A Multi-Set Artificial Immune System for Searching Optima in Dynamic Environments

Jalil Shahabi

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the Degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
August 2012
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Elvan Yılmaz
Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Muhammed Salamah
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Asst. Prof. Dr. Ahmet Ünveren
Supervisor

Examining Committee

1. Asst. Prof. Dr. Adnan ACAN
2. Asst. Prof. Dr. Ahmet ÜNVEREN
3. Asst. Prof. Dr. Yıltan BİTİRİM

ABSTRACT

Artificial Immune Systems (AIS) are computational methods that belong to the computational intelligence family and are inspired by the biological immune system. Many researchers developed immune based models to solve complex computational or engineering problems by using fast exploration ability of the AIS.

The proposed method uses multi-set search mechanisms within AIS to solve the dynamic optimization problems, i.e. moving peak benchmarks. In the moving peak benchmark problems the optimum and environment changes in time. For this reason it is difficult to find optimum. In this thesis distributed detection and fast exploration ability of AIS are combined with multi-set search mechanisms to find optimum solutions of the given dynamic optimization problems.

The given method was compared with different algorithms that solve dynamic optimization problems and the results showed that the proposed method outperforms almost all other methods.

Keywords: dynamic environments, optimization problems, artificial immune systems, moving peaks benchmarks

ÖZ

Yapay Bağışıklık Sistemleri (YBS) hesaplama yöntemleri olup yapay zeka ailesindendir ve biyolojik bağışıklık sistemi ilham alınarak üretilmiş yöntemlerdir. Birçok araştırmacı YBS hızlı keşfetmek yeteneğini kullanarak karmaşık hesaplama veya mühendislik problemlerini çözmek için bağışıklık tabanlı modeller geliştirdi.

Önerilen yöntemde dinamik en iyileme problemlerini çözmek için YBS içinde çoklu-grup arama mekanizmaları kullanıldı. Dinamik problemlere örnek olarak hareketli tepe denektaşları verilebilir. Zaman içinde hareketli tepe denektaşları problemlerinde en iyi ve çevre değişime uğrar. Bu nedenle, en iyiyi bulmak zordur. Bu tez çalışmasında YBS'nin dağıtık algılama ve hızlı keşfetmek yeteneği çoklu-set arama mekanizmaları ile birleştirilerek mevcut dinamik en iyileme problemlerinin en iyi çözümlerini bulmak için kullanıldı.

Verilen yöntem dinamik en iyileme problemlerini çözen farklı algoritmalar ile karşılaştırılmış ve elde edilen sonuçlar doğrultusunda önerilen yöntemin hemen hemen tüm diğer yöntemlerden daha iyi performans gösterdiği ortaya konulmuştur.

Anahtar Kelimeler: dinamik ortamlar, en iyileme problemler, yapay bağışıklık sistemler, hareketli tepe denektaşlar

This thesis is dedicated to my lovely parents who always give opportunity to me to following my dreams, and to my dearest brother Mahmoud who has supported me all the way since the beginning of my studies.

FOR THOSE WHO WERE THERE

AND ARE NOT,

FOR THOSE WHO WERE THERE AND ARE,

FOR HIM, BUT MOST,

FOR HER

ACKNOWLEDGMENT

I would like to thank Asst. Prof. Dr. Ahmet Ünveren for his continuous support and guidance in the preparation of this study. Without his invaluable supervision, all my efforts could have been short-sighted.

I would also like to thank all of my teachers specially Asst. Prof. Dr. Adnan Acan and Assoc. Prof. Dr. Ekrem VAROĞLU who helped me during my education in Cyprus and also all the members of staff at Department of Computer Engineering for the facilities they provided that helped me achieve the results in this thesis.

And also I would like to thank of my dearest friend and my brother Daniyal Yazdani which helped me a lot and never forgot me, and also thanks to my dear friend Mohammad Azhari who supported and helped me a lot and was like a brother in the foreign island.

I owe quite a lot to my family who allowed me to travel all the way from Iran to Cyprus and supported me throughout my studies. I would like to dedicate this study to them as an indication of their significance in this study as well as in my life. I Love them all.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Immune System

Artificial immune system algorithms are methods which have been inspired by immunology theories and Biological observations of the complex mechanisms of living organisms, natural immune defenses against pathogens [1].

These algorithms with characteristics such as dynamic regulation of population size, search space exploration and extraction, and optimization capabilities by maintaining multiple local optimal solutions, in solving many problems have been a point of interest to many researchers, and many different versions of the basic algorithm have been developed and applied for the solution of real world science and engineering problems [2] .

On the other hand, there are some challenges related to this algorithm such as correlation between the cost function and population size, proportion between accuracy and error in the binary exponential range mode, evolution as a random mutation changes, getting stuck of a local optimum and sometimes premature convergence and low convergence rate [3].

In this thesis, the aim is to introduce a new algorithm based on artificial immune systems. This algorithm is designed to solve optimization problem in dynamic environments, particularly the moving peak benchmark problems that are described in chapter 2.

The proposed algorithm used some mechanisms to rectify many problems which exist in artificial immune system algorithms. These mechanisms were inspired by natural immune system of human body to improve performance of artificial immune system algorithms to solve optimization problems in dynamic environments. These mechanisms and the problems that are tried to be solved are explained in chapter 1 and chapter 2.

By comparing the presented algorithm to other algorithms related to dynamic optimization problems, one can say that this is one of the best algorithms in artificial immune system which has been introduced. So far, the results show that this algorithm has better performance than some well-known algorithms in the world of optimization problems. All the results and experiments are shown in chapter 3.

In this chapter, we have a brief review of natural immune system and we will see how immune system can face with new antigens and can improve itself to have a better and faster response to previously seen antigens. You can find these definitions in section 1.1.1. Also, there is a quick look at artificial immune system, and some of the most important algorithms that have been introduced in section 1.1.2 Artificial Immune System (AIS)

1.1.1 Natural Immune System

Immune system is made of a complex structure of cells and blood vessels that have the task of cleansing the body from pathogens. Organs in the body's immune system are called lymphoid organs. Lymph is a Greek word that means clean and limpid flow. Lymphatic vessels and lymph nodes are part of the blood circulatory system to carry lymphocytes, and include a transparent flow of white blood cells and mainly lymph which is showed in **Figure 1-1**.

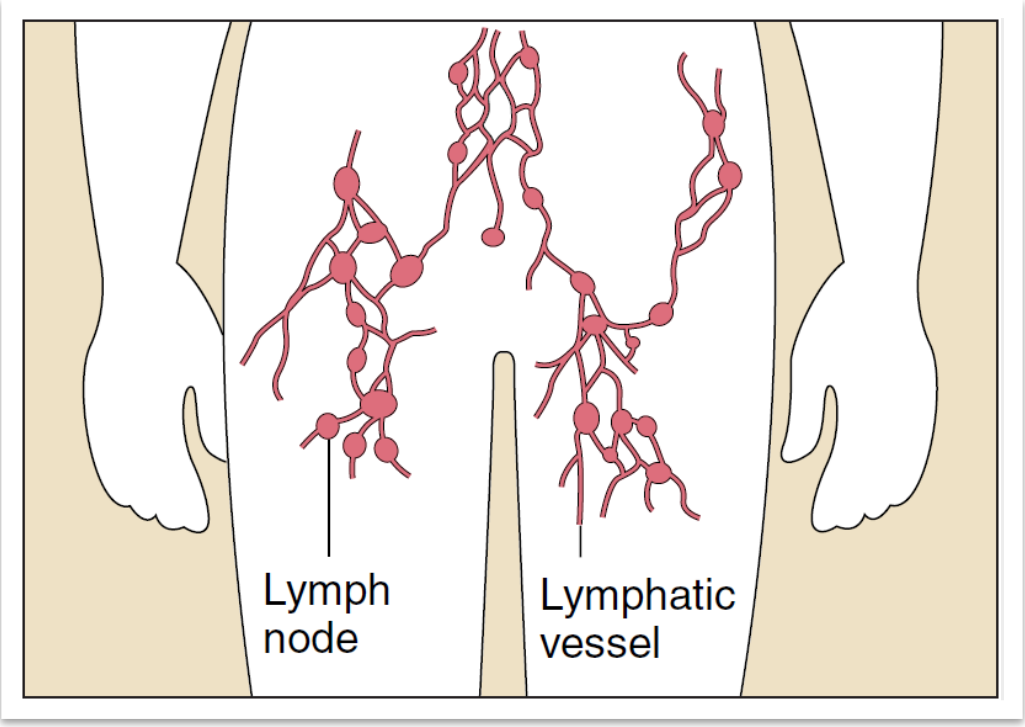


Figure 1-1. Close cooperation lymphatic system and circulatory system **[4]**

Lymphocytes are moved throughout the body by lymphatic vessels and wash all tissues of the body and return to the circulatory system. Lymph nodes mark the lymph vessel network in order to create a confluence of immune cells to defend against aggressors. The spleen which is located at the upper left of the abdomen is a place for immune cells

to confront antigens. Clumps of lymphoid tissue are located in many parts of body such as bone marrow and thymus. Tonsils, adenoids, and appendix are part of the lymph tissue too.

Immune cells and foreign molecules enter through the blood vessels or lymph vessels to lymph nodes. All immune cells that are located in the immune systems are transferred to the blood stream to look for external antigens. Details and structure of the lymph node are shown in Figure 1-2.

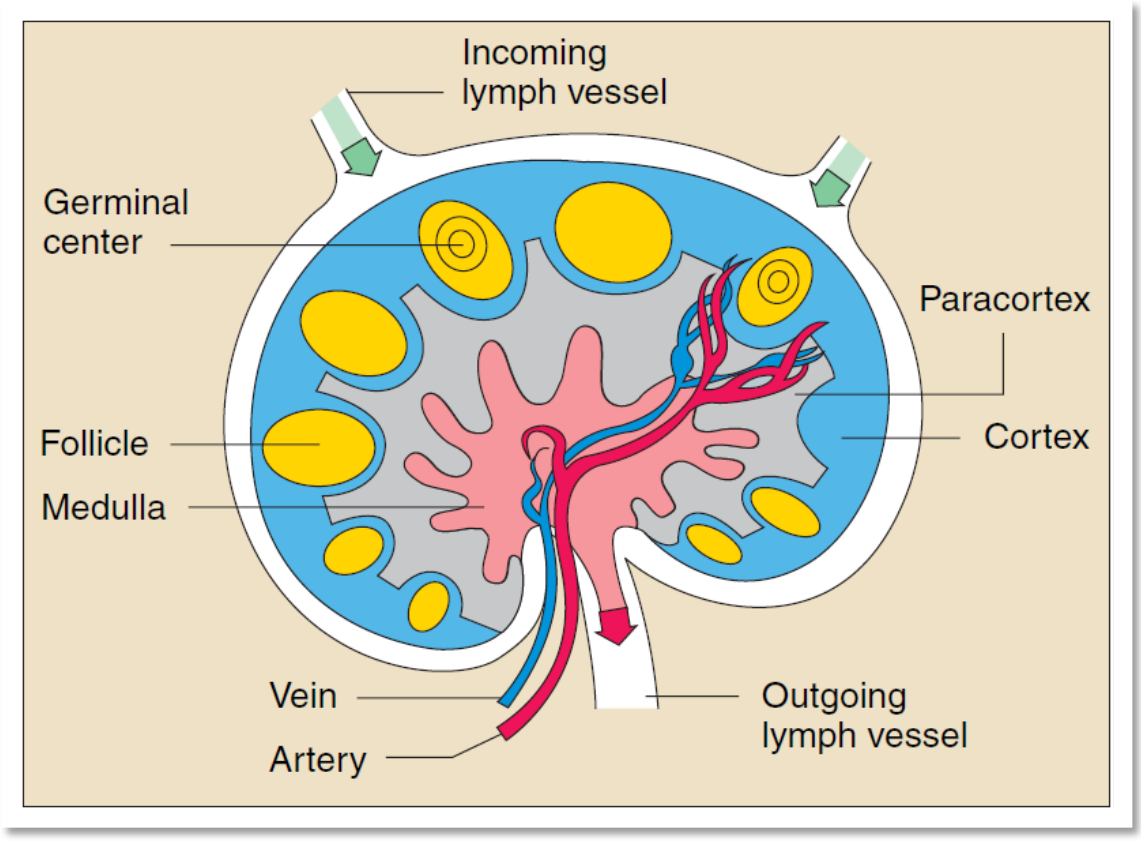


Figure 1-2. Structure of the lymph node [4]

1.1.1.1 Primary Lymphoid Organs

In general, members of the immune system are divided into two groups: primary and secondary lymphoid organs that each of them have their own task. General body's structure and various organs of the immune system are presented in **Figure 1-3**.

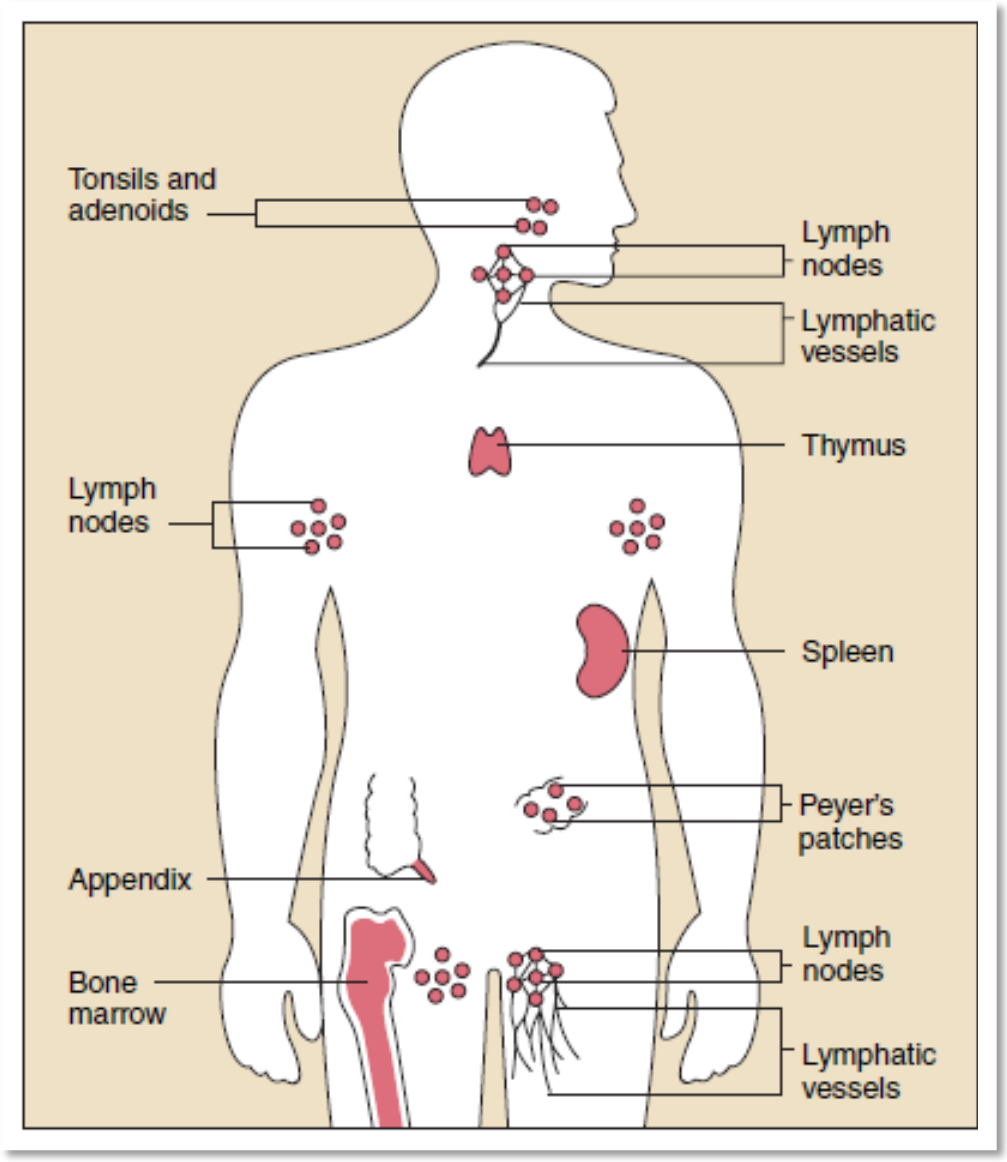


Figure 1-3 Various organs of the immune system in human body **[4]**

The primary lymphoid organs are where lymphocytes have the main evolution. The members of lymphocytes from lymphoid stem cells are improved and then the mature cells are activated and multiplied. In mammals, T cells in the thymus and B lymphocytes in bone marrow and fetal liver are reaching to maturity. Birds have a special area to produce the B-cells that is called bursa of Fabricius. Lymphocytes in primary lymphoid organs, that overcome the antigens a person receives during the life, gain a set of specific antigen receptors. In the primary lymphoid organs, cells with receptors for self-antigens are destroyed, while the T cells in the thymus learn how to recognize their own MHC[1] molecules. When T cells try to distinguish between self and nonself cells, they use a kind of protein that is called MHC [4].

1.1.1.1.1 Bone Marrow

Bone marrow is a soft, sponge-like material that exists within the bone. It contains immature cells that are called stem cells and the production of blood cells is their task. There are three types of blood cells: White blood cells which have immune task; Red blood cells that carry oxygen to the organs and tissues and also collect wasted production from them and Platelets have the task of blood clotting and wound repair.

Most of the stem cells are found in bone marrow but some of them that are called Peripheral blood stem cell (PBSCs) are in the blood flow. With cell division, more cells are produced and after that cells become mature and are converted to white and red blood cells.

[1] Major histocompatibility complex

Blood cells are formed in bone marrow. Bone marrow is located in bone cavity. In children, a marrow of all bones are involved in cells production; but in adults, it is the duty of active part of bone marrow which is called red marrow and is limited to the bones of the body's trunk. Despite the fact that the name of this type of bone marrow is red, they make both the red and white blood cells.

1.1.1.2 Secondary Lymphoid Organs

After production of lymphocytes in primary lymphoid organs, they are stimulated by antigen and immigrate to secondary organs to be distinguished; the spleen is one of the most important members in this group.

1.1.1.2.1 Spleen

Spleen is a wide organ which is located near the stomach, but it is never part of the digestive system and it is more related to the circulatory system.

In a healthy human body, about 10 million red blood cells are destroyed in seconds and of course the red blood cells are needed to be replaced. To do this, three different parts of body work together: bone marrow, liver and spleen. Red bloods cells are stored in the spleen to work in emergency time; also worn-out blood cells are destroyed in spleen. Some of the white blood cells that called lymphocytes are made in the spleen and bone marrow. When a severe shortage of blood occurs suddenly, spleen releases large number of red blood cells to fill the shortage [5].

1.1.1.3 Immune's Type

Human immune system can protect the body against disease-causing agents such as viruses, bacteria, fungi and other parasites. Immune system in living organisms are divided into two categories: innate immunity and adaptive immunity.

1.1.1.3.1 Innate Immunity

Innate immune system was first described by Elie Metchnikoff a century ago. Innate immunity is not faced directly to the invaders, but it fights indirectly with the pathogens that are imported to the body. Innate immune has this feature to immediately deal with pathogens. But in this situation they cannot have modification and correction by generation and mutation [6].

1.1.1.3.2 Adaptive Immunity

Adaptive immune system deals with aggressors by modification of different sets of receptor genes randomly, and with appearance of new invaders they become modified and this mechanism allows the host to generate immunological memory.

Unlike Innate immune system, this system requires modification and reproduction to deal with pathogens which is certainly a time consuming process. Artificial immune systems that are used by researchers in computer science have been inspired by most of these methods and strategies. The adaptive immune system is also called acquired immunity [6].

1.1.1.4 How the Iimmune System Works

Immune system deals with pathogens by a multi-layer structure at several different levels and with respect to innate and adaptive immunity, this structure is shown in Figure 1-4.

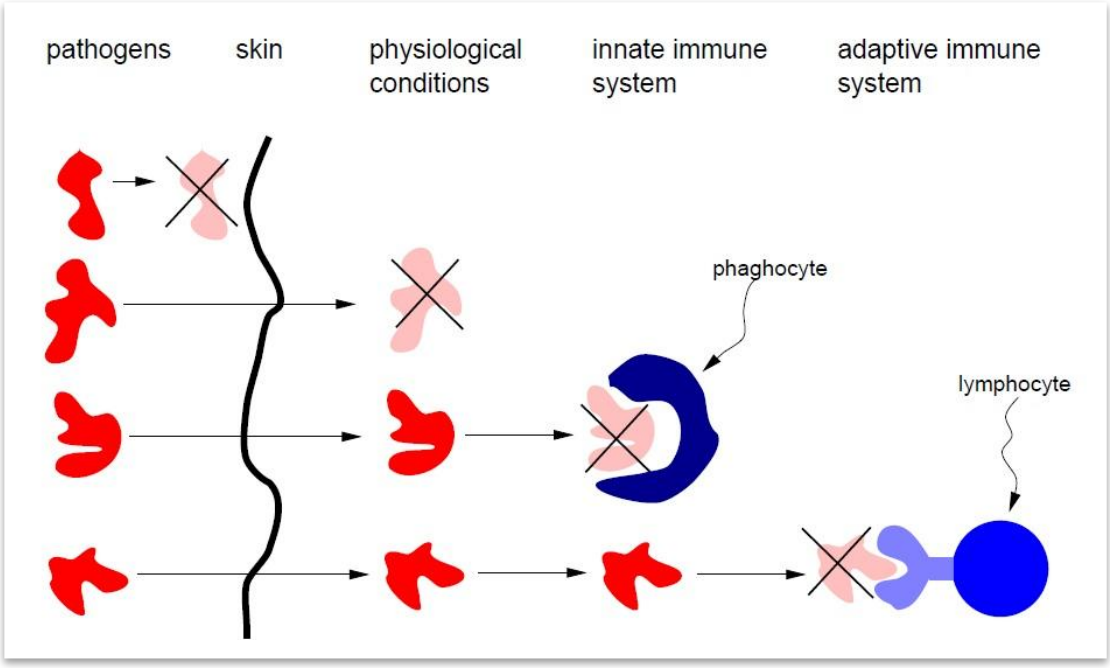


Figure 1-4. Multi-layered structure of the immune system to deal with external factors and pathogenic [7]

Before describing how immune system works in the body, it is needed to introduce and explain the basic concept of immune system.

1.1.1.4.1 Concepts of Basic Components in the Immune System

1.1.1.4.1.1 Hematopoietic Stem Cell

Both red and white blood cells originate from bone marrow as stem cells. Specialized name of these cells are Hematopoietic stem cells because they are source of all blood cells. Stem cells in bone marrow do not have any characteristics of adult blood cells, but they have the ability to divide. Typically when a cell is dividing, it splits into two identical cells. But when a stem cell divides, it divides non-symmetrically. One of the two cells produced is identical to the parent cell, and the other one separates from the parent stem cell and becomes a different cell. On the other hand, for a hematopoietic

stem cell, an offspring is exactly identical to its parent and the other one convert to another hematopoietic stem cell in a specific way [7] [8].

There are multipotent and pluripotent stem cells in bone marrows which are originated from the two ancestral cells: myeloid and lymphoid. The myeloid acts as a common precursor for granulocyte, monocyte, elytroid and megakaryocyte, while lymphoid cell creates T-cells and B-cells.

In addition to self-renewing stem cells and their offspring produced by them, bone marrow included plasma cells. The plasma cells are produced in secondary lymphoid tissues during antigen stimulation and then migrate to the bone marrow. These cells may continue living there and produce antibodies for years.

1.1.1.4.1.2 White Blood Cell[2]

There are cells in the body's immune system which fight against pathogens and their foreign substances. There are a variety of different white blood cell types, but all of them are made by a multipotent cell in the bone marrow which is known as Hematopoietic stem cell. White blood cells exist in the whole body including blood and lymphatic system.

1.1.1.4.1.3 B-Cell

B-cells produce antibodies and release them in blood and lymph stream, which are attached by other immune cells to the foreign antigen for identification and destruction [4]. Each B-cell with a genetic program is designed to provide a specific surface receptor

[2] Leukocyte

for a Specific antigen. B-cells are created from root cells in bone marrow and have their early stages of evolution there. Finally after completion of evolution, the new cell would be a healthy cell and does not react against the body's own cells and is removed from bone marrow and enters the body to identify and deal with the antigens. After the B-cells are activated, they are proliferated and matured into plasma cells.

1.1.1.4.1.4 T-Cell

T-cells are as guards for foreign invaders in the blood and lymph and they mark the antigens, attack and destroy the pathogen. T lymphocytes as well as intermediate cells are responsible for safety. They also work as coordinator and cooperator for all immune responses. T cells with unique cell surface molecules that are called MHC will help to identify the antigen parts [4].

There are several different types of T cells that have different functions such as helper T-cells and killer T-cells. A group of T-cells through interaction with single-core phagocyte help to destroy intracellular pathogens are called TH[3]. T cells interact with B cells to help them divide, differentiate and to produce antibodies. Another batch of cells is responsible for destroying those host cells that were infected by a virus or other intracellular pathogens. This type of operation is called cytotoxicity and these types of T-cells are called cytotoxicit. T-cells are produced in bone marrow and then evolve to mature and migrate to the thymus, which is a member of the body's lymph nodes. Symbolic forms of T-cells are shown in Figure 1-5.

[3] T-cell helper

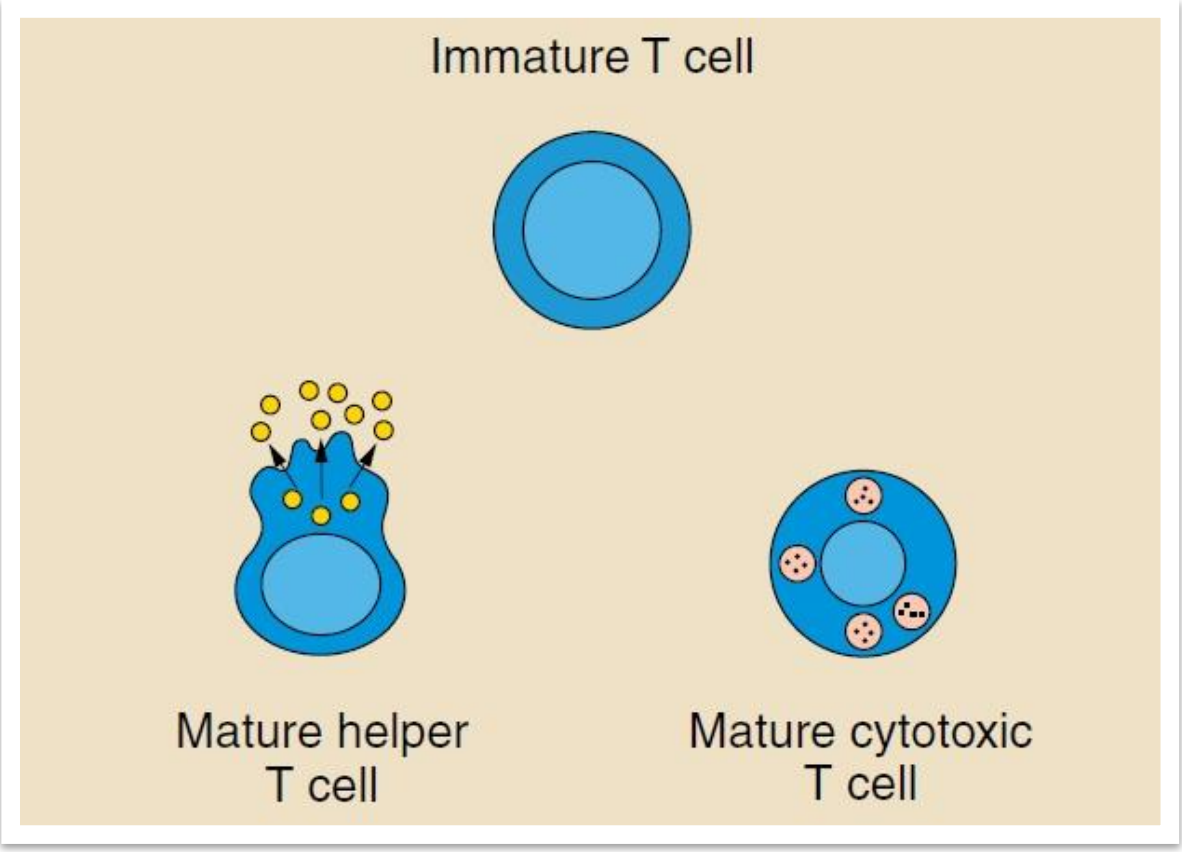


Figure 1-5. Mature the immature T-cells into mature helper T-cells and killer T-cell [4]

1.1.1.4.1.5 Plasma cell

After B-cells are activated, they are proliferated and matured into plasma cells. The number of plasma cells in the blood is low; in fact it is less than 0.1% of the cells in the lymph circulation. In normal conditions, Plasma cells are limited to secondary lymphoid tissues and members, but there are plenty of them in the bone marrow too. Produced antibodies by plasma cells belong to a specific group and immunoglobulin class. Most of the plasma cells have a short life and only live for a few days. The death of these cells occur duo to the apoptosis phenomenon. Maturing B-cells to plasma cells is shown in Figure 1-6.

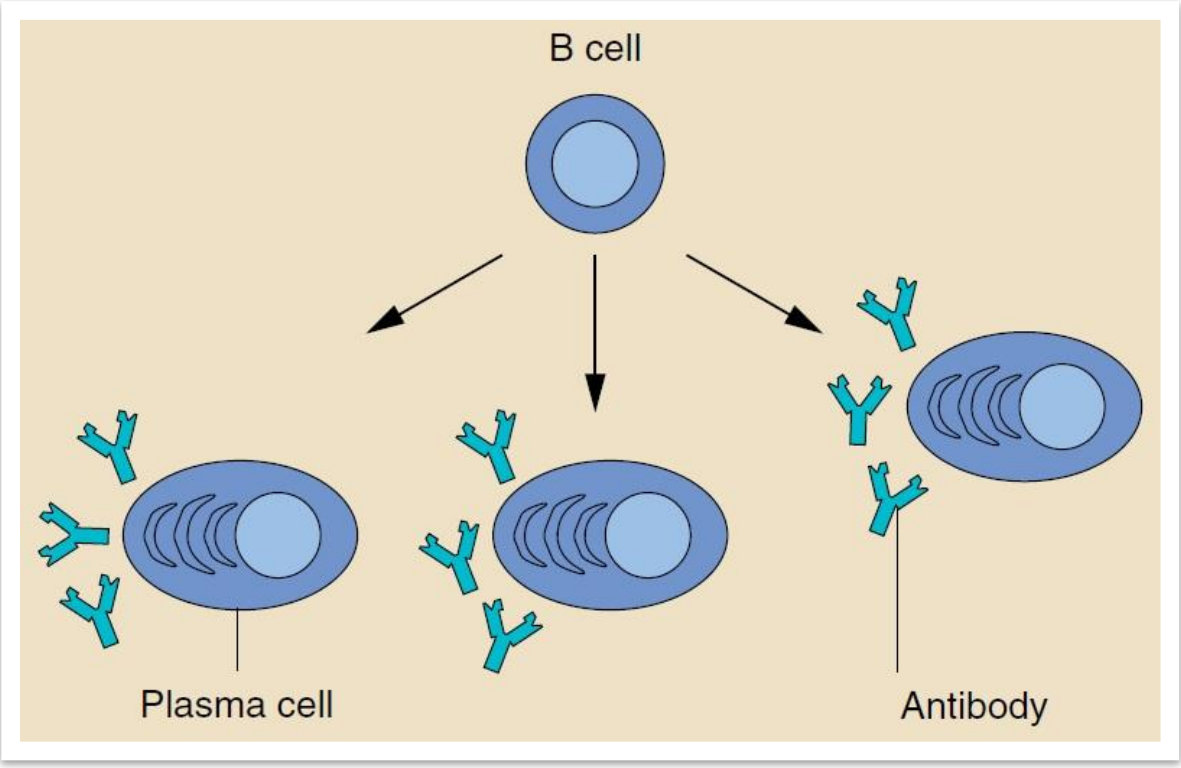


Figure 1-6. Maturing B-cell to plasma cell [4]

1.1.1.4.1.6 Antibody

After B cells are stimulated by antigen or T cells, they turn into plasma cells. In this stage, plasma cells make the large amounts of receptor molecules that are dischargeable. These molecules are known as antibodies.

Since antibodies are similar to the main receptor molecules, they connect to the antigens that activate B-cells in the beginning. Antibodies react to antigens and try to destroy and eliminate foreign invaders. Even other types of antibodies block the viruses to prevent them from entering the cells. The marker molecules that are carried by antigens and antibodies are shown in Figure 1-7.

1.1.1.4.1.7 Antigen

Antigen as a word means antibody generator, and at first was used to describe each molecule that stimulates B-cells to produce a specific antibody. But today usage of these words is much more widespread and covers every molecule that is identified specifically by components of the immune system such as B-cells, T-cells or both.

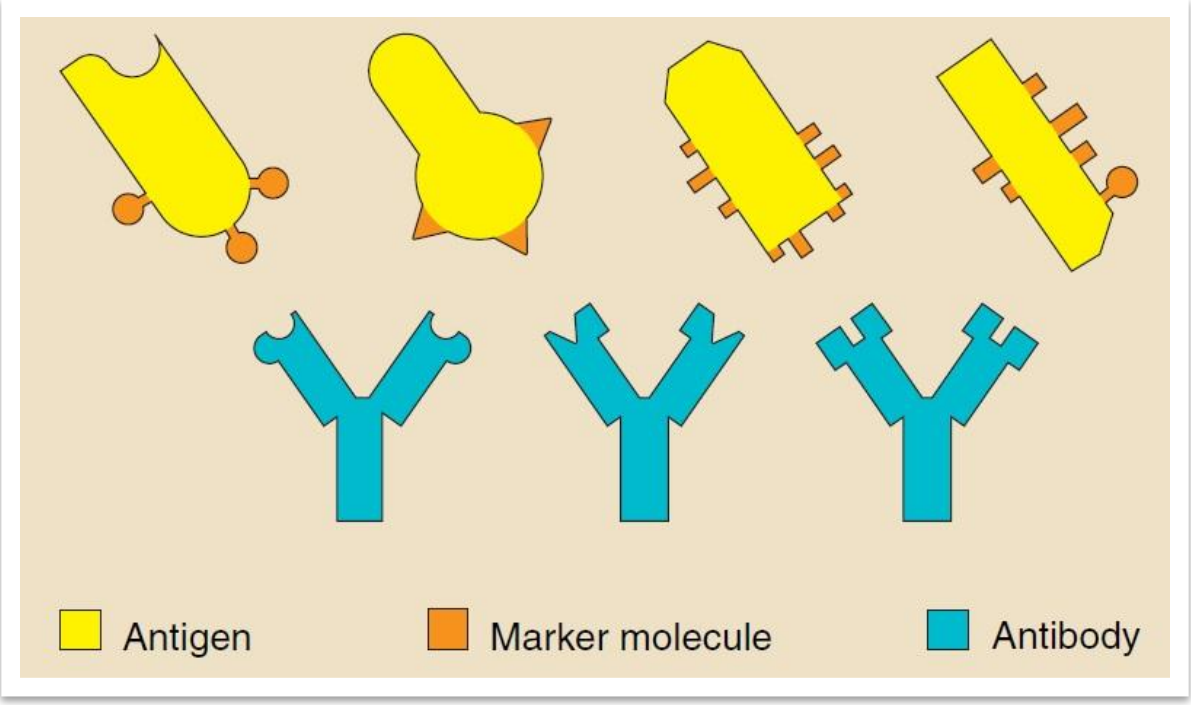


Figure 1-7. Antibody and Antigen's marker molecule to identify as foreign cell [4]

1.1.1.4.2 Immune System Processes

There are several important processes in the immune system which are explained briefly in this section.

1.1.1.4.2.1 Detection in Immune System

Immune cells can detect a pathogen or protein fragment when chemical bonds are established between receptors on the surface of an immune cell and epitopes, and

strength of the bond between these two is called the affinity. The high affinity means receptors of immune cell and epitopes have more similarity and vice versa.

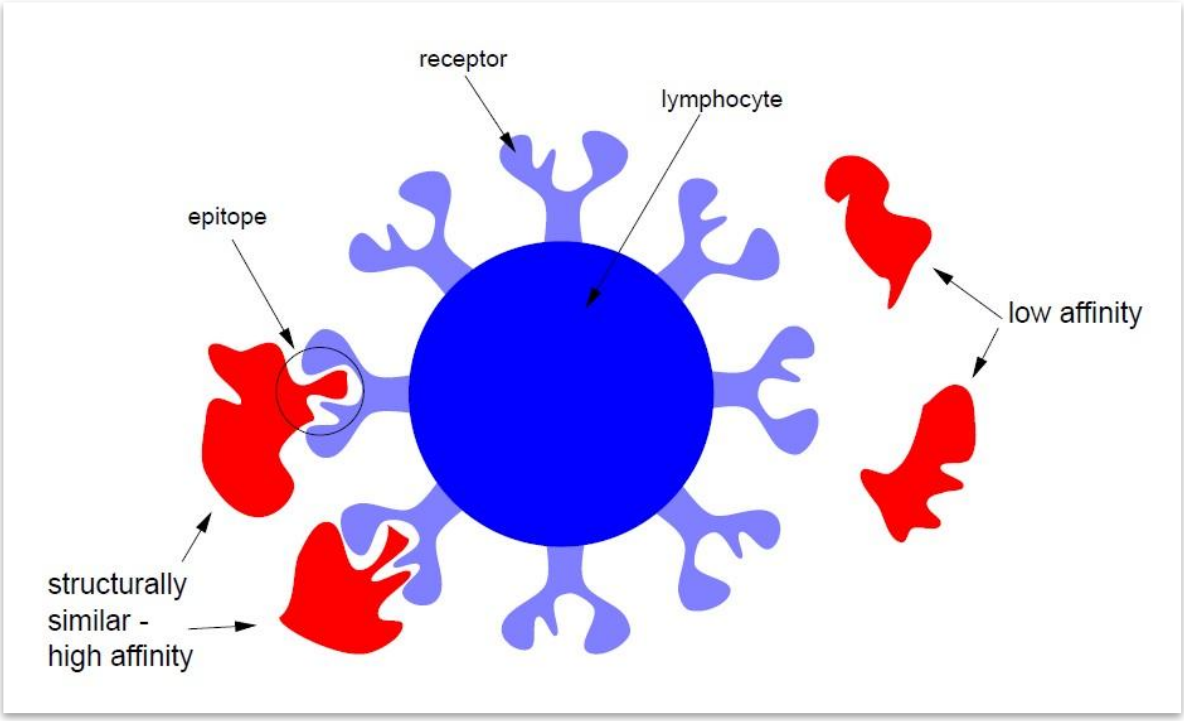


Figure 1-8. Detection in immune system [7]

As can be seen in Figure 1-8, "the pathogens on the left have epitope structures that are complementary **t**o the receptor structures and so the receptors have higher affinities for those epitopes than for the epitopes of the pathogens on the right, which are not complementary" [7].

1.1.1.4.2.2 Self-Cells Detection

In the immune system, to prevent the killing of self-cells, the body tries to eliminate B-cells and T-cells which detect self-cells as pathogens. This process occurs in the thymus. For example if TH-cell can detect a self-cell, it will participate in clonal deletion or negative selection and it will die in this process. This process is shown in Figure 1-9.

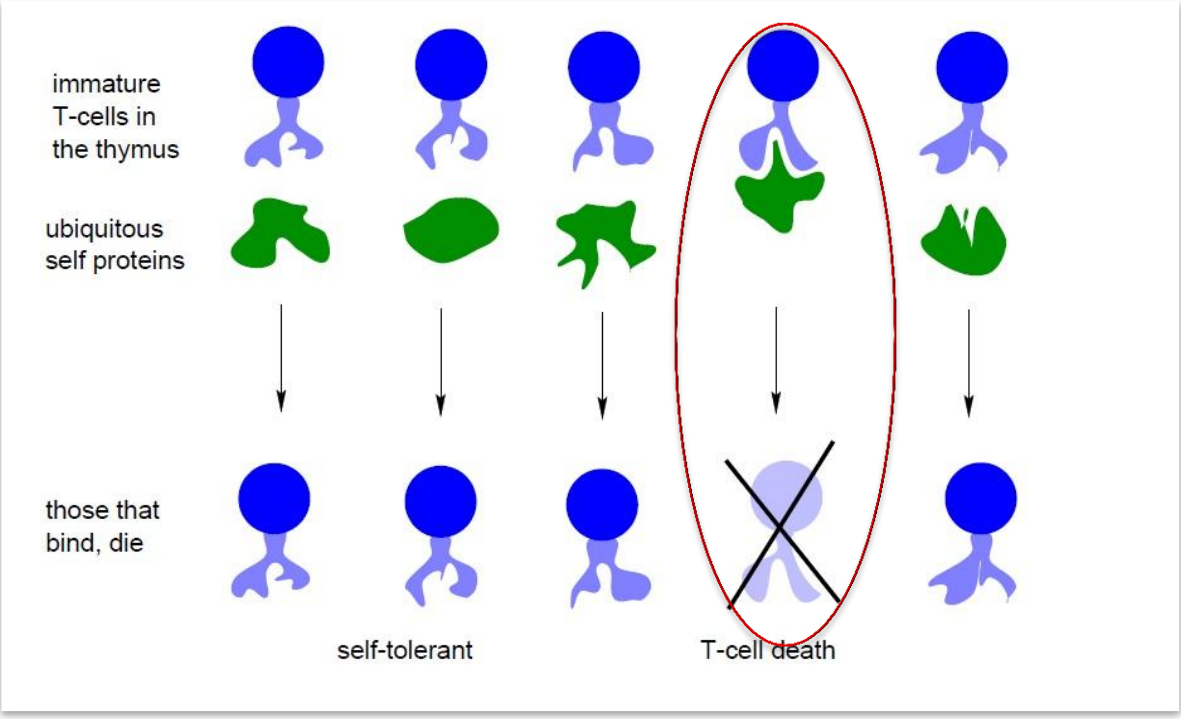


Figure 1-9. Negative selection [7]

1.1.1.4.2.3 Clonal Selection

The body tries to increase the diverse population of T-cells and B-cells and it does this by using clonal selection process. In this process, somatic hyper mutation helps the variation and also competition for pathogen epitopes helps the better selection. Immune system selects a B-cell or T-cell with high affinity and clones it.

It tries to keep the population diversity by mutation. The immune cells with higher affinity have more chance to get cloned to larger groups and vice versa. A sample clone selection and hyper mutation is shown in Figure 1-10.

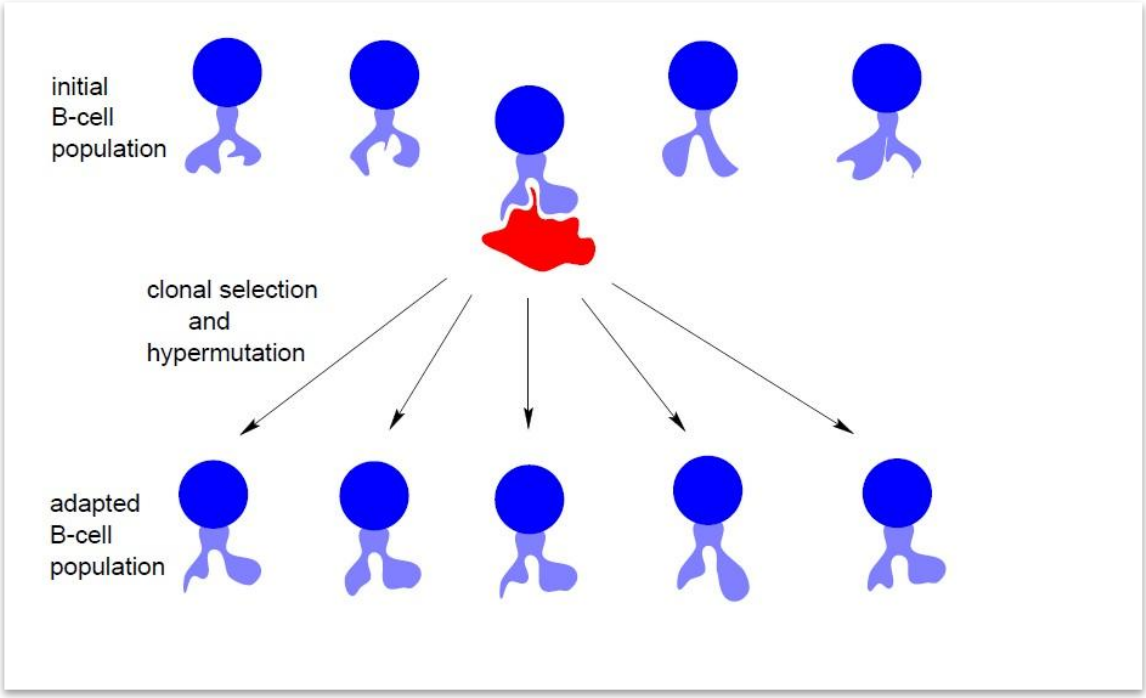


Figure 1-10. Clone selection [7]

1.1.1.4.2.4 Immunological Memory

In the immune system, the B-cells with higher affinity have more chance to proliferate. Thus, they live more than other B-cells in the body, which in this case they can be considered as a memory. These cells can respond to the pathogen faster than the original B-cells that require a longer process to be produced. It can help the body to reach the steady state very quickly.

In the Figure 1-11, you can see the affinity maturation. Activated B-cells have a proliferation process, and they produce mutated clones. The clones with highest affinity will survive and become memory cells or plasma cells.

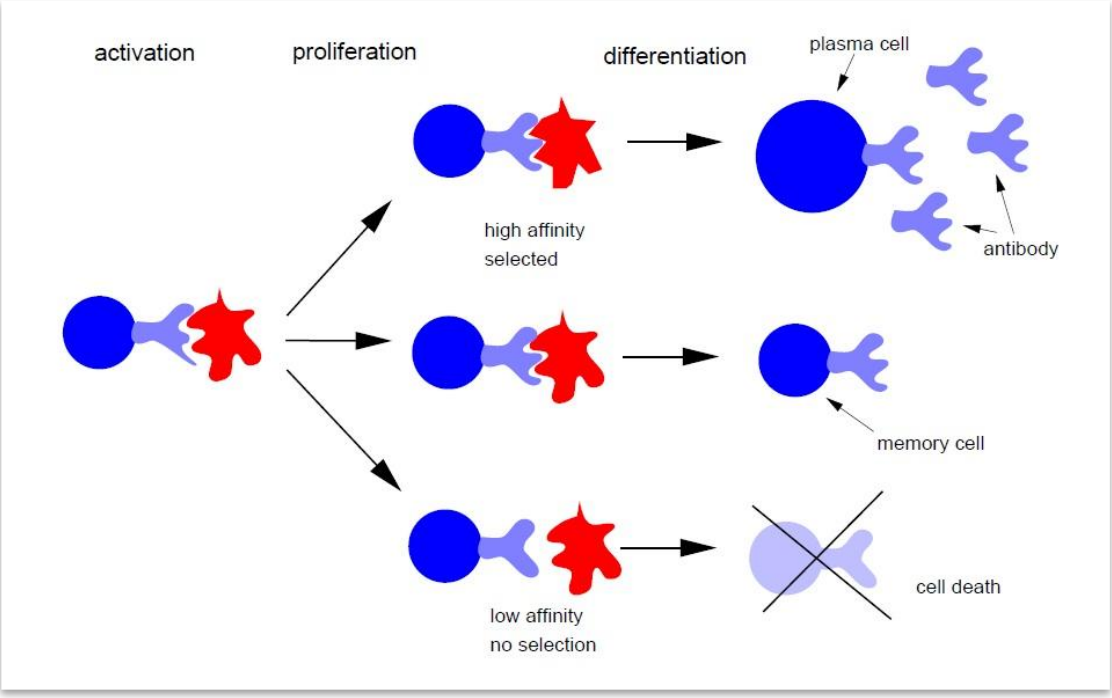


Figure 1-11. Process to produce B-cell memory [7]

1.1.1.5 The Primary Mechanism of Protection in Immune System

As discussed earlier, the human body is protected by immune cells in such way that all immune cells response to antigens which usually are foreign molecules of bacteria or other aggressor agents. The primary mechanism of defense in the immune system is shown in **Figure 1-12**.

APCs[4] like macrophages travel into the body and divide antigens to antigenic peptides by devouring them. Peptide fragments are joined to the MHC[5] protein and are exposed on the cell surface. Other cells like T-cells and T lymphocytes which have receptor molecules have the ability to detect and recognize various combinations of MHC and peptides [9].

[4] Antigen presenting cells
[5] Major histocompatibility complex

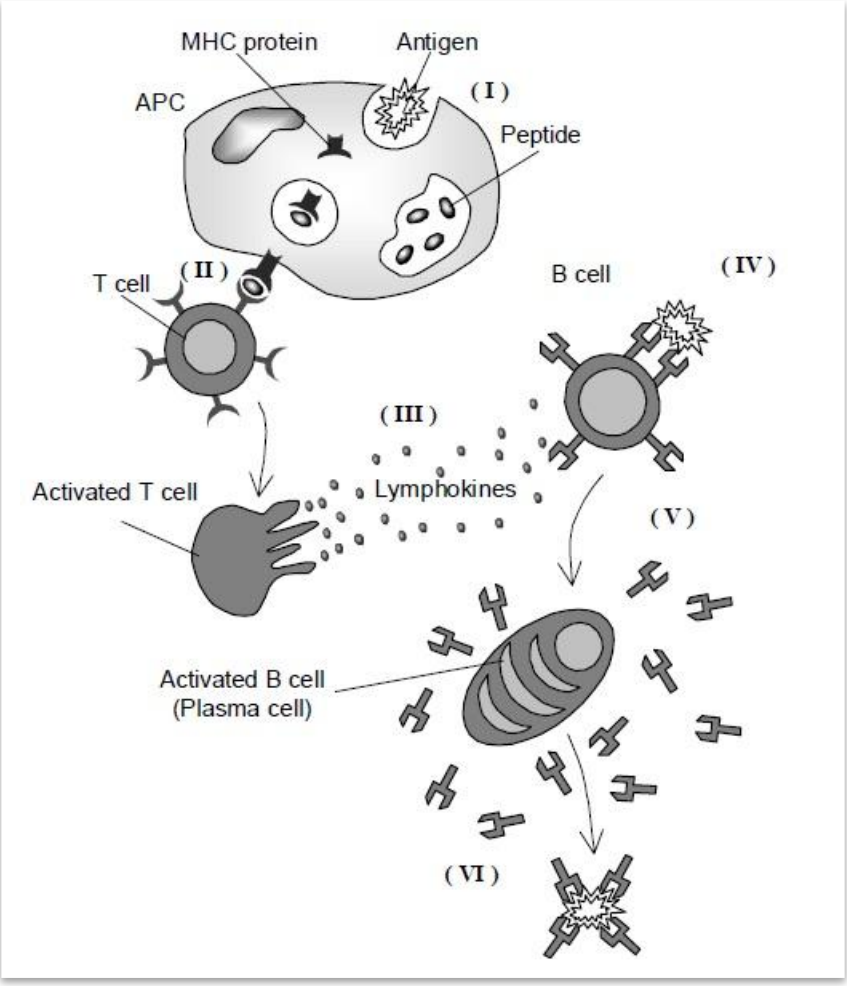


Figure 1-12. The primary mechanism of defense in the immune system **[10]**

T-cells are activated by divided recognition and discharged lymphokines or chemical signals and prepare other components of the immune system for action.

B-lymphocytes have receptor molecules on their surface too, which respond to signals. Unlike T-cells, B-cells can identify some parts of free antigens without MHC molecules.

1.1.1.6 Immune Network Theory

Immune network theory for the first time was proposed in 1974 by Jerne [11]. The part of the antigen that is detected by the antibody is epitope, and the part of antibody molecules whose job is epitope detection called paratope.

Also the part of the antibody with the antigen characteristics is named idiotope. A model of idiotopes which is located in the antibody polypeptide chains region is detected by paratope. In each branch of antibody, there is a paratope and small collection of idiotopes, that are called idiotype [12].

Immune system is defined as a complex network of paratopes that identify a set of idiotopes and a set of idiotopes which are identified by paratopes. Therefore, each element of network has the capability of identification and recognition at the same time. This property creates an immune network in which antibody molecules are connected to each other as free molecules or B-cell receptor molecules.

After an antibody identified an epitope or an idiotope, it can respond positive or negative to this identification signal. The result of a positive response is cell activation, cell proliferation and antibody secretion.

On the other hand, the negative response leads to tolerance or suppression in the cell. Figure 1-13 shows the process of activation and suppression in antibodies chain.

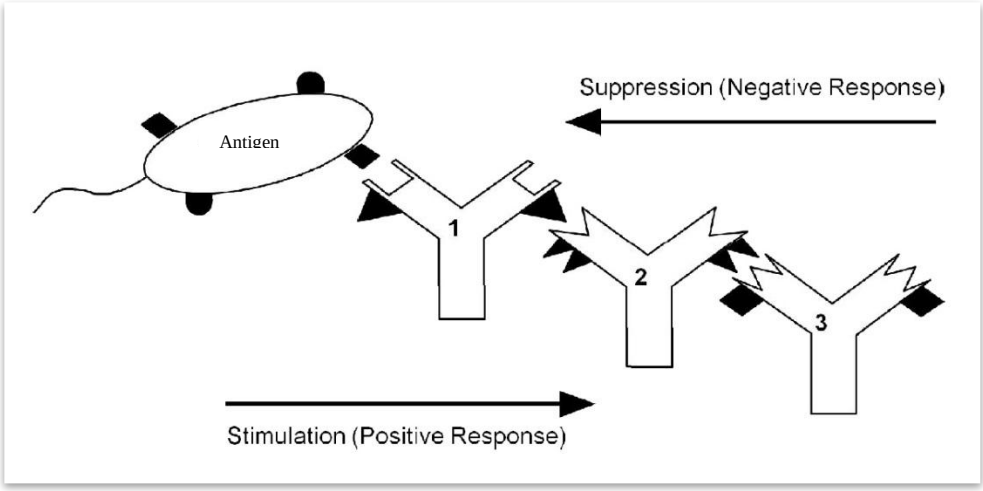


Figure 1-13. Activation and suppression in antibody and antigen [1]

1.1.2 Artificial Immune System (AIS)

Artificial immune system is one of the branches of computer science that is inspired by the immune system of organisms and is presented by different algorithms to solve various problems in computer science. The immune system has different levels. At the first level, it prevents foreign organisms or antigens from entering the body using skin abilities, tears and similar strategies. The second level is the innate immune system where all antigens are treated in the same general way. This level of the immune system works very slowly and is not sufficient to deal with antigens. Adaptive immunity is the next level and in which, for dealing each antigen an appropriate method is applied. This immune level works very fast and can produce large numbers of immune cells to deal with antigens.

Algorithms that are designed in artificial immune system are mostly modeled after the adaptive immune system and these algorithms have been used to solve wide range of computer problems. Artificial immune system algorithms are divided into several

groups: negative selection, clonal selection, danger theory, immune network and B-cell Algorithm. Each one of them is inspired by a different natural immune system. Up to now, these algorithms are used to solve optimization problems, pattern recognition, classification, clustering, network security and other issues of computer science and have obtained good results compared to existing algorithms. The immune system can be a parallel and distributed adaptive system that has been inspired by immunology of natural processes .

1.1.2.1 Artificial Immune System Algorithms

Artificial immune system is one of subsets of methods that are inspired by biological sciences, which itself is a subset of computational intelligence. A brief hierarchical view of computational intelligence and its subsets are shown in **Figure 1-14**.

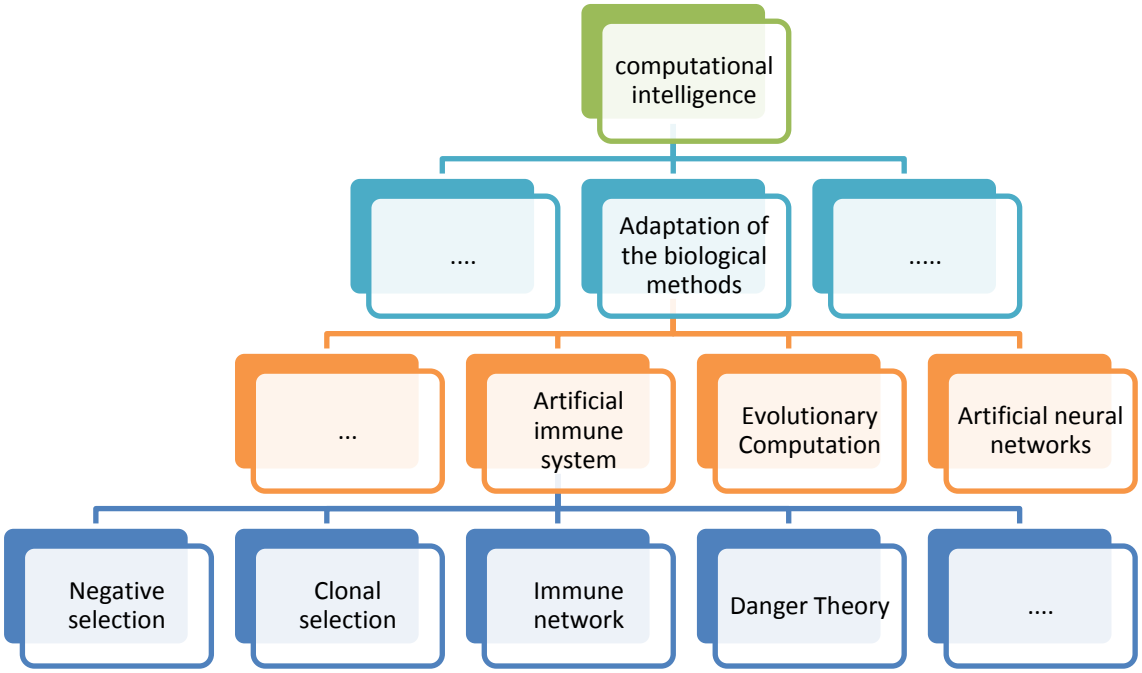


Figure 1-14. AIS position in computational intelligence hierarchy **[13]**

Various applications of artificial immune system are mentioned in many articles and journals in which they are used to solve problems related to the hybrid structures and algorithms with similar mechanisms to immune system [14] [15]. Some computational algorithms based on principles of immune are clonal selection, immune network theory, learning, self-organization, artificial life, cognitive models, multifactorial systems, design and scheduling, pattern recognition, anomaly detection Immune engineering tools and etc. In the following section, some of the most famous algorithms in immune system are briefly explained.

1.1.2.1.1 B-Cell Algorithm (BCA)

The B-cell algorithm is one of naïve and basic algorithms in artificial immune system. There are many researches and articles in which it has been tried to improve this algorithm. The main idea of this algorithm is parallel search in such way that each member of the population (each B-cell) searches its own neighborhood in the search area [16]. The basic B-cell algorithm is shown in **Figure 1-15**.

B-cell algorithm

input : g(v) = function to be optimized
output: P = set of solutions for function
begin
1. Create an initial population P of individuals in shape-space
2. For each v ϵ P, evaluate g(v) and create clone population C with n solutions
3. Select a random member of v'ϵ C and apply the contiguous region hypermutation operator
4. Evaluate g(v'); if g(v') > g(v) then replace v by clone v'
5. Repeat steps 2-4 until stopping criterion is met
end

Figure 1-15. Pseudo-code of the basic B-cell algorithm **[3]**

1.1.2.1.2 Artificial Immune Network Algorithm (Ainet)

The immune network algorithm is one the most famous algorithm in AIS, and there are many versions of this algorithm. The version given before is called the Optimization Artificial Immune Network algorithm (opt-aiNet). Signaling information processing properties and resource maintenance make fundamental of the immune network theory which proposes an additional order of complexity between the cells and molecules under selection. The purpose of the immune network process is preparing a set of distinct pattern detectors for problems domain in such a way that cells similarity (low affinity) means better performing in network [17]. The basic pseudo code of this algorithm is shown in **Figure 1-16**.

Immune Network Algorithm

```
input : N = a set of random detectors, n = number of best
antibodies
output: M = set of generated detectors capable of finding solution
begin
    1. Create an initial random population B
    2. For each solution in B
    2.1 Determine inverse distance for solution in B to each member of N
    2.2 Select n members of B with the highest affinity
    2.3 Clone and mutate each n in proportion
    2.4 Retain the highest affinity of n and place in a set M
    2.5 Perform network dynamics in M to remove weak members of M
    2.6 Generate b random elements and place in B
    3 repeat
end
```

Figure 1-16 Pseudo-code of the basic Artificial immune network algorithm **[3]**

1.1.2.1.3 Clonal Selection Algorithm (CLONALG)

The main idea of clonal selection is inspired by the basic features of natural immune system that response to an antigenic stimulus. The idea is to just select and duplicate

those cells that can recognize antigen [2]. Some of the most important features of this algorithm are:

- Selection and proliferation of cells with the highest level of stimulation (high affinity)
- Elimination of cells with the lowest level of stimulation (low affinity)
- Dependence evolution and selection of the cells, proportional to antigen stimulus
- Save the cells with highest level of stimulation for longer time as memory cell

The main difference between clonal selection and other evaluation algorithms is mutation and how it is being done. The mutation rate is proportional to inverse of the affinity. It means that cells with high affinity have low mutation rate and cells with low affinity are mutated by a higher rate. In both groups of algorithms, the population should be encoded using binary or real numbers similar to evolutionary algorithms, but in clonal selection the binary coding is usually used. The general structure of the clonal selection algorithm is shown in **Figure 1-17**.

Clonal selection Algorithm

1. Fitness evaluation: For each antibody x_i in the population P compute its fitness $f(x_i)$

2. Clonal selection: Choose a reference set $P_a \subset P$ consisting of h antibodies with highest fitness to the antigen

3. Somatic hypermutation.

3.1 For each antibody $x_j \in P_a$ make mutated clones $xc_{j,k}$, k = 1,…,c, compute their fitness and place them in clonal pool C.

3.2 Choose a subset $P_c \subset (P \cup C)$ containing $|P|$ fitness antibodies.

4. Apoptosis: Replace d worst antibodies in P_c by randomly generated solutions.

5. Set $P = P_c$

Figure 1-17. Pseudo-code of Clonal selection algorithm **[18]**

1.1.2.1.4 Negative Selection Algorithm (NSA)

Due to the success achieved by using other artificial immune systems algorithms, researchers were attracted to another aspect of the immune system that does negative selection in the process of the maturing the T-cells. In negative selection process, immature and inappropriate T-cells that are attached to self-cells are eliminated. This process helps the immune system to recognize self from non-self-antigens without any mistakes. Since the process needs to introduce the detection of the specific harmful cells, it allows the immune system to identify previously unseen harmful cells. This algorithm includes 3 levels: self-detection, producing finders and investigating the occurrence of abnormal events. In first level, self-cells are detected by similar methods to other detection methods, and some patterns equal to these are created which are called self-patterns. In second level, some random patterns are produced that have been compared to the self-patterns which were produced in the first level. If a produced random pattern matches with self-pattern, this pattern fails in the process of becoming a detector, so it should be destroyed. Otherwise, it becomes a pattern detector and monitors specified patterns which have been seen in the system before. Some of the important features of negative selection algorithm are:

- Detection of previously unknown attacks
- Probable detection and adjustable
- The inherent distribution detection
- Local detection
- Unique set of detection
- Protection of self and non-self detectors sets

The general pseudo code of negative selection algorithm is shown in **Figure 1-18**.

Negative Selection Algorithm

input : S = set of self-cells
output: D = set of generated detectors
begin
1. Define self as a set S of elements in shape–space $\sum^L$
2. Generate a set D of detectors, such that each fails to match any element in S.
3. Monitor data $\delta \subseteq \sum^L$ by continually matching the detectors in D against δ. If any detector matches with δ, classify _ as a non-self, else as self.

end

Figure 1-18. Pseudo-code of Negative selection algorithm **[3]**

1.1.2.2 Features of Artificial Immune Algorithms

From the perspective of information processing, immune system is a parallel and distributed adaptive system. Immune algorithm has been inspired by theoretical immunology and several processes that occur in it. In general, Immune algorithms use learning, memory and associative retrieval to solve problems and identify related patterns. The immune algorithms have many properties and some of the most important are [10]:

- Uniqueness: each individual has its own immune system independently proportional to its own vulnerabilities and ability
- Recognition of foreigners: the immune system detects and destroy the (harmful) molecules
- Anomaly detection: recognition and response to pathogens that the body has never encountered before
- Distributed detection: cell distribution in the whole body
- Imperfect detection (noise tolerance)
- Reinforcement learning and memory: fast and powerful react to the pathogens which was detected before

We can name versatile, feature extraction, dedicated, self-tolerance, resolution as other properties. Some of the important characteristics in immune system are explained below.

1.1.2.2.1 Mutation

Mutation in the genetic algorithm is used to avoid premature convergence by lost or unseen particle recovery solutions. In immune's algorithm, mutation occurs randomly depending on affinity rate between antibody and antigen. Population with high affinity has lower rate of mutation, but on the other hand, population with low affinity has higher rate of mutation [19]. This strategy tries to search near neighbors individually with higher affinity which is opposite of what happened to individuals with lower affinity. In this situation, the system tries to search in bigger space to probably find a better solution.

The small amount of mutation rate in genetic algorithms is essential and successful and creates diversity in the crossover. The mutation as the only mechanism in immune system that can create diversity in population has a very essential rule. It should create a new population in such way that can search domain space in two cases. In first scenario, it should search near neighbors individually and cover local searches and this happens by using a low rate of mutation. In second scenario, the system should search in a bigger area in domain space which helps to escape from local optima and find the global optimum. In this process it is necessary to protect the most qualified individual (the elite) in cycle. This rate is decreased until it reaches zero and the process of retaining the most qualified individual reaches to its maximum.

There are a lot of strategies for using mutation. Some of the algorithms use binary encoding to present the population. In this case, bits filliping is the mutation technique,

but in various ways. Examples are single or multi point mutations. In the real value string presentation, one or more values can change randomly or the order of the elements is swapped. Additionally, some algorithms create numbers in different ways and are randomly added to or subtracted from the original value [20].

1.1.2.2.2 Adaptive Population Size

In some algorithms, the population size depends on the selection strategy and its calculated intervals can change. If the process of evolution is considered, the population size should be proportional to the process. Boundaries of population size is set in such a way that prevents the stability and keeps diversity of the population by selecting random number of populations to eliminate and create new individuals.

1.1.2.2.3 Secondary Response

Some algorithm use memory to save optimality reports and relative results and when the function is placed in a similar situation, the memory is retrieved for faster and better response.

1.1.2.2.4 Termination Criteria

Termination criteria can be a condition or combination of many conditions that have been considered on many programs and they cause the programs to stop. Some of the most used conditions as termination criteria are:

- Desired accuracy
- Maximum amount of generation (iteration)
- The best individual in current generation remains and duplicate in the next generations
- Population size

1.1.2.3 Problem Environments

There are 2 kinds of problems in computer science which are grouped based on their problem spaces: static environment and dynamic environment. Most of the problems in computer science are static problems. In static environments, domain space and problem solutions stay unchanged. So the agent does not need to adapt to new situations. On the other hand, dynamic environments are more complex and need some strategies to deal with them. A brief review of dynamic environments is explained in below.

1.1.2.3.1 Dynamic Environment

Most real-world problems are changed dynamically over time. As example of these cases it can be pointed to the problems of scheduling or routing that during time where new tasks may be added or subtracted. Because of the characteristics of evolutionary algorithms their advantages and essence have been derived from nature and is used widely to solve optimization problems in uncertain environments. Normal optimization algorithms that are designed for static optimization problems are not able to detect environmental changes and cannot perform successfully. To have a suitable respond in dynamic environments, environment change detection and response to this changes are essential [21].

1.1.2.3.1.1 Detecting Changes in Dynamic Environments

One of the most common methods for detecting changes in dynamic environments is using a point in the environment as a guard. In this case the value of the function at that point is calculated in each cycle and is compared to the function value of the same point in previous iteration. In this case, different values show changes in the environment [22]. But according to the environmental changes that may occur only in the vicinity of the guard, it sometimes may be unable to detect changes. Therefore instead of using a

fixed point in the environment, using several random points as guards is suggested. Certainly accuracy of reorganization will depend on the number and position of the guards [23].

There is another method which suggests monitor changes of the best value of the function. In this method, the global optimum in each period is calculated and compared with the previous period. In this case, difference between the values shows the changes in the environment [24].

1.1.2.3.1.2 Response to Changes in Dynamic Environments

There are a lot of methods that suggest how to deal with the environment changes. To deal with dynamic changes in the environment, one method is removing and replacing all previous values with the new values and look at the new environment as new problem. Some methods have proposed various random replacing such as reproducing some part of the population randomly, initializing the best solutions randomly, and reinitializing the whole of population. Between these methods, reinitializing 10% of the population has good results [24].

Other mechanisms that have been discussed in [25] and [26] use a selection mechanism. It means the whole or some part of the population is generated by the current population which can consider different mechanisms based on elitism. Of other approaches that have been used in various references that can be named are: mechanisms to create diversity in the population [27], using of memory [27] [28], the combination of diversification and memory [29], multi population [30] [31], immigration methods [31] [32] and changes in the parameters of the algorithm [33].

Chapter 2

DYNAMIC OPTIMIZATION PROBLEMS

2.1 Problem Description

To evaluate an algorithm, the most important element is the problem or problems that algorithm tries to solve it. Some of them are static and some of them are dynamic. But in real world, most of the problems are dynamic. It means solution space, objective function, decision variable, and constraint may change during the time. It makes the problem more complex, so it needs different strategies to solve it. A part of these problems are dynamic optimization problems (DOPs). Many researchers have tried to propose some dynamic problems to test and compare their algorithms to each other. Some of the most important test functions are: “the moving peaks benchmark (MPB)” [34] [35], “the DF1 generator” [36], “the single and multi-objective dynamic test problem generator by dynamically combining different objective functions of exiting stationary multi-objective benchmark problems” [37], “exclusive-or (XOR) operator” [38] [39] [31], “dynamic traveling salesman problem (DTSP)” [40] and dynamic multi knapsack problem (DKP) , etc.

To evaluate proposed algorithm, 2 test environments were used and both of them are based on moving peaks benchmark. There is a brief explanation of both of them in section 2.1.1and 2.1.2.

2.1.1 The Moving Peaks Benchmark

In 1999, Jürgen Branke proposed a multidimensional fitness function which consists of several peaks and they change during the time. This fitness function is called "The Moving Peaks Benchmark" [35]. In this benchmark function, locations, heights, width of peaks are changing over time. But these changes happened in 2 models: in one of them, "the optimum shifts slightly" can be found by a local search but on the other one "the height of the peaks changes such that a different peak becomes the maximum peak" which in this case the algorithm should jump to reach to the new optimum. [35]. The function's formula with *n* dimensions and *m* peaks is:

$$\boldsymbol{F}(\vec{\boldsymbol{x}}, \boldsymbol{t}) = \mathbf{max}(\mathbf{B}(\vec{\boldsymbol{x}}), \max_{\boldsymbol{i-1\ldots m}} \boldsymbol{P}(\vec{\boldsymbol{x}}, \boldsymbol{h_i(t)}, \boldsymbol{w_i(t)}, \overrightarrow{\boldsymbol{p_i}}(\boldsymbol{t}))) \qquad (1)$$

Where $\vec{x}\epsilon\mathbb{R}$, $\vec{x} = \{x_1, x_2, \ldots, x_n\}$, t is time, B($\vec{x}$)a time constant, and P is peak's shape. Each peak has height ($h(t)$), width ($w(t)$), and location ($\overrightarrow{p_i}(t)$) which all of the peaks are initialized randomly. To change peaks, new coordinates of each peak is calculated by the following formulas and for λ=0 the changing direction is random and for λ>0 direction depends on the previous direction :

$$\begin{gathered} \boldsymbol{\sigma} \in \boldsymbol{N}(\mathbf{0}, \mathbf{1}) \\ \boldsymbol{h_i(t)} = \boldsymbol{h_i(t-1)} + \boldsymbol{height\_severity} \,.\, \boldsymbol{\sigma} \\ \boldsymbol{w_i(t)} = \boldsymbol{w_i(t-1)} + \boldsymbol{widht\_severity} \,.\, \boldsymbol{\sigma} \\ \overrightarrow{\boldsymbol{p_i}}(\boldsymbol{t}) = \overrightarrow{\boldsymbol{p_i}}(\boldsymbol{t-1}) + \overrightarrow{\boldsymbol{v_i}}(\boldsymbol{t}) \end{gathered} \qquad (2)$$

Where $height\_severity$ and $widht\_severity$ are initialized by the program it shows the severity of height and width changes. $\overrightarrow{v_i}(t)$ is a shift vector and is a linear combination of $\vec{\boldsymbol{r}}$ that is random vector and normalized to length $\boldsymbol{s}$. It is computed by:

$$\overrightarrow{\boldsymbol{v_i}}(\boldsymbol{t}) = \frac{\boldsymbol{s}}{|\vec{\boldsymbol{r}} + \overrightarrow{\boldsymbol{v_i}}(\boldsymbol{t-1})|}\left((\mathbf{1}-\boldsymbol{\lambda})\vec{\boldsymbol{r}} + \boldsymbol{\lambda}\overrightarrow{\boldsymbol{v_i}}(\boldsymbol{t-1})\right) \qquad (3)$$

2.1.2 Generalized Dynamic Benchmark Generator (GDBG) [41]

The GDBG is a test function based on moving peaks benchmark. This function is more complicated than the original function. This function uses rotation method instead of shifting methods to move peaks and change the environment peaks. The GDBG includes 6 different functions to create peaks and each of those function change in 6 different scenarios [41]. In continue, it is tried to give a short review of this test functions and the framework of dynamic changes.

2.1.2.1 Dynamic Changes

Dynamic optimization problems can be defined as:

$$\boldsymbol{F = f(x, \emptyset, t)} \tag{4}$$

Consider *F* as optimization function which is tried to solve and *f* is fitness function. *t* represents the time and feasible solutions are represented as *x*. The solution distributions are determined by system control that here is $\emptyset$. For dynamic change, a new environment can be gained by the formula:

$$\boldsymbol{f(x, \emptyset, t+1) = f(x, \emptyset(t) \oplus \Delta\emptyset, t)} \tag{5}$$

In this equation, $\boldsymbol{\Delta\emptyset}$ is a deviation from the current system control' parameters.

As said before, environment and peaks are changed in 6 different scenarios. These scenarios are "small step change, large step change, random change, chaotic change, recurrent change and recurrent change with noise" [41].

These six change methods are explained in following equations:

T1 (Small change step):

$$\boldsymbol{\Delta\emptyset} = \propto . \|\emptyset\| . \boldsymbol{r} . \emptyset_{\boldsymbol{severity}} \tag{6}$$

T2 (Large change step):

$$\boldsymbol{\Delta\emptyset} = \|\emptyset\|(\propto . \boldsymbol{sign}(\boldsymbol{r}) + (\propto_{\boldsymbol{max}} - \propto) . \boldsymbol{r}) . \emptyset_{\boldsymbol{severity}} \tag{7}$$

T3 (Random change):

$$\boldsymbol{\Delta\emptyset} = \boldsymbol{N}(\mathbf{0}, \mathbf{1}) . \emptyset_{\boldsymbol{severity}} \tag{8}$$

T4 (Chaotic change):

$$\emptyset(\boldsymbol{t} + \mathbf{1}) = \boldsymbol{A} . (\emptyset(\boldsymbol{t}) - \emptyset_{\boldsymbol{min}}) . (\mathbf{1} - (\emptyset(\boldsymbol{t}) - \emptyset_{\boldsymbol{min}})/\|\emptyset\|) \tag{9}$$

T5 (Recurrent change):

$$\emptyset(\boldsymbol{t} + \mathbf{1}) = \emptyset_{\boldsymbol{min}} + \|\emptyset\| \left(\mathbf{sin}\left(\frac{\mathbf{2}\boldsymbol{\pi}}{\boldsymbol{P}}\boldsymbol{t} + \boldsymbol{\varphi}\right) + \mathbf{1}\right)/\mathbf{2} \tag{10}$$

T6 (Recurrent change with noisy):

$$\emptyset(t + 1) = \emptyset_{min} + \|\emptyset\| \left(\sin\left(\frac{2\pi}{P}t + \varphi\right) + 1\right)/2 + N(0,1) . noisy_{severity} \tag{11}$$

"Where $\|\emptyset\|$ the change range of is $\emptyset$, $\emptyset_{severity}$ is a constant number that indicates change severity of $\emptyset$, $\emptyset_{min}$ is the minimum value of $\emptyset$, $noisy_{severity} \in (0, 1)$ is noisy severity in recurrent with noisy change. $\propto \in (0,1)$ and $\propto_{max} \in (0,1)$ are constant values, which are set to 0.04 and 0.1 in the GDBG system. A logistics function is used in the chaotic change type, where *A* is a positive constant between (1.0, 4.0), if $\emptyset$ is a vector, the initial values of the items in $\emptyset$ should be different within $\|\emptyset\|$ in chaotic change. *P* is the period of recurrent change and recurrent change with noise, φ is the initial phase, *r* is a random number in (-1,1), *sign*(*x*) returns 1 when x is greater than 0, returns -1 when x is less than 0, otherwise, returns 0. *N*(0,1) denotes a normally distributed one dimensional random number with mean zero and standard deviation one" [41].

2.1.2.2 Functions Definition

As said before, GDBG is based on moving peaks benchmark, but the problem space and peaks are created by 5 basic functions. In all test functions some fixed parameters were used that was defined by the competition manager. In some case duo to some reasons we had to change them. Dimension that is showed by *n* are fixed in all functions and it equals to 10. Just in rotation peak function, the algorithm was tested with 50 peaks also. In all functions the range of search space is between -5 to 5 in each dimension ($x \in [-5\,, 5]^n$). The original environment change frequency is 10,000 times of dimensions ($change\_freq = 10{,}000 * n$) but in our test, we decrease the change frequency rate to 5,000 times of *n*. [41].

For environment changes step severity (α) is equal to 0.04 and maximum (α_{max}) value would be 0.1. In chaotic scenario, the chaotic constant was considered *A=3.67*. Also in

recurrent change with noisy case, *noisy* $_{severity}$ is 0.8. The description of each function is explained below:

2.1.2.2.1 Rotation Peak Function

This function is exactly same with moving peaks benchmark, but in this version, it uses rotation method instead of shifting methods to move peaks. This is a multi-model function that has the ability to scale. The function controls number of peaks artificially. In our test program, this function is used in 2 situations, one for 10 peaks and another one for 50 peaks. The width ranges from 1 to 10 ($w \in [1, 10]$), and width severity is equal to 0.5 ($\emptyset_{w_{severity}} = 0.5$). The global optimum in each period is calculated by bellow formulas:

$$\boldsymbol{x^*(t) = \overrightarrow{O_i}, F(x^*(t)) = H_i(t)}$$
$$\boldsymbol{H_i(t) = max_j^m H_j} \quad (12)$$

A 3D view of a sample of this function is shown in Figure 2-1.

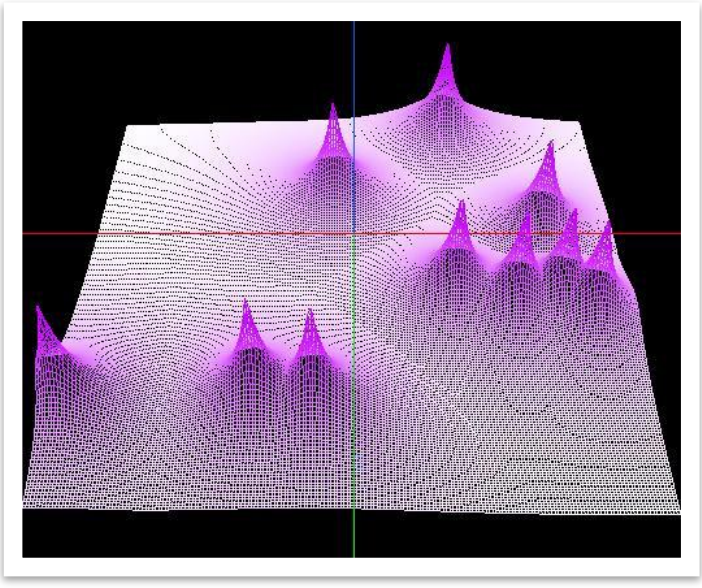


Figure 2-1. 3D perspective of moving peak benchmark **[41]**

2.1.2.2.2 Composition of Sphere's Function

Sphere is one of the most famous functions in mathematics. This function is very interesting in computer science and it has been used to evaluate many algorithms till day. This function creates a round object in 3 dimensions. The function is:

$$\boldsymbol{f(x) = \sum_{i=1}^{n} x_i^2} \qquad \boldsymbol{x \in [-100, 100]} \tag{13}$$

In GDBG function the Sphere's function was used as a basic function to create the environment while $x \in [-5,5]^n$. This function has 10 local optimums. The global optimum can be obtained from these formulas:

$$\boldsymbol{x^*(t) = \overrightarrow{O_i}, F(x^*(t)) = H_i(t)}$$
$$\boldsymbol{H_i(t) = min_j^m H_j} \tag{14}$$

The basic function is: f_1- f_{10} = Sphere's function [41]

A 3D map of this function is shown in Figure 2-2.

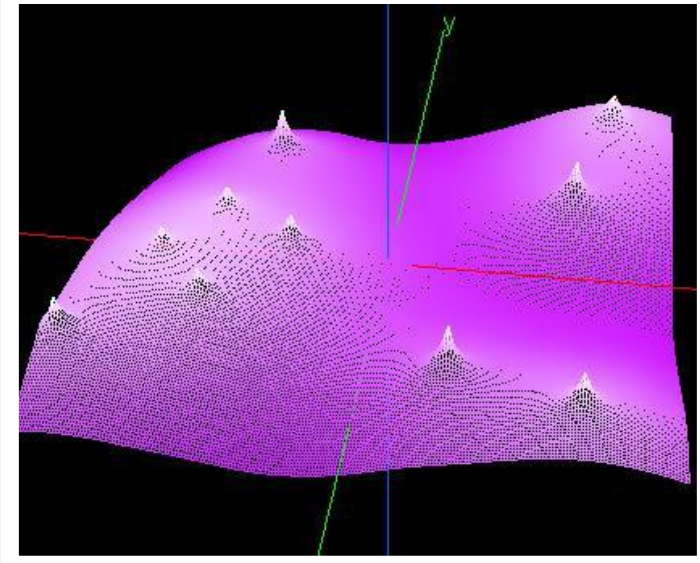


Figure 2-2. A 3D view of Composition of Sphere's function **[41]**

2.1.2.2.3 Composition of Rastrigin's Function

Rastrigin function is a famous function that used as a mathematical optimization function to evaluate algorithm performance and it is defined by:

$$\boldsymbol{f(x) = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10)} \tag{16}$$

The domain space in this function is $x \in [-5,5]^n$. This problem has a huge number of local optima that used rotation method to move them. This function is multi-model function and it is scalable. Global optimum is calculated by the below equation:

$$\boldsymbol{x^*(t) = \overrightarrow{O_i}, F(x^*(t)) = H_i(t)}$$
$$\boldsymbol{H_i(t) = min_j^m H_j} \tag{17}$$

The basic function is: f_1- f_{10} = Rastrigin's function [41]

A 3D view of this function is shown in Figure 2-3.

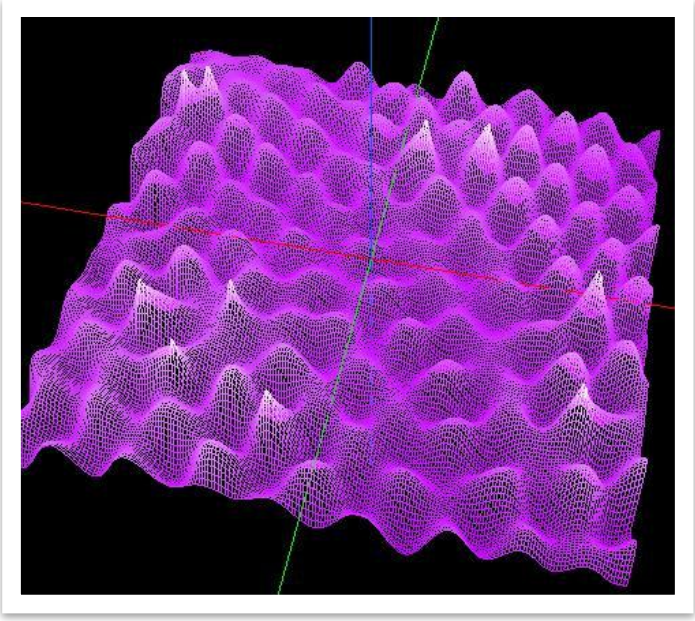


Figure 2-3. 3D view of Rastrigin's function **[41]**

2.1.2.2.4 Composition of Griewank's Function

This function is especially used to test the convergence of optimization algorithms. Griewank's function is defined by:

$$f(x) = \frac{1}{1000}\sum_{i=1}^{n}(x_i)^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{18}$$

The domain space is $x \in [-5,5]^n$ and includes a massive number of local optima. Like other function, this function uses rotation method to move peaks. Global optimum is calculated by:

$$\begin{gathered} \boldsymbol{x^*(t) = \overrightarrow{O_i}, F(x^*(t)) = H_i(t)} \\ \boldsymbol{H_i(t) = min_j^m H_j} \end{gathered} \tag{19}$$

The basic function is: f_1- f_{10} = Griewank's function [41]

You can see a 3D perspective of this function in GDBG function in Figure 2-4.

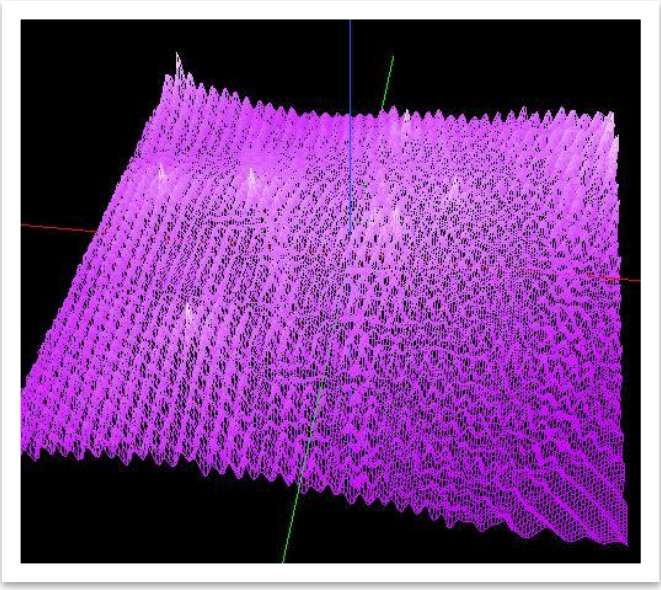


Figure 2-4. 3D view of Griewank's function

2.1.2.2.5 Composition of Ackley's Function

Another function that GDBG uses to create the environment is Ackley's function. Ackley's function is defined by:

$$f(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e \quad (20)$$

In this function $x \in [-5,5]^n$ and the environment contains a huge number of local optima. The global optimum can be obtained of these formulas:

$$x^*(t) = \overrightarrow{O_i}, F(x^*(t)) = H_i(t)$$
$$H_i(t) = min_j^m H_j \quad (21)$$

The basic function is: f_1- f_{10} = Ackley's function [41]

Figure 2-5 shows a 3D view of a sample of this function.

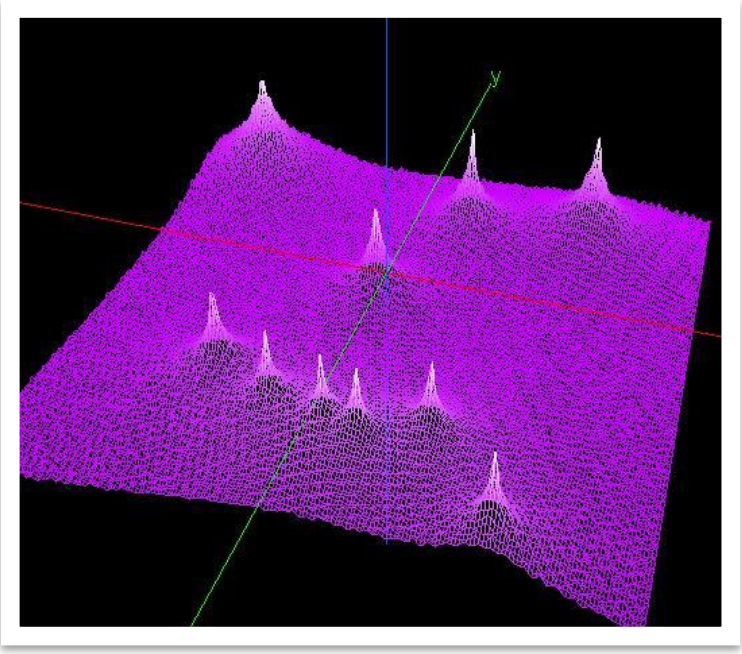


Figure 2-5. 3D view of composition of Ackley's function **[41]**

2.1.2.2.6 Hybrid Composition Function

This function is combination of Ackley's function, Griewank's function, Rastrigin's function, Sphere's function and also Weierstrass's function. Weierstrass's function is defined by:

$$\boldsymbol{f(x)} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \left(\sum_{\boldsymbol{k=0}}^{\boldsymbol{k_{max}}} \left[\boldsymbol{a^k} \cos\left(\boldsymbol{2\pi b^k (x_i + 0.5)} \right) \right] \right) - \boldsymbol{n} \sum_{\boldsymbol{k=0}}^{\boldsymbol{k_{max}}} \left[\boldsymbol{a^k} \cos(\boldsymbol{\pi b^k}) \right] \qquad (22)$$

The hybrid function uses different function properties and mixes them to create an environment that includes massive number of local optima. The global optimum can be get of the bellow formulas:

$$\boldsymbol{x^*(t) = \overrightarrow{O_i}, F\left(x^*(t)\right) = H_i(t)}$$
$$\boldsymbol{H_i(t) = min_j^m H_j} \qquad (23)$$

The basic function is: "f_1 - f_2 =Sphere's function, f_3 - f_4 =Ackley's function, f_5 - f_6 =Griewank's function, f_7 - f_8 =Rastrigin's function and f_9 - f_{10} =Weierstrass's function" [41]. A 3D view of this function is shown in Figure 2-6.

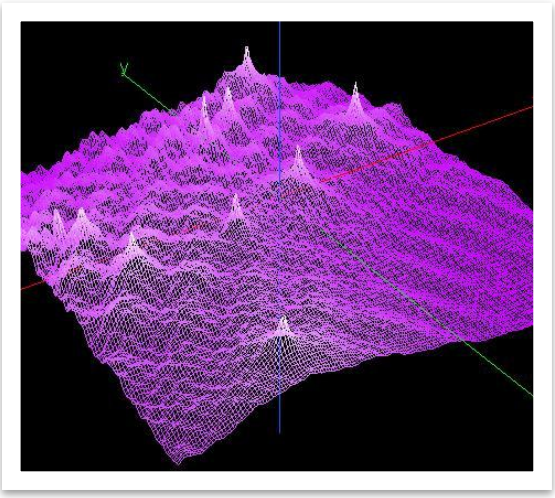


Figure 2-6. 3D view of Hybrid Composition function **[41]**

2.2 Difficulties of Solving Dynamic Problems

Meta-heuristic methods for optimization can solve the problems in static environments more easily than the problems in dynamic environments; these methods have some challenges in dynamic environments that do not exist in static environments. The two most important challenges are diversity loss and outdated memory.

In dynamic environments, the solution's eligibility that was obtained by agents is changing by the environment change and it does not match with the values stored in the memory. In fact, the fitness value that is stored in such a case is not correct and valid to find the solutions. This phenomenon is called *outdated memory*. To solve this problem, two solutions are suggested: re-evaluating memory and forgetting memory [42]. In the forgetting memory method the saved location for each solution is replaced with current location of that solution in the new environment. In the second method, the stored position in memory can be re-evaluated in new environment.

Diversity loss also happens due to the intrinsic nature of meta-heuristic methods for convergence. This issue is more important than the outdated memory. Due to premature convergence, diversity in the environment is reduced and algorithms have the problem to converge to an optimal level in new environment. This issue is created due to the intrinsic nature of these methods to converge to the previous optimal positions and closing the solutions to each other so much.

The easiest way to solve the above problem is re-initialization [43]. In this method, after environment changes we look at the problem as a new problem. This method is very

simple but it is not suitable for solving many dynamic problems. Use of this method only is suggested when major changes happen in the environment. It means the difference between new environment and the previous environment is very high. But in most real dynamic problems, changes in the environment are not very severe and there is a connection between the new environment and previous environment. So using the gained knowledge from previous environment we can increase the efficiency of optimization process in the new environment. As a result of using re-initialization method for solving such problems we lose all the gained knowledge from problem space. To solve diversity loss problem many different methods have been presented that can be divided into two general categories.

2.2.1 Presenting Siversity Method

The final goal of algorithms in this class is that first they allow the diversity loss to occur and then they try to solve it. This group can also be divided into two subgroups:

2.2.1.1 Mutation and Self-adaptation

In this subgroup it is tried to create diversity in the new environment by using self-adaptation and mutation. In [44], an adaptive mutation operator was presented as a mutation factor which is multiplied in normal mutation rate and called Triggered Hyper-mutation. In [45], a chaotic mutation to create diversity in the environment as an adaptation has been used. Also another method is represented in [46] that is solved the mutation step size problem in [44] by adaptation. Replacing the previous good solutions after a change in the environment rather than adding a random solution is a strategy that is presented in [47] to create diversity. A variable relocation method is presented in [48] that relocates the solutions based on the fitness function values when environment changes, it is done for each solution with different radius.

2.2.1.2 Other Approaches

Other methods for creating diversity in the environment after the changes in the environment occurs are presented too. RPSO is a method to randomize the part of the solution or whole of the solutions to detect changes in the environment [49]. An algorithm called Population-Based Incremental Learning (PBIL) was presented in [50] that used a customizable probability vector to produce solutions. This method uses the vector to set learning rate after environment change.

This class of methods is suitable for environments with low or medium change, because the mutation is a local search and is appropriate when the changes are small and local. Some problems of these methods can be noted as unidentifiable changes in most environments, not accurately measuring mutation step size and being inappropriate for the environments with vast and quick changes.

2.2.2 Diversity Maintenance Method

In this method it is always tried to preserved diversity in the environment at all times (before and after the change). Presented algorithms in this section can be divided into three categories:

2.2.2.1 Dynamic Topology

In this group by limiting communication between solutions the speed of algorithm's convergence to the global optimum is reduced. Thus it maintains the diversity in the environment. In [51] a neighborhood structure like grid for maintaining diversity that is called FGPSO was presented, which provides higher performance than RPSO [49] in dynamic environments with high dimensions. In HPSO [52] a hierarchical structure and tree-like is proposed to maintain diversity.

2.2.2.2 Memory-Based

When a periodical or recurrent change in the environment occurs, the past optimal solutions may be very useful for future use. Thus, memory-based methods try to keep such information. Memory-based methods have been mostly proposed for evolutionary methods such as GA, EDA, which have genetic nature.

2.2.2.3 Other Approaches

Other methods to maintain diversity in the environment after the discovery of changing environment is presented too. In [53], a sentinel placement method is used to maintain diversity. In this method, some sentinel series that are distributed in the search space have been used to generate new population. These sentinels always exist in the environment and are not removed so they can be used to identify changes in system. In [54] random immigrant method has been suggested that in every generation, some random solution is added to the population to maintain diversity. Another method to maintain diversity based on fitness sharing is presented in [55].

2.2.3 Hybrid Method

This group includes combination of presenting diversity with diversity maintenance method. So it maintains diversity during run time and also tries to create diversity when environment changes.

Chapter 3

A MULTI-SET ARTIFICIAL IMMUNE SYSTEM FOR SEARCHING OPTIMA IN DYNAMIC ENVIRONMENT

3.1 The Proposed Algorithm

In this chapter a new algorithm based on artificial immune system is proposed to optimize the functions described in chapter 2. The proposed algorithm uses different mechanism to solve dynamic environment challenges and increase the efficiency. In continue of this section, various mechanisms that are used in the proposed algorithm will be described.

3.1.1 Solving the Potential Optimum Coverage Challenge

As was said before, in a dynamic environment there are several peaks; each of them can change global optimum after environment changes. As a result, each of the peaks is considered a potential optimum. So the designed algorithm for optimization in dynamic environments should monitor all the peaks so it can quickly detect the global optimum after environment change has happened.

In the proposed algorithm multi-set mechanism is used to cover all peaks. The mechanism used in this algorithm controls sets is inspired by THs[6] and B-cells [4] cooperation. THs try to help B-cells to identify antigens in the body and become mature cells and convert them to antibodies. There are some sets that are called TH-cell and many other sets that are called antibody. In TH-cell set the number of cells in the set is

[6] T-cell helper [4]

more than the number of cells in antibody sets and their task is finding the peaks in environment. At the beginning of the algorithm there is only a TH-cell in search space and other sets are deactivated. That means their cells are not moving.

At the beginning the TH cells are initialized randomly in the problem space and start to search the environment. To search the environment, cells should be matured proportional with their shapes (shape means position in this algorithm), so they use different mutation rates. To do this, the algorithm uses the below formula as mutation rate which is inspired of PSO movement strategy [56].

$$\boldsymbol{MTrate_{i,j}(T+1) = C * MTL_{i,j}(T) + R^1 * (Pbest_{i,j} - X_{i,j}) + R^2 * (Gbest_{i,j} - X_{i,j})} \qquad (24)$$

Where T is time, $\boldsymbol{MTrate_{i,j}}$ is mutation rate for i^{th} cell in j^{th} dimension. $\boldsymbol{R^1}$ and $\boldsymbol{R^2}$ are two random number in [0,1]. $\boldsymbol{X_{i,j}}$ shows the position of i^{th} cell in j^{th} dimension. *Pbest*[7] is the best affinity value that each cell can catch during cloning process. *Sbest*[8] is the highest affinity value in each antibody set.

After a TH cell converged to a peak, it activates an antibody set and puts it in the peak instead of itself. After antibody was activated and reached the peak that TH cell had found before, covering and chasing of the peak after environment changes becomes the task of that antibody. The antibody set finds the tip of peak where it is located by a local search [57] [58].

[7] Personal best
[8] Set best

As mentioned, the numbers of cells in TH cells set are more than the number of cells in each antibody sets. Therefore when an antibody set is replaced instead of a TH cells, the cells with higher affinity are copied in antibody cells. For example, if the TH set has 10 cells and there are 6 cells in antibody set, top 6 cells from TH cells with higher affinity are chosen and copied in antibody cells. Their mutation rate, position and affinity rate (*Pbest*) be copied too. After antibody set is activated, the TH cells are randomly initialized in the search space. So the local searching and chasing the top of peak are antibody's responsibility and TH cells start the global searching again to find other peaks that are not covered by any antibodies sets. This process will continue until all the peaks are covered by antibodies. If TH cells find a peak that it is not covered by any antibody, put an antibody set there and active that set. When Euclidean distance between the *Sbest* position in m^{th} and $(m+n)^{th}$ iteration is less than a threshold which is called conv_limit, TH cells had found a peak and converged to it. The antibody activation pseudo code is shown in **Figure 3-1**.

Antibody Activation Algorithm

```
If TH cells are converged  then
    list = sort cells in TH cells based on their Pbest_affinity_value in descending order
    activate an antibody set
    For counter=1 to  antibody_set_size
        Antibody_cell_position_i (counter)= list_position(counter);
        Antibody_cell_mutation_rate_i (counter)= list_mutation_rate (counter);
        Antibody_cell_Pbest_i (counter)= list_Pbest (counter);
    Endfor
    Sbest_antibody = Sbest_THcell;
    Reinitialize TH cells;
Endif
```

Figure 3-1. Antibody activation pseudo code

If TH cells find a peak that was found before, without any activation the TH cells will reinitialize again. In fact, a peak that is discovered before is covered by an antibody set. So when the Euclidean distance between TH cell's *Sbest* position and *Sbest* position of antibody's cells is less than a certain amount which is named r_{excl}, the TH cells converged to a peak that was discovered before. The amount of r_{excl} determined is based on the proposed algorithm in [59]. It was shown when the Euclidean distance in both positions is less than the r_{excl}, they are converged to a peak [39]. The r_{excl} is defined by:

$$\boldsymbol{r_{excl}} = \mathbf{0.5} \times \left(\mathbf{1} \Big/ \boldsymbol{p^{\frac{1}{d}}} \right) \tag{25}$$

Where *P* is number of peaks and *d* is number of dimensions in search space. In each cycle, Euclidean distance between location TH cell's *Sbest* and all the *Sbest* of antibody sets is calculated and if the value of this distance with any of the sets is less than rexcl, TH cells are reinitialized. The pseudo code to maintain monopoly among TH cells set and antibody sets is shown in **Figure 3-2**.

TH Elimination Algorithm

```
For each activated_antybody_set i
    If Euclidian distance between Sbest_TH and Sbest_antibody is less than r_excl then
        Reinitialize TH cells;
    Endif
Endfor
```

Figure 3-2. Pseudo code TH Elimination Algorithm

Sometimes it is possible that the TH cells converge before reaching to a peak. This will put an antibody set into the place and activates it. Thus, this antibody set may move toward a peak that is already covered by another activated antibody set, so there will be

two activated antibody sets in a peak. Sometimes it may be that 2 covered peaks are too close to each other and an antibody set leaves its own peak and moves toward the other peak, hence covering that peak by two antibody sets. In this case, residents with more than one set at a peak not only will improve the results, but also merit evaluation would be superfluous. To solve this problem, Euclidean distance between all *Sbest*'s positions of activated antibodies should be calculated. Thus if the distance between the Sbest of two sets is less than r_{excl}, they are in the same peak. In such a case, the set who has the worse affinity value is eliminated and the other one with higher affinity continues its work. The pseudo code of above process is shown in **Figure 3-3**.

Antibody Elimination Algorithm

```
For each activated_antybody_set i
   For each activated_antybody_set j
      If Euclidian distance between Sbest_antibody i and Sbest_antibody j is less than r_excl then
         If Affinity(Sbest_i) <  Affinity (Sbest_j) then
               eliminate antibody set  i;
         Else
              eliminate antibody set  j;
         Endif
      Endif
   Endfor
Endfor
```

Figure 3-3. Pseudo code antibody Elimination Algorithm

3.1.2 Environment Change

One of the major challenges in dynamic environments is environment change detection and adopting algorithms for the problems caused by it. As mentioned in Section 2.2 , after changing the environment, optimization algorithms face two serious problems, diversity loss and outdated memory. Thus, algorithms that are designed to optimize dynamic environments must be able to quickly recognize changes in the environment to

use the mechanisms to solve these two problems. The proposed algorithm for detecting changes in the environment uses a point called *Guard* point. At the beginning of the algorithm a random position in search space is selected as the guard point and its affinity value will be saved. In each cycle of the proposed algorithm, the affinity value of the guard point is calculated and compared with the previous value. If the obtained values in current and previous iteration are equal, then the environment has not changed, but if they were two different values, it shows that environment has changed.

It should be noted that using of guard point is only suitable for those dynamic environments that change globally. But If there are only local changes then no changes may occur in guard position. So in this situation, using of guard point cannot be suitable for environment changes detection and to identify changes global optimum should be used.

After changes were detected in the environment, the algorithm uses some mechanisms to resolve the diversity loss and outdated memory problems. In the proposed algorithm when a changed environment is detected, it first uses mechanisms to increase diversity since diversity loss is happening in our algorithm. Before environment changes, antibody cells are converged to the tip of peaks that they have covered. In this case, the distance between the positions of antibody cells in each set is very close to each other and also their mutation rate will be very close to zero. The distance between the *Pbest* position and *Sbest* position are very close too.

In this situation, after a peak that is covered by an antibody set is moved the position of the antibody's cells and their *Pbest* remain around the peak position and cannot move to the new peak position. The reason of creating this problem is because the current mutation rate is determined based on the previous mutation rate, difference between antibody *Pbest* position and other cells in that antibody set. Also difference between cells *Sbest* position and other cells after changing the environment are close to zero. In this case the variation is very low and the cells cannot adapt themselves to new environment.

To solve this problem, the proposed algorithm uses a new mechanism on antibody sets. In this mechanism, the position of all cells in each activated antibody set is changed after environment change occurs. After the environment changes, the new position of peak is within the spatial with a radius of severity of previous position. So to speed up finding the new position of the peak, the cells should be randomly distributed with uniform distribution around the *Sbest* within the spatial with a radius of severity of previous position. Thus, antibody's cells will expand in a space based on the need to find better values. After changing, the new position of $\boldsymbol{cell}_{\boldsymbol{j}}$ in the antibody $\boldsymbol{set_i}$ is determined by following equation:

$$\boldsymbol{X_{i,j} = Gbest_i + (Rand^D(1,-1) \times P \times Severity)} \qquad (26)$$

Where $\boldsymbol{D}$ is the number of dimensions in problem space, ***Rand*** is a function that produces a D-dimensional vector of random numbers with a uniform distribution in [-1.1]. $\boldsymbol{P}$ determines how the cells are distributed in which radial around the Sbest based

on Severity. In fact, $\boldsymbol{P \times severity}$ shows the maximum distance between the *Sbest* and each cell in each dimension.

As mentioned before, the mutation rate after the environment change is close to zero. In the proposed algorithm to improve the diversity the cells mutation rates are randomly initialized based on severity after environment change is detected. Thus the mutation rate and position of cells could be adjusted based on severity, and antibody sets can be placed in a situation that can quickly find the location of new peaks in the new environment. The mutation rate can be obtained by the following equation

$$\boldsymbol{MTrate_{i,j} = (Rand^{D}(1,-1) \times Q \times Severity)} \quad (27)$$

Where $\boldsymbol{Q}$ is defined as the maximum mutation rate based on percentage of severity.

After the cells positions were determined based on activated antibody *Sbest* and severity, the *Pbest* of each cell changes to its new position ($Pbest_{i,j} = X_{i,j}$). Then the new value of *Pbest* affinity is calculated and the best one in each antibody set is considered as the *Sbest* in that set. Thus the previous memory of cell in activated antibody sets is reset so the outdated memory problem is solved.

After the environment change there is no need to increase diversity in TH cells because when these cells are converging they will reinitialize automatically and their memory should be updated after environment change occurs. Then the searching continues with valid memory. To update the TH cells memory, the merits of each *Pbest* position in TH cells are calculated and the best one will be considered as the *Sbest*. Pseudo-code of this

mechanism that is used after identifying changes in the environment is shown in **Figure 3-4**.

Deal with Environment Change Algorithm

For each activated_antybody_set $_i$
  For each cell $_j$ in antybody_set $_i$
    Update $X_{i,j}$ using Eq. 26
    Update MTrate $_{i,i}$ using Eq. 27
    $Pbest_{i,i} = X_{i,i}$
    Evaluate Affinity($Pbest_{i,j}$);
  Endfor
  $Gbest_i = \arg\max_{Pbest_{i,j}} f(Pbest_{i,j})$
Endfor
For each cell $_k$ in TH-cells
  Evaluate Affinity($Pbest_k$);
Endfor
$Gbest = \arg\max_{Pbest_k} f(Pbest_k)$

Figure 3-4. Pseudo code deal with Environment Change Algorithm

3.1.3 Mechanisms to Increase Performance

In the proposed algorithm, two different mechanisms for increasing the efficiency is used which are described in the following section.

3.1.3.1 Active and Inactive Mechanisms

After environment changes, each of the antibody sets try to find the peak that is resident in them by local searching. Due to the current error value being determined based on the result of a set tasks that has the best *Sbest* among the other sets, the local search results of the best set that is near the highest peak is much more important. Therefore those of other activated antibody sets in current environment do not have a role in determining the results, but the local search is still important for them. In fact, if these sets do not perform their local search after environmental change and several peaks movement, it is

possible that their peak is very far from them and they may lose it. Therefore the local search is essential for the entire activated antibody sets in the environment change.

As noted before, the obtained result of each environment is determined by an antibody set which is near the highest peak. So to improve the result this set should have performed a more accurate local search. One of the ways that local search can be made more precise is giving more opportunity to perform local search. But usually algorithms in dynamic environments do not have much time to adapt themselves with the environment because of frequency change in the environment that is determined based on amount of evaluation. Therefore each of the sets can perform little iteration until the next environment change. This can be a problem when there are a large number of activated sets in the area. In such a case, in each cycle of the algorithm run it performs a lot of fitness evaluation so each set can be executed very few times till the next environment change.

In the proposed algorithm to increase efficiency and give more opportunity to those antibody sets who are nearest to the highest peak, an activating-inactivating mechanism is used for antibody sets. In this mechanism, antibody set can be in two different situations: Activated or Deactivated. The activated antibody set is the set that its cells exist in the search space and performs fitness evaluation to do the search. Also its cells mature during each cycle of the algorithm. On the other hand the deactivated antibody set is the set that its cells are in the search space but do not perform maturing process and fitness evaluation.

In this mechanism, after each change in the environment, all the antibody sets are activated and perform the optimization process. As discussed before, the activated sets that are resident in non-optimal peaks must do local search after each change in the environment to prevent increasing their distance from the peak that they have covered. But after reaching the closest peak continuing the local searching to increase the accuracy is not very useful and it does not affect the result.

In fact, with this mechanism, after an antibody set moves near to a non-optimized peak, it'll be deactivated and they stop to perform affinity evaluation till next environment change. So this mechanism prevents a significant number of futile evaluations and uses them to give more opportunity to the antibody set who resides in the global peak. This causes the local searches around the global optimal peak to perform more and thus improve efficiency and accuracy of the results obtained from the whole algorithm.

In the proposed algorithm to detect whether the antibody set is close enough to its peak or not, their mutation rates are used. In fact, when a set is converging towards the target position and gets near it the mutation rate of the cells decreases.

The proposed algorithm has a parameter called *activated_boundary*, if all cells mutation rates be in range [-*activated_boundary*, *activated_boundary*], then that set is near enough to its peak and it should be deactivated. It should be mentioned that in any environment the set who has the best *Sbest* affinity value among all sets is not deactivated. Also when environment change occurs, all the sets are activated again. Pseudo-code related to activating-inactivating mechanism is shown in **Figure 3-5**.

Activating-Inactivating Algorithm

For each non activated_antybody_set i
  If all dimension of *MTrate* for each Antibody Set j is ϵ [-*activated_boundary*, *activated_boundary*] **then**
    activated [i]=False
  Endif
Endfor

Figure 3-5. Pseudo-code activating-inactivating mechanism

3.1.4 Graphical View on Proposed Algorithm

To better understand this algorithm and see how it works, here is a brief graphical view of running the algorithm in the Moving Peaks Benchmark problem. Figure 3-6 shows a 3D view of sample environment in this problem. As shown in this figure, there are 10 peaks in each environment that one of them can be highest peak which is called global optimum. As said before in problem description, the environment represents many different dimensions that will be discussed in the next chapter but to apprehend the view of the environment, the first two or three dimensions properties were used to create the approximate view of the environment.

Figure 3-7 shows 2D view of the sample environment .In Figure 3-7 (a) TH cells the best cell with highest affinity in this set is shown by star located in sidehill of a peak. After some cloning and searching in the environment, they can be located in near to the peak that is shown in Figure 3-7 (b).

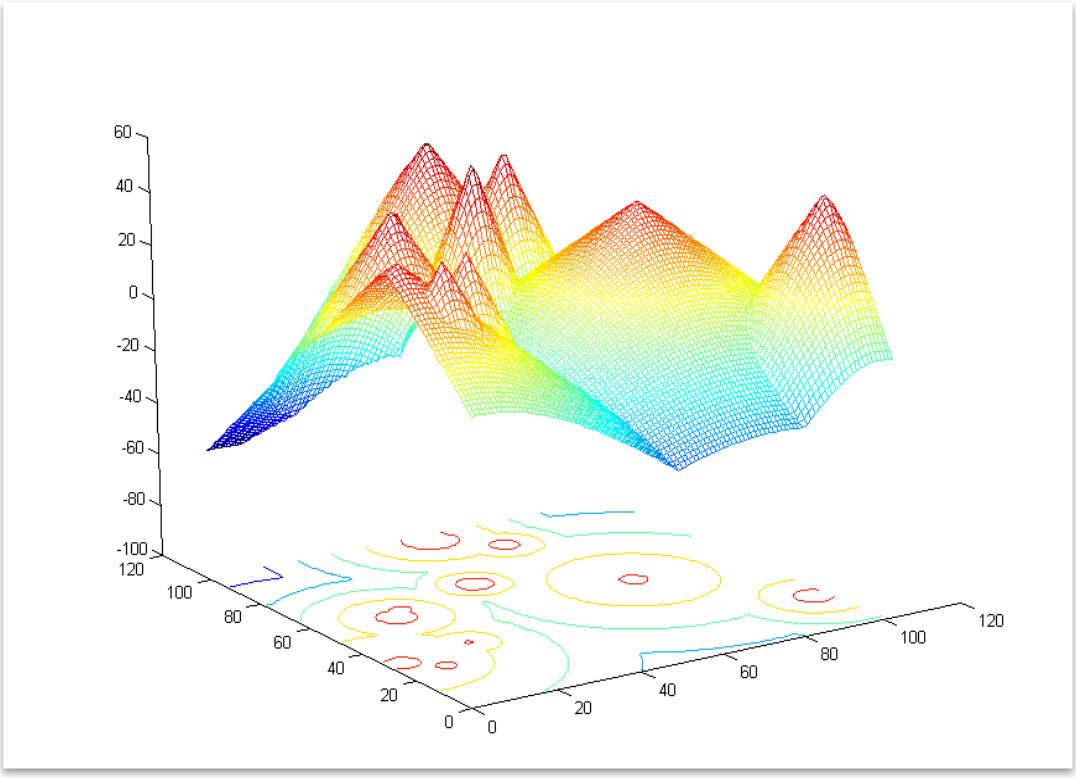


Figure 3-6. 3D view of an environment

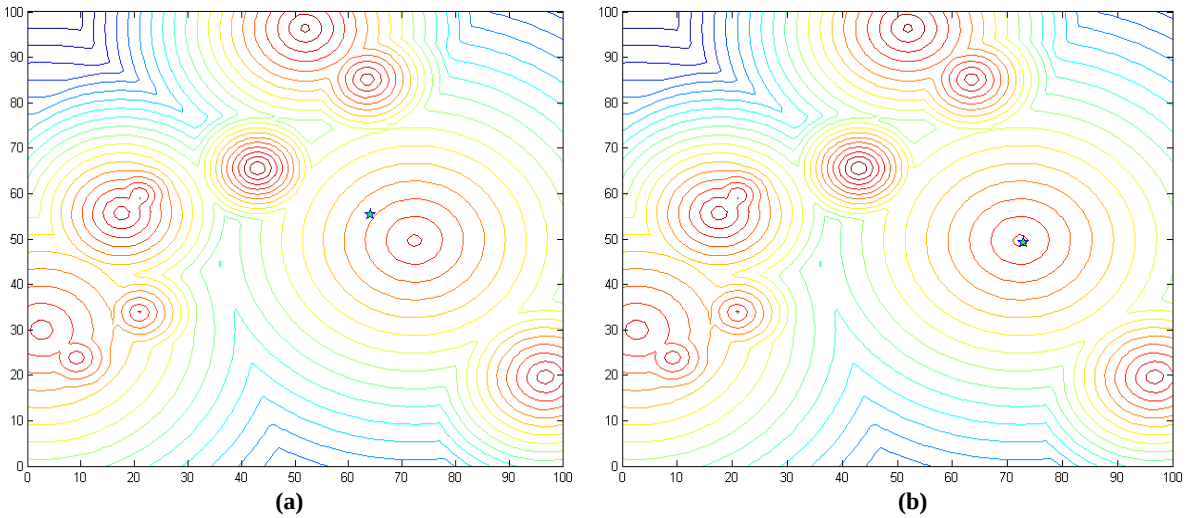


Figure 3-7. Finding peak by TH cells

After TH cells converged to a peak, they activate an antibody set and put that in the peak instead of themselves and TH cells reinitialize randomly again. The blue point in **Figure 3-8** is representing an antibody set. After an antibody set was activated, finding the top of the peak and converging to it is the task of antibody. At the same time TH cells are trying to find another peak and active another antibody set. This process is continuing until all of the peaks are found or the environment changed.

As is shown in **Figure 3-9** after environment changes, antibody sets search in a bigger space around themselves to find potential peaks. If all peaks were founded only the set with highest affinity (global best) continues searching more accurately around the peak to decrease error. Otherwise TH cells continue searching the whole environment to find the remaining peaks and so on.

As can be seen in **Figure 3-8** (a), 5 peaks were found. After environment change in **Figure 3-9** (b) with a local search by antibody sets those peaks are recognized again so TH cells continue searching for the peaks left.

All of these processes are continued until all peaks are found or they reach to the termination conditions. In the Figure 3-10 it is shown environments that all peaks inside are found and covered by antibody sets. The flowchart of this algorithm is shown in **Figure 3-11**.

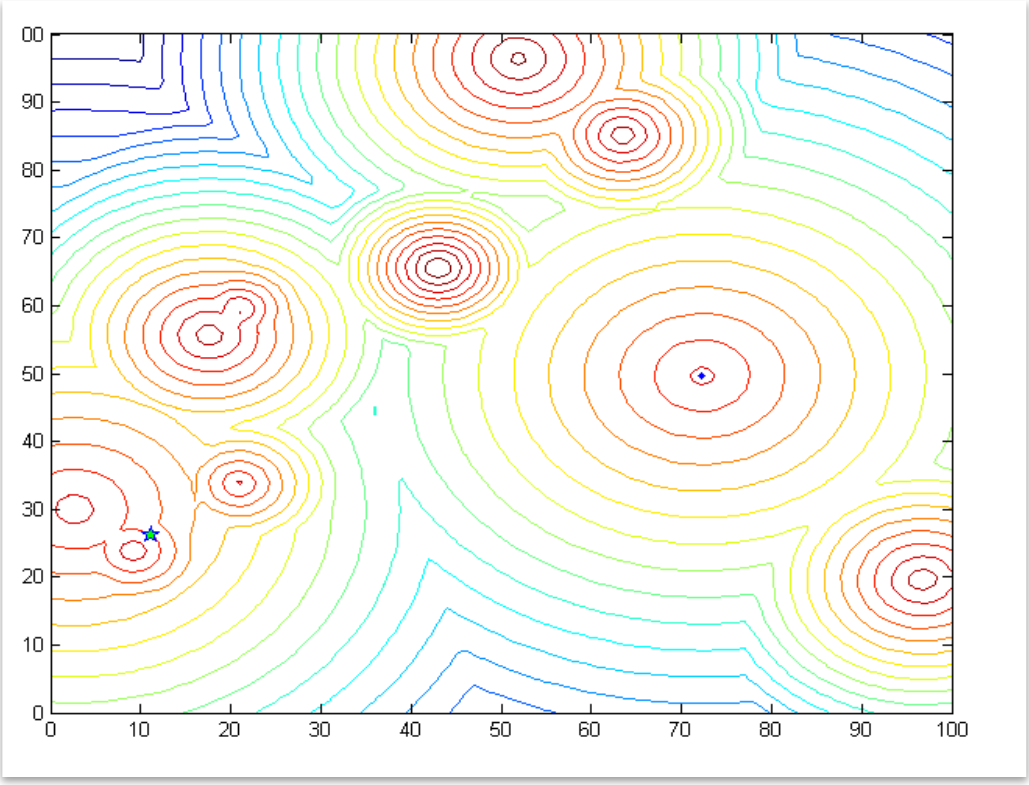
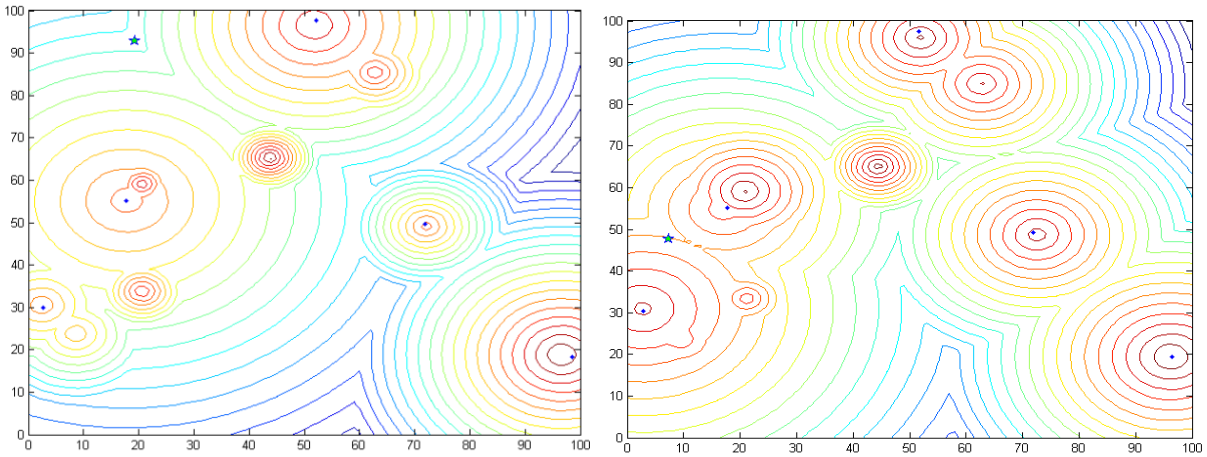


Figure 3-8. Antibody set activating



(a) **(b)**

Figure 3-9. Environment change

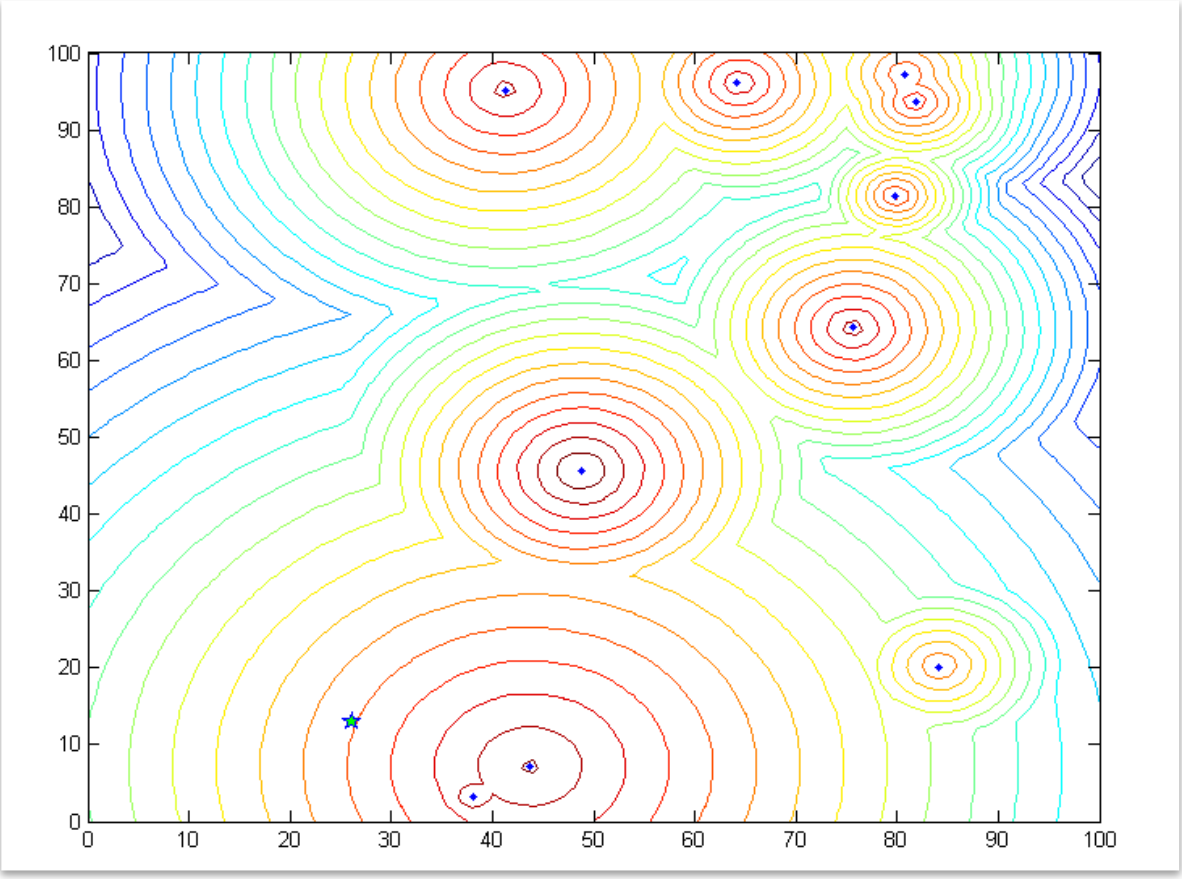


Figure 3-10. All of the peaks are found

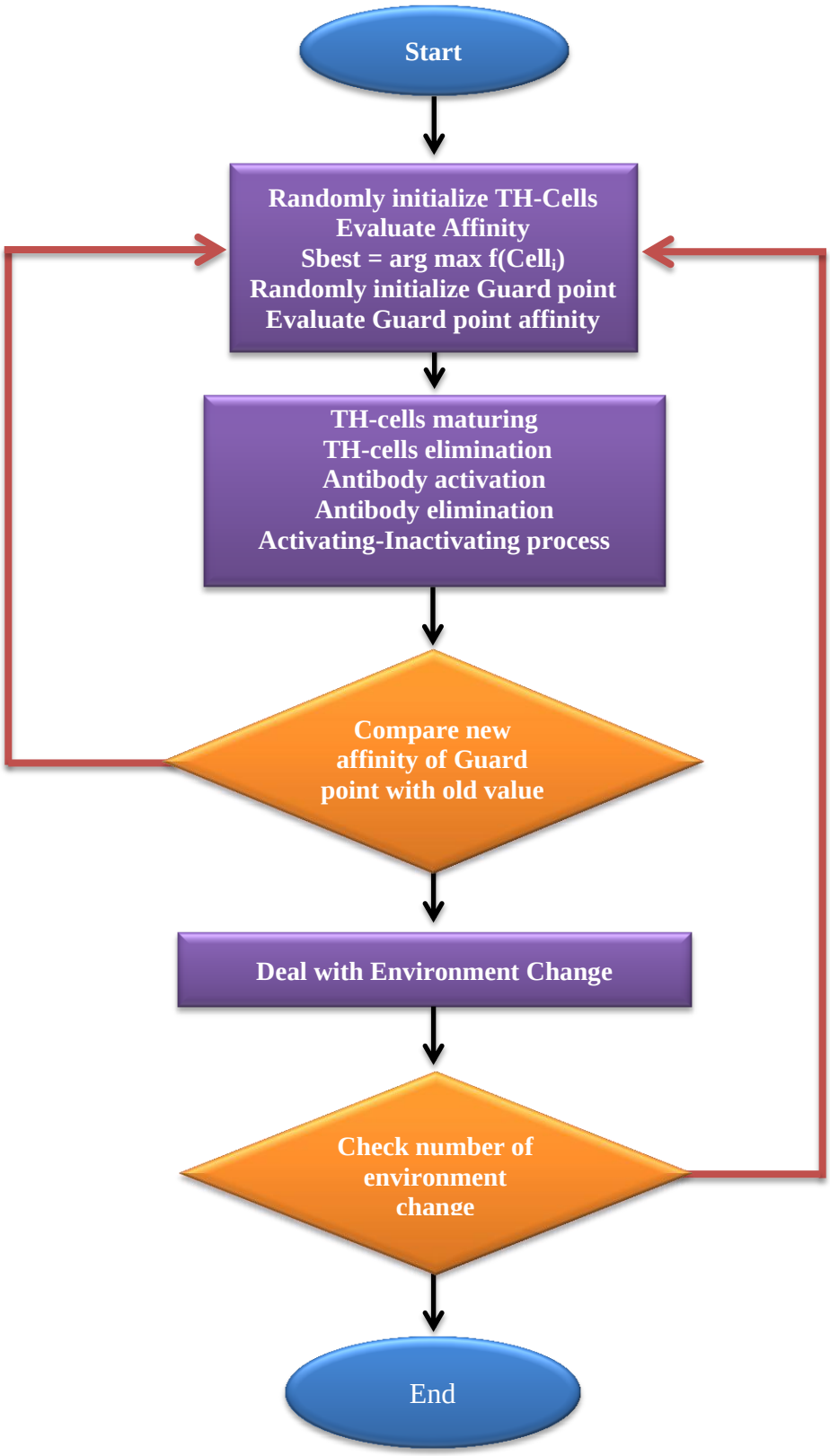


Figure 3-11. Algorithm's flowchart

Chapter 4

EXPERIMENTATION AND RESULTS ANALYSIS

In this part we have a short view on the proposed algorithm performance and we'll see how it responds in different situations. In section 4.1 the results of this algorithm tested on the moving peaks benchmark are shown. In section 4.2 the algorithm was tested on generalized dynamic benchmark generator (GDBG) and we'll see how it responds to these problems.

4.1 Results of Moving Peaks Benchmark Problem

The proposed algorithm was written on Matlab. The computer that ran this test was an Asus laptop with Intel Core ™ i7-Q720 (1.6GHz) CPU and 4GB RAM (DDR3 1066). To perform this test, we used the standard scenario to initialize parameters. These parameters are shown in **Figure 4-1** [59].

Parameter	Setting
Number of peaks ***P***	10
Number of dimensions	5
Peak heights	$\epsilon[30,70]$
Peak widths	$\epsilon[1,12]$
Evaluations in each environment change	5000
Change severity ***s***	1.0
Correlation coefficient	0

Figure 4-1. Standard parameter setting

To evaluate performance of our algorithm we used offline error to compare our algorithm with others and it is equal to average of best affinities at all times founded in optimization process. In other words, offline error is equal to average of all current errors and current error in time *t* is the deviation of the best founded individual by algorithm in time *t* in current environment from optimum. Offline error is a positive number and in the ideal case is equal to zero [60].

4.1.1 Effect of Number of TH Cells on Proposed Algorithm's Performance

In this section, we have a brief review of effects of number of TH cells on proposed algorithm's performance. The performance result of different TH cell population size is shown in **Figure 4-1**. As can be observed, the performance of the proposed algorithm with 10 TH cells is better than others. When the number of cells is less than 10, the performance of algorithm is decreased. In fact by reducing number of TH cells, we reduce speed of convergence and so TH cells find the peak at a later time hence reducing efficiency of the algorithm. On the other hand, by increasing the number of TH cells the performance of algorithm is reduced too. When number of TH cells is increasing, in fact number of evaluation in each environment change is increase so antibody cells have not enough time to search in environment.

Table 4-1. Results of different TH Cell size

Number of TH cells	Offline error ± standard error
5	0.943 ± 0.0721
7	0.68702 ± 0.0692
10	**0.59049 ± 0.0604**
12	0.65267 ± 0.0712
15	0.71542 ± 0.0793
20	0.97163 ± 0.0976
30	1.32108 ± 0.1348
50	1.89049 ± 0.1461

4.1.2 Compare with Other Algorithm

To compare our result with other algorithms, standard scenario is used that is shown in **Figure 4-1**. This scenario is mentioned as second scenario in [60]. There are many algorithm such as differential evolution (DE) [61], Extremal optimization (EO) [62] and Particle Swarm Optimization (PSO) which is used to solve this problem. Some of their results are available in [60]. The results of this problem are shown in Table 4-2.

Table 4-2. Results of Offline error ± Standard error

Authors	Algorithm	Number of peaks	Number of Evaluation	Offline error ± standard error
Bui & Branke **[63]**	EA	50	2500	9.52±0.45
Blackwell & Branke **[61]**	PSO	10	5000	2.16±0.06
Li & Branke **[64]**	PSO	10	5000	1.93±0.06
Mendes & Mohais **[65]**	DE	10	5000	1.75±0.032
Blackwell & Branke **[66]**	PSO	10	5000	1.72±0.06
Moser & Hendtlass **[67]**	EO	10	5000	0.66 ±0.2
Blackwell & Branke **[66]**	mQSO	10	5000	1.85±0.08
Blackwell & Li **[68]**	AmQSO	10	5000	1.51±0.10
Hashemi & Meybodi **[69]**	CLPSO	10	5000	1.78±0.05
Changhe & Shengxiang **[70]**	FMSO	10	5000	3.11±0.06
Hu & Eberhart **[71]**	RPSO	10	5000	12.98±0.48
Blackwell & Branke **[66]**	mCPSO	10	5000	2.08±0.07
Du & Li **[72]**	SPSO	10	5000	2.51±0.09
Bird & Li **[73]**	rSPSO	10	5000	1.50±0.08
Kamosi & Hashemi **[74]**	mPSO	10	5000	1.61±0.12
Kamosi & Hashemi **[75]**	HmPSO	10	5000	1.42±0.04
Liu& Yang& Wang **[76]**	PSO-CP	10	5000	1.31±0.06
Lung & Dumitrescu **[77]**	CESO	10	5000	1.38±0.02
Lung & Dumitrescu **[78]**	ESCA	10	5000	1.54±0.02
Woldesenbet & Yen **[79]**	RVDEA	10	5000	3.54±(-)
Nasiri, & Meybodi **[80]**	SFA	10	5000	1.05±0.04
Rezazadeh & Meybodi **[81]**	APSO	10	5000	1.31±0.03
Noroozi & Hashemi **[82]**	CLDE	10	5000	1.64±0.03
Shahabi & Ünveren	**MDAIS**[9]	**10**	**5000**	**0.59049 ± 0.0604**

[9] Multi-set Dynamic Artificial Immune System

As you can see in Table 4-2, the proposed algorithm has the best performance among 25 algorithms. Only the performance of EO algorithm (Moser and Hendtlass [67]**)** is near to our result. The process of advancing and finding peaks by the proposed algorithm is shown in Figure 4-2.

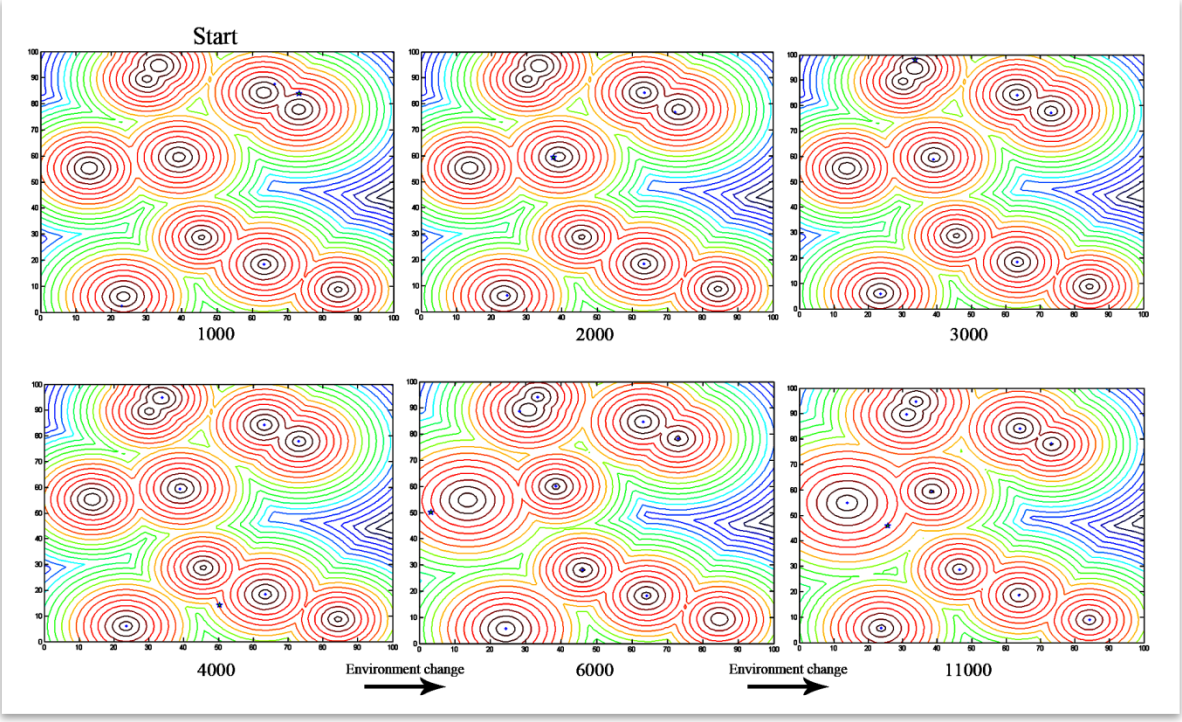


Figure 4-2. Procces of findig peaks by algorithm

4.2 Results of Tests on the Generalized Dynamic Benchmark Generator (GDBG)

To evaluate performance of algorithm in this problem, four formulas are used which are introduced in the following [83]:

$$\boldsymbol{Average\ best(Avg_{best})} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{runs}} \boldsymbol{Min_{j=1}^{num\_change}} \frac{\boldsymbol{E_{i,j}^{last}(t)}}{\boldsymbol{runs}} \qquad (28)$$

$$Average\ \boldsymbol{mean}(\boldsymbol{Avg_{mean}}) = \sum_{\boldsymbol{i=1}}^{\boldsymbol{runs}} \sum_{\boldsymbol{j=1}}^{\boldsymbol{num\_change}} \frac{\boldsymbol{E_{i,j}^{last}(t)}}{(\boldsymbol{runs} * \boldsymbol{num\_change})} \quad (29)$$

$$\boldsymbol{Average\ worst}(\boldsymbol{Avg_{worst}}) = \sum_{\boldsymbol{i=1}}^{\boldsymbol{runs}} \boldsymbol{Max_{j=1}^{num\ change}} \frac{\boldsymbol{E_{i,j}^{last}(t)}}{\boldsymbol{runs}} \quad (30)$$

$$\boldsymbol{STD} = \sqrt{\frac{\boldsymbol{1}}{\boldsymbol{runs} * \boldsymbol{num\ change} - \boldsymbol{1}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{runs}} \sum_{\boldsymbol{j=1}}^{\boldsymbol{num\_change}} \left(\boldsymbol{E_{i,j}^{last}(t)} - \boldsymbol{Avg_{mean}}\right)^{\boldsymbol{2}}} \quad (31)$$

Where number of runs is 20, num_change is equal to number environment change and $E^{last}(t)$ is absolute function error value after reaching to maximum number of evaluations and calculate by following formula [83]:

$$\boldsymbol{E^{last}(t)} = \left|\boldsymbol{f}\left(\boldsymbol{x_{best}(t)}\right) - \boldsymbol{f}(\boldsymbol{x^*(t)})\right| \quad (32)$$

Where $\boldsymbol{f}\left(\boldsymbol{x_{best}(t)}\right)$ the affinity is at best point it time *t* and $\boldsymbol{f}(\boldsymbol{x^*(t)})$ is equal to affinity in global peak. For this problem, parameters were set according to **Figure 4-3**. It should be mentioned that all of the other algorithms used in the following formula and because there are 10 dimensions in this problem they have 10,000 evaluations before environment change [83].

$$\text{Evaluations in each environment change} = 10{,}000 * \text{number of dimensions} \quad (33)$$

But we used 50,000 evaluations in each environment. It means that our algorithm had half opportunity to find the best solution compared to the others algorithms. This shows that our algorithm can find solution at least two times faster than other algorithms.

Parameter	Setting
Number of peaks $\boldsymbol{P}$	10
Number of dimensions	10
Search range	$\epsilon[-5,5]$
Peak widths	$\epsilon[1,10]$
Peak height	$\epsilon[10,100]$
Evaluations in each environment change	10000*dimensions
height severity	5.0
Number of runs	20
Number of environment change	60

Figure 4-3. Parameters setting for GDBG

To compare the results we show every function's result in a different table, each table belongs to a function that is described in section 2.1.2.2. Each function has 4 table which each table shows one of Avg_{best}, Avg_{mean}, Avg_{worst} and STD.

The algorithms that are compared with the proposed algorithm are: Dynamic Artificial Immune (DAI) [84], The Differential Ant-Stigmergy (DAntS) [85], Clustering Particle Swarm Optimizer (CPSO), Standard PSO (SPSO), Simple Genetic Algorithm (SGA) [86], Self-Adaptive Differential Evolution (Self-ADE) [87] and Ensemble of Explicit Memories (EEM) [88]. These tables display Author names, their corresponding algorithm, number of evaluation in each environment change and errors in each change instance. The lower error is representing the better performance. But to have a fair comparison we should look at all results in a function and consider the overall performance.

Table 4-3. Average-best in Function 1 (10 peaks)

Function 1 10 peaks error : Avg_{best}								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	0.0048	0.0027	0.0052	0.0076	0.0052	0.0087
Korosec & Silc **[85]**	DAntS	100,000	4.17 e−13	3.80 e−13	3.80 e−13	6.57 e−13	5.56 e−13	7.90 e−13
Li & Yang [86]	CPSO	100,000	1.054 e−7	5.214 e−8	4.306 e−8	9.721 e−7	2.561 e−7	4.325 e−6
	SGA	100,000	4.01 e−5	4.295 e−5	5.543 e−5	1.799 e−5	1.004 e−5	6.234 e−6
	SPSO	100,000	**0**	**0**	**0**	**0**	**0**	**0**
Brest & Zamuda [87]	**self-ADE**	100,000	**0**	**0**	**0**	**0**	**0**	**0**
Yu & Suganthan **[88]**	EEM	100,000	0.0054	0.00445	0.00435	0.0057	0.01105	0.0104
Shahabi & Ünveren	**MDAIS**	**50,000**	**0**	**0**	**0**	**0**	**0**	**0**

Table 4-4. Average-worst in Function 1 (10 peaks)

Function 1 10 peaks error : Avg_{worst}								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	5.1786	46.1036	41.4286	37.0052	19.5234	71.4790
Korosec & Silc **[85]**	DAntS	100,000	5.51	38.5	39.7	9.17	20.9	47.1
Li & Yang **[86]**	CPSO	100,000	1.244	27.12	28.15	3.239	21.72	26.55
	SGA	100,000	43.2	52.08	45.47	75.39	40.23	80.31
	SPSO	100,000	31	48.23	43.28	72.77	35.77	78.92
Brest & Zamuda [87]	**self-ADE**	100,000	**0.910466**	32.1705	31.7827	**0.919964**	18.392	32.7662
Yu & Suganthan **[88]**	EEM	100,000	35.009	51.032	47.041	13.96	47.763	54.099
Shahabi & Ünveren	**MDAIS**	**50,000**	1.11	**6.4312**	**8.14**	1.8553	**3.3495**	**5.6402**

Table 4-5. Average-mean in Function 1 (10 peaks)

Function 1 10 peaks error : Avg_{mean}								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	0.1353	5.8667	4.2545	5.3563	4.4356	9.9407
Korosec & Silc **[85]**	DAntS	100,000	0.180	4.18	6.37	0.482	2.54	2.34
Li & Yang **[86]**	CPSO	100,000	0.03514	2.718	4.131	0.09444	1.869	1.056
	SGA	100,000	5.609	10.08	13.13	21.22	7.899	29.25
	SPSO	100,000	5.669	10.24	11.73	21.89	6.731	32.01
Brest & Zamuda [87]	**self-ADE**	100,000	0.028813	3.5874	2.99962	**0.015333**	2.17757	1.1457
Yu & Suganthan **[88]**	EEM	100,000	5.7109	10.658	10.87	1.5033	8.2954	8.232
Shahabi & Ünveren	**MDAIS**	**50,000**	**0.024193**	**1.7017**	**2.4409**	0.08553	**0.9174**	**0.8402**

Table 4-6. STD in function 1 (10 peaks)

Function 1 10 peaks error : STD								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	1.0061	10.2772	8.1828	8.9414	5.5545	15.8214
Korosec & Silc **[85]**	DAntS	100,000	1.25	9.07	10.7	1.95	4.80	8.66
Li & Yang **[86]**	CPSO	100,000	0.4262	6.523	8.994	0.7855	4.491	4.805
	SGA	100,000	9.349	13.22	13.87	21.88	9.406	25.68
	SPSO	100,000	7.729	12.62	13.59	20.15	8.75	25.63
Brest & Zamuda [87]	**self-ADE**	100,000	0.442537	7.83849	7.12954	**0.288388**	4.38812	5.72962
Yu & Suganthan **[88]**	EEM	100,000	9.6761	13.851	13.499	3.0008	13.102	14.96
Shahabi & Ünveren	**MDAIS**	**50,000**	**0.3734**	**4.203**	**5.7439**	0.2937	**2.3183**	**3.1321**

As you can observe in function 1 with 10 peaks, the proposed algorithm has the best results in most of the situations. Just in T4 (Chaotic change) case, the Self-ADE algorithm has a better performance. Our result is also better than other algorithms and has a very close performance to Self-ADE performance. In overall, we can say that our algorithm has the best performance in function 1 with 10 peaks.

In continue, function 1 was used again as a measurement for performance. This time the environment was including 50 peaks instead of 10 peaks. The results shown in **Table 4-7** implies that our algorithm still has the best performance among 8 algorithms and just self-ADE algorithm has the nearest results to ours. The only difference is that their number of evaluations is doubled.

Table 4-7. Average-best in Function 1 (50 peaks)

Function 1 50 peaks error : Avg_{best}								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	0.0072	0.0040	0.0057	0.0118	0.0078	0.0104
Korosec & Silc **[85]**	DAntS	100,000	5.97 e−13	5.03 e−13	3.57 e−13	7.73 e−13	8.02 e−13	6.73 E−13
Li & Yang **[86]**	CPSO	100,000	2.447 e−6	2.061 e−7	9.888 e−7	4.353 e−6	2.121 e−6	9.033 e−5
	SGA	100,000	4.01 e−5	4.295 e−5	5.543 e−5	1.799 e−5	1.004 e−5	6.234 e−6
	SPSO	100,000	1.43 e−4	1.435 e−4	2.528 e−4	4.217 e−4	5.458 e−4	0.001029
Brest & Zamuda [87]	**self-ADE**	100,000	**0**	**0**	**0**	**0**	**0**	**0**
Yu & Suganthan **[88]**	EEM	100,000	0.0063	0.00535	0.00505	0.00585	0.0197	0.0164
Shahabi & Ünveren	**MDAIS**	**50,000**	**0**	**0**	**0**	**0**	**0**	**0**

Table 4-8. Average-worst in Function 1 (50 peaks)

Function 1 50 peaks error : Avg_{worst}								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	4.5776	29.9379	33.7780	37.9725	24.1907	62.4719
Korosec & Silc **[85]**	DAntS	100,000	7.67	29.1	31.0	5.58	11.6	35.1
Li & Yang **[86]**	CPSO	100,000	4.922	22.08	25.65	1.974	9.606	22.08
	SGA	100,000	40.16	44.75	47.84	70.65	28.03	78.24
	SPSO	100,000	33.32	46.08	45.33	69.84	28.23	78.32
Brest & Zamuda [87]	**self-ADE**	100,000	3.92056	30.1958	27.6823	**1.21212**	9.08941	33.1204
Yu & Suganthan **[88]**	EEM	100,000	26.538	50.227	44.899	13.497	21.09	27.041
Shahabi & Ünveren	**MDAIS**	**50,000**	**3.5051**	**14.4249**	**10.3486**	1.9921	**0.7403**	**6.0172**

Table 4-9 Average-mean in Function 1 (50 peaks)

Function 1 50 peaks error : Avg_{mean}								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	0.3644	4.7485	5.2531	2.6565	2.8641	6.8330
Korosec & Silc **[85]**	DAntS	100,000	0.442	4.86	8.42	0.509	1.18	2.07
Li & Yang **[86]**	CPSO	100,000	0.2624	3.279	6.319	0.125	0.8481	1.482
	SGA	100,000	7.614	11.3	15.24	17.93	5.293	34.93
	SPSO	100,000	7.95	12.29	14.89	20.96	5.426	36.27
Brest & Zamuda [87]	**self-ADE**	100,000	0.172355	4.08618	4.29209	**0.0877388**	0.948359	1.76542
Yu & Suganthan **[88]**	EEM	100,000	5.7391	13.285	15.896	1.4109	2.2653	3.1577
Shahabi & Ünveren	**MDAIS**	**50,000**	**0.124**	**2.9001**	**2.1662**	0.1024	**0.2811**	**0.0901**

Table 4-10. STD in Function 1 (50 peaks)

Function 1 50 peaks error : *STD*								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	0.9275	6.7580	6.6830	5.9773	4.1579	11.8790
Korosec & Silc **[85]**	DAntS	100,000	1.39	7.00	9.56	1.09	2.18	5.97
Li & Yang **[86]**	CPSO	100,000	0.9362	5.303	7.442	0.3859	1.779	4.393
	SGA	100,000	9.754	11.26	13.04	19.04	6.186	26.54
	SPSO	100,000	8.162	11.55	12.5	19.02	6.348	26.24
Brest & Zamuda [87]	**self-ADE**	100,000	0.763932	6.4546	6.74538	**0.24613**	1.76552	5.82652
Yu & Suganthan **[88]**	EEM	100,000	6.8424	12.944	13.365	2.4466	4.239	5.6002
Shahabi & Ünveren	**MDAIS**	**50,000**	**0.6123**	**4.754**	**5.103**	0.3922	**0.8261**	**0.312**

As can be seen in **Table 4-8**, our algorithm still has the best results. These results of average-worst show that our algorithm has less error than others even with half evaluation size. Just in T4 (Chaotic change) the self-ADE algorithm is a little better.

In **Table 4-9**, the best results of average-mean belong to our algorithm. This table can be considered as main parameter to compare the algorithms and proves that our algorithm has a better performance than the original immune system algorithm and others algorithms.

The STD errors that are shown in **Table 4-10** implies that the proposed algorithm has the best result for solving Function 1 even with 50 peaks.

In general, we can say that the proposed algorithm has better performance than others for solving Function 1 in different situation, just in T4 (Chaotic change) there is an algorithm which had better performance than ours with minor difference.

Following that, the results of Sphere's function are shown. This function which is tested by 8 different algorithms was introduced in chapter 2. In **Table 4-11** that shows average-best is clear only those two algorithms have zero error for all conditions. These two algorithms are self-ADE and our algorithm. These two algorithms may have the same performance in finding global peak, but our algorithm has the same results with only half number of evaluations.

Table 4-11. Average-best in Function 2 (10 peaks)

Function 2 **10 peaks error : Avg_{best}**								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	0.0534	0.0678	0.0813	0.0596	0.1032	0.0582
Korosec & Silc **[85]**	DAntS	100,000	1.97 e−11	2.34 e−11	2.72 e−11	1.41 e−11	3.59 e−11	1.65 e−11
Li & Yang **[86]**	CPSO	100,000	9.377 e−05	7.423 e−05	4.651 e−05	1.121 e−05	7.792 e−05	1.087 e−04
	SGA	100,000	1.909 e−03	3.022 e−03	5.739 e−03	2.071 e−03	9.138 e−03	3.432 e−03
	SPSO	100,000	1.016 e−013	**0**	4.334 e−014	7.523 e−014	**0**	**0**
Brest & Zamuda [87]	**self-ADE**	100,000	**0**	**0**	**0**	**0**	**0**	**0**
Yu & Suganthan **[88]**	EEM	100,000	0.1266	0.1383	0.13615	0.132	0.12985	0.1195
Shahabi & Ünveren	**MDAIS**	**50,000**	**0**	**0**	**0**	**0**	**0**	**0**

Table 4-12. Average-worst in Function 2 (10 peaks)

Function 2 10 peaks error : Avg_{woest}								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	**0.2102**	68.0774	473.8170	14.0593	441.2040	51.9411
Korosec & Silc **[85]**	DAntS	100,000	33.9	403	356	16.5	433	249
Li & Yang **[86]**	CPSO	100,000	19.26	144.1	158.3	**10.18**	320.7	26.08
	SGA	100,000	150.5	565.5	543.6	124.8	511	289.4
	SPSO	100,000	272.3	561	539.4	279.3	515.6	541.6
Brest & Zamuda [87]	self-ADE	100,000	15.4426	435.019	468.43	10.6608	459.147	49.5327
Yu & Suganthan **[88]**	EEM	100,000	38.758	45.346	29.778	32.751	34.247	35.26
Shahabi & Ünveren	**MDAIS**	**50,000**	20.1	**35.0502**	**10.414**	38.6683	**27.942**	**19.31**

Table 4-13. Average-mean in Function 2 (10 peaks)

Function 2 10 peaks error : Avg_{mean}								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc [84]	DAI	100,000	0.0984	8.1209	17.9979	1.0652	101.3840	6.5192
Korosec & Silc **[85]**	DAntS	100,000	3.30	25.6	18.9	1.45	49.6	2.11
Li & Yang **[86]**	**CPSO**	100,000	1.247	10.1	10.27	**0.5664**	25.14	1.987
	SGA	100,000	33.05	182.9	128.5	32.85	191.7	43.25
	SPSO	100,000	45.79	186.9	135.8	53.57	186.5	73.34
Brest & Zamuda **[87]**	self-ADE	100,000	0.963039	43.0004	50.1906	0.793141	67.0523	3.36653
Yu & Suganthan **[88]**	EEM	100,000	6.2147	7.2236	4.9885	4.2067	3.5058	3.478
Shahabi & Ünveren	**MDAIS**	**50,000**	**0.8253**	**6.3629**	**1.904**	3.7801	**1.651**	**1.2217**

Table 4-14. STD in Function 2 (10 peaks)

Function 2 10 peaks error : *STD*								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	**DAI**	100,000	**0.0291**	14.3832	62.2259	2.8269	134.5180	13.8172
Korosec & Silc **[85]**	DAntS	100,000	8.78	83.2	67.8	3.83	112	5.29
Li & Yang **[86]**	**CPSO**	100,000	4.178	35.06	33.45	**2.137**	64.25	5.217
	SGA	100,000	53.75	218.9	188.7	35.12	200.6	69.84
	SPSO	100,000	59.34	212.7	185.4	60.58	198.1	99.96
Brest & Zamuda [87]	self-ADE	100,000	3.08329	114.944	124.015	2.53425	130.146	12.9738
Yu & Suganthan **[88]**	EEM	100,000	9.6292	11.024	8.245	7.5828	7.3318	7.5956
Shahabi & Ünveren	**MDAIS**	**50,000**	2.1943	**7.0383**	**1.2047**	2.3425	**6.254**	**3.127**

As can be seen in **Table 4-12**, the best result in average-worst belongs to the proposed algorithm. In this test, our algorithm had some bad answer in T1 (Small change step) and T4 (Chaotic change). These bad results also had effect on the other results and as shown in **Table 4-13** (average-mean) and **Table 4-14** (STD) the best performance in T1 and T4 belong to DAI and CPSO. We think that the results can improve with the whole number of evaluations. But these results are still very good and acceptable.

In the third function (Rastrigin's function) our algorithm could not find best results of average-best. As shown in **Table 4-15**, the best results in this table belong to self-ADE. Our algorithm is in the second place in best performance. It still has very good results and they are very close to the best results.

Table 4-15. Average-best in Function 3 (10 peaks)

Function 3 10 peaks error : Avg_{best}								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	674.0810	943.8250	943.781	727.1850	907.9080	691.9480
Korosec & Silc **[85]**	DAntS	100,000	3.39 e−11	43.4	1.38	4.51 e−11	3.08	4.21 e−11
Li & Yang **[86]**	CPSO	100,000	0.003947	126.2	42.89	7.909 e−005	228.5	4.356
	SGA	100,000	0.009432	0.3146	2.045	0.5873	36.15	0.075
	SPSO	100,000	1.427	211.4	20.9	3.82	13.59	3.782
Brest & Zamuda [87]	**self-ADE**	100,000	**0**	**9.70434 e−8**	**3.13019 e−10**	**0**	**5.35102 e−10**	**8.17124 e−14**
Yu & Suganthan **[88]**	EEM	100,000	0.1996	0.17025	0.17975	0.20085	0.16635	0.1431
Shahabi & Ünveren	MDAIS	50,000	6.734 e−12	3.1431 e−6	9.2384 e−08	5.7098 e−8	7.1948 e−10	3.770 e−9

Table 4-16. Average-worst in Function 3 (10 peaks)

Function 3 10 peaks error : Avg_{worst}								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	1103.66	1270.5	1240.51	1644.55	1202.09	1834.17
Korosec & Silc **[85]**	DAntS	100,000	435	988	937	1170	923	1470
Li & Yang **[86]**	CPSO	100,000	711.2	1008	966.1	1204	974.2	1424
	SGA	100,000	786.1	1036	991.7	1286	970.5	1380
	SPSO	100,000	864.1	1068	1024	1396	990.2	1509
Brest & Zamuda [87]	**self-ADE**	100,000	**238.417**	938.858	944.695	922.236	874.852	1226.38
Yu & Suganthan **[88]**	EEM	100,000	512.53	504.83	501.49	555.05	507.77	506.33
Shahabi & Ünveren	**MDAIS**	**50,000**	354.32	**463.71**	**315.44**	**477.05**	**409.48**	**315.71**

Table 4-17. Average-mean in Function 3 (10 peaks)

Function 3 10 peaks error : Avg_{mean}								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	810.83	1078.7500	1073.43	1031.53	1023.9	1186.9
Korosec & Silc **[85]**	DAntS	100,000	15.7	824	688	435	697	626
Li & Yang **[86]**	CPSO	100,000	137.5	855.1	765.9	430.6	859.7	753
	SGA	100,000	158.1	638.7	573.9	419.5	741.9	491.7
	SPSO	100,000	553.6	900.8	827.1	709	829.1	803.5
Brest & Zamuda [87]	self-ADE	100,000	**11.3927**	558.497	572.105	65.7409	475.768	243.27
Yu & Suganthan **[88]**	EEM	100,000	151.98	140.47	136.67	164.96	95.123	107.54
Shahabi & Ünveren	**MDAIS**	**50,000**	93.62	**75.131**	**90.715**	**154.9**	**77.34**	**81.911**

Table 4-18. STD in Function 3 (10 peaks)

Function 3 10 peaks error : STD								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	**DAI**	100,000	66.1085	**64.1245**	**64.9950**	274.7490	**57.8713**	292.2960
Korosec & Silc **[85]**	DAntS	100,000	67.1	204	298	441	315	460
Li & Yang **[86]**	CPSO	100,000	221.6	161	235.8	432.2	121.5	361.7
	SGA	100,000	264.5	399.6	399.8	444.2	278.8	464.3
	SPSO	100,000	298.1	148.8	212.6	385.8	186.7	375
Brest & Zamuda [87]	**self-ADE**	100,000	**58.1106**	384.621	386.09	208.925	379.89	384.98
Yu & Suganthan **[88]**	EEM	100,000	190.71	182.71	183.86	216.4	151.82	158.36
Shahabi & Ünveren	**MDAIS**	**50,000**	79.22	104.1	157.6	**172.82**	103.46	**94.61**

On the other hand, the proposed algorithm has best performance in the average-worst results that are shown in **Table 4-16**. To compare with other algorithms, our results are much better than other algorithms and in some cases our errors are half or even third of the other results.

In **Table 4-17** that presents the average-mean results, the best performance again belongs to our algorithm.

In the STD errors, the ADI algorithm has the best performance. The proposed algorithm only gained the best results in T4 (Chaotic change) and T6 (Recurrent change with noisy).

Table 4-19. Average-best in Function 4 (10 peaks)

Function 4 **10 peaks error : Avg_{best}**								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	0.0679	0.1222	0.0864	0.0543	0.1497	0.0618
Korosec & Silc **[85]**	DAntS	100,000	2.01 e−11	2.95 e−11	2.87 e−11	1.85 e−11	5.89 e−11	2.09 e−11
Li & Yang **[86]**	CPSO	100,000	6.36 e−5	1.868 e−4	1.03 e−4	9.346 e−6	4.07 e−3	8.616 e−5
	SGA	100,000	2.697 e−3	3.439 e−3	7.537 e−3	1.855 e−3	4.842 e−2	3.322 e−3
	SPSO	100,000	**0**	**0**	**0**	0.3056	**0**	**0**
Brest & Zamuda [87]	**self-ADE**	**100,000**	**0**	**0**	**0**	**0**	**0**	**0**
Yu & Suganthan **[88]**	EEM	100,000	0.13325	0.1386	0.13335	0.13045	0.13	0.1118
Shahabi & Ünveren	**MDAIS**	**50,000**	**0**	**0**	**0**	**0**	**0**	**0**

Table 4-20. Average-worst in Function 4 (10 peaks)

Function 4 10 peaks error : Avg_{worst}								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	26.0705	586.2790	580.6420	51.9689	562.5500	336.7740
Korosec & Silc **[85]**	DAntS	100,000	57.6	505	540	18.8	528	39.7
Li & Yang **[86]**	CPSO	100,000	29.38	459.8	389.4	**14.62**	481	63.06
	SGA	100,000	296.5	643.3	624.3	376.2	590.9	595.3
	SPSO	100,000	376.3	656.1	612.9	460.3	576.1	684.4
Brest & Zamuda [87]	**self-ADE**	100,000	**19.623**	475.7	544.92	16.6057	510.193	28.4483
Yu & Suganthan **[88]**	EEM	100,000	37.581	47.009	36.414	34.924	31.496	35.28
Shahabi & Ünveren	**MDAIS**	**50,000**	20.15	**23.6**	**27.41**	23.19	**26.88**	**18.73**

Table 4-21. Average-mean in Function 4 (10 peaks)

Function 4 10 peaks error : Avg_{mean}								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	**DAI**	100,000	**1.4227**	122.4410	98.6688	4.2632	304.5660	12.6334
Korosec & Silc **[85]**	DAntS	100,000	5.60	65.6	53.6	1.85	108	2.98
Li & Yang **[86]**	**CPSO**	100,000	2.677	37.15	36.67	**0.7926**	67.17	4.881
	SGA	100,000	4.881	272.9	230.1	52.76	335.5	57.38
	SPSO	100,000	55.05	289.7	223.6	73.85	285	98.15
Brest & Zamuda [87]	**self-ADE**	100,000	1.48568	49.5044	51.9448	1.50584	69.4395	**2.35478**
Yu & Suganthan **[88]**	EEM	100,000	6.601	8.1906	7.1991	5.0355	3.121	3.5162
Shahabi & Ünveren	**MDAIS**	**50,000**	3.054	**6.601**	**4.117**	2.571	**2.39**	2.491

Table 4-22. STD in Function 4 (10 peaks)

Function 4 10 peaks error : *STD*								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	4.5459	201.6270	196.6950	9.7255	203.2430	55.8386
Korosec & Silc **[85]**	DAntS	100,000	26.5	160	140	4.22	178	7.59
Li & Yang **[86]**	**CPSO**	100,000	7.055	99.43	97.18	**2.775**	130.3	15.39
	SGA	100,000	80.15	270.7	251.2	96.98	223.7	116.6
	SPSO	100,000	92.64	263	245.1	104.8	228.1	148.4
Brest & Zamuda [87]	**self-ADE**	100,000	**4.47652**	135.248	141.78	4.10062	144.041	5.78252
Yu & Suganthan **[88]**	EEM	100,000	10.032	11.923	10.145	8.3325	6.6867	7.3484
Shahabi & Ünveren	**MDAIS**	**50,000**	4.775	**9.631**	**6.71**	7.84	**4.311**	**4.27**

The proposed algorithm shows a good performance to find global optimum peaks in function 4 (Griewank's function). As shown in Table 4-19, Self-ADE and the proposed algorithm have the best performances in this error measurement. Additionally, The CPSO has a close performance too.

The results of average-worst are shown in Table 4-20. In this case, our algorithm finds better solution than others and in worst case it has the best performance at least in 4 of 6 different situations.

As can be seen in Table 4-21, DAI algorithm has the best performance in T1. IN T4, the best result belongs to CPSO algorithm and self-ADE is the best in T6. In other cases the best performance in average-mean belongs to our algorithm.

In general we can say that the proposed algorithm had a very successful performance in function 4. The results that are shown in **Table 4-22** can support this statement. In this table, the proposed algorithm has less errors than other algorithms in T2 (Large change step), T3 (Random change), T5 (Recurrent change) and T6 (Recurrent change with noisy).

There is a similar situation in average-best results in function 5 (Ackley's function) with function 4. According to **Table 4-23**, the only algorithm that has zero errors in all situations is our algorithm. Also the SPSO algorithm has a close performance and it has the same results in 5 of 6 situations but we should note that it has twice the number of evaluations in each environment changes.

Table 4-23. Average-best in Function 5 (10 peaks)

Function 5 10 peaks error : Avg_{best}								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	0.2511	0.4368	0.3469	0.2172	11.5370	0.3173
Korosec & Silc **[85]**	DAntS	100,000	3.22 e−11	3.74 e−11	3.86 e−11	2.69 e−11	5.99 e−11	2.85 e−11
Li & Yang **[86]**	CPSO	100,000	1.584 e−4	3.224 e−4	3.337 e−4	4.85 e−6	1.377 e−4	2.077 e−4
	SGA	100,000	6.832 e−3	7.609 e−3	5.71 e−3	3.871 e−3	8.463 e−3	5.129 e−3
	SPSO	100,000	5.857 e−007	**0**	**0**	**0**	**0**	**0**
Brest & Zamuda [87]	self-ADE	100,000	4.10338 e−14	4.16556 e−14	4.15668 e−14	4.08562 e−14	4.24549 e−14	4.08562 e−14
Yu & Suganthan **[88]**	EEM	100,000	0.20075	0.18235	0.19615	0.2484	0.2035	0.184
Shahabi & Ünveren	**MDAIS**	**50,000**	**0**	**0**	**0**	**0**	**0**	**0**

Table 4-24. Average-worst in Function 5 (10 peaks)

Function 5 10 peaks error : Avg_{worst}								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	1728.1	705.152	786.275	1375.1	1927.64	1910.64
Korosec & Silc **[85]**	DAntS	100,000	17.1	22.2	16.0	8.10	29.0	8.75
Li & Yang **[86]**	CPSO	100,000	25.41	31.76	27.77	26.66	63.2	42.54
	SGA	100,000	80.54	82.92	75.17	89.64	64.14	89.61
	SPSO	100,000	554.7	500.4	360.5	740	94.04	945
Brest & Zamuda [87]	**self-ADE**	100,000	**4.89413**	9.6899	10.1371	**4.75098**	9.28981	4.78684
Yu & Suganthan **[88]**	EEM	100,000	44.887	54.133	36.438	39.928	55.669	56.092
Shahabi & Ünveren	**MDAIS**	**50,000**	7.94	**6.53**	**7.71**	6.36	**5.48**	**4.66**

Table 4-25. Average-mean in Function 5 (10 peaks)

Function 5 10 peaks error : Avg_{mean}								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	40.8943	34.4531	34.942	120.637	943.2230	480.337
Korosec & Silc **[85]**	DAntS	100,000	0.955	0.990	0.949	0.392	2.30	0.467
Li & Yang **[86]**	CPSO	100,000	1.855	2.879	3.403	1.095	7.986	4.053
	SGA	100,000	27.99	29.57	25.4	33.96	24.42	31.77
	SPSO	100,000	62.22	58.85	44.51	91.95	29.03	116.9
Brest & Zamuda [87]	**self-ADE**	100,000	**0.159877**	0.333918	0.357925	**0.108105**	0.409275	0.229676
Yu & Suganthan **[88]**	EEM	100,000	7.9042	10.091	7.2867	6.2507	8.2195	7.9011
Shahabi & Ünveren	**MDAIS**	**50,000**	0.5133	**0.1982**	**0.285**	0.344	**0.204**	**0.1752**

Table 4-26. STD in Function 5 (10 peaks)

Function 5 10 peaks error : *STD*								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	480.337	119.896	115.025	293.542	633.318	610.802
Korosec & Silc **[85]**	DAntS	100,000	3.43	4.05	3.31	1.61	6.36	1.73
Li & Yang **[86]**	CPSO	100,000	5.181	6.787	6.448	4.865	13.81	8.371
	SGA	100,000	24.23	25.31	21.92	30.98	19.39	30.97
	SPSO	100,000	104	99.23	149.7	22.24	193.1	152.7
Brest & Zamuda [87]	**self-ADE**	100,000	**1.02554**	1.64364	1.83299	**0.826746**	1.90991	0.935494
Yu & Suganthan **[88]**	EEM	100,000	11.287	13.28	10.201	10.116	13.016	12.911
Shahabi & Ünveren	**MDAIS**	**50,000**	1.294	**1.381**	**0.632**	1.106	**0.927**	**0.719**

In average-worst results that are shown in Table 4-24, the lowest errors in most of the situations belong to proposed algorithm. The Self-ADE algorithm is better than our algorithm just in T1(Small change step) and T4 (Chaotic change). The same situation has happened in average-mean and STD results that can be seen in **Table 4-25** and **Table 4-26**. In function 5 (Ackley's function), again our algorithm has better performance than other 7 algorithms, even it achieved these results by half of evaluations that other algorithms had.

Finally the last function that we used to compare the proposed algorithm with other algorithm is Hybrid Composition function. The results of this function are shown at below.

Table 4-27. Average-best in Function 6 (10 peaks)

Function 6 10 peaks error : Avg_{best}								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	0.1003	0.2871	0.2873	0.1074	38.4149	0.1442
Korosec & Silc **[85]**	DAntS	100,000	2.36 e−11	3.58 e−11	3.69 e−11	2.55 e−11	6.37 e−11	2.56 e−11
Li & Yang **[86]**	CPSO	100,000	1.693 e−4	1.26 e−4	6.566 e−4	1.28 e−5	1.835 e−3	2.852 e−4
	SGA	100,000	0.01314	4.445 e−3	9.612 e−3	0.1484	9.385 e−3	5.041 e−3
	SPSO	100,000	1.457	9.832 e−14	9.699 e−14	4.941 e−12	**0**	**0**
Brest & Zamuda [87]	**self-ADE**	100,000	**0**	**0**	**0**	**0**	**0**	**0**
Yu & Suganthan **[88]**	EEM	100,000	0.1493	0.14815	0.15235	0.15565	0.1404	0.12375
Shahabi & Ünveren	**MDAIS**	**50,000**	**0**	**0**	**0**	**0**	**0**	**0**

Table 4-28. Average-worst in Function 6 (10 peaks)

Function 6 10 peaks error : Avg_{worst}								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	418.3680	937.339	1018.53	906.233	1101.5	1324.42
Korosec & Silc **[85]**	DAntS	100,000	48.3	554	529	81.6	499	249
Li & Yang **[86]**	CPSO	100,000	37.79	258.5	504.8	131.8	628.8	265.7
	SGA	100,000	247.2	824.8	783.6	309.6	785	530.1
	SPSO	100,000	546.3	842.4	806.1	682.1	817.1	748.6
Brest & Zamuda [87]	self-ADE	100,000	32.7204	51.8665	84.519	38.7914	191.895	45.0354
Yu & Suganthan **[88]**	EEM	100,000	94.921	73.431	58.111	50.242	100.68	84.709
Shahabi & Ünveren	**MDAIS**	**50,000**	**24.631**	**37.105**	**32.774**	**34.05**	**70.637**	**31.829**

Table 4-29. Average-mean in Function 6 (10 peaks)

Function 6 10 peaks error : Avg_{mean}								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	20.4434	391.196	456.441	83.9698	845.862	482.207
Korosec & Silc **[85]**	DAntS	100,000	8.87	37.0	26.7	9.74	37.9	13.3
Li & Yang **[86]**	CPSO	100,000	6.725	21.57	27.13	9.27	71.57	23.67
	SGA	100,000	39.41	138.6	98.51	53.53	170.1	52.1
	SPSO	100,000	71.15	158.7	140.3	120.7	162.8	113.8
Brest & Zamuda [87]	self-ADE	100,000	6.22948	10.3083	10.954	6.78734	14.9455	7.8028
Yu & Suganthan **[88]**	EEM	100,000	17.303	18.732	16.005	11.753	26.311	24.558
Shahabi & Ünveren	**MDAIS**	**50,000**	**4.992**	**12.636**	**9.430**	**5.918**	**8.044**	**6.337**

Table 4-30. STD in Function 6 (10 peaks)

Function 6 10 peaks error : STD								
Authors	**Algorithm**	**Num of Evaluation**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
Zuben & Franc **[84]**	DAI	100,000	79.323	395.435	405.038	220.177	251.208	434.421
Korosec & Silc **[85]**	DAntS	100,000	13.3	122	98.4	22.0	118	57.4
Li & Yang **[86]**	CPSO	100,000	9.974	63.51	83.98	24.23	160.3	51.55
	SGA	100,000	65.84	254.3	208.8	100.2	274.6	87.99
	SPSO	100,000	118.1	260.8	240.7	173.3	275	164.6
Brest & Zamuda [87]	self-ADE	100,000	164.6	13.2307	23.2974	10.1702	45.208	10.9555
Yu & Suganthan **[88]**	EEM	100,000	22.801	19.006	17.397	13.594	29.318	26.794
Shahabi & Ünveren	**MDAIS**	**50,000**	**7.5112**	**11.807**	**15.674**	**9.871**	**24.683**	**9.284**

As can be seen in **Table 4-27**, the best results belong to the Self-ADE and the proposed algorithm. Only these two algorithms could have the lowest possible error in average-best error and it is equal to zero in all six situations.

In average-worst results that are shown in **Table 4-28**, the absolute winner is our algorithm. In this error measurement, our algorithm can decrease the original artificial immune system algorithm error between 15 to 40 times and only with half of evaluation size. It can easily show the power and speed of our algorithm. This situation repeats in average-mean table. The results that can be seen in **Table 4-29** shows in the function 5, the proposed algorithm can find peaks easier and faster than other algorithms in different situations.

In the last table, the results of STD errors are shown. Also in this case our algorithm can get the best results. If we have a look at all of the results in all problems in different situations, we see that our algorithm has the best performance in most of situations. It could improve the original artificial immune algorithm. In general we can claim that our algorithm has one of the best algorithms in the world and in many tests that have been done it has the best performance in general.

Chapter 5

CONCLUSION

In this thesis, we introduced a powerful algorithm to solve optimization problem in dynamic environments. This algorithm was inspired by artificial immune system algorithms and was upgraded by multi-set techniques to improve its performance.

This algorithm designed for problem solving that have many potential solutions but at same time only one of them is the best solution. For example moving peak benchmark that has many peaks at same time only one is the highest and it is the global optimum. This algorithm tries to find the best solution among the all solution by finding all of the possible solution and searching the near spaces. It also tries to find the best new solution after each environment change by monitoring the previous solution.

This algorithm was tested by two different problems. In the first problem it was compared with 24 algorithms and it had the best performance among them. In second problem which includes 6 different functions and 6 environment change methods, the proposed algorithm showed really good performance and it had the best performance in general.

In the proposed algorithm there we tried not to use any initial knowledge of the problem space as much as possible, but like other algorithms it needed shift severity. For the

future, we can suggest adding some online learning algorithms or adaptive mechanism to improve the algorithm in such a way that it does not need any initial information about problem space.

REFERENCES

[1] L. N. DeCastro, J. Timmis, *Artificial Immune Systems: A New Computational Intelligence Approach*. London: Springer, 2002.

[2] U. Aickelin, D. Dasgupta, "ARTIFICIAL IMMUNE SYSTEMS," in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, Edmund K. Burke and Graham Kendall, Eds., ch. 13.

[3] J. Timmis , T. Stibor , E. Clark , and A. Hone , "Theoretical Advances in Artificial Immune Systems," *Theoretical Computer Science*, vol. 403, pp. 11-32, August 2008.

[4] J. Kelly, *Understanding the Immune System How It Works*.: National Institute of Allergy and Infectious Diseases, 2007.

[5] P. Delves, S. Martin, I. Roitt, and D. Burton, *Essential Immunology*, 11th ed.: Blackwell Publishing, 2006.

[6] D. Kitamura, *How the Immune System Recognizes Self and Nonself Immunoreceptors and Their Signaling*. Japan: Springer, 2008.

[7] S. A. Hofmeyr, "An Interpretative Introduction to the Immune System," in *In Design Principles for the Immune System and Other Distributed Autonomous Systems*, I. Cohen and L. Segel, Eds.: Oxford University Press, 2000.

[8] W. R. Clark, *In Defense of Self:How the Immune System Works in Managing Health and Desease*.: Oxford University Press, 2007.

[9] G. J. V. Nossal, "Life, Death and the Immune System," *Scientific American*, vol. 269, pp. 21-30, 1993.

[10] F. J. Von, L. N. De Castro Zuben, "Artificial Immune Systems: Part I - Basic Theory and Applications," School of Computing and Electrical Engineering, State University of Campinas, Brazil, Technical Report TR – DCA 01/99, 1999.

[11] A. S. Perelson, "Immune Network Theory," *Immunol Rev*, pp. 5–36, August 1989.

[12] G. M. Edelman, N. K. Jerne, *Clonal Selection in a Lymphocyte Network: Cellular Selection and Regulation in the Immune Response*. New York: Raven Press, 1974.

[13] F. J. Benini, H. Varela, "The Immune Recmitment Mechanism: A Selective Evolutionary Mechanism," in *4th International Conference on Genetic Algorithms*, San Diego, CA, July 1991, pp. 520- 526.

[14] Carlos A. Coello Coello,F. Fabio, R. Maurizio,.

[15] Carlos A. Coello Coello, N. C. Cort´es, "Hybridizing a Genetic Algorithm with an Artificial Immune System for Global Optimization," , February 17, 2004.

[16] T.Krzystof , W.T.Stawomir, "Immune-based algorithms for dynamic optimization," *Elsevier*, vol. 179, pp. 1495-1515, November 2009.

[17] B. Jason, *Clever Algorithms: Nature-Inspired Programming Recipes*.: LuLu, 2011.

[18] K. trojanowski, S. T. Wierzchon, "Immune-based algorithms fo dynamic optimization," *information Sciences*, vol. 179, pp. 1495-1515, 2009.

[19] CARLOS A. COELLO COELLO, N. C. CORT´ES, "Solving Multiobjective Optimization Problems Using an Artificial Immune System," *Genetic Programming and Evolvable Machines. Springer*, vol. 6, pp. 163–190, 2005.

[20] J.Timmis , L.N. De Castro, "Artificial Immune Systems: A Novel Paradigm to Pattern Recognition," in *Artificial Neural Networks in Pattern Recognition*, C. Fyfe, Ed.: University of Paisley, 2002, pp. 67-84.

[21] Carlos A. Coello Coello, S.C. Esquivel, V.S.Areagon, "A T-cell algorithm for solving dynamic optimization problems," *Information Sciences*, pp. 3614-3637, April 2011.

[22] G.Dozier,A. Carlisle, "Adapting Particle Swarm Optimization to Dynamic Environments,"" in *International Conference on Artificial Intelligence*, Las Vegas,USA, 2000, pp. 429-434.

[23] G.Dozier, A.Carlisle, "Tracking Changing Extrema with Adaptive Particle Swarm Optimizer," in *2002 World Automation Congress*, Orlando, FL, USA, 2002, pp. 265-270.

[24] R. C.Eberhart, H.Xiaohui, "Adaptive Particle Swarm Optimization: Detection and Response to Dynamic Systems," in *IEEE Congress on Evolutionary Computation*, Honolulu, Hawaii, USA, 2002, pp. 1666-1670.

[25] A. P. Engelbrecht, F. V. Bergh, "A Cooperative Approach to Particle Swarm Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, pp. 225-239, 2004.

[26] D. B Fogel, T. Michalewicz, T. Back, *Evolutionary Computation 1 - Basic Algorithms and Operators*. Bristol, UK: Institute of Physics (IoP) Publishing, 2000.

[27] H. Richter, "Memory Design for Constrained Dynamic Optimization Problems," in *Applications of Evolutionary Computation*, 2010, pp. 552–561.

[28] S. Richter, H. Yang, "Learning Behavior in Abstract Memory Schemes for Dynamic Optimization Problems," *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, vol. 13, pp. 1163–1173, 2009.

[29] H. Cheng, and F. Wang, S. Yang, "Genetic Algorithms With Immigrants and Memory Schemes for Dynamic Shortest Path Routing Problems in Mobile Ad Hoc Networks," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 40, 2010.

[30] B. Wang, and Y. Wang, C. Hu, "Multi-Swarm Particle Swarm Optimiser with Cauchy Mutation for Dynamic Optimisation Problems," *International Journal of Innovative Computing and Applications*, vol. 2, pp. 123–132, 2009.

[31] X. Yao, S. Yang, "Population-Based Incremental Learning With Associative Memory for Dynamic Environments," *IEEE Transactions on Evolutionary Computation*, vol. 12, pp. 542-561, 2008.

[32] K. Tang, T. Chen, X. Yao, X. Yu, "Empirical Analysis of Evolutionary Algorithms with Immigrants Schemes for Dynamic Optimization," *Memetic Computing*, vol. 1, pp. 3-24, 2009.

[33] J. J. Grefenstette, H.G. Cobb, "Genetic Algorithms for Tracking Changing Environments," in *Proceedings of the 5th International Conference on Genetic Algorithms*, 1993, pp. 523-530.

[34] J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems," in *IEEE Congress on Evolutionary Computation*, vol. 3, 1999, pp. 1875–1882.

[35] Jürgen Branke. (1999, December) The Moving Peaks Benchmark. [Online]. http://people.aifb.kit.edu/jbr/MovPeaks/movpeaks/

[36] R. W. Morrison , K. A. De Jong, "A test problem generator for non-stationary environments," in *Evol. Comput*, 1999, pp. 2047-2053.

[37] Y. Jin , B. Sendhoff, "Constructing dynamic optimization test problems using the multiobjective optimization concept," in *EvoWorkshop 2004*, 2004, pp. 526-536.

[38] S. Yang, "Non-stationary problem optimization using the primal-dual genetic algorithm," in *IEEE Congr. on Evol. Comput*, 2003, pp. 2246-2253.

[39] S. Yang, X. Yao, "Experimental study on population-based incremental learning algorithms for dynamic optimization problems," in *Soft Comput*, 2005, pp. 815-834.

[40] C. Li, M. Yang, L. Kang, "A new approach to solving dynamic TSP," in *Simulated Evolution and Learning*, 2006, pp. 236-243.

[41] C. Li, Sh. Yang, D. A. Pelta, "Benchmark Generator for the IEEE WCCI-2012 Competition on Evolutionary Computation for Dynamic Optimization Problems," , 2011.

[42] A. Carlisle , G. Dozier, "Adapting particle swarm optimization to dynamic environments," in *The International Conference on Artificial Intelligence*, 2000.

[43] K. Krishnakumar, "Micro genetic algorithms for stationary and nonstationary function optimization," in *Proceedings of SPIE International Conference Adaptative Systems*, 1989, pp. 289-296.

[44] H. G. Cobb, "An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments," *NRL Memorandum Report*, vol. 6760, pp. 523-529, 1990.

[45] T. Nanayakkara, K. Watanabe, K. Izumi, "Evolving in dynamic environments through adaptive chaotic mutation," *Proc. of the 4th Internat. Symposium on ArtiJficial Life and Robotic*, vol. 2, pp. 520-523, 1999.

[46] F. Vavak, K. Jukes, T. Fogarty, "Performance of a genetic algorithm with variable local search range relative to frequency of the environmental changes," in *Genetic Programming*, 1998, pp. 22-25.

[47] E. L. Yu, P. N. Suganthan, "Evolutionary programming with ensemble of explicit

memories for dynamic optimization," in *Evolutionary Computation, 2009. CEC '09. IEEE Congress*, 2009, pp. 431-438.

[48] Y. G. Woldesenbet , G. G. Yen, "Dynamic evolutionary algorithm with variable relocation," *IEEE Transactions on Evolutionary Computation*, vol. 13, pp. 500-513, 2009.

[49] X. Hu, R. C. Eberhart, "Adaptive particle swarm optimization: detection and response to dynamic systems," *IEEE Congress on Evolutionary Computation,CEC2002*, pp. 1666-1670, 2002.

[50] S. Yang , X. Yao, "Experimental study on population-based incremental learning algorithms for dynamic optimization problem," *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, vol. 9, pp. 815-834, 2005.

[51] J. Kennedy , R. Mendes, "Population structure and particle swarm performance," , 2002, pp. 1671-1676.

[52] S. Janson , M. Middendorf, "A hierarchical particle swarm optimizer for dynamic optimization problems," in *Applications of evolutionary computing*, 2004, pp. 513-524.

[53] R.W.Morrison, "Designing evolutionary algorithms for dynamic environments," *Springer-Verlag New York Inc*, 2004.

[54] J. Grefenstette, "Genetic algorithms for changing environments," *Parallel Problem Solving from Nature*, vol. 2, pp. 137-144, 1992.

[55] H. Andersen, "An investigation into genetic algorithms, and the relationship between speciation and the tracking of optima in dynamic functions," in *Honours thesis*. Brisbane, Australia: Queensland University of Technology, 1991.

[56] J. Kennedy, R. Eberhart, "Particle Swarm Optimization," *IEEE International Conference on Neural Networks*, vol. 4, pp. 1942-1948, November 1995.

[57] K. Liaskos, "Hybridizing Evolutionary Testing with Artificial Immune Systems and Local Search," in *IEEE International Conference on Software Testing Verification and Validation Workshop*, 2008, pp. 211 - 220.

[58] R. Javadzadeh, Z. Afsahi, M.R. Meybodi, "Improved Artificial Immune System Algorithm with Local Search," in *World Academy of Science, Engineering and Technology*, 2009, pp. 654-657.

[59] T. Blackwell ,J. Branke, "Multiswarms, exclusion, and anti-convergence in dynamic environments," *IEEE Transactions on Evolutionary Computation*, vol. 10, pp. 459-472, 2006.

[60] The Moving Peaks Benchmark. [Online]. http://people.aifb.kit.edu/jbr/MovPeaks/

[61] T. Blackwell, J. Branke, "Multi-swarm optimization in dynamic environments," *Applications of Evolutionary Computing,Springer*, vol. 3005, pp. 489-500, 2004.

[62] S.Boettcher, A.G.Percus, "Extremal optimization: Methods derived from Co-Evolution," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 1999, pp. 825-832.

[63] L.T.Bui, J. Branke, H. A. Abbass, "Multiobjective optimization for dynamic environments," in *Congress on Evolutionary Computation*, IEEE 2005, pp. 2349 – 2356.

[64] X. Li, J. Branke, T. Blackwell, "Particle Swarm with Speciation and Adaptation in a Dynamic Environment," in *Proceedings of Genetic and Evolutionary Computation Conference*, 2006.

[65] R. Mendes, A. Mohais, "DynDE: a differential evolution for dynamic optimization problems," in *IEEE Congress on Evolutionary Computation*, 2005, pp. 2808-2815.

[66] T. Blackwell, J. Branke, "Multi-swarms, Exclusion and Anti-Convergence in Dynamic Environments," in *IEEE Transactions on Evolutionary Computation*, 2006, pp. 51-58.

[67] I. Moser, T. Hendtlass, "A Simple and Efficient Multi-Component Algorithm for Solving Dynamic Function Optimisation Problems," in *IEEE CEC*, 2007.

[68] T. Blackwell, J. Branke, X. Li, "Particle swarms for dynamic optimization problems," in *Swarm Intelligence*, 2008, pp. 193-217.

[69] A. Hashemi, M. Meybodi, "Cellular Pso: A Pso for Dynamic Environments," in *Advances in Computation and Intelligence*, 2009, pp. 422-433.

[70] L. Changhe, Y. Shengxiang, "Fast Multi-Swarm Optimization for Dynamic Optimization Problems Natural Computation," in *Fourth International Conference on Natural Computation*, 2008, pp. 624-628.

[71] X. Hu, R. C. Eberhart, "Adaptive particle swarm optimization: detection and response to dynamic systems," in *IEEE Congress on Evolutionary Computation*, 2002, pp. 1666-1670.

[72] W. Du, B. Li, "Multi-strategy ensemble particle swarm optimization for dynamic optimization Information Sciences," in *Information Sciences, vol. 178*, 2008, pp. 3096-3109.

[73] S. Bird, X. Li, "Using regression to improve local convergence," in *IEEE Congress on Evolutionary Computation*, 2007, pp. 592-599.

[74] M. Kamosi, A. Hashemi, M. Meybodi, "A New Particle Swarm Optimization Algorithm for Dynamic Environments," in *Swarm, Evolutionary, and Memetic Computing*, 2010, pp. 129-138.

[75] M. Kamosi, A. B. Hashemi, M. R. Meybodi, "A Hibernating Multi-Swarm Optimization Algorithm for Dynamic Environments," in *Proceedings of World Congress on Nature and Biologically Inspired Computing(NaBIC2010)*, Kitakyushu, Japan, 2010, pp. 370-376.

[76] L. Liu, S. Yang, D. Wang, "Particle Swarm Optimization With Composite Particles in Dynamic Environments ," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 40, pp. 1634-1648, 2010.

[77] R. I. Lung, D. Dumitrescu, "A collaborative model for tracking optima in dynamic environments," in *2007 Congr. Evol. Comput*, 2007, pp. 564–567.

[78] R. I. Lung, D. Dumitrescu, "Evolutionary swarm cooperative optimization in dynamic environments," in *Natural Comput*, 2010, pp. 83–94.

[79] Y. G. Woldesenbet, G. G. Yen, "Dynamic evolutionary algorithm with variable relocation," *IEEE Transactions on Evolutionary Computation*, vol. 13, pp. 500-513, 2009.

[80] B. Nasiri , M. R. Meybodi, "Speciation based Firefly Algorithm for Optimization in Dynamic Environments," *International Journal of Artificial Intelligence*, vol. 8, pp. 118-132, 2012.

[81] I. Rezazadeh,M. R Meybodi, A. Naebi, "Adaptive Particle Swarm Optimization Algorithm for Dynamic Environments," *Lecture Notes in Computer Science*, vol. 6728, pp. 120-129, 2011.

[82] N. Noroozi,A. B. Hashemi, M. R. Meybodi, "CellularDE: A Cellular Based Differential Evolution for Dynamic Optimization Problems," *Adaptive and Natural Computing Algorithms, Lecture Notes in Computer Science, Springer*, vol. 6593, pp. 340-349.

[83] C. Li, S. Yang, T. T. Nguyen, E. L. Yu, X. Yao, Y. Jin, H.-G. Beyer, P. N. Suganthan, "Benchmark Generator for CEC'2009 Competition on Dynamic Optimization," 2008.

[84] F.O. de Franc, F.J. Von Zuben, "A Dynamic Artificial Immune Algorithm Applied to Challenging Benchmarking Problems," in *IEEE Congress on Evolutionary*

Computation, 2009.

[85] P. Korosec, J. Silc, "The Differential Ant-Stigmergy Algorithm Applied to Dynamic Optimization Problems," in *IEEE Congress on Evolutionary Computation (CEC 2009)*, 2009.

[86] C. Li, S. Yang, "A Clustering Particle Swarm Optimizer for Dynamic Optimization," in *IEEE Congress on Evolutionary Computation (CEC 2009)*, 2009.

[87] J. Brest, A. Zamuda, B. Boskovic, M. S. Maucec, V. Zumer, "Dynamic Optimization using Self-Adaptive Differential Evolution," in *IEEE Congress on Evolutionary Computation (CEC 2009)*, 2009.

[88] E. L. Yu, P. N. Suganthan, "Evolutionary Programming with Ensemble of Explicit Memories for Dynamic Optimization," in *IEEE Congress on Evolutionary Computation (CEC 2009)*, 2009.