.00	2.97	2.97	0.00	2.68	2.68	0.00	3.46	3.46	0.00
80	2.19	2.19	0.00	1.41	1.41	0.00	1.10	1.10	0.00	2.28	2.28	0.00	1.10	1.10	0.00
100	0.00	0.00	0.00	0.89	0.89	0.00	0.00	0.00	0.00	1.41	1.41	0.00	0.00	0.00	0.00

Figure 1: Standard deviation test: Pattern Recognition

	Sphere				Matyas				Booth				Beale			
	CLONALG		Cell		CLONALG			Cell Rescue	CLONALG		Cell Rescue		CLONALG		Cell Rescue	
Gen No.	x	y	x	y	x	y	x	y	x	y	x	y	x	y	x	y
1	1.51	0.35	1.51	0.35	3.41	3.52	3.41	3.52	2.08	1.72	2.08	1.72	5.85	0.87	5.85	0.87
2	0.13	0.20	0.22	0.09	0.45	0.36	0.43	0.58	0.81	0.93	1.38	1.12	4.63	0.94	1.16	0.27
3	0.06	0.13	0.15	0.15	0.45	0.24	0.49	0.58	0.26	0.22	0.48	0.47	0.60	0.10	1.00	0.25
4	0.05	0.04	0.03	0.14	0.29	0.22	0.54	0.56	0.22	0.21	0.44	0.22	0.31	0.10	0.62	0.22
5	0.05	0.03	0.04	0.05	0.29	0.22	0.57	0.55	0.18	0.21	0.19	0.18	0.39	0.10	0.65	0.20
6	0.02	0.02	0.04	0.02	0.10	0.12	0.51	0.57	0.18	0.19	0.19	0.17	0.31	0.07	0.58	0.15
7	0.02	0.02	0.03	0.03	0.15	0.14	0.49	0.55	0.18	0.17	0.11	0.08	0.31	0.07	0.59	0.15
8	0.02	0.02	0.03	0.03	0.16	0.16	0.52	0.52	0.15	0.17	0.09	0.10	0.32	0.07	0.54	0.15
9	0.02	0.02	0.02	0.03	0.15	0.15	0.48	0.49	0.15	0.17	0.09	0.10	0.31	0.07	0.54	0.15
10	0.02	0.02	0.01	0.02	0.13	0.15	0.47	0.41	0.15	0.16	0.08	0.07	0.31	0.07	0.54	0.15
30	0.00	0.00	0.00	0.00	0.12	0.12	0.07	0.07	0.06	0.07	0.02	0.03	0.23	0.06	0.39	0.10
50	0.00	0.00	0.00	0.00	0.09	0.09	0.04	0.04	0.04	0.04	0.00	0.00	0.19	0.05	0.31	0.08
80	0.00	0.00	0.00	0.00	0.05	0.05	0.02	0.02	0.02	0.02	0.00	0.00	0.18	0.05	0.25	0.06
100	0.00	0.00	0.00	0.00	0.05	0.04	0.01	0.01	0.01	0.01	0.00	0.00	0.17	0.04	0.21	0.05

Figure 2: Standard deviation test: Function Optimisation 1

Levi function N. 13				Ackley				Three-hump			
CLONALG		Cell Rescue		CLONALG		Cell Rescue		CLONALG		Cell Rescue	
x	y	x	y	x	y	x	y	x	y	x	y
1.10	3.09	1.12	3.16	1.92	0.71	1.92	0.71	1.90	0.59	1.91	0.59
1.03	0.75	0.71	0.82	0.07	0.05	0.11	0.08	0.16	0.27	0.16	0.28
0.58	0.55	0.56	0.98	0.05	0.08	0.07	0.08	0.17	0.13	0.16	0.18
0.38	0.34	0.33	1.35	0.06	0.03	0.06	0.04	0.03	0.08	0.08	0.14
0.39	0.16	0.31	1.79	0.01	0.01	0.02	0.07	0.03	0.03	0.06	0.12
0.37	0.15	0.33	2.24	0.01	0.01	0.02	0.04	0.03	0.03	0.03	0.09
0.02	0.13	0.31	2.68	0.01	0.01	0.03	0.01	0.02	0.03	0.01	0.05
0.04	0.13	0.04	3.13	0.01	0.00	0.01	0.01	0.01	0.03	0.03	0.05
0.04	0.13	0.30	3.57	0.01	0.00	0.01	0.01	0.02	0.01	0.03	0.05
0.03	0.13	0.31	4.02	0.01	0.00	0.00	0.01	0.01	0.01	0.01	0.04
0.00	0.01	0.00	12.97	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.01	0.00	21.91	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.01	0.00	35.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.01	0.00	44.27	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Figure 3: Standard deviation test: Function Optimisation 2

Maintenance Manual

Prerequisites:

- Python 3.6 together with NumPy and json packages installed

Executing the code:

- Download and extract the zip folder to the desired location.
- To execute CLONALG (pattern recognition) open and run the PatternRecognition.py. The algorithm is by default trained with patterns imported from the GlassPatterns.txt and tested with patterns from GlassPatterns2.txt and all parameters are set up to match the patterns. To run the algorithm with new patterns the variable "PatternsLearn" needs to be change. It can be done by either hardcoding the patterns within the program i.e. setting PatternsLearn to a list of paterns (a pattern needs to be a binary numpy list).

For example: PatternsLearn = [np.array([1,0,0,1,0,0,1,0,0]), np.array([0,1,0, 0,1,0, 0,1,0])] - the variable stores 2 encoded patterns representing simple bitmap for character "1" encoded as a list of bits. Alternatively, the file called "GlassPatterns" can be change with a list of new patterns. New patterns must be in binary form and should be in order: patterns of class A, patterns of class B, etc. If the classification is desired then it must be marked within the code how many patterns are there for every class. Additionally, following parameters and variables must be justify to the chosen patterns set: Popul size, Individual length, Memory size, Selection size, Affinity threshold, Control parameter, clone rate, Number generations (detailed explanation of every variable is provided within the report in chapter 3).

- To execute Cell Rescue (pattern recognition) open and run the same file but comment lines used for CLONALG and use lines for reshuffling (see comments). To use method 1 of reshuffling call the function "reshuffle1", to use method 2 call the function "reshuffle2" etc. To run the algorithm with new patterns follow the same steps as above.
- To execute CLONALG (function optimisation) open and run the FunctionOptimisation.py . To minimise another function go to the calculate affinity method and implement the formula of the function similarly to the presented examples. Then also change parameters appropriately.
- To execute Cell rescue (function optimisation) follow the same steps as for Cell Rescue (pattern recognition)