



**Virus Detection
Using Artificial Immune System
with Genetic Algorithm**

الكشف عن الفيروسات باستخدام نظام المناعة
الاصطناعي والخوارزمية الجينية

**Prepared by
Suha M. A. Afaneh**

**Supervisors
Prof. Alaa AL-Hamami
Prof. Raed Abu Zitar**

**A Dissertation Submitted in Partial Fulfillment of the
Requirements for Degree of Doctor of Philosophy in
Computer Science**

**Department of Computer Science
College of Computer Sciences and Informatics
Amman Arab University**

August 2010

AUTHORIZATION

I, the undersigned, Suha M. A. Afaneh, authorize Amman Arab University for Graduate Studies to reproduce this dissertation in whole or in part for purposes of research.

Name:

..... Suha Afaneh

Signature:

..... 

Date:

..... 21-Aug-2010

DISSERTATION COMMITTEE APPROVAL


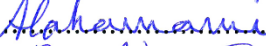

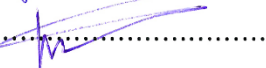
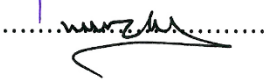
Date: 21- Aug- 2010

Name: Suha M. A. Afaneh

Degree: Doctors of Philosophy in Computer Science

Title: Virus Detection Using Artificial Immune System with Genetic Algorithm

This dissertation, with the title “Virus Detection Using Artificial Immune System with Genetic Algorithm”, was presented, examined, and approved on: 21-Aug-2010

Examining Committee	Title	Signature
Prof. Nadim Obaied	Chair	
Prof. Alaa Al-Hamami	Supervisor & Memeber	
Prof. Raed Abu Zitar	Co-supervisor & Memeber	
Dr. Bassel Kasasbieh	Member	
Dr. Mezher Alaney	Member	

Acknowledgments

A long journey of making this dissertation starts with small steps. For that I want to thank two great men who supported me, each in a different way. For Dr Alaa Hamami I say, thank you for your great support and encouragement which helped me to move on inspite of the many obstacles which faced this work. For Dr Raed Abu Zitar I say thank you for your valuable advices which was present every moment I felt the gap in my destination. Especially that you have suggested this interesting topic. For both of you, I must say a word of appreciation toward all of your efforts to help me in my journey.

Nevertheless, great deeds can not be done with out the assistance of their surroundings, to all of these people I have to say more than thank you, because you have filled the spaces I left behind and your smiles kept me focused on my target for Mais Afaneh, Dr Khaled Alqawasmy, Firas Afaneh, Tahany El-Najy, Shafiqa Bahlawan, Khaled J. Al-Attar, Ferial Al-Alaf, Dimah Fadda and Ahmad Saadeh. Nothing more than a word of gratitude can be said.

Dedication

To the great woman who pushes and supports me all over the way.

To whom I call the light of my day, the source of my happiness, my pearl.

Dear mother Souhaila Bahlawan, this work is for you.

To my husband Abed, my soul mate, and the love of my life.

To the father of my two lovely children (Sama and Karam); the man who makes every thing comes true and possible for me. He is my mirror which reflects my

true image of being; a human, a wife, a mother, a student, and a working woman.

He has been, with my children the source of my inspiration and my constant changing power toward the best.

To the soul of my father, Ameen Afaneh, I miss you and I wish if you were here with me.

Table of contents

Acknowledgments	IV
Dedication.....	V
Table of contents	VI
List of Figures	VIII
List of Tables	X
List of Abbreviations	XII
Abstract.....	XIII
Arabic Abstract	XV
Chapter One Introduction	1
1.1 Introduction.....	1
1.2 The Statement of the Problem.....	3
1.2.1 Research Questions	5
1.2.2. Definitions.....	6
1.2.3 Thesis Contributions	7
1.3 The Thesis Structure	8
Chapter Two Literature Reviews and Related Works	10
2.1. Literature views.....	10
2.1.1. The Immune System (IS).....	10
2.1.2. The Artificial Immune System (AIS).....	14
2.1.3. The Viruses.....	19
2.1.4. The Anti-virus	25
2.1.5 Genetic Algorithm	26
2.2. Related Works	28
2.2.1. Studies about the Artificial Immune System	28
2.2.2 Studies about Computer Viruses	30
2.2.3. Studies about Computer Viruses with AIS	33
Chapter Three Methodology	39
3.1. Data sets	39
3.2. Research tools.....	39
3.3. Research Stages	39
3.3.1. The design and implementation of the VDC algorithm.....	40
3.3.2 The testing of the VDC algorithm.....	51
3.3.3 The optimization of the VDC algorithm by using the GA.....	56
3.3.4 The testing of the optimized VDC algorithm based on GA.....	58
Chapter Four The VDC algorithm Results and Analysis	59
4.1 Training of the VDC algorithm	59
4.1.1 The Training Phase Analysis	84
4.2 Matching the VDC algorithm.....	86
4.2.1 The Matching Phase Analysis	98
Chapter Five The Optimization of the VDC Algorithm using the GA	108
5.1. The Optimized VDC Algorithm based on GA.....	108
5.1.1 GA Optimization	109
5.1.2 GA Training	114
5.1.3 GA Matching.....	124

5.2. The Comparison between the Standard VDC algorithm and the Optimized VDC algorithm based on GA	136
Chapter Six Conclusions and Future Work.....	146
6.1. Conclusions	146
6.2. Comparison with Related Works	147
6.3. Limitations	148
6.4. Future Work.....	149
The References	151
Appendixes.....	158
Appendix A:	158
Appendix B: Parts of the Matching Results.....	161
Appendix C: Parts of the GA Matching Results	171
Appendix D: Malware Types.....	174
Appendix E Antivirus Software Generations	178

List of Figures

Figure 1: Major steps of the research	5
Figure 2 : The Basic Algorithm of Clonal Selection	7
Figure 3 : The cells and secretions of the immune system	11
Figure 4 : How the immune system defends the body	12
Figure 5: Antibody molecule: V-region & C-region.	13
Figure 6: The Basic Algorithm of Negative Selection	15
Figure 7: The Clonal selection	16
Figure 8: The Basic Algorithm of Network Theory	18
Figure 9: Idiomatic Network	19
Figure 10: A simple virus structure.....	21
Figure 11: A classic parasitic virus	22
Figure 12: The Genetic Algorithm flowchart	26
Figure 13: Censoring - Generation of Valid Detector Set	33
Figure 14: Monitor Protected Strings for Changes	33
Figure 15: Kephart Immune System	35
Figure 16: The Stages of research.....	40
Figure 17: The VDC algorithm main steps flowchart	42
Figure 18: Cloning, Hypermutation, and Reselection	43
Figure 19: The VDC algorithm pseudo code	44
Figure 20: Matching the VDC algorithm flowchart	53
Figure 21: The pseudo code of the matching of the VDC algorithm	54
Figure 22: The GA as Parameters Optimizer	57
Figure 23: The flowchart of optimizing the VDC algorithm by the GA	58
Figure 24: The Initial Population	60
Figure 25: The first run Mean fitness & Best fitness	61
Figure 26: The first run final population	62
Figure 27: The second run Mean fitness & Best fitness	63
Figure 28: The second run final population	64
Figure 29: The third run Mean fitness & Best fitness	65
Figure 30: The third run final population.....	66
Figure 31: The forth run Mean fitness & Best fitness	67
Figure 32: The forth run final population	68
Figure 33: The fifth run Mean fitness & Best fitness	69
Figure 34: The fifth run final population	70
Figure 35: The sixth run Mean fitness & Best fitness	71
Figure 36: The sixth run final population	72
Figure 37: The seventh run Mean fitness & Best fitness	73
Figure 38: The seventh run final population	74
Figure 39: The eighth run Mean fitness & Best fitness	75
Figure 40: The eighth run final population.....	76
Figure 41: The ninth run Mean fitness & Best fitness	77
Figure 42: The ninth run final population	78
Figure 43: The tenth run Mean fitness & Best fitness	79
Figure 44: The tenth run final population.....	80
Figure 45: The eleventh run Mean fitness & Best fitness	81
Figure 46: The eleventh run final population	82
Figure 47: The twelfth run Mean fitness & Best fitness	83
Figure 48: The twelfth run final population	84
Figure 49: The Mean fitness of matching Sig1 run with 0% infected files	87
Figure 50: The Best fitness of matching Sig1 run with 0% infected files	88
Figure 51: The Mean fitness of matching Sig6 run with 5% infected files	89

Figure 52: The Best fitness of matching Sig6 run with 5% infected files	90
Figure 53: The Mean fitness of matching Sig1 run with 25% infected files	91
Figure 54: The Best fitness of matching Sig1 run with 25% infected files	92
Figure 55: The Mean fitness of matching Sig4 run with 50% infected files	93
Figure 56: The Best fitness of matching Sig4 run with 50% infected files	94
Figure 57: The Mean fitness of matching Sig1 run with 75% infected files	95
Figure 58: The Best fitness of matching Sig1 run with 75% infected files.....	96
Figure 59: The Mean fitness of matching Sig2 run with 100% infected files	97
Figure 60: The Best fitness of matching Sig2 run with 100% infected files	98
Figure 61: The Current Best Individual in the first GA optimization run	110
Figure 62: The Mean & Best Fitness in the first GA optimization run	111
Figure 63: The Current Best Individual in the second GA optimization run	111
Figure 64: The Mean & Best Fitness in the second GA optimization run	112
Figure 65: The Current Best Individual in the third GA optimization run	112
Figure 66: The Mean & Best Fitness in the third GA optimization run	113
Figure 67: The Current Best Individual in the forth GA optimization run	113
Figure 68: The Mean & Best Fitness in the forth GA optimization run	114
Figure 69: The first GA training run Mean Fitness & Best Fitness	116
Figure 70: The first GA training run final population	117
Figure 71: The second GA training run Mean Fitness & Best Fitness	118
Figure 72: The second GA training run final population	119
Figure 73: The third GA training run Mean Fitness & Best Fitness	120
Figure 74: The third GA training run final population	121
Figure 75: The forth GA training run Mean Fitness & Best Fitness	122
Figure 76: The forth GA training run final population	123
Figure 77: The Mean fitness of matching Sig1 run with 0% infected files	125
Figure 78: The Best fitness of matching Sig1 run with 0% infected files	126
Figure 79: The Mean fitness of matching SigGA4 run with 5% i nfecte d files.....	127
Figure 80: The Best fitness of matching SigGA4 run with 5% infected files	127
Figure 81: The Mean fitness of matching SigGA1 run with 25% infected files	128
Figure 82: The Best fitness of matching SigGA1 run with 25% infected files	129
Figure 83: The Mean fitness of matching SigGA4 run with 50% infected files	130
Figure 84: The Best fitness of matching SigGA4 run with 50% infected files	131
Figure 85: The Mean fitness of matching SigGA1 run with 75% infected files	132
Figure 86: The Best fitness of matching SigGA1 run with 75% infected files	132
Figure 87: The Mean fitness of matching SigGA1 run with 100% infected files	133
Figure 88: The Best fitness of matching SigGA1 run with 100% infected files.....	134

List of Tables

Table 1: The differences between CLONALG and VDC algorithm	50
Table 2: Files' pool contents	51
Table 3: The parameters values of the training phase	52
Table 4: The Parameters values of the matching phase	55
Table 5: The first Training run results	61
Table 6: The second Training run results	62
Table 7: The third Training run results	65
Table 8: The forth Training run results	67
Table 9: The fifth Training run results	69
Table 10: The sixth Training run results	70
Table 11: The seventh Training run results	73
Table 12: The eighth Training run results	75
Table 13: The ninth Training run results	77
Table 14: The tenth Training run results	79
Table 15: The eleventh Training run results	81
Table 16: The twelfth Training run results	83
Table 17: The Summary of the training results	85
Table 18: The results of the matching of Sig1 with 0% infected files	87
Table 19: The summary of matching of Sig5 and Sig8 with 0% infected files	88
Table 20: The results of the matching of Sig6 with 5% infected files	89
Table 21: The summary of the matching of Sig7, Sig10 and Sig11 with 5% infected files	90
Table 22: The results of the matching of Sig1 with 25% infected files	91
Table 23: The summary of the matching of Sig2 and Sig12 with 25% infected files ...	92
Table 24: The results of the matching of Sig4 with 50% infected files	93
Table 25: The summary of the matching of Sig5 and Sig8 with 50% infected files	94
Table 26: The results of the matching of Sig1 with 75% infected files	95
Table 27: The summary of the matching of Sig3, Sig6, Sig9 and Sig12 with 75% infected files	96
Table 28: The results of the matching of Sig2 with 100% infected files	97
Table 29: The summary of the matching of Sig4, Sig7, Sig10, Sig11 and Sig12 with 100% infected files	98
Table 30: The matching results	101
Table 31: The detection rate of the matching results	106
Table 32: The GA Optimization Runs Specifications	110
Table 33: The GA optimization results	114
Table 34: The GA training runs specifications	115
Table 35: The first GA training run results	116
Table 36: The second GA training run results	118
Table 37: The third GA training run results	120
Table 38: The forth GA training run results	122
Table 39: The Summary of the GA training results	124
Table 40: The GA matching runs specifications	124
Table 41: The results of the matching of SigGA2 with 0% infected files	125
Table 42: The results of the matching of SigGA4 with 5% infected files	126
Table 43: The results of the matching of SigGA1 with 25% infected files	128
Table 44: The summary of the GA matching results of SigGA2 with 25% infected files	129
Table 45: The results of the matching of SigGA4 with 50% infected files	130
Table 46: The results of the matching of SigGA1 with 75% infected files	131
Table 47: The results of the matching of SigGA1 with 100% infected files	133

Table 48: The summary of the GA matching of SigGA2, SigGA3 and SigGA4 with 100% infected files	134
Table 49: The GA Matching results.....	135
Table 50: The GA Detection Rate of the GA matching Results	136
Table 51: The comparison according to the Mean fitness and the Δ Mean fitness	137
Table 52: The Detection Speed summary	144

List of Abbreviations

AIS	Artificial Immune System
ALCFG	Arbitrary Length of Control Flow Graph
APCs	Antigen Presenting Cells
ANN	Artificial Neural Networks
CD	Compact Disc
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DoS	Denial of Service
DVD	Digital Video Disc or Digital Versatile Disc
EC	Evolutionary Computation
Fat	multiplying factor
GA	Genetic Algorithm
ICSA	Immune Clonal Strategy Algorithm
IS	Immune System
LAN	Local Area Network
MHC	Major Histocompatibility Complex
NK	Natural Killer
NGVCK	Next Generation Virus Creation Kit
NOP	No Operation
PC	Personal Computer
Pm	Hypermutation probability
REALGO	REtrovirus ALGORithm
ROC	Receiver Operating Characteristic
RNA	Ribonucleic Acid
SMS	Short Message Service
USB	Universal Serial Bus
VCL	Virus Creation Laboratory
VDC	Virus Detection Clonal
XOR	eXclusive OR

Virus Detection Using Artificial Immune System with Genetic Algorithm

Prepared by: Suha M. A. Afaneh

*Supervisors: Prof. Alaa Al-Hmami
Prof. Raed Abu Zitar*

Abstract

The protection against viruses is becoming increasingly difficult day by day, and they form risks on every one who uses computers, especially large companies and institutions. The viruses' intelligence is accumulated with time, and their signatures are changing continuously, which has made the Anti-viruses mission more complicated. Consequently, the issue of detecting viruses has been considered a hot and important topic.

This dissertation aims to develop an algorithm, which is based on the concepts of the Artificial Immune System to detect viruses.

Several studies have been concerned with the Artificial Immune System, which is inspired by the natural immune system of humans and animals. This subject is relatively considered recent, and is not matured yet. This system has been applied in different fields, most importantly viruses.

An algorithm has been suggested in this dissertation, which is based on the Artificial Immune System. A clonal selection Algorithm has been developed to detect viruses, which has been written, programmed and called the Virus detection Clonal (VDC) algorithm.

The VDC algorithm consists of three basic steps: cloning, Hypermutation and re-selection stochastically. Within the step of the reselection stochastically; there lay the virus's detection process, where the viruses' signatures are matched with the files.

The developed VDC algorithm is subjected to testing of two phases; training and matching. Two main parameters are determined; one of them is setting the number of signatures per clone (*Fat*), while the other defines the Hypermutation probability (*Pm*).

Later on the researcher used Genetic Algorithm as a tool, to improve the developed algorithm in searching the values of the main parameters (*Fat* and *Pm*) to reproduce better results.

The dissertation results have shown that the detection rate of viruses, by using the developed algorithm, is 94.4%. As for the detection rate of false positives, it has reached 0%. These rates are confirmed by the Genetic Algorithm.

The Dissertation has concluded that the developed algorithm (VDC), which is created to detect viruses, is good, and can be used in this field. The researcher has recommended that the developed algorithm can be utilized to be applied on other types of Malware that have signatures.

الكشف عن الفيروسات باستخدام نظام المناعة الاصطناعي والخوارزمية الجينية

إعداد: سهى عفانة
إشراف: الأستاذ الدكتور علاء الحمامي
الأستاذ الدكتور رائد أبو زعيتر
الملخص

Arabic Abstract

هدفت هذه الدراسة إلى تطوير خوارزمية استندت إلى مفاهيم نظام المناعة الاصطناعي للكشف عن الفيروسات.

اهتمت دراسات متعددة بنظام المناعة الاصطناعي الذي ألهم من نظام المناعة الطبيعية لدى الإنسان و الحيوان. و يعتبر هذا الموضوع حديثاً نسبياً و لما ينضج بعد. و طبق هذا النظام في مجالات مختلفة أهمها الفيروسات التي أصبحت الحماية منها تتزايد صعوبة يوماً بعد يوم ، وتشكل خطراً على كل من يستخدم الحاسوب، خاصة الشركات و المؤسسات الكبرى. كما غدا ذكاء الفيروسات يتراكم مع مرور الوقت، وبصماتها تتغير باستمرار مما جعل مهمة مضادات الفيروسات أكثر تعقيداً، لذا فإن موضوع الكشف عنها يعتبر من المواضيع الساخنة و الهامة.

اقترحت هذه الدراسة خوارزمية استندت إلى نظام المناعة الاصطناعي فطورت الباحثة خوارزمية الاختيار النسخي (Clonal Selection Algorithm) للكشف عن الفيروسات ، كتبتها وبرمجتها وأسمتها خوارزمية (VDC) .

تتكون الخوارزمية من ثلاث خطوات رئيسة هي : الاستنساخ، و التطفير(عمل طفرات)، وإعادة الاختيار عشوائياً. و ضمن خطوة إعادة الاختيار عشوائياً توجد عملية الكشف عن الفيروسات، حيث تتطابق كل من بصمات الفيروسات مع الملفات.

أخضعت الخوارزمية المطورة إلى الفحص الذي مر بمرحلتين هما : التعلم و التطابق، واختير متغيران رئيسان: أحدهما يحدد عدد البصمات في المستعمرة الواحدة (*Fat*)، و الثاني يحدد احتمالية التطفير (*Pm*).

ثم استخدمت الباحثة الخوارزمية الجينية كأداة لتحسين الخوارزمية المطورة، وذلك عن طريق البحث عن قيم المتغيريين الرئيسيين لتوليد أفضل النتائج .

أظهرت نتائج الدراسة أن معدل نسبة الكشف عن الفيروسات باستخدام الخوارزمية المطورة هي 94.4 % . أما نسبة الكشف عن ملفات نظيفة من الفيروسات على أنها مصابة (False Positive) فقد بلغت صفر بالمئة. وهاتان النسبتان أكدتهما الخوارزمية الجينية.

وخلصت الدراسة إلى أن الخوارزمية المطورة (VDC) التي أعدتها الباحثة للكشف عن الفيروسات جيدة، ويمكن استخدامها في هذا المجال. و أوصت بالإفادة منها في تطبيقاتها على مجالات فيروسية أخرى.

Chapter One Introduction

1.1 Introduction

Over the last few decades, there has been an ever increasing interest in the area of the biological inspired systems; such as Artificial Neural Networks (ANN), Genetic Algorithms (GA), Evolutionary Computation (EC), and Artificial Immune System (AIS). The AIS is relatively recent innovation, and its main idea is to build a system based on the immune system in human beings and other animals.

The biological immune system is a massively parallel system, which is robust, complex, and adaptive system, which is able to deal with changes in individual bodies, changes in environment, and even to adapt rapidly by defending the body from the foreign pathogens (infection elements such as microbes, virus, bacteria, tumor cells, ... etc).

There are two inter-related systems by which the body identifies foreign pathogens: the innate immune system and the adaptive immune system. The innate immune system is so called because the body is born with the ability to recognize certain microbes and immediately destroy them. The innate immune system can destroy many pathogens on first encounter. The adaptive immune system enables the body to recognize and respond to any pathogen, even if it has never faced the invader before. "The most important aspect of innate immune recognition is the fact that it induces the expression of co-stimulatory signals in Antigen Presenting Cells (APCs) that will lead to T cell activation, promoting the start of the adaptive immune response" [Castro, 2000a, Castro, 2002b and Aickelin, 2004].

The tissues and organs that compose the immune system are distributed throughout the body. They are known as lymphoid organs, which can be divided into primary (or central), responsible for the production of new lymphocytes, and secondary (or peripheral) where the lymphocyte repertoires match pathogen's antigen (each pathogen has a specific receptors called antigens).

The immune system's characteristics have been exploited to build algorithms, called AIS. The AIS is a diverse area of research that attempts to bridge the gap between immunology and engineering, and is developed through the application

of techniques, such as mathematical and computational modeling of immunology, abstraction from those models into algorithm (and system) design and implementation in the context of engineering. The AIS has become known as an area of computer science and engineering that uses immune system metaphors for the creation of novel solutions to problems, so it is a type of optimization algorithm inspired by the principles and processes of the vertebrate immune system. The AIS has several concepts: Clonal Selection, Negative Selection, and Network Immune Theory. This research deals with the clonal selection algorithm and more precisely the CLONALG, "which is primarily derived to perform machine-learning and pattern recognition tasks, and it is adapted to solve optimization problems, emphasizing multimodal and combinatorial optimization" [Castro, 2002b].

The AIS is used in many applications; such as Pattern recognition, Robotics, Control, Optimization, Learning, and Virus Detection. This research concentrates on the last mentioned application, which is virus using the clonal selection algorithm.

In 1984, mathematician Dr. Frederick Cohen introduced the term "computer virus"; therefore he became the "father" of computer viruses because of his early studies of them. Cohen introduced the computer virus based on the recommendation of his advisor, Professor Leonard Adleman, who picked the name from science fiction novels. Cohen's informal definition of a computer virus was: "A virus is a program that is able to infect other programs by modifying them to include a possibly evolved copy of itself" [Cohen, 1984]. The virus is one type of malware, while the malware is a variety of forms of hostile, intrusive, or annoying software or program code. Malware includes (beside computer viruses) worms, Trojan horses, logic bombs, backdoor, and other malicious and unwanted software. This research refers to all types of malware as it does with computer viruses. They can be harmless, such as a message on the screen, or damaging, such as the destruction of programs and data files.

The most common and straightforward method of preventing viral attacks on the network is virus signature technology. Virus signature databases have doubled or tripled over the past few years to keep up with the ever-increasing volume of malware; unfortunately, new viruses still get through, no matter how large the signatures become, or how fast they are updated.

Virus signature technology, despite the inherent limitation, is still an important part of the anti-virus strategy, and makes up a significant part of the defense, but having only a single line of defense is a very risky position. The vulnerability of virus signature technology is that it must be supplemented—not replaced—with complementary technologies to make sure that every possible attack vector is covered, preferably on multiple fronts. The inherent limitation of virus signatures is that it requires new viruses to be caught, included in the database, and updated on each individual system. There is a natural time lag between when a virus is first released into the wild, and when it is included in the database. There may be an additional time lag between when it is included in the database, and the individual enterprise updates their system [Secure, 2008].

This research proposes an artificial algorithm to fight viruses adaptively using the characteristics of our immune system, and then to optimize the parameters using the GA, which is "a search technique used in computing to find exact or approximate solutions to optimization and search problems. Genetic algorithms are categorized as global search heuristics" [Wikipedia, 2010].

This research shall be useful for antivirus companies and other researchers who are interested in this field.

1.2 The Statement of the Problem

It is getting harder everyday to protect the data from the dangers posed by computer viruses. These malicious programs have evolved into multiple forms and can be contracted through a variety of ways, including opening email attachments, opening spam, visiting corrupt websites, or they can be transferred via a Local Area Network (LAN). Such software has been used to compromise computer systems, to destroy their information, and to render them useless. It has been also used to gather information, such as passwords and credit card

numbers, and to distribute information, all without the knowledge of the system's users. As more and more novice users obtain sophisticated computers with high-speed connections to the Internet, the potential for further abuse is great. One shortcoming is that we must obtain a copy of a malicious program before extracting the pattern necessary for its detection. Obtaining copies of new or unknown malicious programs usually entails them to infect or attack a computer system. To complicate matters, writing malicious programs has become easier: there are virus kits freely available on the Internet. Individuals who write viruses have become more expert, often using mechanisms to change or obfuscate their code to produce so called polymorphic viruses. Indeed, researchers have recently discovered that simple obfuscation techniques foil commercial programs for virus detection. These challenges have prompted some researchers to investigate learning methods for detecting new or unknown viruses, and more generally, malicious code.

As all manner of information migrate online, malware has kept on track to become a huge source of individual threats. As security professionals close off points of access, attackers develop more sophisticated attacks in a continuously evolving game of cat and mouse. Today, profit models from malware are comparable to any seen in the legitimate world. But there is hope. Some studies have shown that while 25% of consumers facing Personal Computers (PCs) are infected by some sort of malware, the infection rate of the commercial PC sector is around half that rate. This difference is most likely a direct result of the efforts of security professionals working in commercial sites to defend against these threats [Creeger, 2010].

As mentioned previously, the computer viruses become cleverer day after day, and their signatures are varying continuously, so detecting them is becoming harder by the antivirus, as well, the signatures' databases and knowledge bases are enormously increasing, hence, they are not sufficient. Therefore, the problem of virus detection is a hot key issue. This research proposes an artificial solution to fight computer viruses adaptively, using the characteristics of the proposed immune system by producing a Virus Detection Clonal (VDC) algorithm, and then to optimize the parameters using the GA.

Figure (1) illustrates the major steps of the research. Building the VDC algorithm includes the design, implementation and testing of the algorithm. After that, the VDC algorithm is tuned by using the GA. This step includes the optimization and testing. The detailed steps are described in chapter three.

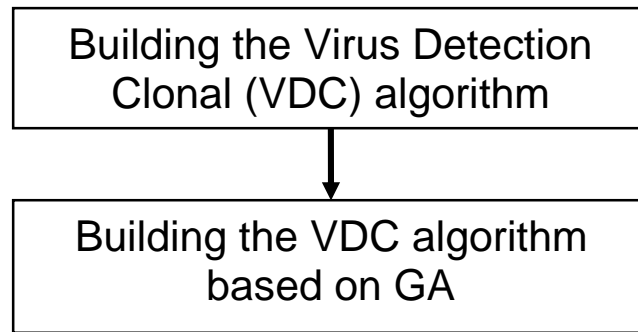


Figure 1: Major steps of the research

1.2.1 Research Questions

This research answers the following questions, which are related to the statement of the problem:

- 1) Will the proposed algorithm – the Virus Detection Clonal (VDC) algorithm be good in detecting computer viruses?
- 2) Will the tuning process by GA improve the VDC algorithm accuracy, and speed or not?
- 3) Are the VDC and GA applicable for solving the problem of computer viruses detection?

1.2.2. Definitions

The Immune System (IS):

It can be defined as a complex of cells, molecules and organs that represent an identification mechanism capable of perceiving and combating dysfunction in our own cells (infectious self) and the action of exogenous infectious microorganisms (infectious non-self). The interaction among the IS and several other systems and organs allows the regulation of the body and guaranteeing its stable functioning [Castro, 1999].

The Artificial Immune System (AIS):

Artificial immune systems can be defined as an abstract or metaphorical computational systems developed using ideas, theories, and components, extracted from the immune system (Natural). Most AIS aim at solving complex computational or engineering problems, such as pattern recognition, elimination, and optimization [Castro, 2002b].

Virus:

A computer virus is a computer program that can copy itself and infect a computer without permission or knowledge of the user [Wikipedia, 2010].

Anti-virus:

Antivirus software is a computer program that attempts to identify, neutralize or eliminate malicious software (malware) [Wikipedia, 2010].

Genetic Algorithm (GA):

A Genetic Algorithm (GA) is a search technique used in computation to find exact or approximate solutions (supervised or unsupervised) to optimization and search problems. Genetic algorithms are categorized as global search heuristics. Genetic algorithms are a particular class of evolutionary algorithms (EA) that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover [Wikipedia, 2010].

1.2.3 Thesis Contributions

The contribution of this research resides in the modernity of this field. Scientists have been interested in the AIS in the past few years. Although the virus problem is old, it has been considered as a progressing problem and very important since it affects every individual that uses computers. This means that millions of users are involved. Besides, the international studies consider it as an immature topic [Garrett, 2005]. So this research is about:

- 1) Producing virus detection algorithm which is called VDC algorithm that employs the Clonal selection algorithm illustrated in Figure (2), using signature scanner method.
- 2) Optimizing the parameters, that are used as the population strings in the GA, to increase the accuracy of the detection algorithm.

```
input  : S = set of patterns to be recognised, n the number of worst elements to select for removal
output : M = set of memory detectors capable of classifying unseen patterns

begin

  Create an initial random set of antibodies, A

  forall patterns in S do
    Determine the affinity with each antibody in A
    Generate clones of a subset of the antibodies in A with the highest affinity.
    The number of clones for an antibody is proportional to its affinity
    Mutate attributes of these clones to the set A , and place a copy of the highest
    affinity antibodies in A into the memory set, M
    Replace the n lowest affinity antibodies in A with new randomly generated antibodies
  end

end
```

Figure 2 : The Basic Algorithm of Clonal Selection

[EPSRC, 2008]

The AIS is still immature as a tool that can be efficiently used to learn and discover solutions for narrow domain knowledge based problems. Few works have been done on tuning this AIS and optimizing its performance, especially on the Clonal Selection algorithm, and more precisely to be used with virus detection. A challenging application such as the viruses' detection is a suitable benchmark for testing the tuned AIS compared to standard AIS. Moreover, the AIS lack formal description and stochastic analysis that helps in understanding the nature of this

real life adopted algorithm. The AIS optimizer is the GA that is a general-purpose optimization algorithm, which can be hybridized with the AIS to come up with efficient dynamic machine learning tool.

The negative selection algorithm (the self-non-self algorithm) has been used for virus detection [Forrest, 1994, Kephart, 1994a, D'haeseleer, 1996, Hang, 2005, Edge, 2006, Pietzowski, 2006 and Yu, 2009]. On the other hand, the clonal selection algorithm has not been used with this application yet, as according to the researcher knowledge, after searching the internet and the specialized journals. Thus, applying the clonal selection algorithm with virus detection is a brand new contribution.

The clonal selection principle describes the basic features of an immune response to an antigenic stimulus. It establishes the idea that only those cells that recognize the antigen proliferate, thus being selected against those that do not. The main features of the clonal selection theory are that: New cells are (cloned) copies of their parents, subject to a mutation mechanism (Hypermutation), Self-reactive cells are eliminated, and Proliferation and differentiation of mature cells on contact with antigens. When an antibody strongly matches an antigen the corresponding B-cell is stimulated to produce clones of itself that then produce more antibodies [Aickelin, 2004]. In this work the antigens is the computer viruses inside the infected files and the antibodies are the signatures. The signatures with high matching values (fitness) are selected to have the Cloning and the Hypermutation and the reselection processes; so that the cloning makes copies for the signatures with best fitness, then they are mutated to provide the ability of detecting viruses which are different in some genes, even if they did not attack previously (to defend adaptively). And in this research the reselection stochastically is added to the Clonal Selection Algorithm to guarantee choosing the best mutated signatures to be added to the signatures' pool.

1.3 The Thesis Structure

This thesis includes six chapters, where chapter one views the introduction of the research problem, its questions, the contribution of the research and the definitions of the main terminologies used in it. Chapter two presents the literature

review with the explanation of the Immune System, the Artificial Immune System, the viruses and their classifications, the antivirus and the Genetic Algorithm. After that, the related works which includes studies on the Artificial Immune System and the computer viruses, then studies that combine them together, are reviewed. Chapter three demonstrates the employed algorithm, how it is developed to be used in virus detection, the strategies that have been utilized in algorithm testing, and the optimization using GA. The results and analysis extracted from the testing of the algorithm are displayed in chapter four, while the results and analysis derived from the optimization using the GA are presented in chapter five. Finally chapter six includes the conclusions and recommendations for future researches.

Chapter Two

Literature Reviews and Related Works

This chapter includes two main sections: first, literature reviews, second, related works. In the literature reviews part there is an explanation of the following topics: the Immune System (IS) in the humans and animals, the Artificial Immune System (AIS), the viruses, the antivirus and the Genetic Algorithm (GA).

The immune system part considers its components such as cells and secretions, the basic defense mechanism, the antibodies and their role in the immune system, and finally the affinity definition.

The artificial immune system part handles its definition and the main following concepts: the negative selection, the clonal selection, and the network theory.

When it comes to the viruses, some of the subjects are included like the virus phases, the virus structure, the virus types, and the virus signature. The antivirus is defined, and then the generations are listed. After that the genetic algorithm part reviews its definition, and describes the main steps to perform it.

The second section demonstrates the related works, as some of these studies have explained the AIS; in regard to developing the AIS algorithms, applying them in different fields or optimizing them by GA. Some of the studies are interested in the extraction of new good viruses' signatures or proposing methods or techniques to detect viruses. The rest of the studies in this section are concerned with the AIS in the field of virus detection, as new algorithms or systems are proposed to detect viruses using the negative selection algorithm.

2.1. Literature Reviews

This section gives a brief description for the immune system, the artificial immune system, the viruses, the anti viruses and the genetic algorithm.

2.1.1. The Immune System (IS)

It can be defined as a complex of cells, molecules and organs that represent an identification mechanism capable of perceiving and combating the dysfunction in our own cells (infectious self) and the action of exogenous infectious

microorganisms (infectious non-self). The interaction among the IS and several other systems and organs allows the regulation of the body, and guaranteeing its stable functioning.

The immune system is composed of a tremendous variety of cells and secretions; which are complement, phagocytes, granulocytes and their relatives, and lymphocytes. The lymphocytes can be divided into: B cells, T cells, and natural killer cells. The T cells can be subdivided into three major subclasses: T helper, T killer, and T suppressor, as illustrated in Figure (3).

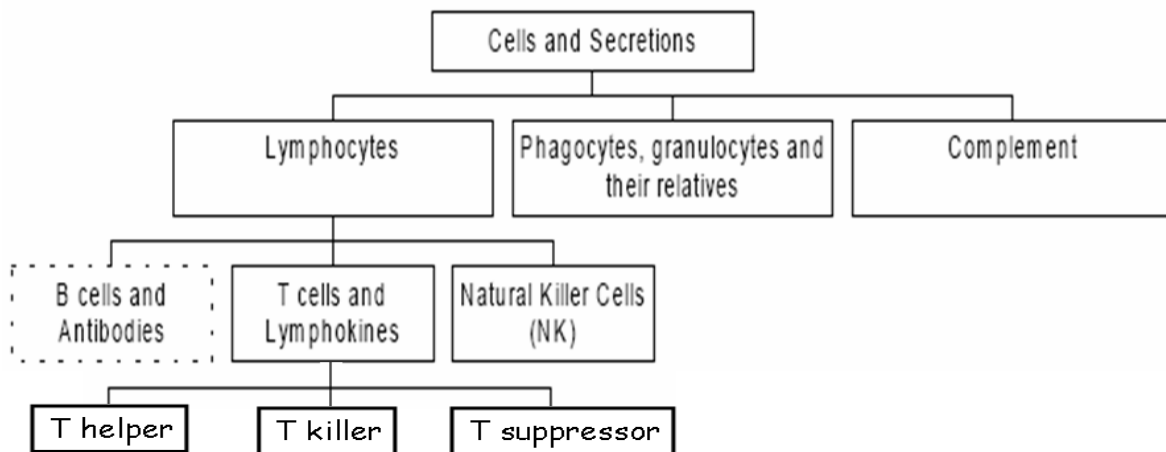


Figure 3 : The cells and secretions of the immune system

The main functions of the B cells include the production and secretion of antibodies as a response to pathogens. Each B cell is programmed to produce a specific antibody. The antibodies are specific proteins that recognize and bind to another particular protein. The production and binding of antibodies is usually a way of signaling other cells to kill, ingest or remove the bound substance. The T cells function includes the regulation of other cells' actions and directly attacking the host infected cells. The T helper cells are essential to the activation of all other cells in the immune system. The T killer cells are capable of eliminating microbial invaders, viruses or cancerous cells. Once activated, they inject noxious chemicals into the other cells, perforating their surface membrane and causing their destruction. The suppressor T lymphocytes are vital for the maintenance of the immune response, as they inhibit the action of other immune cells, and without their activity, immunity will certainly lose control resulting in allergic reactions and autoimmune diseases. The Natural Killer cells (NK) constitute another kind of lethal lymphocytes. Like the T killer cells, they contain granules

filled with powerful chemicals. On the other hand, unlike the T killer cells, they do not need to recognize a specific antigen before they start acting. They attack mainly tumors and protect against a great variety of infectious microbes [Castro, 2000a].

Figure (4) presents a simplified version of the basic immune mechanisms of defense through the following steps:

- I. Specialized Antigen Presenting Cells (APCs), which are phagocytes, roam the body, ingesting and digesting the antigens they find and fragmenting them into antigenic peptides.
- II. Pieces of these peptides are joined to Major Histocompatibility Complex (MHC) molecules and are displayed on the surface of the cell. Other white blood cells, called T cells, have receptor molecules that enable each of them to recognize a different peptide-MHC combination.

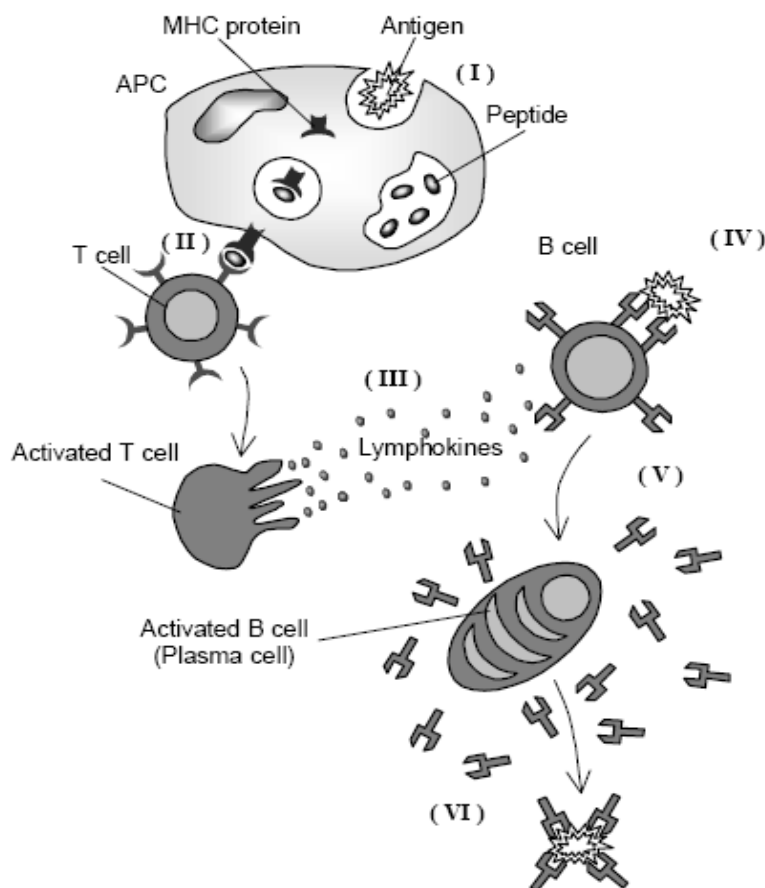


Figure 4 : How the immune system defends the body

[Castro, 1999]

- III. T cells activated by that recognition divide and secrete chemical signals that mobilize other components of the immune system.

IV. The B lymphocytes, which also have receptor molecules of a single specificity on their surface, respond to those signals. Unlike the receptors of T cells, however, those of B cells can recognize parts of the antigens free in solution, without MHC molecules.

V. When activated, the B cells divide and differentiate into plasma cells that secrete antibody proteins, which are soluble forms of their receptors.

VI. By binding to the antigens they find, antibodies can neutralize them or precipitate their destruction by complement enzymes or by scavenging cells.

Some T and B cells become memory cells that persist in the circulation, and boost the immune system's readiness to eliminate the same antigen if it presents itself in the future. Because the genes for antibodies in B cells frequently suffer mutation and editing, the antibody response improves after repeated immunizations, this phenomenon called affinity maturation [Castro, 1999].

The antibodies play a central role in the immune system. Antigens are diverse in structure, forcing the antibody repertoire to get larger. The basic unit of an antibody is composed of two regions; the variable region, or V-region, which is primarily responsible for antigen recognition and contains particularly variable sub-regions whose residues have been implicated in actual antigen contact. The constant regions, or C-regions, these regions are responsible for a variety of effector's functions, as illustrated in Figure (5).

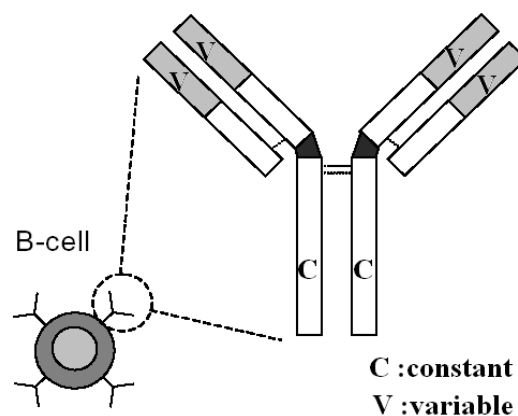


Figure 5: Antibody molecule: V-region & C-region.

[Castro, 1999]

Affinity:

The interaction of an antibody and an antigen is evaluated via a distance measured between their attribute strings; this measure distance is called affinity measure. One of the following can measure the affinity:

Euclidean distance

$$D = \sqrt{\sum_{i=1}^L (Ab_i - Ag_i)^2} \dots\dots\dots (2.1)$$

Manhattan distance

$$D = \sum_{i=1}^L |Ab_i - Ag_i| \dots\dots\dots (2.2)$$

Hamming distance

$$D = \sum_{i=1}^L \delta ; \text{ where } \delta = \begin{cases} 1 & \text{if } Ab_i \neq Ag_i \\ 0 & \text{otherwise} \end{cases} \dots\dots\dots (2.3)$$

Where D is affinity between an antibody and an antigen, Ab_i is an antibody where $Ab = \langle ab_1, ab_2, \dots, ab_L \rangle$, Ag_i is an antigen where $Ag = \langle ag_1, ag_2, \dots, ag_L \rangle$ [Castro, 1999].

2.1.2. The Artificial Immune System (AIS)

Artificial Immune System can be defined as "abstract or metaphorical computational system that is developed using ideas, theories, and components, extracted from the immune system. Most AIS aims to solve complex computational or engineering problems, such as pattern recognition, elimination, and optimization" [Castro, 2002b].

The AIS has several concepts: negative selection, clonal selection, and network immune theory.

The Negative Selection:

The process of deleting self-reactive lymphocytes is termed clonal deletion, and is carried out via a mechanism called negative selection that operates on lymphocytes during their maturation. For T-cells, this mainly occurs in the thymus, which provides an environment rich in antigen presenting cells that present self-antigens. Immature T-cells that strongly bind these self-antigens undergo a controlled death. Thus, the T-cells that survive this process shall be un-reactive to self-antigens. The property of lymphocytes that do not react to the self is called immunological tolerance.

Negative selection algorithms are inspired by the main mechanism in the thymus that produces a set of mature T-cells capable of binding only non-self antigens. The first negative selection algorithm was proposed by Forrest *et al* (1994) to detect data manipulation caused by a virus in a computer system. The starting point of this algorithm is to produce a set of self-strings, S , that define the normal state of the system. The task then is to generate a set of detectors, D , that only bind/recognize the complement of S , as illustrated in Figure (6). These detectors can then be applied to new data in order to classify them as being self or non-self [Castro, 2001, Castro, 2002a and EPSRC, 2008].

```

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

begin

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

end

```

Figure 6: The Basic Algorithm of Negative Selection

[EPSRC, 2008]

The Clonal Selection:

When stimulated, a B cell proliferates and secretes its receptor molecules as free antibodies. Antibodies thus can either be free or receptors attached to cells. Secretion requires that B cells become activated, undergo proliferation (cloning) and finally differentiate into plasma and memory cells. A clone is a cell, or a set of cells, which are the progeny of a single cell. A plasma cell is the one capable of secreting antibody with high rates, and a memory cell is the cell with high affinity with the antigen that will be rescued for a faster and stronger response to a previously seen (or related) antigen. Those cells that recognize antigens grow in concentration and affinity (affinity maturation), while those that do not die out.

This basic process of pattern recognition and selection is known as clonal selection and is similar to natural selection, except that it occurs on a rapid time scale on the order of days or weeks, within our bodies, as illustrated in Figure (7) [Castro, 2000a, and Castro, 2002a].

There are two important features of affinity maturation in B-cells that can be exploited from the computational viewpoint. The first one is that the proliferation of B-cells is proportional to the affinity of the antigen that binds it, thus the higher the affinity, the more clones produced. Secondly, the mutations suffered by the antibody of a B-cell are inversely proportional to the affinity of the antigen it binds. When applied to pattern matching, a set of patterns, S , to be matched are considered to be antigens. The task is to then produce a set of memory antibodies, M , that match the members in S . This is achieved via the algorithm of Figure (2) [EPSRC,2008].

The clonal selection principle is used to explain the basic features of an adaptive immune response to an antigenic stimulus. It establishes the idea that only those cells that recognize the antigens are selected to proliferate. The selected cells are subject to an affinity maturation process, which improves their affinity to the selective antigens [Castro, 2002b].

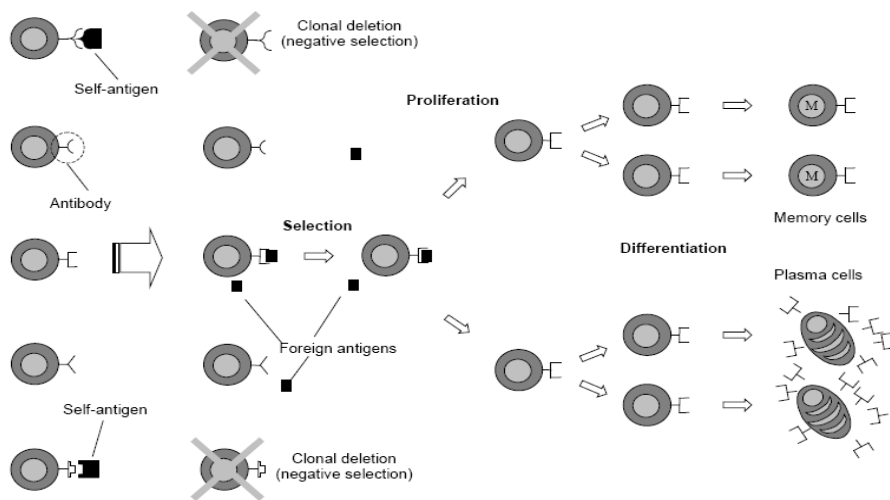


Figure 7: The Clonal selection

[Castro, 2000a]

The Network Theory:

The immune network theory helps to explain some of the observed emergent properties of the immune system, such as learning and memory.

The premise of immune network theory is that any lymphocyte receptor within an organism can be recognized by a subset of the total receptor repertoire. The receptors of this recognizing set have their own recognizing set and so on, thus an immune network of interactions is formed. Immune networks are often referred to as idiotypic networks. In the absence of foreign antigen, the immune system must display a behavior or activity resulting from interactions with itself, and from these immunological behavior interactions such as tolerance and memory emerge, as illustrated in Figure (8) [EPSRC, 2008].

The antibody molecules recognize a portion of the antigen called epitope. An idio type is defined as the set of epitopes displayed by the variable regions of a set of antibody molecules, and an idiotope is each single idiotypic epitope. While each B cell is known to have a single type of antibody, antigens typically have several different types of epitopes, and can be recognized by several different antibodies.

```

input  : S = set of patterns to be recognised, nt network affinity threshold,
         ct clonal pool threshold, h number of highest affinity clones, a number of
         new antibodies to introduce
output : N = set of memory detectors capable of classifying unseen patterns

begin

Create an initial random set of network antibodies, N
repeat

  forall patterns in S do
    Determine the affinity with each antibody in N
    Generate clones of a subset of the antibodies in N with the highest affinity. The number of clones for
    an antibody is proportional to its affinity
    Mutate attributes of these clones to the set A, a and place h number of
    the highest affinity clones into a clonal memory set, C
    Eliminate all elements of C whose affinity with the antigen is less than a predefined threshold ct
    Determine the affinity amongst all the antibodies in C and eliminate those antibodies whose affinity with each
    other is less than the threshold ct
    Incorporate the remaining clones of C into N
  end

  Determine the affinity between each pair of antibodies in N and eliminate all antibodies whose affinity
  is less than the threshold nt
  Introduce a random number of randomly generated antibodies and place into N

end until a stopping criteria has been met
end

```

Figure 8: The Basic Algorithm of Network Theory

[EPSRC, 2008]

The antibody portion responsible for matching (recognizing) an antigen is called paratope, also known as V-region, for variable regions. It is variable because it can alter its shape to achieve a better match with a given antigen. The strength and specificity of the Ag-Ab interaction is measured by the affinity of their match. As illustrated in Figure (9) [Castro, 2000a and Castro, 2002a].

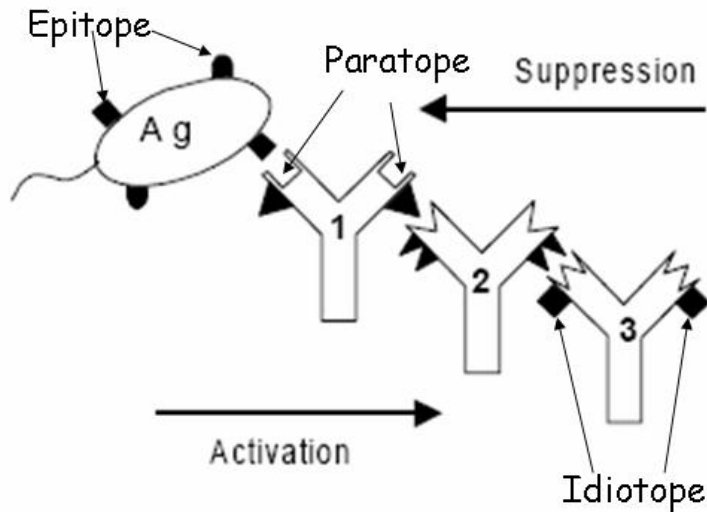


Figure 9: Idiomatic Network

[Castro, 1999]

2.1.3. The Viruses

Virus is a computer program written by a person that attaches itself to a program, propagates copies of itself to other programs, and infect any computer without the permission or knowledge of the user. The virus can spread when its host is taken to the target computer either by being sent over a network or the Internet, or carried on a removable medium such as a floppy disk, CD, DVD, or USB drive. Malware is software that is intentionally included or inserted in a system for a harmful purpose. The term "virus" is also commonly but erroneously used to refer to other types of malware. These types are listed in Appendix D.

This research deals with viruses. A computer virus carries in its instructions code the recipe for making copies of it-self. The typical virus becomes embedded in a program on a computer. Then, whenever the infected computer comes into contact with an uninfected piece of software, a fresh copy of the virus passes into the new program. Thus, the infection can be spread from computer to computer by unsuspecting users who either swap disks or send programs to one another over a network. In a network environment, the ability to access applications and systems services on other computers provides a perfect culture for the spread of virus.

A virus can do anything that other programs do. The only difference is that it attaches itself to another program and executes secretly when the host program is run. Once the virus is executing, it can perform any function, such as erasing files and programs.

During its lifetime, a typical virus goes through the following phases:

1. **Dormant phase** - the virus is idle
2. **Propagation phase** - the virus places an identical copy of itself into other programs
3. **Triggering phase** – the virus is activated to perform the function for which it is intended
4. **Execution phase** – the function is performed.

A virus can be pre-pended or post-pended to an executable program, or it can be embedded in some other fashion. The key to its operation is that the infected program, when invoked, will first execute the virus code and then execute the original code of the program.

A very general depiction of virus structure is shown in Figure (10). In this case, the virus code, V, is pre-pended to infected programs, and it is assumed that the entry point to the program, when invoked, is the first line of the program [Stalling, 2007].

An infected program begins with the virus code and works as follows: the first line of code is a jump to the main virus program. The second line is a special marker that is used by the virus to determine whether a potential victim program has already been infected with this virus. When the program is invoked, control is immediately transferred to the main virus program. The virus program first seeks out uninfected executable files and infects them. Next, the virus may perform some action, usually detrimental to the system. This action can be performed every time the program is invoked, or it can be a logic bomb that is triggered only under certain conditions. Finally, the virus transfers control to the original program. If the infection phase of the program is reasonably rapid, a user is unlikely to notice any difference between the execution of an infected and uninfected program.

```

program V :=
{goto main;
 1234567;

  subroutine infect-executable :=
    {loop:
      file := get-random-executable-file;
      if (first-line-of-file = 1234567)
        then goto loop
        else prepend V to file; }

  subroutine do-damage :=
    {whatever damage is to be done}

  subroutine trigger-pulled :=
    {return true if some condition holds}

main:  main-program :=
      {infect-executable;
      if trigger-pulled then do-damage;
      goto next;}

next:

}

```

Figure 10: A simple virus structure

[Stalling,2007]

There has been a continuous arms race between virus writers and writers of antivirus software since viruses first appeared. As effective countermeasures are developed for existing types of viruses, new types have been developed. The following categories are suggested as being among the most significant types of viruses:

- **Parasitic virus:** the traditional and still most common form of virus. A parasitic virus attaches itself to the executable files and replicates, when the infected program is executed, by finding other executable files to infect. As shown in Figure (11) such viruses overwrite the top of the host with their own code and save the top of the original host program in the end, usually virus-size long.

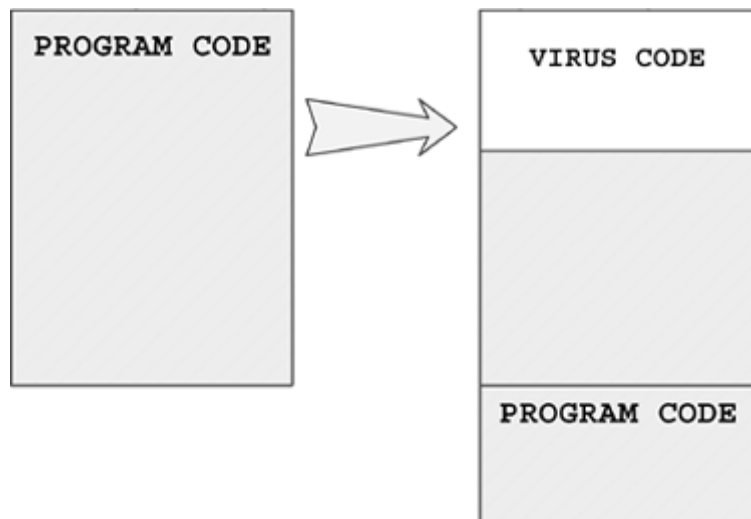


Figure 11: A classic parasitic virus

[Szor,2005]

- **Memory-resident virus:** a virus which remains in the memory after the initialization of the virus code, and infects every program that is executed in the main memory.
- **Boot sector virus:** a virus which infects a master boot record or boot record and spreads when a system is booted from the disk containing the virus. Knowing that today the boot infection technique is rarely used.
- **Stealth virus:** a virus is explicitly designed to hide from detection by antivirus software, by intercepting the anti-virus software's request to read the file and passing the request to the virus, instead of the operating system. The virus can then return an uninfected version of the file to the anti-virus software, so that it seems that the file is benign.
- **Polymorphic virus:** a virus that mutates with every infection, making detection by the "signature" of the virus impossible; it creates copies during replication that are functionally equivalent, but have distinctly different bit patterns. In this case, the "signature" of the virus varies with each copy. To achieve this variation, the virus may randomly insert superfluous instructions, or interchange the order of independent instructions. Encryption is the most common method to hide a code. With encryption, the main body of the code (also called its payload) is encrypted and appears meaningless. For the code to function as before, a decryption function is added to the code. When the

- code is executed this function reads the payload and decrypts it before executing it in turn. Encryption alone is not polymorphism. To gain polymorphic behavior, the encryptor/decryptor pair is mutated with each copy of the code. This allows different versions of some code while all functions the same. Polymorphic viruses can mutate their decryptors to a high number of different instances that can take millions of different forms; this means that the Virus writers usually waste time to create a new polymorphic virus. While a researcher is able to deal with the detection of such virus in a shorter time. "There are a surprisingly low number of efficient external polymorphic engines" [Szor, 2005].
- **Metamorphic virus:** as with a polymorphic virus, a metamorphic virus mutates with every infection. The difference is that a metamorphic virus rewrites itself completely in each iteration, increasing the difficulty of detection. While, polymorphic virus ciphers its original code to avoid pattern recognition. Metamorphic viruses may change their behavior as well as their appearance. Often, it does this by translating its own code into a temporary representation, editing the temporary representation of itself, and then writing itself back to a normal code again. This procedure is done with the virus itself, and thus also the metamorphic engine itself undergoes changes. This is used by some viruses when they are about to infect new files, and the result is that the "children" will never look like their "parents".
- **Virus-creation toolkit:** a tool which enables non-expert users to create quickly a number of different viruses. Although viruses created with toolkits tend to be less sophisticated than viruses designed from scratch, the absolute number of new viruses that can be generated creates a problem for antivirus schemes.
- **Macro Viruses:** a macro is an executable program that is written in a macro language; which is usually some form of a basic programming language. The macro virus takes advantage of a feature found in office applications, so that the programs may run automatically when the document is opened. Successive releases of Office provide increased protection against macro viruses; hence, they no longer are an epidemic viruses' threat.

- **E-Mail Viruses: an email virus** is rapidly spreading virus via emails, they can make use of Microsoft Word macro embedded in an attachment. If the recipient opens the e-mail attachment, the Word macro is activated. Then the e-mail virus sends itself to everyone on the mailing list in the user's e-mail package, and then the virus does local damage. Or they can be activated merely by opening an e-mail that contains the virus rather than opening an attachment. The virus uses the visual Basic scripting language supported by the e-mail package.

Virus signature:

Signature is a characteristic byte-pattern that is a part of a certain virus, or family of viruses, or short identifiers which consist of sequences of bytes in the machine code of the virus. "A good signature is one that is found in every object infected by the virus" [Kephart, 1994b].

In the antivirus world, a signature is an algorithm or hash (a number derived from a string of text) that uniquely identifies a specific virus. Depending on the type of scanner being used, it may be a static hash which, in its simplest form, is a calculated numerical value of a snippet of code unique to the virus. Or, less commonly, the algorithm may be behavior-based, i.e. if this file tries to do X, Y, Z, they must be flagged as suspicious and prompt the user for a decision. Depending on the antivirus vendor, a signature may be referred to as a signature, a definition file, or a DAT file.

A single signature may be consistent among a large number of viruses. This allows the scanner to detect a brand new virus that has never even seen before. This ability is commonly referred to as either heuristics or generic detection. The ability to detect heuristically or generically is significant, given that most scanners now include in excess of 250k signatures, and the numbers of new viruses being discovered continues to increase dramatically year after year.

The reoccurring need to update each time a new virus is discovered, because new signatures must be created. This new virus may not be detectable by an existing signature, or may be detectable but cannot be properly removed, because its behavior is not totally consistent with previously known threats. After the new signature has been created and tested by the antivirus vendor, it is

pushed out to the customer in the form of signature updates. These updates add the detection capability to the scan engine. In some cases, a previously provided signature might be removed or replaced with a new signature to offer better overall detection or disinfection capabilities.

Depending on the scanning vendor, updates may be offered hourly, or daily, or sometimes even weekly. This period depends on the scanner type. For example, adware and spyware are not nearly as prolific as viruses, thus typically an adware/spyware scanner may only provide weekly signature updates (or even less often). Conversely, a virus scanner must contend with thousands of new threats discovered each month and therefore, signature updates should be offered at least daily.

Of course, it is simply not practical to release an individual signature for each new virus discovered, thus antivirus vendors tend to release on a set schedule, covering all of the new malware they have encountered during that time frame. If a particularly prevalent or menacing threat is discovered between their regularly scheduled updates, the vendors will typically analyze the malware, create the signature, test it, and release it out-of-band (which means, release it outside of their normal update schedule) [Landesman, 2008].

2.1.4. The Anti-virus

Antivirus software is a computer program that attempts to identify, neutralize or eliminate malicious software (malware) [Wikipedia, 2010].

According to Stalling (2007) there are four generations of antivirus software:

First generation: simple scanners.

Second generation: heuristic scanners.

Third generation: Activity Traps.

Fourth-generation: full-featured protection.

These generations are described briefly in appendix E.

2.1.5 Genetic Algorithm

Goldberg (1989) described Genetic Algorithms as: search procedures based on the mechanism of natural selection and natural genetics, i.e. "they are general search and optimisation algorithms that use the theories of evolution as a tool to solve problems in science and engineering. This creates an evolving spopulation of candidate solutions to the particular problem, using operations inspired by natural genetic variation and natural selection". Figure (12) illustrates the main four steps for this algorithm.

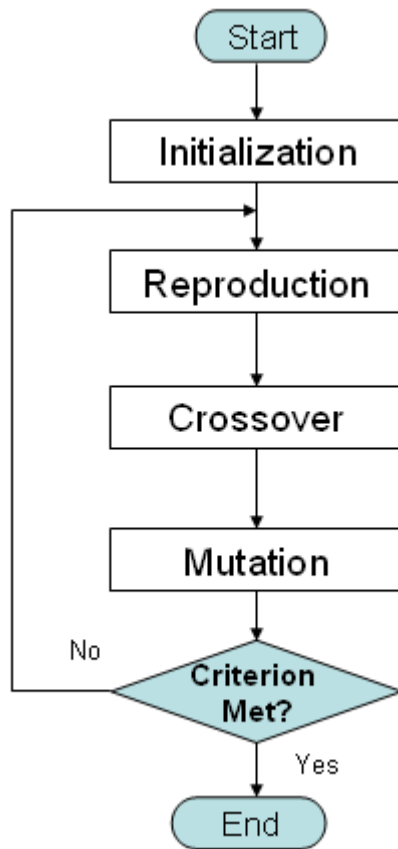


Figure 12: The Genetic Algorithm flowchart

Initialization

Initially many individual solutions are randomly generated to form an initial population. The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Traditionally, the population is generated randomly, covering the entire range of possible solutions (the search space). Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found.

Reproduction

Reproduction is a process in which individual strings are copied according to their fitness function values; the strings with higher values have higher probability of contributing in the next generation.

The probability of a string to be selected is:

$$P(i) = \frac{F(i)}{\sum_{j=1}^n F(j)} \dots\dots\dots (2.4)$$

Where: $P(i)$: is the probability that the i^{th} candidate string is selected.

$F(i)$: is the fitness of the i^{th} string.

n : is the total number of strings in the population.

The strings with higher probability are copied to the new population with the same number of strings in the current population.

Crossover

Each pair of the selected strings is subjected to the probability of crossover (p_c), by selecting two strings randomly as two parents, then choosing a random integer number (between 1 and $l-1$; where l is the length of string). As an example, consider the following two strings with the integer number 4.

$A_1 = 0110 \ 1111$
 $A_2 = 1110 \ 0000$

The bits up to this point (4) in the first individual get swapped with the corresponding bits from the second individual, to form two new strings

$A'_1 = 0110 \ 0000$
 $A'_2 = 1110 \ 1111$

This way is a single-point crossover. There is another way by choosing a number of crossover points randomly. The bits between every second grouping of bits (i.e. bits between every second crossover point) are swapped between two individuals to produce the new strings. This is called Multi-point crossover.

Mutation

As in crossover, the mutation operator also has the effect of creating new population members. It can help creating strings that will not otherwise be formed

by selection and crossover alone. The mutation simply means changing a 1 to 0 and vice versa according to a small probability (p_m), and it is usually (~ 0.001). Each iteration of this process is called a generation. The entire set of generations is called a run. The fittest member over the entire run is typically taken as the required solution.

In this research the GA is used to tune parameters of the AIS system.

2.2. Related Works

The related studies are classified into three parts: the artificial immune system, the computer virus, and the computer virus with AIS.

2.2.1. Studies about the Artificial Immune System

Castro and Zuben have published many papers in this subject. In their paper in 2000 they explored basic aspects of the immune system, and proposed a novel immune network model with the main goals of clustering and filtering redundant data from problems described by a set of discrete samples. Their concern was to show that immune concepts could be used to develop novel computational tools for data processing. As important results of their model, the network evolved was capable of reducing redundancy, describing data structure, shapes and their cluster inter-relations [Castro, 2000a]. Also in their paper in 2002, they proposed a computational implementation of the clonal selection principle that explicitly took into account the affinity maturation of the immune response. The general algorithm, named CLONALG, was primarily derived to perform machine-learning and pattern recognition tasks, and then it was adapted to solve optimization problems, CLONALG was also contrasted with evolution strategies and genetic algorithms [Castro, 2002b].

Castro and Timmis (2002) work introduced AIS as computational intelligence paradigm to perform pattern recognition. And they concluded a comparison between AIS and artificial neural networks as pattern recognition paradigms. They reviewed three classes of artificial immune system algorithms to perform

pattern recognition: 1) negative selection, 2) clonal selection, and 3) immune network models. In negative selection, a pattern recognition system was designed by learning information about the complement set of the patterns to be recognized. Clonal selection algorithms learnt to recognize patterns through an evolutionary-like procedure. Finally, immune network models were peculiar because they carried information about the patterns to be recognized and, also, they had knowledge of themselves, i.e., a notion of self-identification. All algorithms were population based with the knowledge distributed among the components of the system.

According to Castro and Timmis, Most computational immunology algorithms, which composed particular cases of artificial immune systems, were based upon the negative selection algorithm to protect computers and networks of computers from viruses, unauthorized users, etc. Additionally, the application of other models, including the immune network and clonal selection algorithms, to other types of pattern recognition applications, such as character recognition, data analysis, clustering and classification were discussed. Then it was followed with a theoretical comparison between artificial immune systems and neural network models for pattern recognition. Aspects such as the basic units composing each system, their respective types of adaptation mechanisms, the types of memory presented, and how they presented generalization capabilities were stressed.

In Yang's paper (2006), he investigated several GAs inspired by the ideas of biological immune system and transformation schemes for dynamic optimization problems. Diversity and memory were mechanisms integrated into genetic algorithms to enhance their performance for problem optimization in dynamic environments. Yang proposed an aligned transformation operator, and combined it to the immune system based genetic algorithm to deal with dynamic environments. Using a series of systematically constructed dynamic test problems, experiments were carried out to compare several immune system based genetic algorithms, including the proposed one, and two standard genetic algorithms enhanced with memory and random immigrants respectively.

In the paper of Liu et al (2006), they presented a novel artificial intelligent algorithm, named Immune Clonal Strategy Algorithm (ICSA). The new immune operator, Clonal Operator, inspired by the Immune System was discussed firstly. Three different mutation mechanisms were used in ICSA; Gauss Mutation, Cauchy Mutation, and Mean Mutation, and then based on these three methods a comparison with Classical Evolutionary Strategy on a set of benchmark functions was made, the numerical results showed that ICSA was capable of avoiding prematurity, increasing the converging speed and keeping the variety of solutions. The clonal operator is an antibody random map induced by the affinity including three steps: clone, clonal mutation and clonal selection. Here, the affinity between antibody and antigen are similar to the definitions of the objective function and restrictive condition, the possible solution, match between solution and the fitting function in AIS. They found that the essential of the clonal operator was producing a variation population around the parents according to their affinity, and then the searching area was enlarged. Compared with Classical Evolutionary Strategy, ICSA was convergent faster and the diversity was much better.

2.2.2 Studies about Computer Viruses

In their paper, Kephart and Arnold (1994b) had developed a statistical method for automatically extracting good signatures from the machine code of a virus. The basic idea was to characterize statistically a large corpus of programs, and then to use this information to estimate false-positive probabilities for proposed virus signatures. In effect, the algorithm extrapolated from the corpus to the much larger universe of executable programs that did or might exist. In practice, signatures extracted by this method were very unlikely to generate false positives, even when the scanner that had employed them permitted some mismatches. That was accomplished in two phases. First, a set of signatures which were likely to appear in each instance of the virus was generated. Second, one or a few signatures that minimized the false-positive probability were chosen from this set.

Chess and White in their paper for IBM Company (2000) pointed out that there were computer viruses with no algorithms which could be detected. Every widely-deployed virus detection program in use that day claimed to find a virus in at least some non-viral objects (a false positive), because the methods used for detection were approximate, based on the presence of a particular binary string in a certain place, on the calculation of the finite-size checksum of a macro, on a certain pattern of changes to a file, and so on. Producers of anti-virus software of course tried to minimize the number of actual non-viral programs that were falsely detected. But no one worried about the fact that the algorithms used to detect viruses produced false positives on an enormous number of non-viral objects that had never been presented on any actual user's computer.

According to Chess and White, acceptable virus detection, in the real world, involves detecting all viable instances of the virus in question, and preferably some number of minor variants of it, while falsely detecting the virus in only a vanishingly small number of innocent programs that are actually present on a computer somewhere. It is helpful to have a formal characterization of this more realistic notion of detection; theorists in the area of computer virus protection may usefully work toward such a characterization.

Kolter and Maloof (2006) described the use of machine learning and data mining to detect and classify malicious executables as they appeared in the wild. They gathered 1,971 benign (system and non-system executables) and 1,651 malicious executables and encoded each as a training example using n-grams of byte codes as features. Such processing resulted in more than 255 million distinct n-grams. After selecting the most relevant n-grams for prediction, they evaluated a variety of inductive methods, including naive Bayes, decision trees, support vector machines, and boosting. Ultimately, boosted decision trees outperformed other methods with an area under the Receiver Operating Characteristic (ROC) curve of 0.996. Results suggested that their methodology would scale to larger collections of executables. They also evaluated how well the methods classified executables based on the function of their payload, such as opening a backdoor and mass-mailing. Areas under the ROC curve for detecting payload function

were in the neighborhood of 0.9, which were smaller than those for the detection task. However, they attributed this drop in performance to fewer training examples and to the challenge of obtaining properly labeled examples, rather than to the failing of the methodology or to some inherited difficulty of the classification task. Finally, they applied detectors to 291 malicious executables discovered after they gathered their original collection, and boosted decision trees which achieved a true-positive rate of 0.98 for a desired false-positive rate of 0.05. This result was particularly important, for it suggested that their methodology could be used as the basis for an operational system, for detecting previously undiscovered malicious executables.

The paper of Preda et al (2007) took the position that the key to malware identification lied in their semantics, not like the malware detectors that were presented at that time which worked by checking for "signatures", and attempted to capture (syntactic) the characteristics of the machine-level byte sequence of the malware. This reliance on a syntactic approach made such detectors vulnerable to code obfuscations, increasingly used by malware writers that altered syntactic properties of the malware byte sequence without significantly affecting their execution behavior. Therefore, they had proposed a semantics-based framework for reasoning about malware detectors and proving properties such as soundness and completeness of these detectors. Their approach used trace semantics to characterize the behaviors of malware, as well as the program being checked for infection, and used abstract interpretation to "hide" irrelevant aspects of these behaviors.

Al Daoud et al (2009) proposed an efficient and novel method based on Arbitrary Length of Control Flow Graphs (ALCFG) and similarity of the aligned ALCFG matrix. They used the metamorphic viruses that were generated by two tools; namely: Next Generation Virus Creation Kit (NGVCK0.30) and Virus Creation Lab for Windows 32 (VCL32). The results showed that all the generated metamorphic viruses could be detected by using the suggested approach, while less than 62% were detected by well-known antivirus software.

2.2.3. Studies about Computer Viruses with AIS

Forrest (1994) had worked on self and non-self discrimination in computer viruses; he had proposed an algorithm of two phases: the first phase was responsible for generating a set of detectors; where each detector was a string that did not match any of the protected data. This censoring phase is illustrated in Figure (13). The second phase was responsible for monitoring the protected data by comparing them with the detectors. As shown in Figure (14).

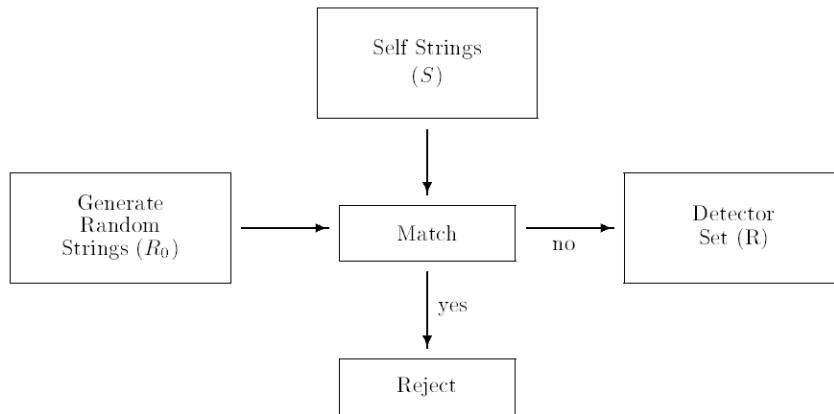


Figure 13: Censoring - Generation of Valid Detector Set

[Forrest, 1994]

The matching process had not considered being in a perfect match manner, since it was extremely hard to be found between strings of any reasonable length; a partial matching rule had been needed. By using a matching rule that looked for r contiguous matched between symbols in corresponding positions.

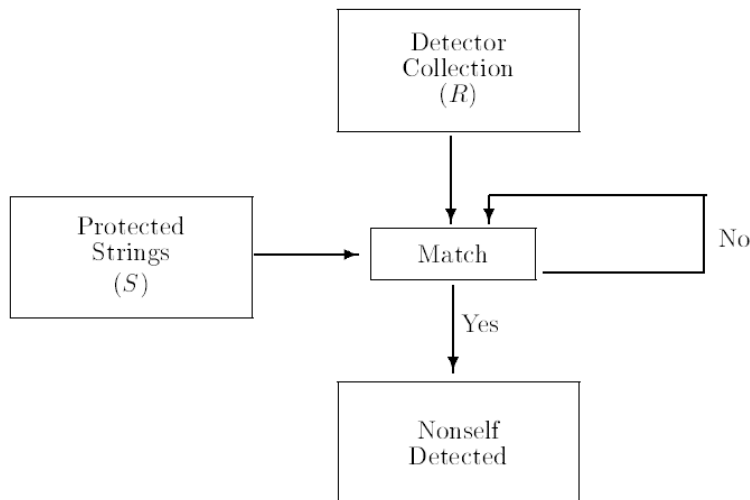


Figure 14: Monitor Protected Strings for Changes

[Forrest, 1994]

In the system developed by Kephart (1994a), a set of antibodies that previously did not encounter computer viruses or worms (agents) was generated so as to promote a faster and stronger response to future infecting agents. He was also concerned about minimizing the risk of an autoimmune response, in which the computer immune system would mistakenly identify legitimate software as being undesirable.

A particular virus was recognized via an exact or fuzzy match to a relatively short sequence of bytes occurring in the virus (signature). The process by which the proposed computer immune system established whether new software contained a virus had several stages. Integrity monitors, which used checksums to check for any changes to programs and data files, had a notion of self that was: any differences between the original and current versions of any file were flagged, as were any new program. However, evidence of a non-self entity was not by itself enough to trigger a computer immune response. Mechanisms that employed the complementary strategy of “knowing the enemies” were also brought into play.

The capture of a virus sample by decoy programs was somewhat analogous to the ingestion of antigen by APCs. In the computer immune system, the infected decoys were then processed by another component of the immune system, called a signature extractor, so as to develop a recognizer for the virus.

The computer immune system had an additional task to attempt to extract from the decoys information about how the virus attached to its host, so that infected hosts could be repaired (if possible). Hence, the system automatically developed both a recognizer and a repair algorithm appropriate to the virus.

Viral self-replication was dealt with self-replication, in the sense that, detection of a virus by a single computer could trigger a wave of kill signals that propagated along the path taken by the virus, destroying the virus in its wake. Figure (15) depicts the main components and their respective function within the immune system.

If a virus-like anomaly was detected by the immune system, the first response would be to trigger a scan for known viruses. If the anomaly could not be attributed to a known virus, the immune system would try to lure any virus that

might be presented in the system to infect a diverse suite of decoy programs. From time to time, each of the decoy programs was examined to see if it had been modified. If one or more had been modified, it was almost certain that an unknown virus was loose in the system, and each of the modified decoys contained a sample of that virus. The next step would be to extract a signature for the virus automatically. In addition, another automatic virus analysis tool under development in laboratory would determine how the virus attached to host programs, and extract information that would allow any program infected by the virus to be repaired.

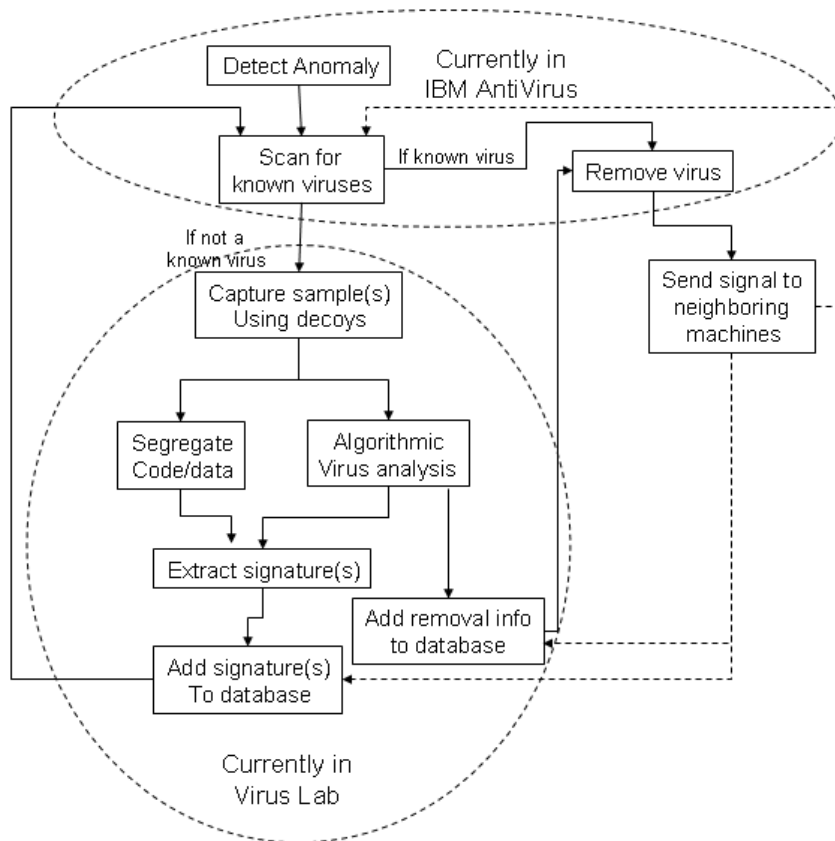


Figure 15: Kephart Immune System

[Kephart, 1994c]

Having automatically developed both a recognizer and a repair algorithm appropriate to the virus, the information could be added to the corresponding databases. If the virus was ever encountered again, the immune system would recognize it immediately as a known virus. A computer with an immune system

could be thought of as ill during its first encounter with a virus, since a considerable amount of time and energy (or CPU cycles) would be expended to analyze the virus. An additional feature, kill signal, would be used by a computer to inform neighboring computers on the network that it was infected. The signal would also convey to the recipient any signature or repair information that might be of use in detecting and eradicating the virus. If the recipient found that it was infected, it would send the signal to its neighbors, and so on. If the recipient is not infected, it will not pass along the signal, but at least it will receive the database updates; which effectively will immunize it against that virus. This approach unified a wide variety of computer and data security problems of distinguishing self from other.

Edge et al (2006) created an algorithm to be used for computer virus detection; they had developed an Artificial Immune System Genetic Algorithm which is called REtrovirus ALGOriithm (REALGO) based on the human immune system's use of reverse transcription Ribonucleic Acid (RNA). The REALGO algorithm provided memory such that during a complex search the algorithm could revert back to, and attempt to mutate in a different "direction", in order to escape local minima. In lieu of non-existing virus generic templates, validation was addressed by using an appropriate variety of function optimizations with landscapes believed to be similar to that of virus detection.

The results showed that the REALGO algorithm was superior for optimizing complex functions but not necessarily for easier ones. This was due to the fact that the REALGO algorithm added complexity to the search that was not needed for simple searches. Once the complexity of the search landscape was greater than that of the algorithm, the REALGO algorithm became superior. Preliminary results had shown that the REALGO algorithm did indeed provide a superior search for complex landscapes, due to its ability to revert back to a previous good solution if the search stagnated. Rather than resetting to a new starting point, the search was able to attempt a search in a new direction from this previous good solution, without having to waste generations for the initial convergence.

The next step was to integrate it into a complete virus detector [Edge, 2006].

Unterleitner proposed in his book (2008) a model of the computer immune system (CIS), which was based on several mechanisms of the human immune system. His system targeted the Internet worms, different kinds of viruses and shell codes which were possibly polymorph. The implementation of his work was intended to shield computers in a LAN from new network driven attack attempts. Each network node was equipped with a sensor, which has been used to train an individual set of detectors. That set evolved in response to the network traffic at that node. Consequently, the set of detectors was different in each node, which led to having the whole network system highly diverse.

According to Unterleitner, applying the CIS with multiple independent sensors across a network ensured a distributed detection system which was not centrally or hierarchically controlled. His implementation took advantage of the network intrusion detection system called Snort, which provided the basis for processing the network packets. His model proposed a hybrid detection system, which was a combination between the Misuse detection and the Anomaly detection.

Two different methods were checked if they had been able to significantly separate self elements from non-self elements. The first method was the widely used Pearson correlation coefficient that was based on associations between data sets and the algorithm. The second method included four algorithms: Hamming Distance, Levenshtein or Edit Distance, R-Contiguous symbols and Longest Common Subsequence.

The suggested Anomaly detection system was performing well, if three requirements had been applied to the training of the detector set. The first requirement was to have a stable definition for the Self set and it influenced the training success. The next requirement was the number of detectors in the set, which must be chosen properly. This requirement affected the detection rate, as the detection capability of the system was influenced by the size of the detector set. For example for the big Self set, a higher number of detectors had to be selected. The last requirement was the right proportion of Self to Non-self in the inspected data, which had an effect on the practical applicability of the detection system.

The system detected attacks, if all three requirements were met. If the implementation used a detector set that had been trained on some fields of the Packet Header, then this the attacks would be detected.

Yu et al (2009) had presented a novel Windows PE virus detection approach that drew inspiration from artificial immune system and the structure of the relocation module of the virus. The structure of Windows PE virus was sufficiently analyzed. The dynamic evolution of self and non-self, the presentation of the antigen, and the generation of antibody were proposed. The experiment was conducted and its results indicated that this approach did not only have relatively higher detection rate of unknown Windows PE virus than the earlier known methods, but also had better capability of self-adaptive and self-learning.

The experiment was conducted in the computer virus and anti-virus laboratory, computer network and information security institute of Sichuan University. Since there was no benchmark data set available for the detection of computer viruses, unlike intrusion detection, the data sets including 100 viruses and 500 benign executables were collected from the website VX Heavens, and from system32 folder in windows, respectively. The experimental results showed that the proposed approach which was non-signature based not only had a higher detection rate, low false-positive rate and low omitting rate, but also its efficiency was better than the currently mature antivirus products.

Chapter Three Methodology

This chapter represents the methodology of creating the proposed algorithm: Virus Detection Clonal (VDC) algorithm, which is inspired from the Clonal Selection Algorithm. This chapter includes three sections: data sets, research tools and research stages which are emanated from the literature reviews and related works.

3.1. Data sets

Since there is no benchmark data set available for the detection of computer viruses, unlike Intrusion detection, the data sets including 100 viruses' signatures have been collected from the website VX Heavens (2010). The 500 benign files have been gathered from the windows XP files. The researcher has formatted a PC then installed Windows XP, after that, she has chosen 500 files to guarantee the disinfection. These signatures and files are used to fill the virus signatures' pool and files' pool respectively.

3.2. Research tools

At first, the MATLAB (R2007a) version 6.5 has been used under Windows XP to implement the algorithm. Then the MATLAB (R2009b) version 7.9 has been employed to continue the rest of the implementation, because it has been found that the MATLAB (R2009b) version 7.9 enhances the memory management. The Clonal Selection Algorithm "CLONALG" which was created by Castro and Zuben (2000a). At the stage of using GA, the Genetic Algorithm tool under MATLAB has been used for the parameters tuning.

3.3. Research Stages

The research goes through five stages, where MATLAB is used. As illustrated in Figure (16):

1. The Virus Detection Clonal (VDC) algorithm is designed and implemented, where the Signature Scanner method with Clonal Selection algorithm (CLONALG) is used.

2. The VDC algorithm is tested.
3. The VDC algorithm is optimized by using the GA to tune the system parameters.
4. The optimized VDC algorithm based on GA is tested.
5. The standard VDC algorithm is compared with the GA based algorithm.
This comparison depends on the accuracy criteria; the number of the correct detections and false positives.

The first four stages are described in the following sections, and the comparison stage is described in chapter 5.

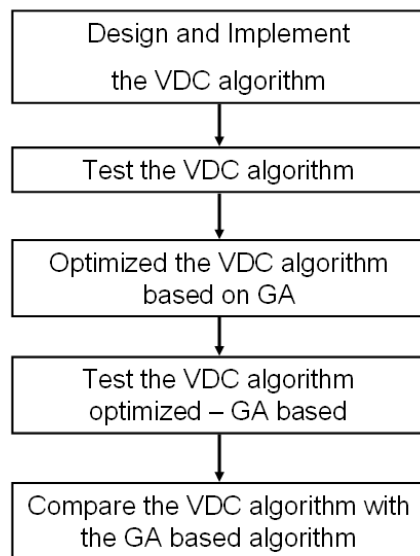


Figure 16: The Stages of research

3.3.1. The design and implementation of the VDC algorithm

The CLONALG by Castro and Zuben, which is the algorithm that has been an inspiration for the proposed algorithm (VDC), is explained in the case of pattern recognition, where a set of patterns to be recognized (P), and the basic steps of the CLONALG are:

1. A population of individuals (M) is randomly initialized.
2. For each pattern of P a match with each element of the population M is done to determine its affinity.

3. A (n) of the best highest affinity elements of M is selected.
4. The cloning is applied for these n individuals by making copies of these individuals proportional to their affinity; the higher the affinity the higher the number of copies.
5. These copies are mutated with a rate proportional to their affinity, and then stored in the temporary population (M').
6. The affinity is determined for the mutated clones.
7. The highest affinity one is reselected to be a candidate, then compared with its respective element in M' , if it is larger it is replaced.
8. The d lowest affinity individuals of M are replaced.

In this algorithm, it is assumed that the n highest affinity individuals are sorted in ascending order after Step 3, so that the amount of clones generated for all these n selected elements is given by the following equation:

$$N_c = \sum_{i=1}^n \text{round}\left(\frac{\beta \cdot N}{i}\right) \dots\dots\dots (3.1)$$

Where N_c is the total amount of clones generated for each of the patterns (P), β is a multiplying factor, N is the total amount of elements (M), n is the number of selected elements in M with the highest affinity, to apply the cloning on them, and round is the operator that rounds its argument towards the closest integer. Each term of this sum corresponds to the clone size of each selected element (M), e.g., for $N = 100$ and $\beta = 1$, the highest affinity element ($i = 1$) produces 100 clones, while the second highest affinity element produces 50 clones, and so on [Castro, 2002b].

The VDC algorithm is inspired from the CLONALG described above, and the differences are reviewed after explaining the VDC algorithm. Figure (17) illustrates the flowchart of the VDC algorithm, which has the following main steps: Cloning, Hypermutation and Reselection stochastically, these steps are detailed at Figure (18).

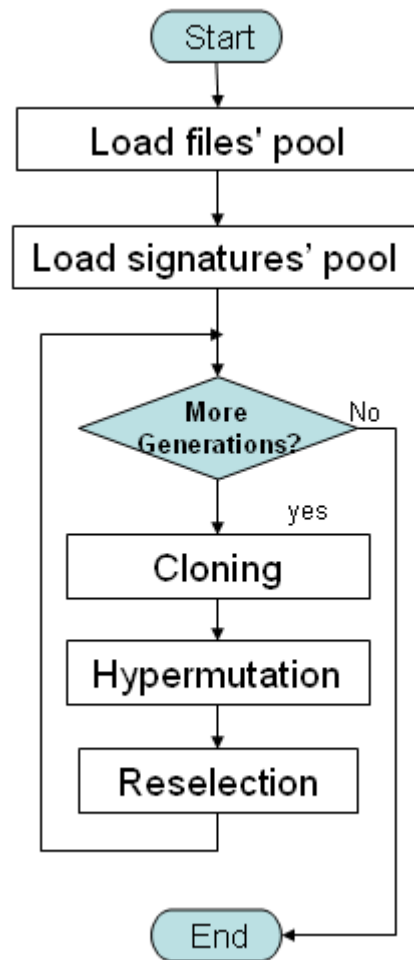


Figure 17: The VDC algorithm main steps flowchart

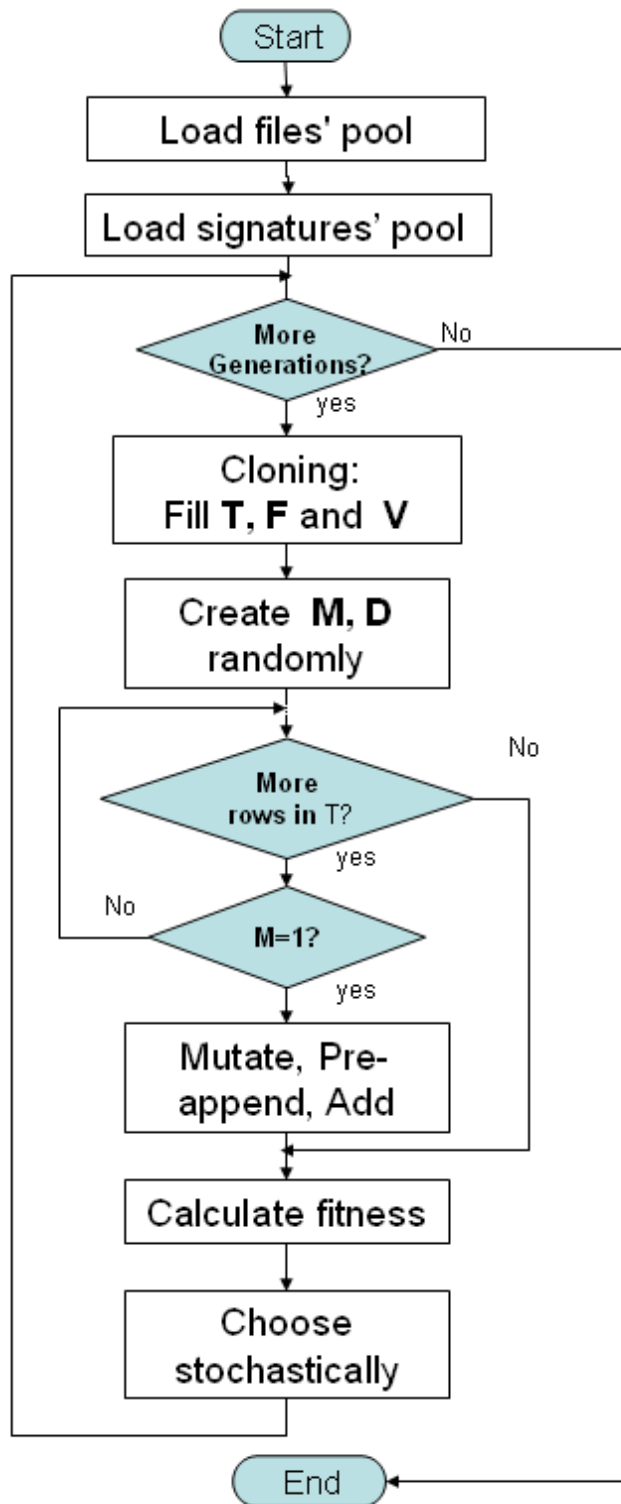


Figure 18: Cloning, Hypermutation, and Reselection

The pseudo code of the VDC algorithm is illustrated at Figure (19); it represents the steps that compose the VDC algorithm. These steps are described in details after that.

- 1 • Load files' pool (filename, file_content)
- 2 • Open signatures' pool (virus name, signature, initial fitness)
- 3 • Loop condition
- 4 ○ Cloning:
- 5 ▪ Making (**Fat** * **N**) copies for each signature in **T**, and their fitness in **F**, and the virus name in **V**
- 6 ○ Hypermutation:
- 7 ▪ **M**: creating random values(**RN**); $1 \geq \text{RN} \geq 0$ and if $\text{RN} \leq \text{Pm}$ then **M**=1 else **M**=0
- 8 ▪ **D**: creating random numbers (-1 or 0 or 1)
- 9 ▪ Loop for each row in **T**
- 10 • If **M** = 1 then
- 11 ✓ Mutation of one character by random character in a random position
- 12 ✓ 'mut_' is pre-appended to the virus name in this row
- 13 ✓ **D** is added to the fitness in this row
- 14 ○ Re-selection:
- 15 ▪ Calculate Fitness:
- 16 • For each file on the files' pool (not eliminated)
- 17 ○ The file content is matched with the signatures in **T**; if they are matched then
- 18 ✓ δ is added to the fitness of the matched signature
- 19 ✓ The matched file is eliminated from the files' pool
- 20 ▪ The signatures are selected according to their fitness stochastically

Figure 19: The VDC algorithm pseudo code

The full description of the pseudo code steps are:

1. Load files' pool (filename, file_content):

The files' pool contains Windows files, and for each file the "file name" and the "contents" are contained in the pool.

2. Open signatures' pool (virus name, signature, initial fitness):

The virus signatures' pool contains the "virus name", the "virus signature", and "initial fitness" for each signature. The initial fitness is a random number between 4 and 210.

3. Loop condition:

This loop iterates until the *IT* reaches the *Gen*. Where *IT* is the iteration number and *Gen* is the number of all generations, where *Gen* is a parameter, defined in the algorithm. The steps from 4 to 20 are executed within this loop.

4. Cloning:

The signatures' pool is sorted in descending order according to their fitness, before performing the cloning. The cloning process is applied on the half size of the signatures' pool with the highest fitness, which is initially 100 signatures (the half is 50 signatures with higher fitness). This size is increased from one generation to another. The researcher chooses the half size only, based on the experiments she applied, as by using all the signatures the size of the signatures' pool becomes enlarged, which leads to slowing the algorithm. Besides, the signatures with higher fitness are supposed to be widespread and that's why they are chosen. Knowing that the half size of the signatures' pool equation (half) is:

$$half = floor\left(\frac{n}{2}\right) \dots\dots\dots (3.3)$$

Where *n* is the signatures' pool size.

5. Making ($Fat * N$) copies for each signature in T , and their fitness in F , and the virus name in V :

Where Fat is the multiplying factor; the probability of the number of the copies in each clone, and N is the number of the signatures in the signatures' pool. For example; if the $Fat = 0.1$ and $N = 100$, then the number of elements in each clone is 10, and if the $Fat = 0.05$ and $N = 100$, then the number of elements in each clone is 5. Knowing that, the copies of the same signature form one clone.

Hence, in this step for each signature ($Fat * N$) of copies that is made; T has the copies of the signature, F has copies of the fitness, and V has copies of the virus name.

6. Hypermutation:

The Hypermutation is the mechanism of making random changes to the virus signatures inside T , and occasionally one such change leads to an increase in the fitness, because higher fitness variants are selected in later steps.

The steps from 6 to 13 clarify the Hypermutation step.

7. M : creating random values(RN); $1 \geq RN \geq 0$ and if $RN \leq Pm$ then $M=1$ else $M=0$

The Pm is the Hypermutation probability. M is a vector with the number of rows that equals the number of rows in T , ($M_{(rowcount(T) \times 1)}$). The values in M are 0 or 1 according to the random number (RN) after comparing it with the Pm value. If $RN \leq Pm$ then $M=1$ else $M=0$; Where $1 \geq RN \geq 0$. For example, if $Pm=0.2$, and if the random number is 0.08 then $M=1$, and if the random number is 0.34 then $M=0$.

8. D : creating random numbers (-1 or 0 or 1)

D is a vector with the number of rows that equals the number of rows in T , ($D_{(rowcount(T) \times 1)}$). The values inside D are -1 or 0 or 1, because these values are added to the fitness of the mutated signatures in the following steps, where

the fitness of the mutated signature can be better, worse or the same as the fitness of the signature before mutation, which reflects the randomness. So if $D=-1$, this means the fitness is decreased by 1, and if $D=0$ the fitness remains the same, and if $D=1$ the fitness increases by 1.

9. Loop for each row in T

The steps 10-13 are executed for each copy of the signatures in T .

10. If $M = 1$ then

If the value of M is 1 then the mutation steps are performed as represented in steps from 11 to 13. Knowing that, if $M=0$ then the next row in T is executed in step 9.

11. Mutate one character by a random character in a random position

One character in the signature is replaced with a random character; the ASCII code for this random character is between 48 and 122, where the characters that are equivalent to these ASCII codes are the following: (1, 2, 3, 4, 5, 6, 7, 8, 9, :, <, =, >, ?, @, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, [, \,], ^, _, ` , a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z), and the replacement position is also chosen randomly. For example, if the signature is '8e5ef1aec91259d70c5e62cdf42c36e ddc8cc9cbe45313d0' after mutation it can be '8e5ef1aec91259d70c5e62kdf42c36e ddc8cc9cbe45313d0'; the c is replaced by k. or if the signature is '8e5ef1aec91259d70c5e62cdf42c36e ddc8cc9cbe45313d0' after mutation it can be '8e5ef1aec91259d70c5e62cdf42c36e ddc8ccdcbe45313d0'; the 9 is replaced by d, and so on.

12. 'mut_' is pre-appended to the virus name in this row

The virus name in V for the mutated signature in T is pre-appended with the "mut_" to distinguish it from non mutated signatures. For example, if the virus

name is 'Virus.1C.Tanga.a', then it is changed to 'mut_Virus.1C.Tanga.a'.

13. *D is added to the fitness in this row*

The fitness in F for the mutated signature in T is changed by adding the D value mentioned in step 8, hence the new fitness is either increased by 1 or kept the same or decreased by 1.

14. *Re-selection*

This step is about choosing the next generation, whereas the signatures are selected according to their fitness stochastically. The steps from 15 to 20 demonstrate this.

15. *Calculate Fitness*

The fitness function is a counter for the matches between the signatures in T and the files inside the files' pool in addition to the initial fitness:

$$f(x) = f_0(x) + \delta \cdot \sum_{i=1}^z \text{matchfunction}(x, y_i) \quad \dots\dots\dots (3.4)$$

Where $f_0(x)$: the initial fitness for signature x .

y_i : The i^{th} file.

δ : multiplying factor with a value of 10.

z : The number of all files in the files' pool.

The steps from 16 to 19 explain the calculation of fitness function.

16. *For each file on the file's pool (not eliminated)*

A loop is made for each file in the files' pool, and it is checked, if the file is not eliminated then steps 17, 18, and 19 are done.

17. *The file content is matched with the signatures in T; if they are matched then*

Each of the file content inside the files' pool is matched with all the signatures in T . The match function is:

$$\text{matchfunction} = \begin{cases} 0; \text{ no match} \\ 1; \text{ match found} \end{cases} \quad \dots\dots\dots (3.5)$$

And if a match exists, then the steps 18 and 19 are executed.

18. δ is added to the fitness of the matched signature

The fitness value in F for this signature is changed by adding δ , where δ equals 10 in this algorithm to give the detection process higher weight than given to the mutation process (Mutation adds 1 to the fitness at most).

19. The matched file is eliminated from the files' pool

The matched file is eliminated from the files' pool since it is infected; to get rid of the redundancy issue.

20. The signatures are selected according to the fitness stochastically.

The stochastic selection process is:

1. A random number is created for each generation (iteration in Gen) R , which is called the selection threshold, and its values are between 0.6 and 1; to make sure that the Best fitness is selected.
2. Each fitness in the clone is divided by the maximum fitness of this clone.
3. If the value in step 2 $\geq R$, and the signature does not exist in the original signatures' pool (in step 2 initially) then the fitness of this signature is appended to a temporary matrix.
4. The temporary matrix is sorted in descending order.
5. The best new 11 signatures are selected to be added to the original signatures' pool. The appended new signatures are determined by 11 in order to prevent the enlargement of the signatures' pool.

After that the execution continues back to step 3.

The fitness function for the whole algorithm is:

$$f(x) = f_0(x) + \delta \cdot \sum_{i=1}^{\infty} matchfunction(x, y_i) + \sum_{j=1}^t D_j \quad \dots\dots\dots (3.6)$$

Where: $f_0(x)$: the initial fitness for signature x

y_i the i^{th} file

δ : multiplying factor with a value of 10.

z : the number of all files in the files' pool.

$$matchfunction = \begin{cases} 0; & \text{no match} \\ 1; & \text{match found} \end{cases} \dots\dots\dots (3.7)$$

D_j : if $M = 1$ then $D_j = (0 \text{ or } 1 \text{ or } -1)$ randomly

t : the no. of signatures in the original signatures' pool * the no. of signatures in each clone.

After describing the CLONALG and the VDC algorithm, Table (1) demonstrates the main differences between these two algorithms.

Table 1: The differences between CLONALG and VDC algorithm

Category	CLONALG	VDC algorithm
<i>P</i>	Patterns to be recognized	Files to be searched
<i>M</i>	Randomly initialized	Viruses' signatures as described in section 3.1
Affinity	The match between elements in <i>M</i> and patterns in <i>P</i>	Fitness function values in equation 3.6
Number of elements to be cloned	<i>n</i>	The half size of signatures' pool
The number of elements in each clone	proportional to the elements affinity in equation 3.1	Fixed and it is (<i>Fat</i> * <i>N</i>) for each clone
Mutation	Proportional to the elements affinity	Not proportional
Lowest elements in M	Replaced the lowest <i>d</i>	Instead of replacement, adding the best 11 elements'

¹ The replacement is not an option due to the sensitivity of the application, as when dealing with viruses, even if the virus is not widespread, it is important for the algorithm to be able to detect it.

This section has described the design and implementation of the VDC algorithm. Yet, the next section includes the testing strategy for VDC algorithm.

3.3.2 The testing of the VDC algorithm

The strategy of testing is demonstrated in this section, and the results are figured in chapter four. There are two phases: training and matching.

The training phase takes in consideration the filling of the signatures' pools with the new signatures after applying the VDC algorithm in addition to the already known signatures (original signatures that was gathered from VX Heaven website before mutation).

To apply the VDC algorithm the files' pools are needed to complete the matching process between files and signatures.

At the beginning, all files contained in the files' pool are benign to test the virus detection, and then 5% of the files are infected, after that 25%, 50%, 75%, 100% of the files are infected, as figured in Table (2). For the training process the files' pool with 5%, 25% and 75% infected files are used (to leave the other three files' pools for the matching phase without being used in the training phase), then all the six files' pools are employed at the matching process.

Table 2: Files' pool contents

No. of all files	No. Of benign files	No. of infected files	Infection Rate
500	500	0	0%
500	475	25	5%
500	375	125	25%
500	250	250	50%
500	125	375	75%
500	0	500	100%

A set of infected files is prepared by infecting them with viruses from the dataset mentioned in section 3.1. In the 5% files' pool, the 25 files are selected randomly

from the set of prepared infected files. Then for the 25% files pool, the 125 files are selected from the rest of the set (with out retaining the 25 files back). After that, for the 50% files' pool the 375 are selected from the rest, and so on.

Several parameters in the VDC algorithm are changed to search for better performance, with different values for each parameter: *Learning Gen*, *Pm* and *Fat*. The *learning Gen* values are 100, 150 or 300, the *Pm* values are 0.05, 0.1 or 0.2, and the *Fat* values are either 0.05 or 0.1. These parameters are chosen as examples but not exclusive, because the probabilities of the parameters values are infinite, and according to the researcher diligence. The parameters values are described at Table (3). The resulting signatures' pools are: Sig1, Sig2, Sig3, Sig4, Sig5, Sig6, Sig7, Sig8, Sig9, Sig10, Sig11 and Sig12, which are used in the matching phase.

Table 3: The parameters values of the training phase

Parameters			Signatures' pools
<i>Learning Gen</i>	<i>Pm</i>	<i>Fat</i>	
5% infected files			
100	0.05	0.05	Sig1
100	0.1	0.05	Sig2
300	0.05	0.1	Sig3
100	0.05	0.1	Sig12
25% infected files			
100	0.05	0.05	Sig4
300	0.1	0.1	Sig5
150	0.2	0.05	Sig6
100	0.2	0.05	Sig10
75% infected files			
100	0.2	0.05	Sig7
150	0.1	0.1	Sig8
300	0.05	0.1	Sig9
100	0.2	0.1	Sig11

After finishing the training phase, the matching phase starts. The matching is concerned with the calculation of fitness function, which means searching for matches between the files' pool and the virus signatures' pool. Therefore, the matching algorithm does not contain the Hypermutation, Cloning nor reselection steps. As illustrated in the matching flow chart in Figure (20).

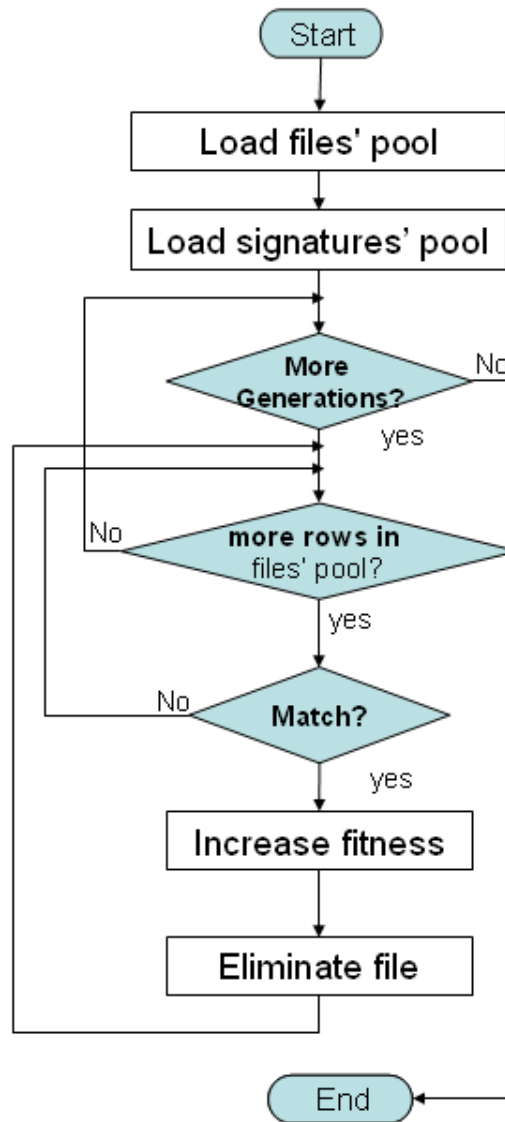


Figure 20: Matching the VDC algorithm flowchart

Figure (21) illustrates the pseudo code of the matching phase. The below steps are explained previously in section 3.3.1. except in step 7 where the increase value on the fitness is 1 (not 10).

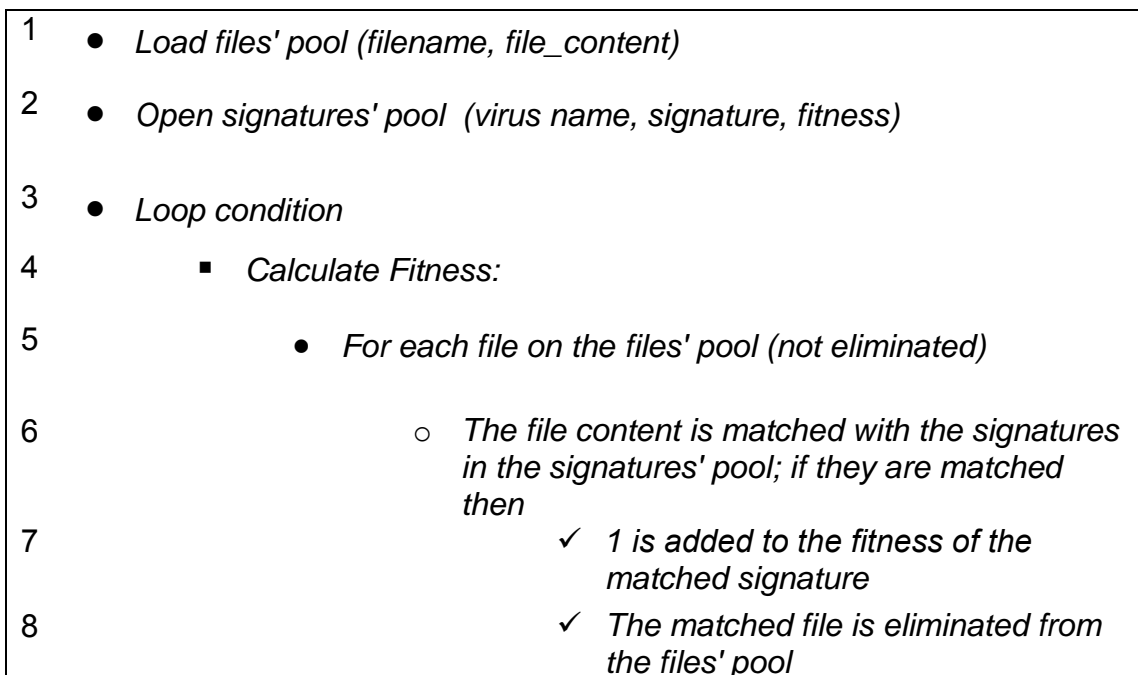


Figure 21: The pseudo code of the matching of the VDC algorithm

The fitness function is a counter for the matches between the virus signatures and the files:

$$f(x) = f_0(x) + \sum_{i=1}^z \text{matchfunction}(x, y_i) \dots\dots\dots (3.8)$$

Where: $f_0(x)$: the initial fitness for signature x

y_i : The i^{th} file

z : The number of all files in the files' pool

The match function is shown in equation 3.7.

The signatures' pools that are obtained in Table (3) are employed with the files' pools (0%, 5%, 25%, 50%, 75%, and 100%) in the matching process as illustrated in Table (4). Knowing that, *matching Gen*=100 for all of them.

Table 4: The Parameters values of the matching phase

Files' pools	Signatures' pools
0%	Sig1
0%	Sig5
0%	Sig8
5%	Sig6
5%	Sig7
5%	Sig10
5%	Sig11
25%	Sig1
25%	Sig2
25%	Sig12
50%	Sig4
50%	Sig5
50%	Sig8
75%	Sig1
75%	Sig3
75%	Sig6
75%	Sig9
75%	Sig12
100%	Sig2
100%	Sig4
100%	Sig7
100%	Sig10
100%	Sig11
100%	Sig12

The matching starts on the files' pool 0%, which contains 500 benign files, and the signatures' pools Sig1, Sig5 and Sig8 as illustrated in Table (4), to examine the concept of the false positive (when detecting benign files as infected files).

For the five files' pools that are left, the matching is performed by running each file's pool with the corresponding signatures' pools in Table (4), then:

- When matching files' pools with 5% and 75% of infected files, new 100 files are added to the files' pool at the matching iteration (*matching Gen*) of 5.
- When matching files' pools with 25%, 50% and 100% of infected files, new 100 files are added to the files' pool at the matching iteration (*matching Gen*) of 50. Regardless of the time when the new 100 files are added to the files' pool, the algorithm must be able to detect the infected files.

The 100 files include benign files and infected files. The infected files are categorized into three:

1. Files with signatures that already exist in the original files pool. The signatures are used at the training phase.
2. Files with signatures that do not exist in the original files pool. The signatures are not used at the training phase.
3. Files that have signatures with mutations. These mutated signatures are obtained from the signatures' pools that are produced in the training phase.

This section has described the testing strategy of the VDC algorithm. Yet, the next section includes the optimization of the VDC algorithm by using the GA.

3.3.3 The optimization of the VDC algorithm by using the GA

The Genetic Algorithm Toolbox in the MATLAB is employed to tune the parameters of the VDC algorithm. These parameters are: the multiplying factor which determines the number of elements in each clone (Fat), and the Hypermutation factor (Pm). These two parameters construct the input string. The output string is the Mean fitness which is a real number that represents the Mean fitness while executing the VDC algorithm. The fitness function of the genetic algorithm is a file called "myfun" which contains the call for the VDC algorithm (.m file) pre-appended by the minus sign (-) to maximize the searching by the GA, because GA by default minimizes the searching and by adding the minus(-), it searches for the maximum.

Figure (22) illustrates how the GA treats the VDC algorithm from the previous section as a black box.

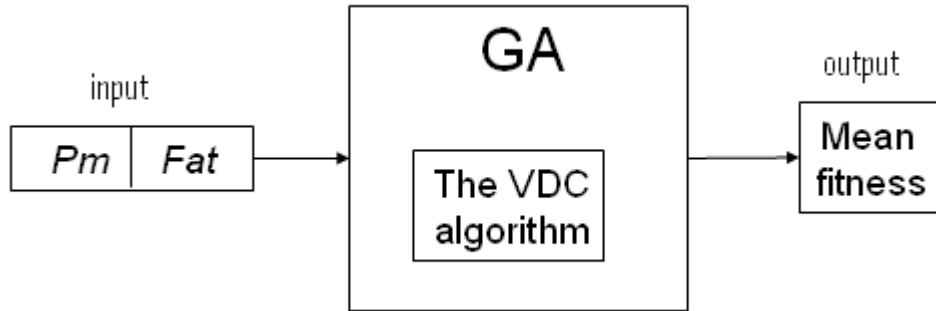


Figure 22: The GA as Parameters Optimizer

The VDC algorithm (.m file) function prototype is:

Function [meanfx] = VDC_algorithm(param)

The *param* contains the two parameters; $Pm = param(1)$ and $Fat = param(2)$. The lower bound of these parameters = [0.01; 0.02] and the upper bound = [1; 1], where the first entry is for the first parameter and the second entry is for the second parameter.

The lower bound for the *Fat* has been chosen with the value of 0.02, as the number of elements in each clone = $Fat * N$ and $N = 100$, which represents the initial population that has been collected from the VXHeaven website, so when $Fat = 0.02$, this means that there is at least 2 elements in the clone.

Regarding the *Pm*, the value of 0.01 is chosen as a lower bound, because if it has been less than this value, the *Pm* is closer to zero, so the effect of Hypermutation does not appear. So if the value of $Pm = 0.01$, this means that from each 100 copies of signatures, Hypermutation is executed at one copy at least.

The upper bound represents the maximum value of the parameters *Fat* and *Pm* which is equals 1.

Figure (23) illustrates the flow chart that represents the process of optimizing the VDC algorithm using the GA.

The GA is supposed to find the best parameters' values for the VDC algorithm (*Pm* and *Fat*), and these values are used to run the VDC algorithm (when using the parameters' values that are resulted from using the GA as an optimizer in the

VDC algorithm. This algorithm is called the optimized VDC algorithm based on GA). The results of the optimized VDC algorithm based on GA and the comparison between the VDC algorithm and this algorithm are included in chapter five.

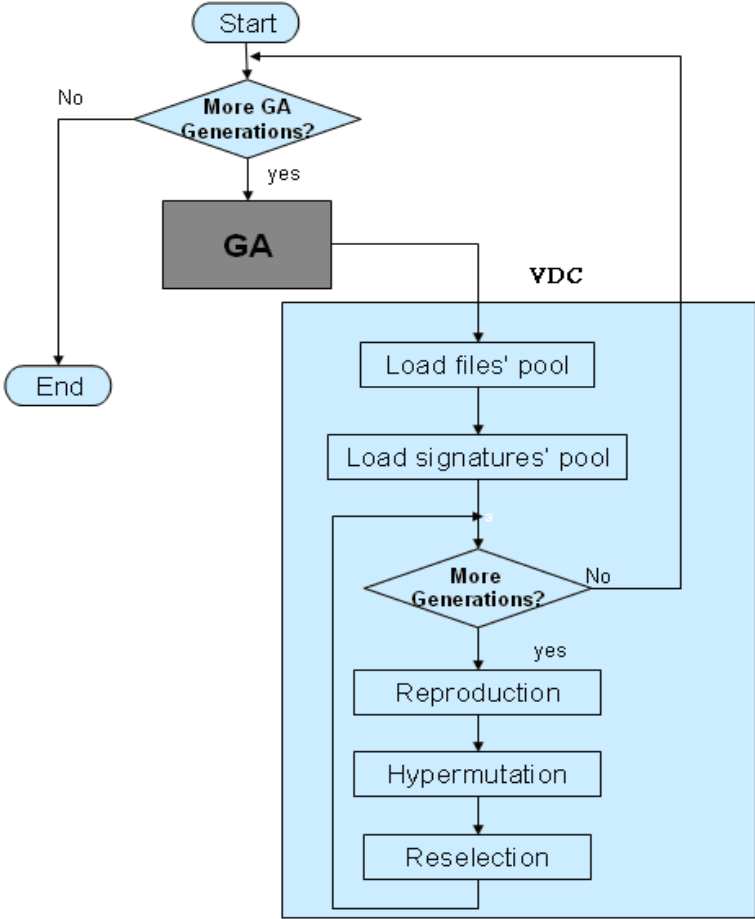


Figure 23: The flowchart of optimizing the VDC algorithm by the GA

3.3.4 The testing of the optimized VDC algorithm based on GA

The testing strategy of the optimized VDC algorithm based on GA has two phases: training and matching.

The training phase takes in consideration filling signatures' pools, using the parameters from the previous section - after applying the GA based algorithm. Then the matching phase tests these signatures' pools with the files' pools mentioned in Table (1). The matching flow chart is illustrated in Figure (20).

This chapter has shown the strategy of the research. The results are explained in chapter four and five, whereas chapter six demonstrates the conclusion and recommendations for future researches.

Chapter Four

The VDC algorithm Results and Analysis

The proposed algorithm is the CLONALG algorithm after updating it to detect viruses as a signature scanner. This algorithm is created as mentioned in chapter three to produce the Virus Detection Clonal (VDC) Algorithm. This chapter answers the first question of the research questions in section 2.1, which is "**will the proposed AIS algorithm – The Virus Detection Clonal (VDC) Algorithm - be good in detecting computer viruses?**" by demonstrating the results. These results are divided into two categories: the training phase and the matching phase.

4.1 Training of the VDC algorithm

The training phase applies the VDC algorithm on different files' pools (5%, 25% and 75% of infected files – to leave the other three files' pools for the matching phase without being used in the training phase) with different values of parameters; these values have been chosen according to the experience of the researcher from the experiments on the VDC algorithm. The parameters are *Learning Gen*, *Pm* and *Fat*. The *Learning Gen* (number of generations) has the values: 100, 150 or 500, the *Pm* (Hypermutation probability) has the values: 0.05, 0.1 or 0.2, and the number of elements per clone is $Fat * N$; where $N=100$ for all runs, and *Fat* has the values: 0.05 or 0.1. Whilst, when $Fat=0.05$ the number of elements per clone=5, and when $Fat=0.1$ the number of elements per clone=10. The produced signatures' pools are: Sig1, Sig2, Sig3, Sig4, Sig5, Sig6, Sig7, Sig8, Sig9, Sig10, Sig11 and Sig12 as represented in Table (3), and these signatures' pools are used in the matching phase next section. Knowing that, the fitness is calculated based on equation 3.6.

Figure (24) illustrates the initial population of the signatures' pools for all of the twelve runs; because all runs read the same 100 signatures -that have been gathered from the VX Heaven website- as initial population, so the population size is 100 for all of them. This figure is used later on to compare the initial population of the signatures' pools with the final population after applying the training runs.

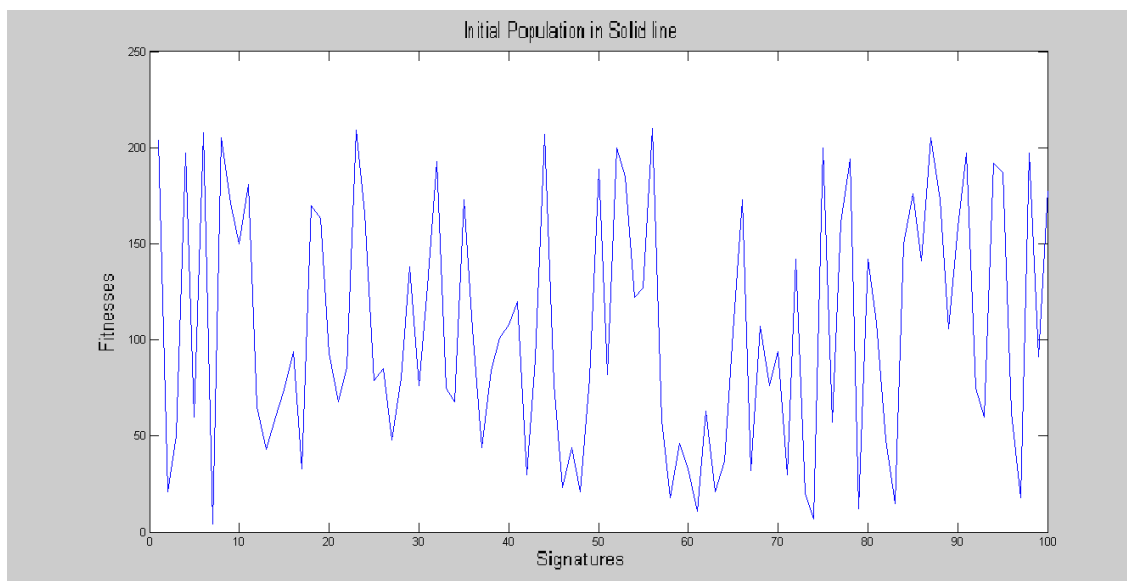


Figure 24: The Initial Population

The **first** run parameters are: *Learning Gen*=100, *Pm*=0.05 and *Fat* = 0.05, and the files' pool with 5% of infected files, and the produced signatures' pool is saved as **Sig1**.

Table (5) shows the results of the first run, where each of the iterations display the number of signatures in the signatures' pool. The number of signatures value is 100 on the first iteration and it is increased by 11 signatures or less in each iteration. The reason that the number of signatures that are added is less than eleven is in some iterations, the best mutated signatures that corresponds with the selection threshold may be less than 11. As shown in Table (5), the number of signatures in iteration 101 is 1198, but if 11 signatures are added in each of the iterations, then the number of signatures is supposed to be 1200.

The selection threshold ranges between 0.6 and 1.0; this threshold determines the selection process stochastically, as described in step 20 in the VDC algorithm in section 3.1.

The Mean fitness, first increases at a rapid rate, then the increase continues at a slower rate; the Mean fitness in the beginning increases at a higher changing rate reaching 9.97, then the rate decreases to reach 6.28 in the second iteration, to become 5.25 in the third iteration and so on, till it reaches around 0.11 in the last five iterations, as shown in Table (5). (The whole table is viewed in Appendix A).

Whereas the Best fitness increases quickly in the first iteration only and later on

the increase becomes slower; in the first iteration the changing rate is 25 for the Best fitness, after that it increases by 1 with several iterations. Figure (25) represents the Mean fitness and the Best fitness.

Table 5: The first Training run results

Iteration number	No of signatures	Selection threshold	Mean fitness	Best fitness
1	100	0.90	160.14	210.00
2	111	0.90	170.11	235.00
3	122	1.00	176.39	236.00
4	131	0.60	181.64	236.00
5	142	0.70	186.43	236.00
...
96	1143	0.90	246.61	253.00
97	1154	1.00	246.72	253.00
98	1165	0.70	246.82	253.00
99	1176	0.90	246.92	253.00
100	1187	0.70	247.03	253.00
101	1198	1.00	247.15	253.00

As a result the number of detected infected files is 17 out of 25, (as it is the training phase, the researcher decides to take half the size of the signatures' pool as mentioned in section 3.1), and the Mean fitness = 247.1486.

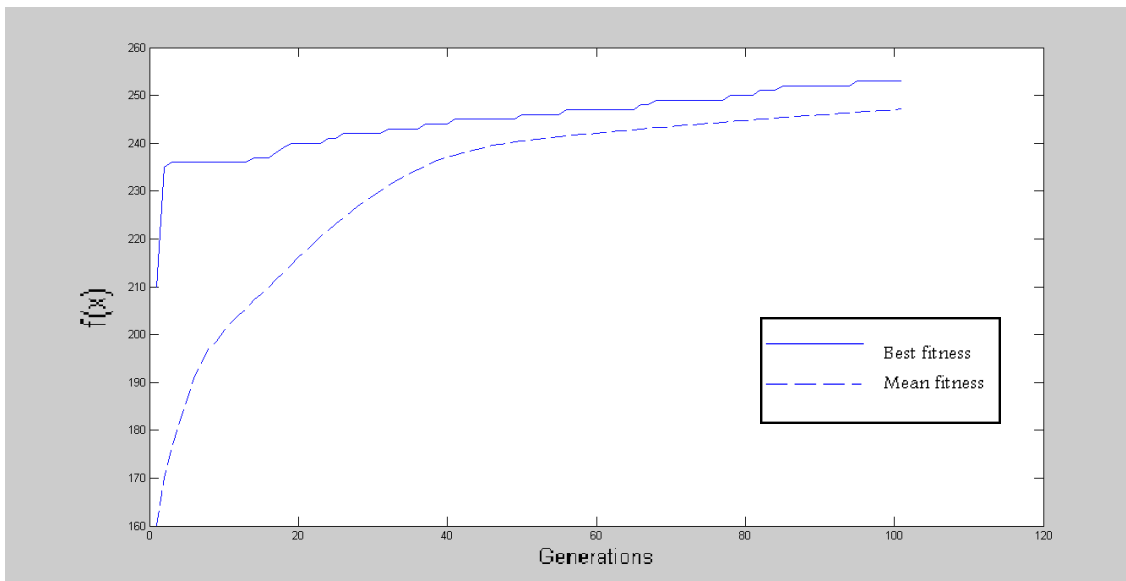


Figure 25: The first run Mean fitness & Best fitness

Figure (26) demonstrates the initial population in solid line, which is the same in Figure (24), and the final population including the new signatures after Hypermutation is in dotted line. As shown the fitness of the new signatures after Hypermutation is higher.

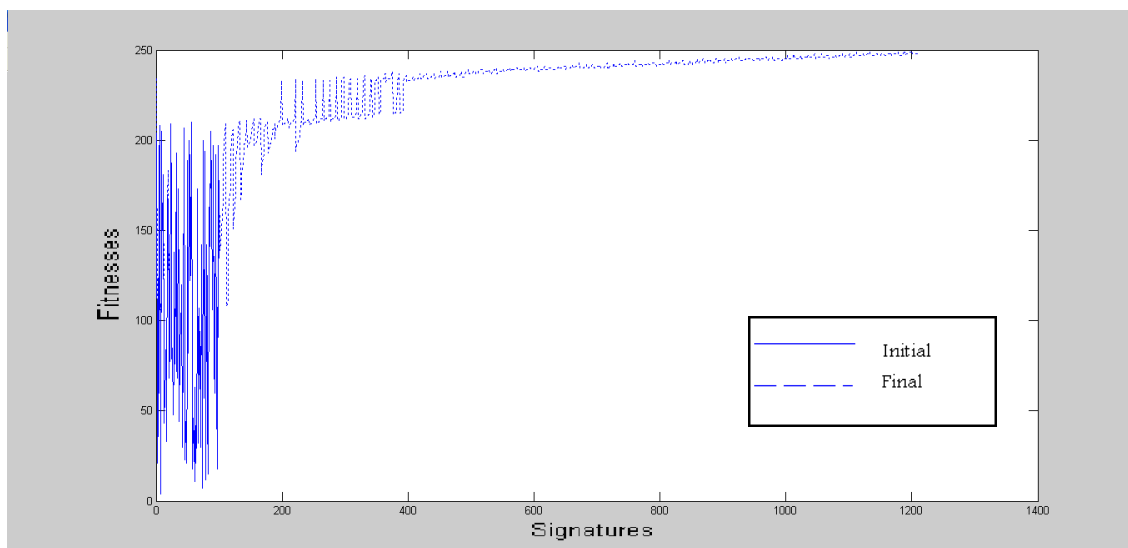


Figure 26: The first run final population

The **second** run parameters are: *Learning Gen*=100, *Pm*=0.1 and *Fat* = 0.05, and the files' pool with 5% of infected files, and the produced signatures' pool is saved as **Sig2**.

Table (6) shows the results of the second run, where each iteration displays the number of signatures in the signatures' pool. The number of signatures value is 100 on the first iteration and it is increased by 11 signatures or less in each iteration. As shown in Table (6), the number of signatures in the last iteration is 1200.

The selection threshold ranges between 0.6 and 1.0; this threshold determines the selection process stochastically, as described in step 20 in the VDC algorithm in section 3.1.

Table 6: The second Training run results

Iteration number	No of signatures	Selection threshold	Mean fitness	Best fitness
1	100	0.70	160.12	210.00
2	111	0.70	173.85	224.00
3	122	1.00	185.36	224.00
4	133	1.00	191.65	225.00
5	144	0.60	196.44	225.00
...
96	1145	0.80	241.28	250.00
97	1156	1.00	241.47	250.00
98	1167	0.80	241.68	251.00
99	1178	0.60	241.87	251.00
100	1189	0.90	242.08	251.00
101	1200	0.70	242.29	251.00

The Mean fitness, first increases at a rapid rate, then the increase continues at a slower rate; the Mean fitness in the beginning increases at a higher changing rate reaching 13.73, then the rate decreases to reach 11.51 in the second iteration, to become 6.29 in the third iteration and so on, till it reaches around 0.20 in the last five iterations, as shown in Table (6).

Whereas the Best fitness increases quickly in the first iteration only and later on the increase becomes slower; in the first iteration the changing rate is 14 for the Best fitness, after that it increases by 1 with several iterations. Figure (27) represents the Mean fitness and the Best fitness.

As a result the number of detected infected files is 17 out of 25, and the Mean fitness = 242.2893.

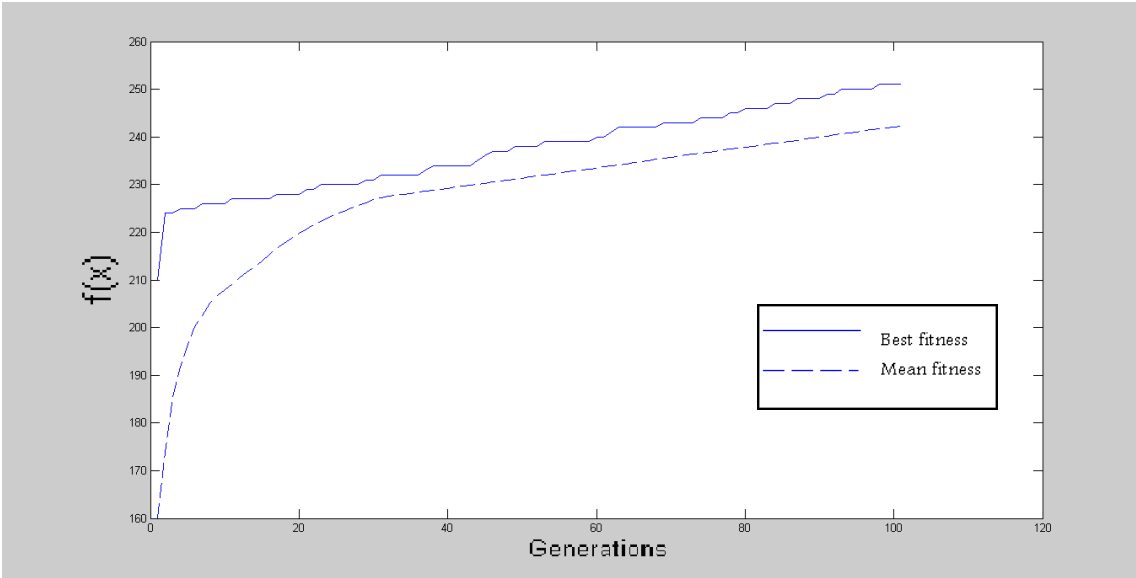


Figure 27: The second run Mean fitness & Best fitness

The initial population is demonstrated in solid line at Figure (28), which is the same in Figure (24), and the final population including the new signatures after Hypermutation is in dotted line. As shown the fitness of the new signatures after Hypermutation is higher.

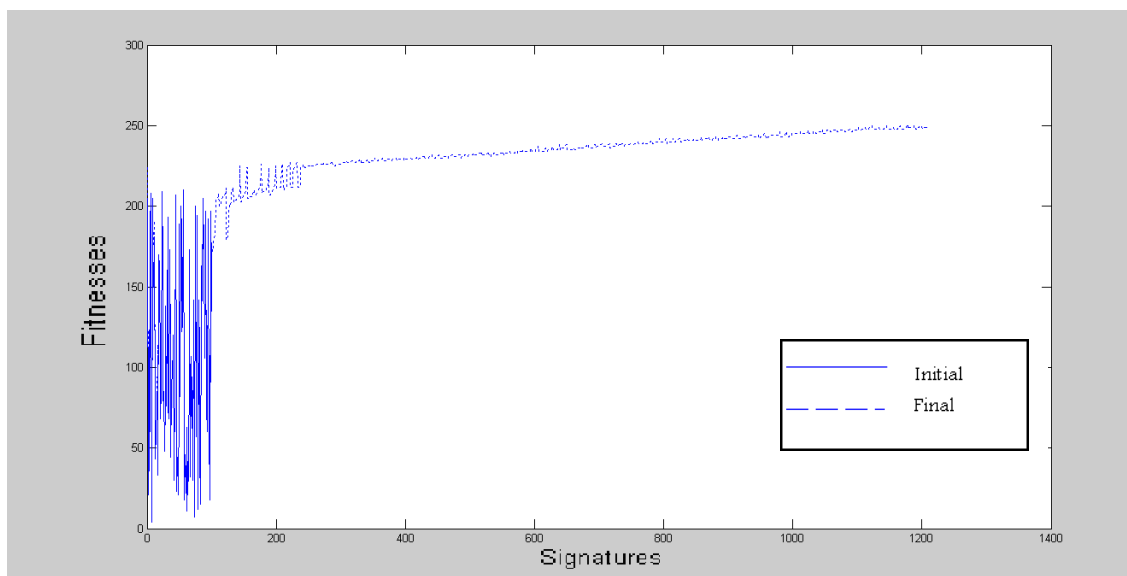


Figure 28: The second run final population

The **third** run parameters are: *Learning Gen*=300, *Pm*=0.05 and *Fat* = 0.1, and the files' pool with 5% of infected files, and the produced signatures' pool is saved as **Sig3**.

Table (7) shows the results of the third run, where each iteration displays the number of signatures in the signatures' pool. The number of signatures value is 100 on the first iteration and it is increased by 11 signatures or less in each iteration. As shown in Table (7), the number of signatures in the last iteration is 3400. The selection threshold ranges between 0.6 and 1.0; this threshold determines the selection process stochastically, as described in step 20 in the VDC algorithm in section 3.1.

The Mean fitness, first increases at a rapid rate, then the increase continues at a slower rate; the Mean fitness in the beginning increases at a higher changing rate reaching 8.85, then the rate increase to reach 10.59 in the second iteration, to become 8.68 in the third iteration, till it reaches around 0.16 in the last five iterations, as shown in Table (7).

Whereas the Best fitness increases quickly in the first iteration only and later on the increase becomes slower; in the first iteration the changing rate for the Best fitness is 15, after that it increases by 1 with several iterations. Figure (29) represents the Mean fitness and the Best fitness.

Table 7: The third Training run results

Iteration number	No of signatures	Selection threshold	Mean fitness	Best fitness
1	100	1.00	160.10	210.00
2	111	0.80	168.95	225.00
3	122	0.60	179.54	225.00
4	133	0.80	188.22	225.00
5	144	0.70	194.48	226.00
...
296	3345	0.70	274.47	292.00
297	3356	0.60	274.63	293.00
298	3367	0.80	274.79	293.00
299	3378	0.80	274.95	293.00
300	3389	1.00	275.11	293.00
301	3400	1.00	275.27	293.00

As a result the number of detected infected files is 17 out of 25, and the Mean fitness = 275.2660.

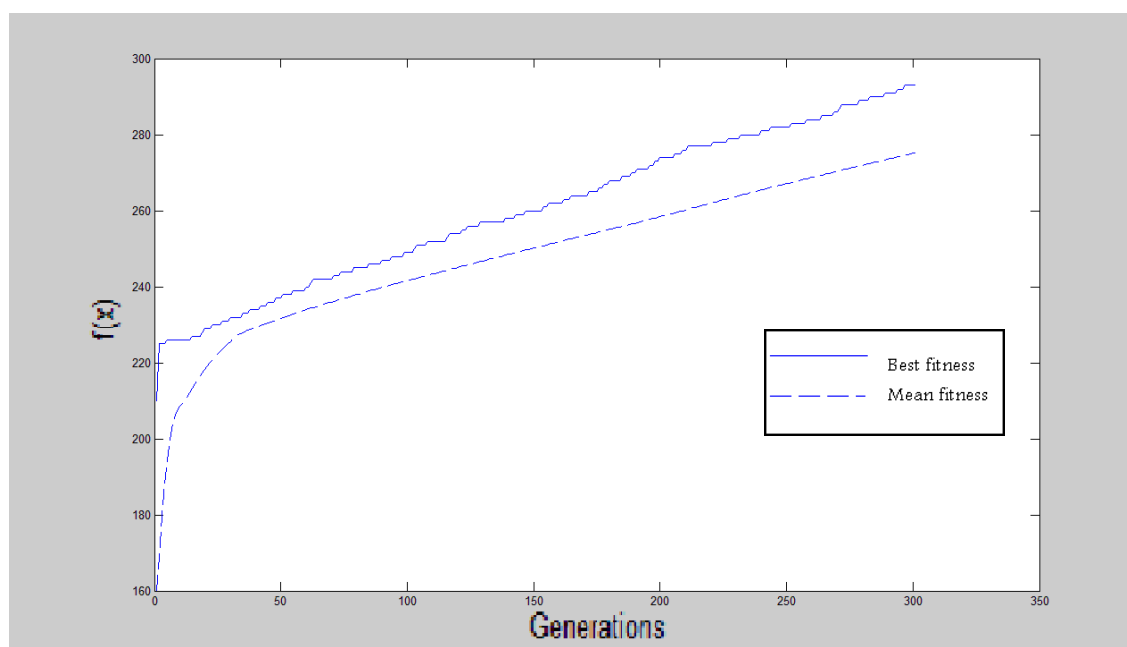


Figure 29: The third run Mean fitness & Best fitness

Figure (30) represents the initial population in solid line, which is the same in Figure (24), and the final population including the new signatures after Hypermutation is in dotted line. As shown the fitness of the new signatures after Hypermutation is higher.

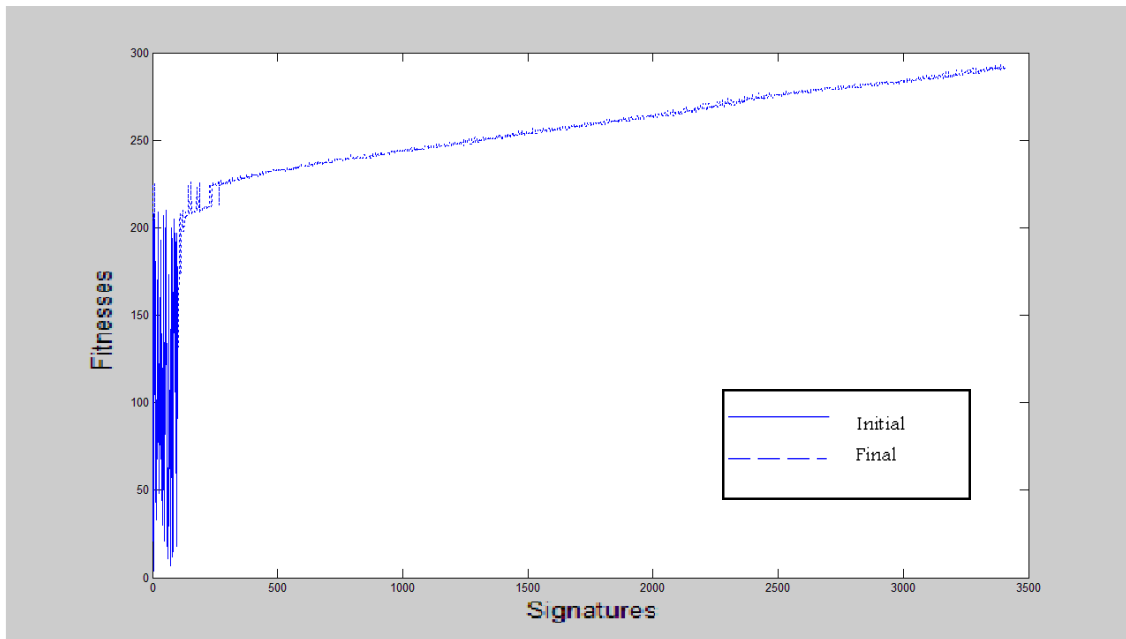


Figure 30: The third run final population

The **Forth** run parameters are: *Learning Gen*=100, *Pm*=0.05 and *Fat* = 0.05 and the files' pool with 25% of infected files, and the produced signatures' pool is saved as **Sig4**.

Table (8) shows the results of the forth run, where each iteration displays the number of signatures in the signatures' pool. The number of signatures value is 100 on the first iteration and it is increased by 11 signatures or less in each iteration. As shown in Table (8), the number of signatures in the last iteration is 1200. The selection threshold ranges between 0.6 and 1.0; this threshold determines the selection process stochastically, as described in step 20 in the VDC algorithm in section 3.1.

The Mean fitness, first increases at a rapid rate, then the increase continues at a slower rate; the Mean fitness in the beginning increases at a higher changing rate reaching 10.2, then the rate increases to reach 18.36 in the second iteration, to become 7.52 in the third iteration, till it reaches around 0.14 in the last five iterations, as shown in Table (8).

Whereas the Best fitness increases quickly in the first iteration only and later on the increase becomes slower; in the first iteration the changing rate is 30 for the Best fitness, after that it increases by 1 with several iterations. Figure (31) represents the Mean fitness and the Best fitness.

Table 8: The forth Training run results

Iteration number	No of signatures	Selection threshold	Mean fitness	Best fitness
1	100	0.60	160.12	210.00
2	111	0.60	170.32	440.00
3	122	0.70	188.68	440.00
4	133	0.80	196.20	440.00
5	144	0.70	205.57	440.00
...
96	1145	1.00	448.94	455.00
97	1156	0.70	449.07	455.00
98	1167	0.60	449.20	456.00
99	1178	0.90	449.33	456.00
100	1189	0.80	449.48	456.00
101	1200	0.80	449.61	456.00

As a result the number of detected infected files is 103 out of 125, and the Mean fitness = 449.6050.

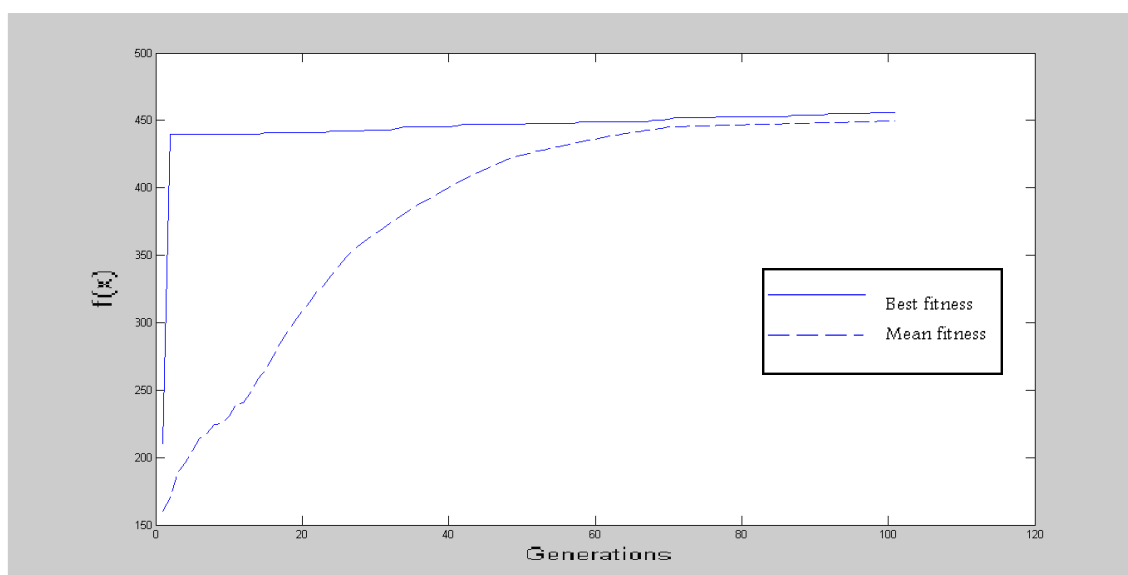


Figure 31: The forth run Mean fitness & Best fitness

Figure (32) exhibits the initial population in solid line, which is the same in Figure (24), and the final population including the new signatures after Hypermutation is in dotted line. As shown the fitness of the new signatures after Hypermutation is higher.

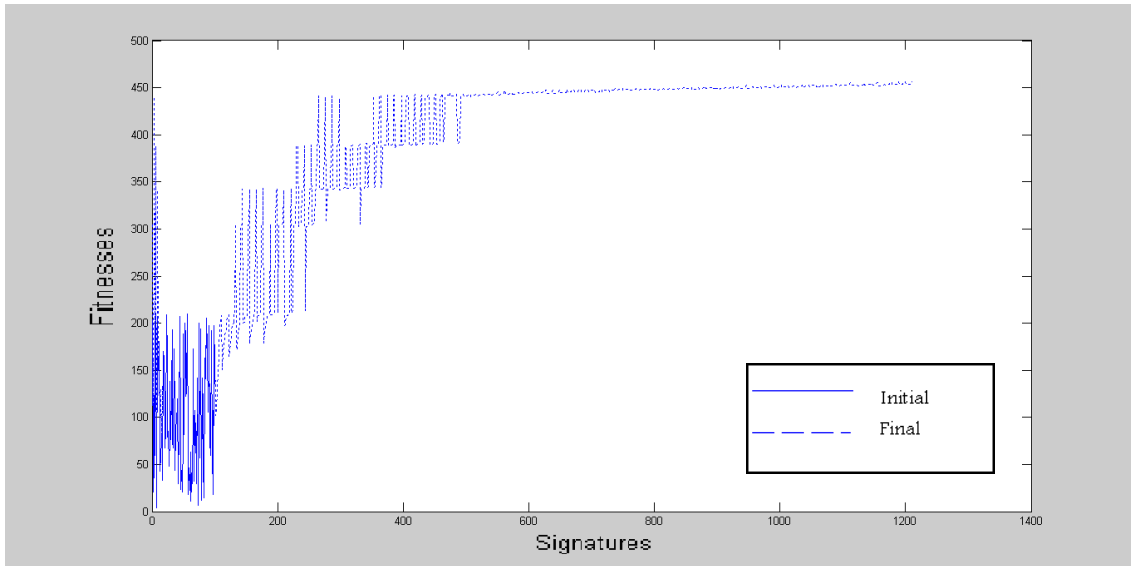


Figure 32: The forth run final population

The **fifth** run parameters are: *Learning Gen*=300, *Pm*=0.1 and *Fat* = 0.1, and the files' pool with 25% of infected files, and the produced signatures' pool is saved as **Sig5**.

Table (9) shows the results of the fifth run, where each iteration displays the number of signatures in the signatures' pool. The number of signatures value is 100 on the first iteration and it is increased by 11 signatures or less in each iteration. As shown in Table (9), the number of signatures in the last iteration is 3400. The selection threshold ranges between 0.6 and 1.0; this threshold determines the selection process stochastically, as described in step 20 in the VDC algorithm in section 3.1.

The Mean fitness, first increases at a rapid rate, then the increase continues at a slower rate; the Mean fitness in the beginning increases at a higher changing rate reaching 17.17, then the rate increases to reach 23.61 in the second iteration, to become 24.60 in the third iteration, till it reaches around 0.25 in the last five iterations, as shown in Table (9).

Whereas the Best fitness increases quickly in the first iteration only and later on the increase becomes slower; in the first iteration the changing rate for the Best fitness is 30, after that it increases by 1 with several iterations. Figure (33) represents the Mean fitness and the Best fitness.

Table 9: The fifth Training run results

Iteration number	No of signatures	Selection threshold	Mean fitness	Best fitness
1	100	0.80	160.11	210.00
2	111	0.90	177.28	440.00
3	122	0.80	200.89	440.00
4	133	0.90	225.49	441.00
5	144	0.60	252.07	441.00
...
296	3345	1.00	523.94	550.00
297	3356	1.00	524.18	551.00
298	3367	1.00	524.44	551.00
299	3378	1.00	524.68	551.00
300	3389	0.80	524.94	552.00
301	3400	0.80	525.19	552.00

As a result the number of detected infected files is 103 out of 125, and the Mean fitness = 525.1859.

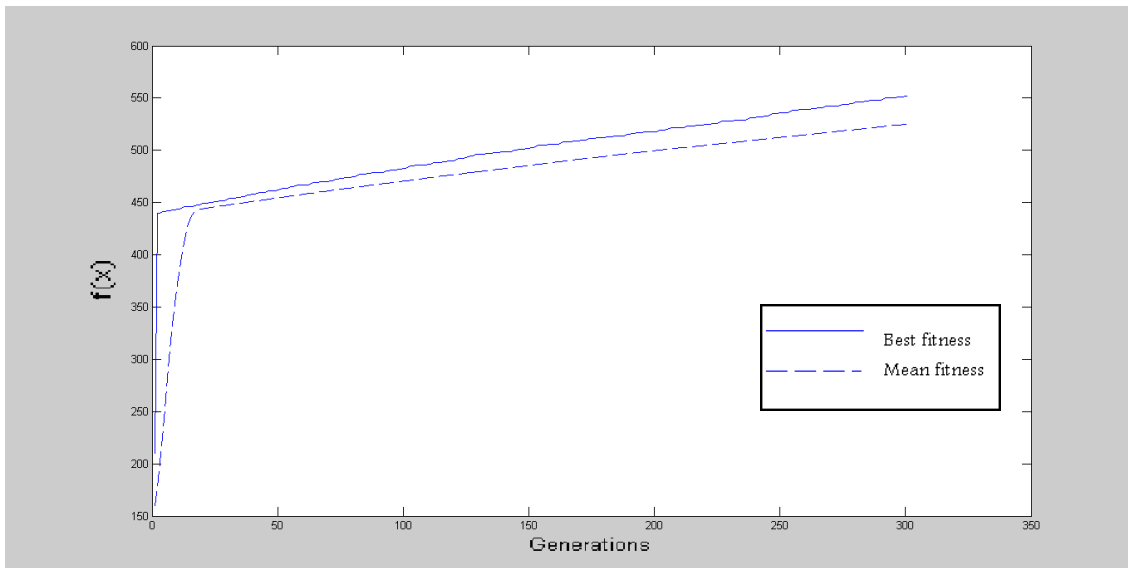


Figure 33: The fifth run Mean fitness & Best fitness

Figure (34) demonstrates the initial population in solid line, which is the same in Figure (24), and the final population including the new signatures after Hypermutation is in dotted line. As shown the fitness of the new signatures after Hypermutation is higher.

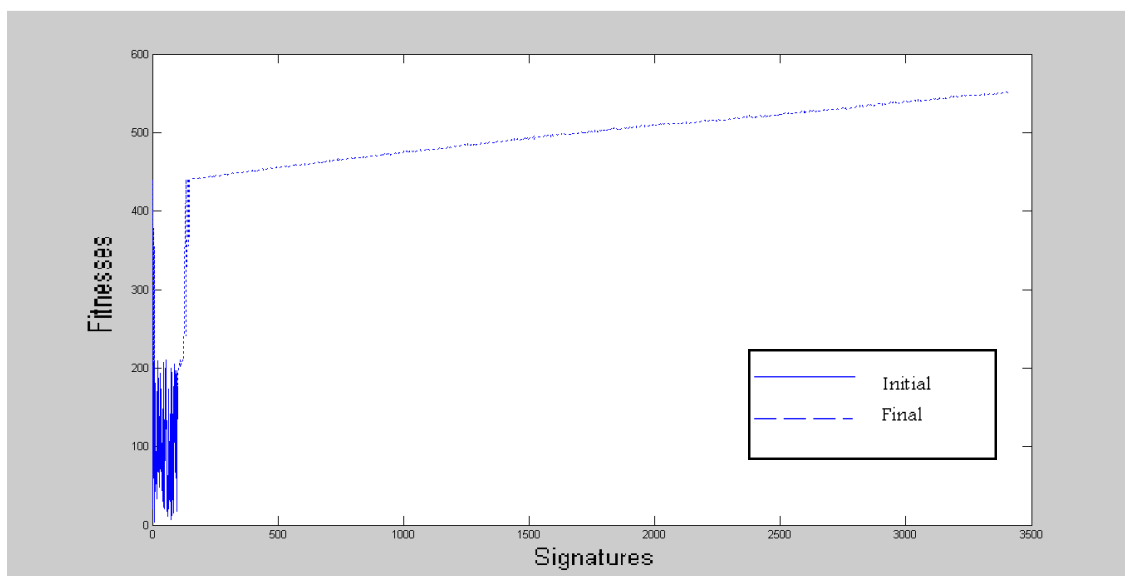


Figure 34: The fifth run final population

The **sixth** run parameters are: *Learning Gen*=150, *Pm*=0.2 and *Fat* = 0.05, and the files' pool with 25% of infected, files and the produced signatures' pool is saved as **Sig6**.

Table (10) shows the results of the sixth run, where each iteration displays the number of signatures in the signatures' pool. The number of signatures value is 100 on the first iteration and it is increased by 11 signatures or less in each iteration. As shown in Table (10), the number of signatures in the last iteration is 1750.

The selection threshold ranges between 0.6 and 1.0; this threshold determines the selection process stochastically, as described in step 20 in the VDC algorithm in section 3.1.

Table 10: The sixth Training run results

Iteration number	No of signatures	Selection threshold	Mean fitness	Best fitness
1	100	0.70	160.11	210.00
2	111	0.70	179.63	598.00
3	122	0.70	200.41	598.00
4	133	0.60	213.71	598.00
5	144	1.00	222.95	599.00
...
146	1695	1.00	641.52	655.00
147	1706	0.70	641.76	656.00
148	1717	0.90	642.03	656.00
149	1728	0.70	642.28	656.00
150	1739	0.70	642.54	656.00
151	1750	1.00	642.79	657.00

The Mean fitness, first increases at a rapid rate, then the increase continues at a slower rate; the Mean fitness in the beginning increases at a higher changing rate reaching 19.52, then the rate increases to reach 20.78 in the second iteration, to become 13.3 in the third iteration, till it reaches around 0.25 in the last five iterations, as shown in Table (10).

Whereas the Best fitness increases quickly in the first iteration only and later on the increase becomes slower; in the first iteration the changing rate for the Best fitness is 388, after that it increases by 1 with several iterations. Figure (35) represents the Mean fitness and the Best fitness.

As a result the number of detected infected files is 103 out of 125, and the Mean fitness = 642.7941.

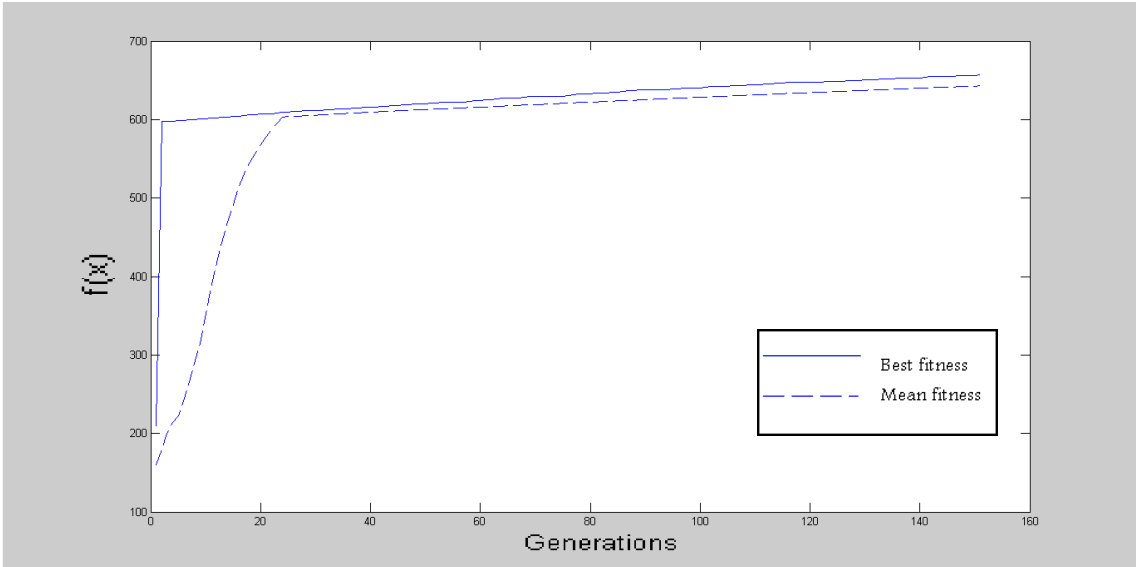


Figure 35: The sixth run Mean fitness & Best fitness

Figure (36) demonstrates the initial population in solid line, which is the same in Figure (24), and the final population including the new signatures after Hypermutation is in dotted line. As shown the fitness of the new signatures after Hypermutation is higher.

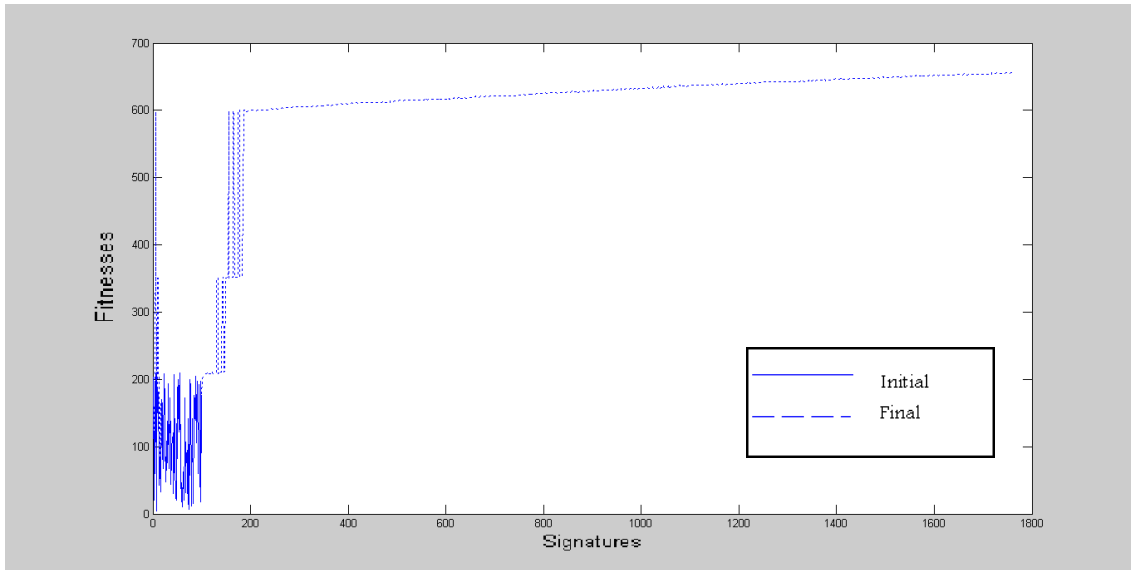


Figure 36: The sixth run final population

The **seventh** run parameters are: *Learning Gen*=100, *Pm*=0.2 and *Fat* = 0.05, and the files' pool with 75% of infected files, and the produced signatures' pool is saved as **Sig7**.

Table (11) shows the results of the seventh run, where each iteration displays the number of signatures in the signatures' pool. The number of signatures value is 100 on the first iteration and it is increased by 11 signatures or less in each iteration. As shown in Table (11), the number of signatures in the last iteration is 1200. The selection threshold ranges between 0.6 and 1.0; this threshold determines the selection process stochastically, as described in step 20 in the VDC algorithm in section 3.1.

The Mean fitness, first increases at a rapid rate, then the increase continues at a slower rate; the Mean fitness in the beginning increases at a higher changing rate reaching 28.17, then the rate increases to reach 47.82 in the second iteration, to become 31.18 in the third iteration, till it reaches around 0.29 in the last five iterations, as shown in Table (11).

Whereas the Best fitness increases quickly in the first iteration only and later on the increase becomes slower; in the first iteration the changing rate is 598 for the Best fitness, after that it increases by 1 with several iterations. Figure (37) represents the Mean fitness and the Best fitness.

Table 11: The seventh Training run results

Iteration number	No of signatures	Selection threshold	Mean fitness	Best fitness
1	100	0.70	160.11	210.00
2	111	0.70	188.28	808.00
3	122	0.70	236.10	808.00
4	133	0.60	267.28	808.00
5	144	1.00	300.86	809.00
...
96	1145	0.80	837.22	849.00
97	1156	0.70	837.53	850.00
98	1167	0.80	837.84	850.00
99	1178	0.60	838.12	850.00
100	1189	0.60	838.43	850.00
101	1200	0.70	838.74	851.00

As a result the number of detected infected files is 357 out of 375, and the Mean fitness = 838.7423.

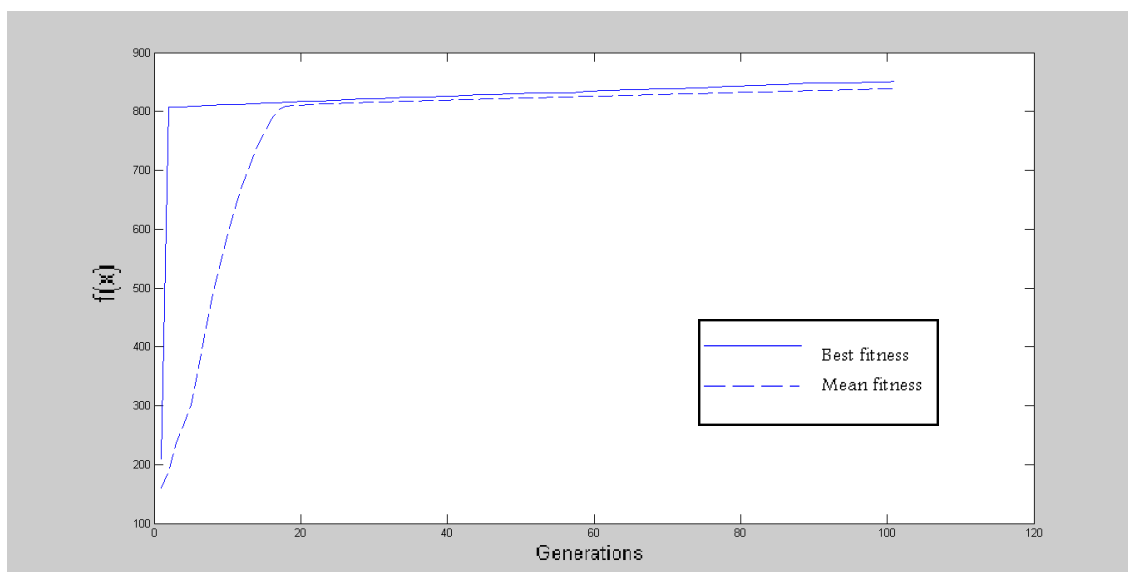


Figure 37: The seventh run Mean fitness & Best fitness

Figure (38) demonstrates the initial population in solid line, which is the same in Figure (24), and the final population including the new signatures after Hypermutation is in dotted line. As shown the fitness of the new signatures after Hypermutation is higher.

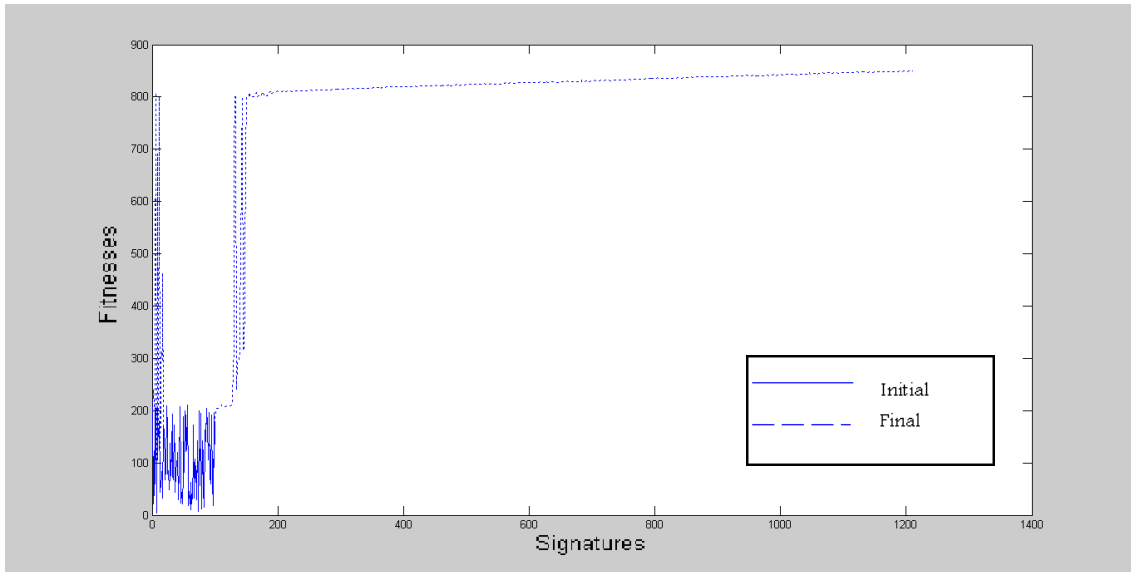


Figure 38: The seventh run final population

The **eighth** run parameters are: *Learning Gen*=150, *Pm*=0.1 and *Fat* = 0.1, and the files' pool with 75% of infected files, and the produced signatures' pool is saved as **Sig8**.

Table (12) shows the results of the eighth run, where each iteration displays the number of signatures in the signatures' pool. The number of signatures value is 100 on the first iteration and it is increased by 11 signatures or less in each iteration. As shown in Table (12), the number of signatures in the last iteration is 1750. The selection threshold ranges between 0.6 and 1.0; this threshold determines the selection process stochastically, as described in step 20 in the VDC algorithm in section 3.1.

The Mean fitness, first increases at a rapid rate, then the increase continues at a slower rate; the Mean fitness in the beginning increases at a higher changing rate reaching 46.42, then the rate increases to reach 63.06 in the second iteration, to become 6.29 in the third iteration, till it reaches around 0.29 in the last five iterations, as shown in Table (12).

Whereas the Best fitness increases quickly in the first iteration only and later on the increase becomes slower; in the first iteration the changing rate for the Best fitness is 1010, after that it increases by 1 with several iterations. Figure (39) represents the Mean fitness and the Best fitness.

Table 12: The eighth Training run results

Iteration number	No of signatures	Selection threshold	Mean fitness	Best fitness
1	100	0.80	160.11	210.00
2	111	0.90	179.99	1220.00
3	122	0.80	226.41	1220.00
4	133	0.90	289.47	1221.00
5	144	0.60	372.35	1221.00
...
146	1695	0.60	1264.41	1281.00
147	1706	1.00	1264.70	1281.00
148	1717	0.90	1265.01	1281.00
149	1728	0.60	1265.28	1282.00
150	1739	0.80	1265.58	1282.00
151	1750	0.90	1265.86	1282.00

As a result the number of detected infected files is 252 out of 375, and the Mean fitness = 1265.9000.

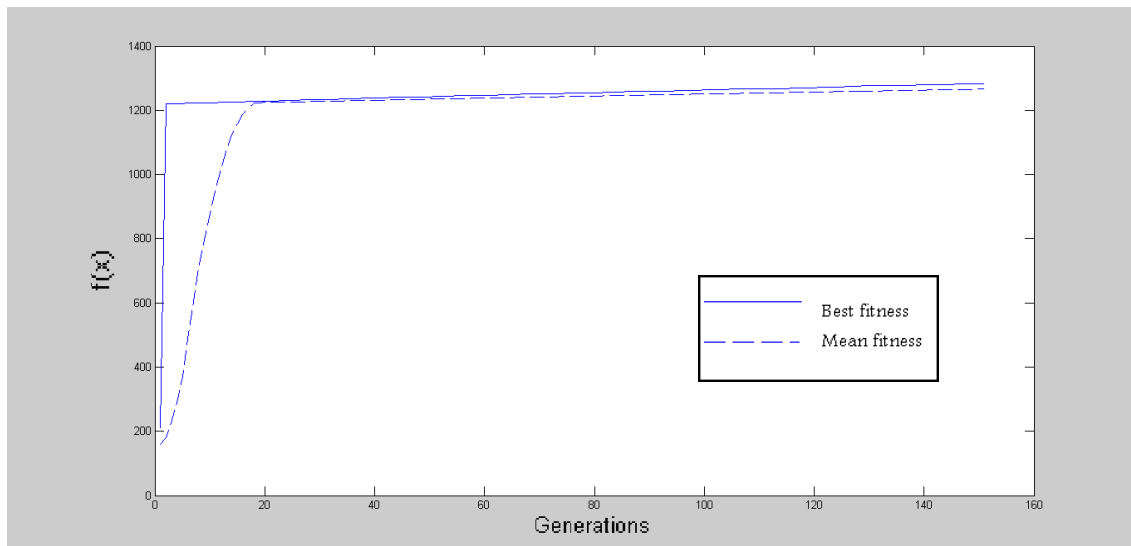


Figure 39: The eighth run Mean fitness & Best fitness

Figure (40) demonstrates the initial population in solid line, which is the same in Figure (24), and the final population including the new signatures after Hypermutation is in dotted line. As shown the fitness of the new signatures after Hypermutation is higher.

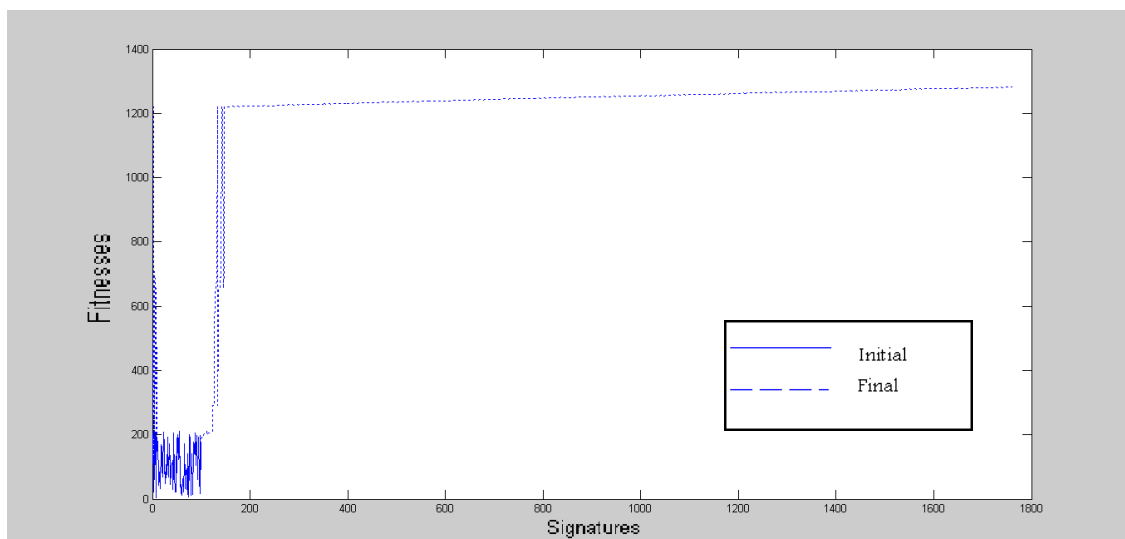


Figure 40: The eighth run final population

The **ninth** run parameters are: *Learning Gen*=300, *Pm*=0.05 and *Fat* = 0.1, and the files' pool with 75% of infected files, and the produced signatures' pool is saved as **Sig9**.

Table (13) shows the results of the ninth run, where each iteration displays the number of signatures in the signatures' pool. The number of signatures value is 100 on the first iteration and it is increased by 11 signatures or less in each iteration. As shown in Table (13), the number of signatures in the last iteration is 3400. The selection threshold ranges between 0.6 and 1.0; this threshold determines the selection process stochastically, as described in step 20 in the VDC algorithm in section 3.1.

The Mean fitness, first increases at a rapid rate, then the increase continues at a slower rate; the Mean fitness in the beginning increases at a higher changing rate reaching 17.81, then the rate increases to reach 43.02 in the second iteration, to become 10.21 in the third iteration, till it reaches around 0.15 in the last five iterations, as shown in Table (13). Whereas the Best fitness increases quickly in the first iteration only and later on the increase becomes slower; in the first iteration the changing rate is 981 for the Best fitness, after that it increases by 1 with several iterations. Figure (41) represents the Mean fitness and the Best fitness.

Table 13: The ninth Training run results

Iteration number	No of signatures	Selection threshold	Mean fitness	Best fitness
1	100	0.80	160.11	210.00
2	111	0.90	177.92	1191.00
3	122	0.90	220.94	1191.00
4	133	0.90	231.15	1191.00
5	144	0.90	241.01	1191.00
...
296	3345	0.60	1242.22	1259.00
297	3356	0.90	1242.36	1259.00
298	3367	0.70	1242.52	1260.00
299	3378	1.00	1242.68	1260.00
300	3389	0.60	1242.83	1260.00
301	3400	0.60	1242.97	1261.00

As a result the number of detected infected files is 267 out of 375, and the Mean fitness = 1243.0000.

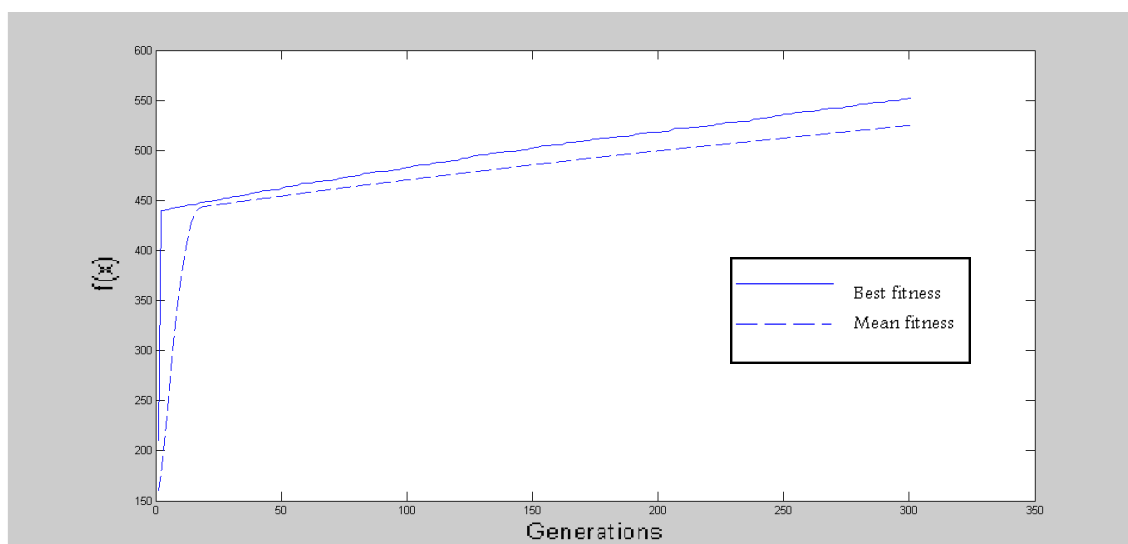


Figure 41: The ninth run Mean fitness & Best fitness

Figure (42) demonstrates the initial population in solid line, which is the same in Figure (24), and the final population including the new signatures after Hypermutation is in dotted line. As shown the fitness of the new signatures after Hypermutation is higher.

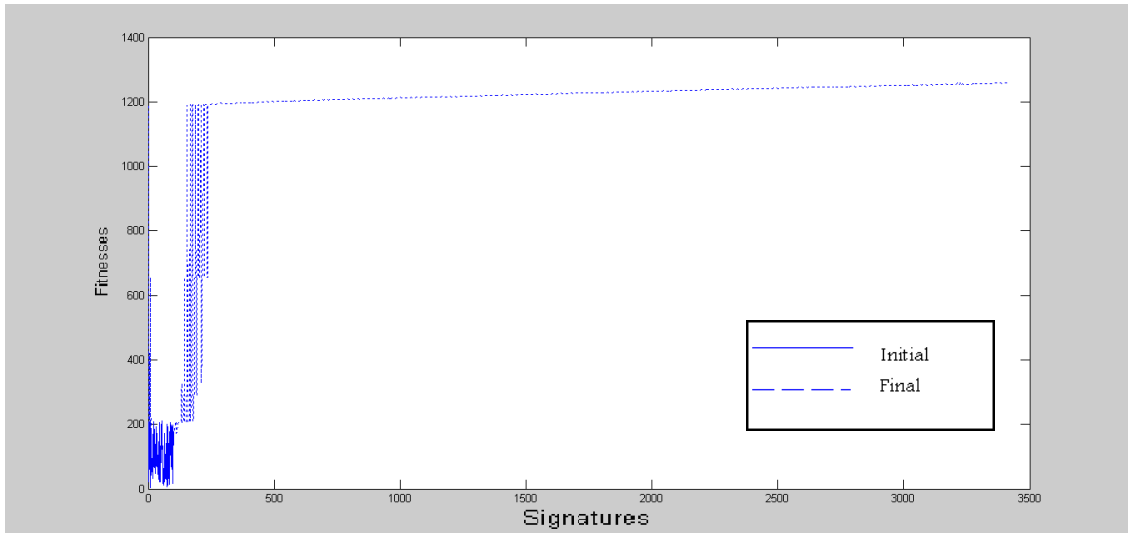


Figure 42: The ninth run final population

The **tenth** run parameters are: *Learning Gen*=100, *Pm*=0.2 and *Fat* = 0.05 and the files' pool with 25% of infected files, and the produced signatures' pool is saved as **Sig10**.

Table (14) shows the results of the tenth run, where each iteration displays the number of signatures in the signatures' pool. The number of signatures value is 100 on the first iteration and it is increased by 11 signatures or less in each iteration. As shown in Table (14), the number of signatures in the last iteration is 1200. The selection threshold ranges between 0.6 and 1.0; this threshold determines the selection process stochastically, as described in step 20 in the VDC algorithm in section 3.1.

The Mean fitness, first increases at a rapid rate, then the increase continues at a slower rate; the Mean fitness in the beginning increases at a higher changing rate reaching 19.52, then the rate increases to reach 20.78 in the second iteration, to become 13.3 in the third iteration, till it reaches around 0.30 in the last five iterations, as shown in Table (14).

Whereas the Best fitness increases quickly in the first iteration only and later on the increase becomes slower; in the first iteration the changing rate is 388 for the Best fitness, after that it increases by 1 with several iterations. Figure (43) represents the Mean fitness and the Best fitness.

Table 14: The tenth Training run results

Iteration number	No of signatures	Selection threshold	Mean fitness	Best fitness
1	100	0.70	160.11	210.00
2	111	0.70	179.63	598.00
3	122	0.70	200.41	598.00
4	133	0.60	213.71	598.00
5	144	1.00	222.95	599.00
...
96	1145	0.80	627.22	639.00
97	1156	0.70	627.53	640.00
98	1167	0.80	627.84	640.00
99	1178	0.60	628.12	640.00
100	1189	0.60	628.43	640.00
101	1200	0.70	628.74	641.00

As a result the number of detected infected files is 103 out of 125, and the Mean fitness = 628.7423.

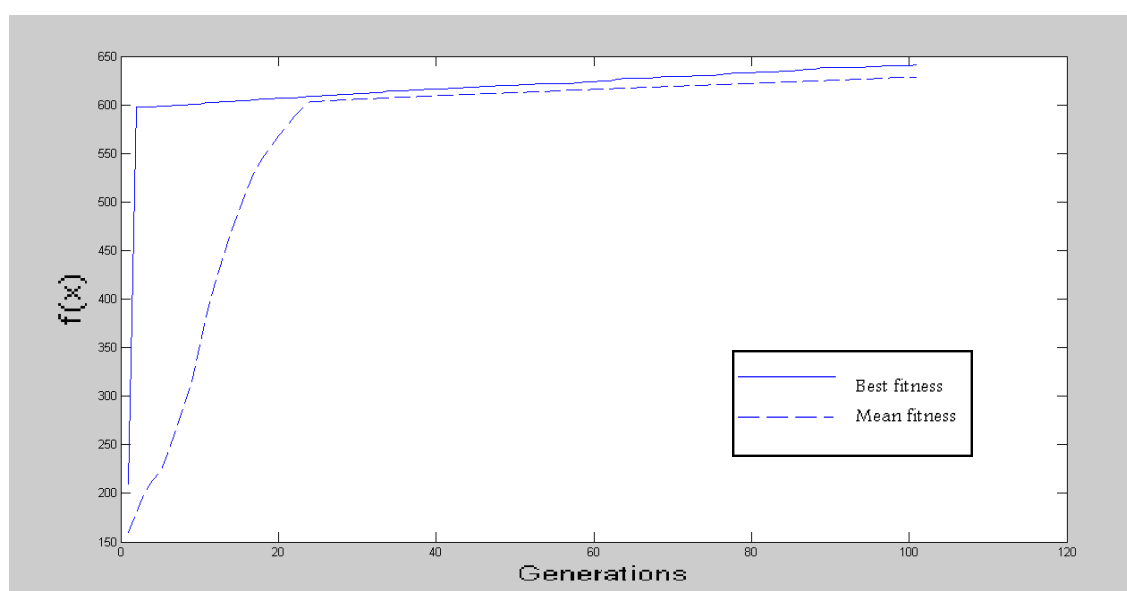


Figure 43: The tenth run Mean fitness & Best fitness

The initial population is demonstrated in solid line at Figure (44), which is the same in Figure (24), and the final population including the new signatures after Hypermutation is in dotted line. As shown the fitness of the new signatures after Hypermutation is higher.

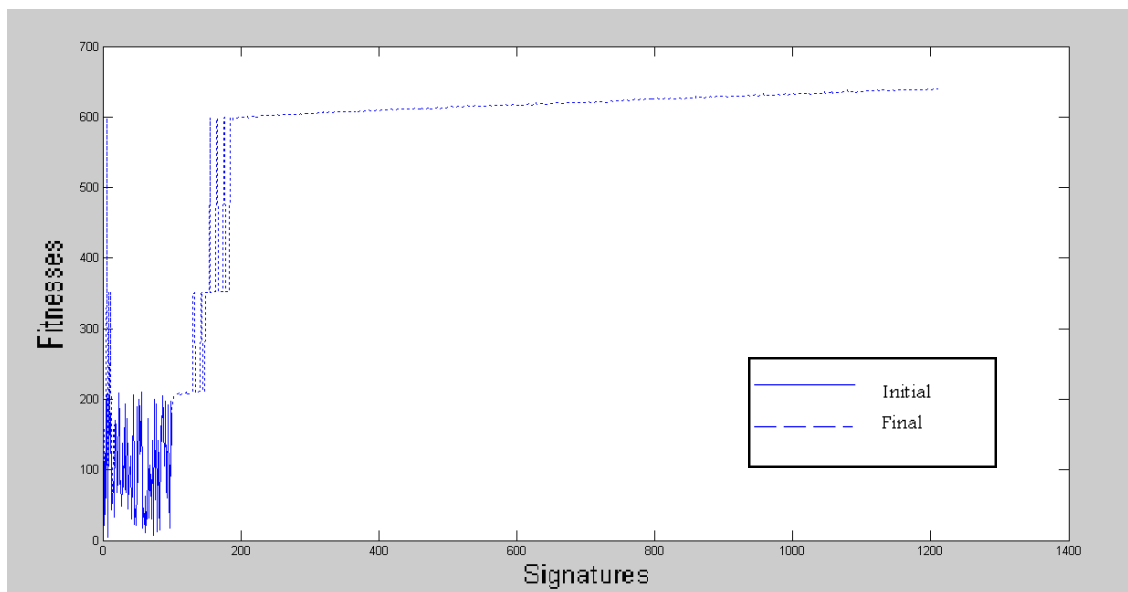


Figure 44: The tenth run final population

The **eleventh** run parameters are: *Learning Gen*=100, *Pm*=0.2 and *Fat* = 0.1, and the files' pool with 75% of infected files, and the produced signatures' pool is saved as **Sig11**. Table (15) shows the results of the eleventh run, where each iteration displays the number of signatures in the signatures' pool. The number of signatures value is 100 on the first iteration and it is increased by 11 signatures or less in each iteration. As shown in Table (15), the number of signatures in the last iteration is 1200.

The selection threshold ranges between 0.6 and 1.0; this threshold determines the selection process stochastically, as described in step 20 in the VDC algorithm in section 3.1.

The Mean fitness, first increases at a rapid rate, then the increase continues at a slower rate; the Mean fitness in the beginning increases at a higher changing rate reaching 22.52, then the rate increases to reach 56.90 in the second iteration, to become 85.06 in the third iteration, till it reaches around 0.42 in the last five iterations, as shown in Table (15). Whereas the Best fitness increases quickly in the first iteration only and later on the increase becomes slower; in the first iteration the changing rate is 617 for the Best fitness, after that it increases by 1 with several iterations. Figure (45) represents the Mean fitness and the Best fitness.

Table 15: The eleventh Training run results

Iteration number	No of signatures	Selection threshold	Mean fitness	Best fitness
1	100	0.80	160.11	211.00
2	111	0.90	182.63	828.00
3	122	0.80	239.53	829.00
4	133	0.90	324.59	829.00
5	144	0.60	411.88	830.00
...
96	1145	1.00	869.65	886.00
97	1156	0.90	870.05	886.00
98	1167	0.70	870.50	887.00
99	1178	1.00	870.90	887.00
100	1189	0.80	871.34	888.00
101	1200	0.90	871.76	889.00

As a result the number of detected infected files is 341 out of 375, and the Mean fitness = 871.7612.

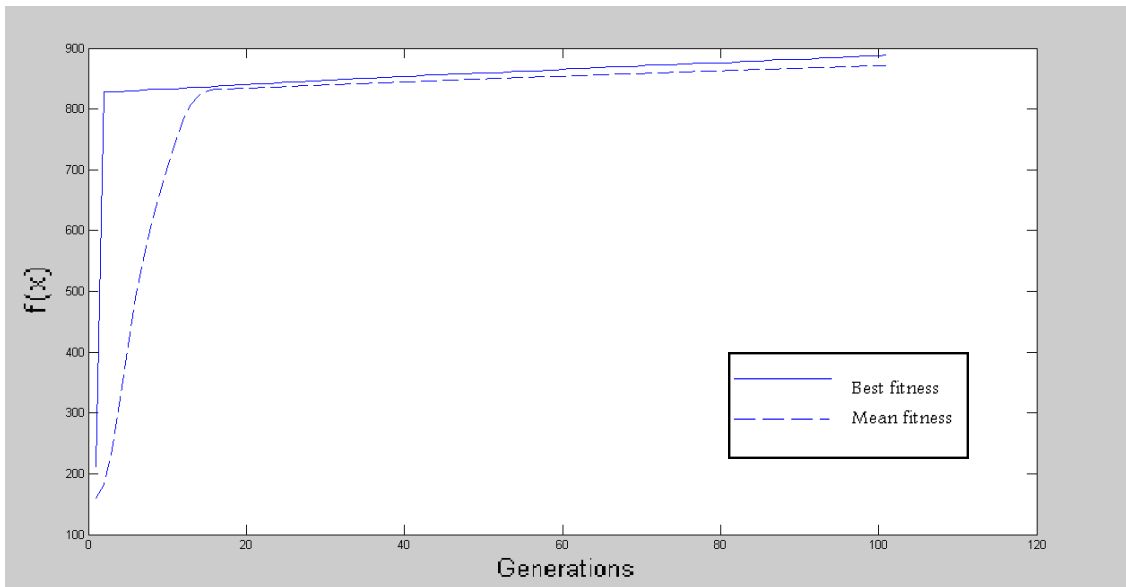


Figure 45: The eleventh run Mean fitness & Best fitness

Figure (46) demonstrates the initial population in solid line, which is the same in Figure (24), and the final population including the new signatures after Hypermutation is in dotted line. As shown the fitness of the new signatures after Hypermutation is higher.

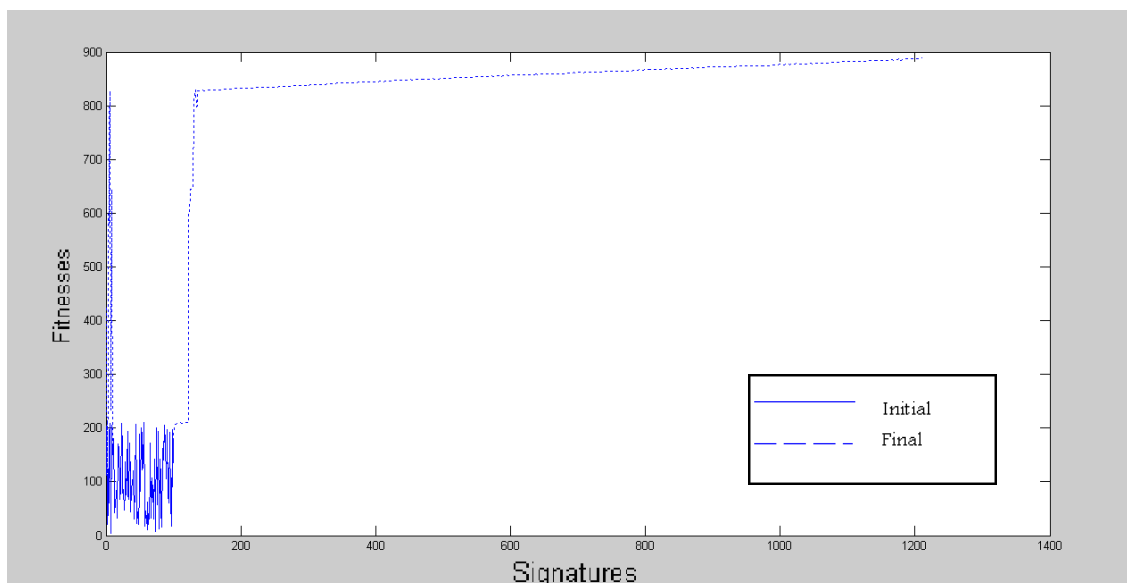


Figure 46: The eleventh run final population

The **twelfth** run parameters are: *Learning Gen*=100, *Pm*=0.05 and *Fat* = 0.1, and the files' pool with 5% of infected files, and the produced signatures' pool is saved as **Sig12**. Table (16) shows the results of the twelfth run, where each iteration displays the number of signatures in the signatures' pool. The number of signatures value is 100 on the first iteration and it is increased by 11 signatures or less in each iteration. As shown in Table (16), the number of signatures in the last iteration is 1200.

The selection threshold ranges between 0.6 and 1.0; this threshold determines the selection process stochastically, as described in step 20 in the VDC algorithm in section 3.1.

The Mean fitness, first increases at a rapid rate, then the increase continues at a slower rate; the Mean fitness in the beginning increases at a higher changing rate reaching 13.54, then the rate decreases to reach 9.47 in the second iteration, to become 8.42 in the third iteration, till it reaches around 0.19 in the last five iterations, as shown in Table (16). Whereas the Best fitness increases quickly in the first iteration only and later on the increase becomes slower; in the first iteration the changing rate is 15 for the Best fitness, after that it increases by 1 with several iterations. Figure (47) represents the Mean fitness and the Best fitness.

Table 16: The twelfth Training run results

Iteration number	No. of signatures	Selection threshold	Mean fitness	Best fitness
1	100	0.80	160.11	210.00
2	111	0.90	173.65	225.00
3	122	0.90	183.12	225.00
4	133	0.90	191.54	225.00
5	144	0.90	196.75	225.00
...
96	1145	0.60	242.62	251.00
97	1156	0.80	242.81	251.00
98	1167	0.80	243.02	251.00
99	1178	0.80	243.20	252.00
100	1189	0.90	243.40	252.00
101	1200	0.60	243.59	252.00

As a result the number of detected infected files is 17 out of 25, and the Mean fitness =243.5857.

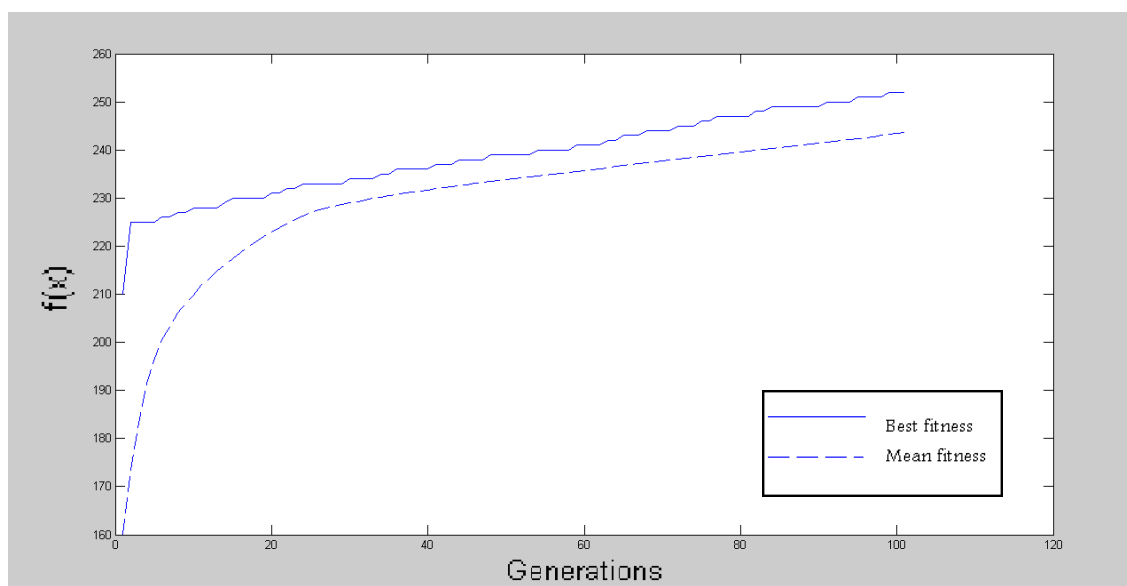


Figure 47: The twelfth run Mean fitness & Best fitness

The initial population is demonstrated in solid line at Figure (48), which is the same in Figure (24), and the final population including the new signatures after Hypermutation is in dotted line. As shown the fitness of the new signatures after Hypermutation is higher.

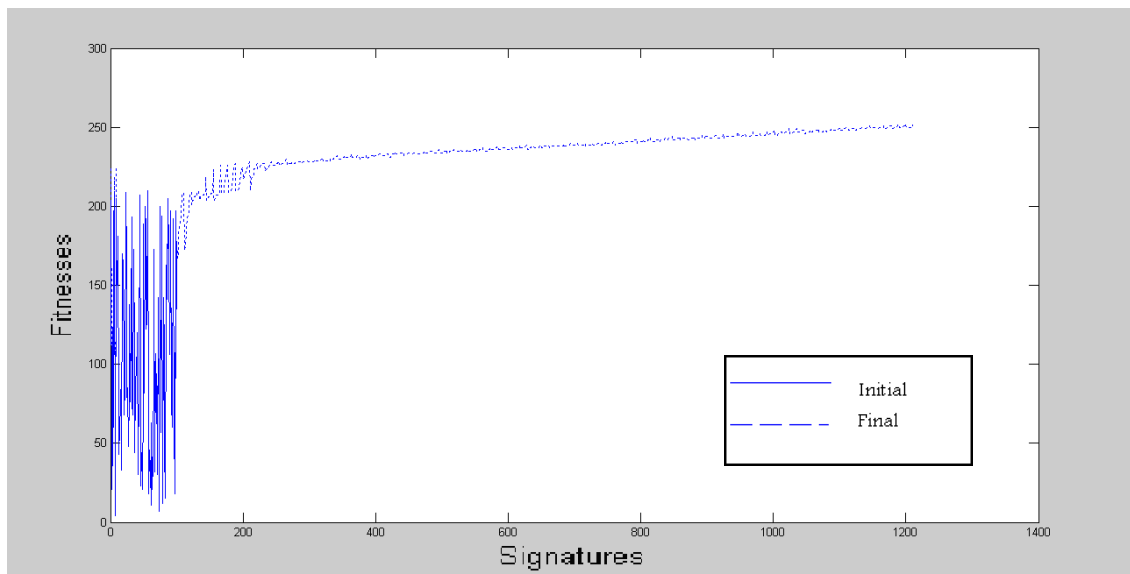


Figure 48: The twelfth run final population

4.1.1 The Training Phase Analysis

At the end of the Training phase, which includes 12 runs, the results of these runs are summarized in Table (17). This table shows the Mean fitness and the number of signatures for all runs that are already conducted. As noticed in this Table, there are changes in the Mean fitness and the number of signatures, and that's due to the changes in the variables values (*Learning Gen*, *Pm*, *Fat* and Training pool). As for the variable *Pm*, it has the values 0.05, 0.1 or 0.2 and this change affects the Hypermutation probability that changes signatures, to produce new mutations, in the attempt to develop new signatures.

While for the variable *Fat*, with its values of 0.05 or 0.1, when *Fat* =0.05 the number of elements per clone is 5, and when *Fat*=0.1 the number of elements per clone is 10, so when the number of elements increases at the clone, it affects the number of signatures copies that enters the Hypermutation process, by increasing them. This means that the effect of Hypermutation is controlled by these two parameters together (*Pm* and *Fat*), and each Hypermutation affects the fitness by adding 1 at most.

For the *learning Gen* variable, it represents the number of generations which is processed by the algorithm. When the number of generations increase, the percentage of the new signatures that are added to the signatures' pool

increases, and that's because in each iteration the highest fitness of 11 new signatures (hypermuted) are selected stochastically. Knowing that the number of new signatures can be less than 11 in one generation, when the number of all new signatures that corresponds with the value of the selection threshold between 0.6 and 1.0 are less than 11 in that generation. For example, the case with Sig1, where the number of signatures is 1198, while it must be 1200; if 11 signatures are added in each generation, (11 Signatures * 100 generations + the 100 signatures in the initial population). Typically, if the *learning Gen*=100, the number of signatures is 1200, and when the *learning Gen* = 150, the number of signatures is 1750, and finally when the *learning Gen* = 300, the number of signatures is 3400. The *Leaning Gen* affects the Mean fitness.

Whereas for the Training pool variable, which has the values of 5%, 25% or 75% of infected files, it affects the Mean fitness, whenever the infected files are increased, the added value on the fitness increases due to the effect of the detection; as each detection adds δ (=10) on the fitness. Knowing that, the effect of the Hypermutation on the fitness is by adding 1 at most in each Hypermutation. So in the Training pool whenever the number of infected files increases, the Mean fitness increases, and this variable has the higher effect on the Mean fitness as shown in Table (17).

Table 17: The Summary of the training results

Parameters			Signatures' pool	Mean fitness	No. of signatures
Learning Gen	Pm	Fat			
5% infected files					
100	0.05	0.05	Sig1	247.1486	1198
100	0.1	0.05	Sig2	242.2893	1200
300	0.05	0.1	Sig3	275.2660	3400
100	0.05	0.1	Sig12	243.5857	1200
25% infected files					
100	0.05	0.05	Sig4	449.6050	1200
300	0.1	0.1	Sig5	525.1859	3400
150	0.2	0.05	Sig6	642.7941	1750
100	0.2	0.05	Sig10	628.7423	1200
75% infected files					
100	0.2	0.05	Sig7	838.7423	1200
150	0.1	0.1	Sig8	1265.9	1750

300	0.05	0.1	Sig9	1243.0000	3400
100	0.2	0.1	Sig11	871.7612	1200

The fitness increases by either the detection (each detection adds δ), or by the Hypermutation (each Hypermutation increases 1 at most). The detection depends on the Training pool (number of infected files). The Hypermutation is controlled by the *Pm* and *Fat*, and when the *Learning Gen* increases, the times of Hypermutation increase, hence the fitness increase.

The training phase is only done to produce the signatures' pools (Sig1... Sig12) for the matching phase, hence the space is lacking the opportunity for future conclusions, because the Cloning, Hypermutation and Detection are just carried on the half size of the signatures' pool.

4.2 Matching the VDC algorithm

The matching phase runs all the six files' pools mentioned in section 3.3.2, to test the signatures' pools which are filled in the training phase previously, and they are: Sig1, Sig2, Sig3, Sig4, Sig5, Sig6, Sig7, Sig8, Sig9, Sig10, Sig11 and Sig12. The description of these pools is represented at Table (2).

To test the concept of the false positive in the beginning, Sig1, Sig5 and Sig8 are run on the files' pool with 0% of infected files, then matching each of Sig6, Sig7, Sig10 and Sig11 on the files' pool with 5% of infected files follows, with the addition of 100 files when the *matching Gen* = 5. Followed by matching each of Sig1, Sig2 and Sig12 on the files' pool with 25% of infected files follows, with the addition of 100 files when the *matching Gen* = 50. After that, matching each of Sig4, Sig5 and Sig8 on the files' pool with 50% of infected files commences, with the addition of 100 files when the *matching Gen* = 50. Next, matching each of Sig1, Sig3, Sig6, Sig9 and Sig12 on the files' pool with 75% of infected files begins, with the addition of 100 files when the *matching Gen* = 5. After ward, matching each of Sig2, Sig4, Sig7, Sig10, Sig11 and Sig12 on the files' pool with 100% of infected files commences, with the addition of 100 files when the *matching Gen* = 50.

It must be noted that the 100 files that are added after the first iteration contain benign and infected files.

The infected files are come from three sources: the first source is files with signatures used at the training phase that already exist in the original files' pool. The second source is the files with signatures which are not used at the training phase and do not exist in the original files' pool. The third source is the files with mutated signatures that are produced in the training phase. Knowing that, the *matching Gen* for all matching runs equals 100.

The matching of **Sig1** with the files' pool with 0% infected files (all the files are benign). The results are shown in Table (18).

Table 18: The results of the matching of Sig1 with 0% infected files

Iteration number	Mean fitness	Best fitness
1	226.6609	253.00
2	226.6609	253.00
3	226.6609	253.00
4	226.6609	253.00
5	226.6609	253.00
...
96	226.6609	253.00
97	226.6609	253.00
98	226.6609	253.00
99	226.6609	253.00
100	226.6609	253.00
101	226.6609	253.00

As a result, the number of infected files = 0 and the Mean fitness = 226.6609.

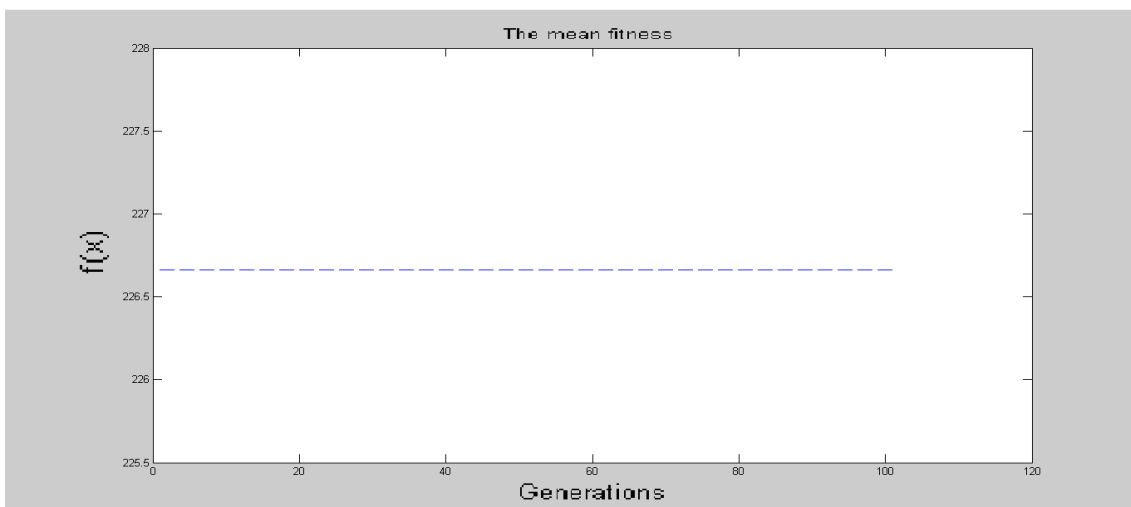


Figure 49: The Mean fitness of matching Sig1 run with 0% infected files

Table (18) and Figures (49) and (50) illustrate that none of the files are detected as infected files, and as noticed the Mean fitness and Best fitness do not change. This is due to the fact that all files are benign. Hence, the detection rate is 100%. The result properly reflects the reality, and it is an accepted result.

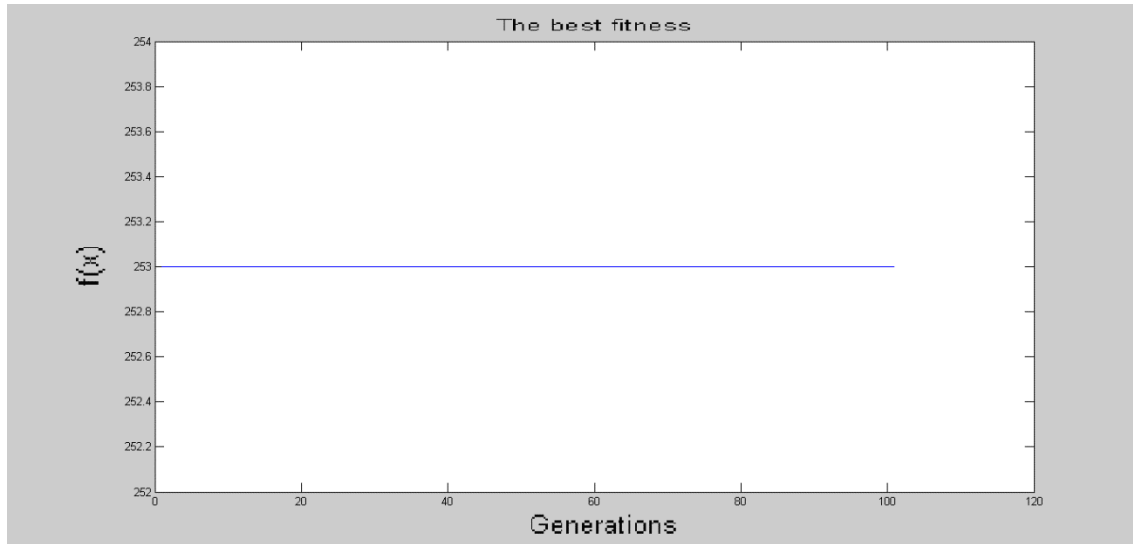


Figure 50: The Best fitness of matching Sig1 run with 0% infected files

The matching results of **Sig5** and **Sig8** with the files' pool with 0% infected files (all the files are benign) are summarized Table (19).

Table 19: The summary of matching of Sig5 and Sig8 with 0% infected files

Signatures pool	Matching Pool	Detected Files	Mean fitness	Best fitness	Detection Rate
Sig5	0%	0	485.3166	552.00	100%
Sig8	0%	0	1167.21	1283.00	100%

In both matching runs for Sig5 and Sig8, none of the files are detected as infected files, and the Mean fitness and Best fitness do not change. This is due to the fact that all files are benign. Hence, the detection rate is 100%.

The matching of **Sig6** with the files pool with 5% infected files. At iteration number 5 new 100 files are added to the files' pool, with 5% of these files are infected. The results are shown in Table (20).

Table 20: The results of the matching of Sig6 with 5% infected files

Iteration number	Mean fitness	Best fitness
1	585.44804	657.00
2	585.46167	657.00
3	585.46167	657.00
4	585.46167	657.00
5	585.46167	657.00
6	585.46394	657.00
...
97	585.46394	657.00
98	585.46394	657.00
99	585.46394	657.00
100	585.46394	657.00
101	585.46394	657.00

As a result, the number of infected files= 28 and the Mean fitness = 585.46394.

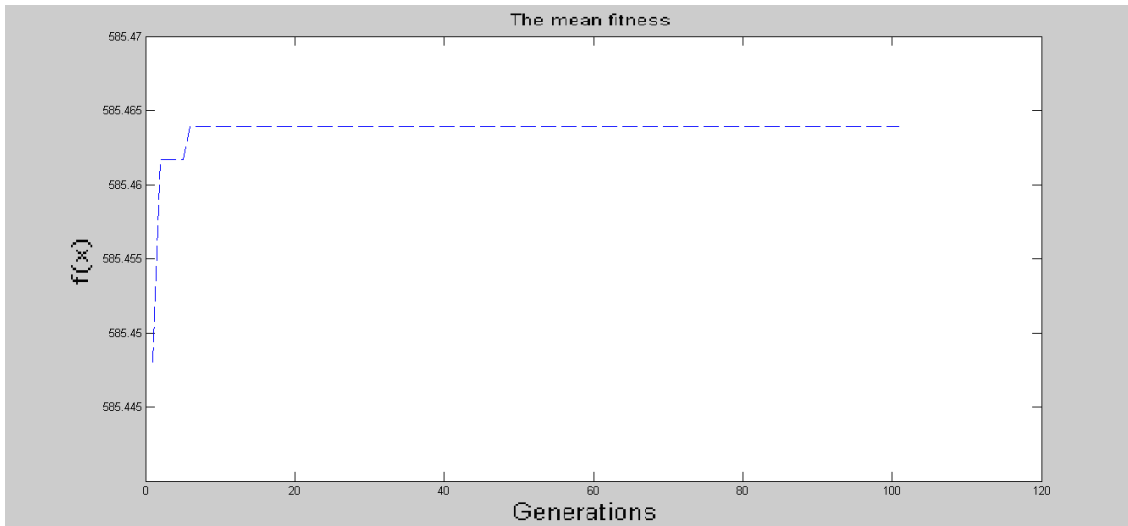


Figure 51: The Mean fitness of matching Sig6 run with 5% infected files

Table (20) and Figure (51) show that the Mean fitness increases by 0.01363 in the first iteration whereas it increases at the iteration number 5 by 0.00227. The Δ Mean fitness appears in the first and fifth iteration only, as a result of the detection process. In the first iteration the matching algorithm detects the original files in the files' pool (500), after that in the fifth iteration, when the 100 files are added, the matching algorithm detects them.

The Best fitness does not change; this explains the straight line in Figure (52). The number of detected files is 28 out of 30 (25+5) with a Detection rate of 93.3%, where the 25 infected files are in the original pool, and the 5 are from the new added pool. As a result this detection rate is accepted.

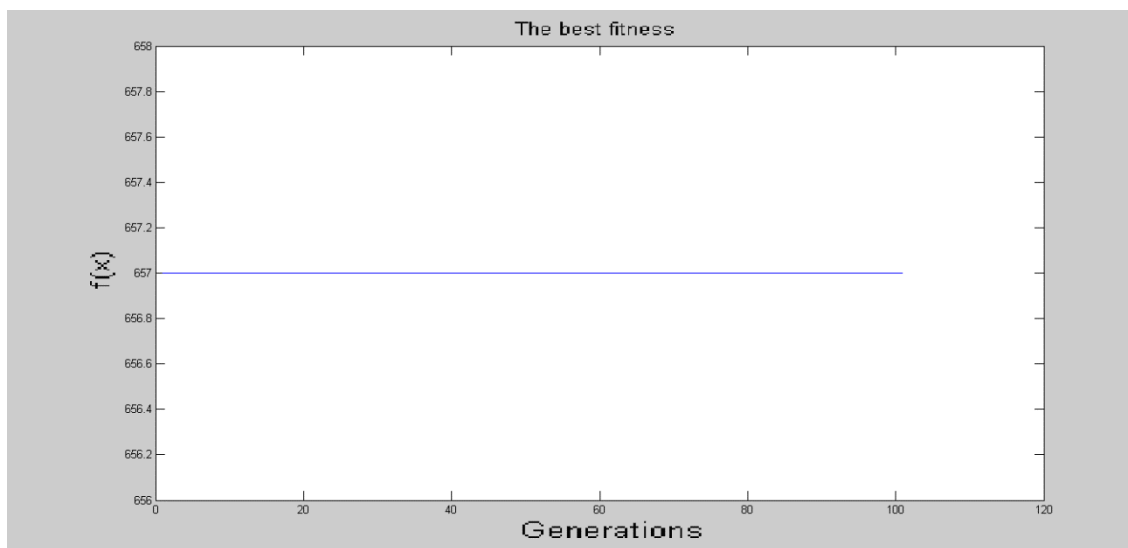


Figure 52: The Best fitness of matching Sig6 run with 5% infected files

The matching results of **Sig7**, **Sig10** and **Sig11** with the files pool with 5% infected files are summarized in Table (21). At iteration number 5 new 100 files are added to the files' pool, with 5% of these files are infected.

Table 21: The summary of the matching of Sig7, Sig10 and Sig11 with 5% infected files

Signatures pool	Matching Pool	Detected Files	Mean fitness	Best fitness	Detection Rate
Sig7	5%	28	924.6069	1042.00	93.3%
Sig10	5%	28	556.9083	641.00	93.3%
Sig11	5%	28	786.5623	889.00	93.3%

For these three matching runs the Mean fitness increases at two places; at the first iteration and iteration number 5. But the Best fitness does not change. The details of Sig7 are displayed in Appendix (B1).

The matching of **Sig1** with the files pool with 25% infected files. At iteration number 50 new 100 files are added to the files' pool, with 25% of these files are infected. The results are shown in Table (22).

Table 22: The results of the matching of Sig1 with 25% infected files

Iteration number	Mean fitness	Best fitness
1	226.66088	253.00
2	226.76344	263.00
3	226.76344	263.00
4	226.76344	263.00
5	226.76344	263.00
...
50	226.76344	263.00
51	226.77833	264.00
...
98	226.77833	264.00
99	226.77833	264.00
100	226.77833	264.00
101	226.77833	264.00

As a result, the number of infected files= 142 and the Mean fitness = 226.7783.

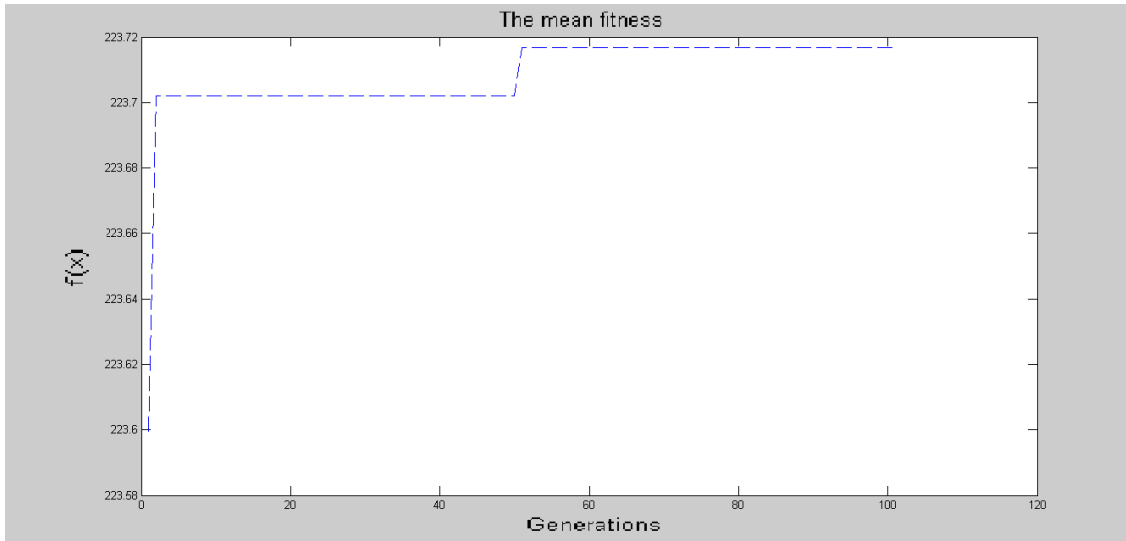


Figure 53: The Mean fitness of matching Sig1 run with 25% infected files

Table (22) and Figure (53) show that the Mean fitness increases by 0.10256 in the first iteration whereas it increases at the iteration number 50 by 0.01489.

The Best fitness increases at the first iteration by 10 whereas it increases at iteration number 50 by 1 as illustrated in Figure (54). The number of detected files is 142 out of 150 (125+25) with a Detection rate of 94.7%, where the 125 infected files are in the original pool, and the 25 are from the new added pool. As a result this detection rate is accepted.

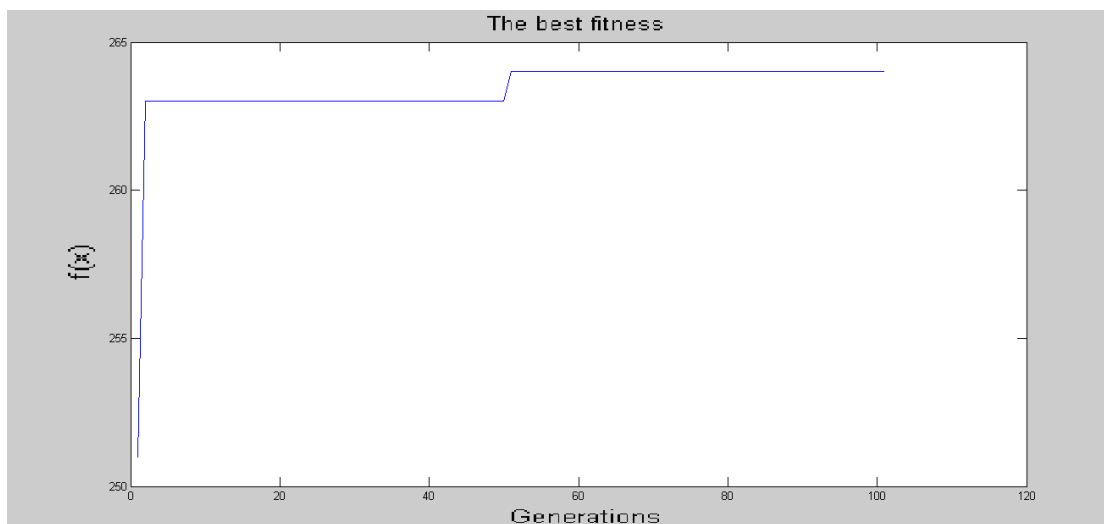


Figure 54: The Best fitness of matching Sig1 run with 25% infected files

The matching results of **Sig2** and **Sig12** with the files pool with 25% infected files are summarized in Table (23). At iteration number 50 new 100 files are added to the files' pool, with 25% of these files are infected.

Table 23: The summary of the matching of Sig2 and Sig12 with 25% infected files

Signatures pool	Matching Pool	Detected Files	Mean fitness	Best fitness	Detection Rate
Sig2	25%	142	223.7168	264.00	94.7%
Sig12	25%	142	225.3997	264.00	94.7%

The Mean fitness and the Best fitness increase in two places; in the first iteration and iteration number 50. The run of Sig2 is represented in Appendix (B2) and Sig12 in Appendix (B3).

The matching of **Sig4** with the files pool with 50% infected files. At iteration number 50 new 100 files are added to the files' pool, with 50% of these files are infected. The results are shown in Table (24).

Table 24: The results of the matching of Sig4 with 50% infected files

Iteration number	Mean fitness	Best fitness
1	383.23534	456.00
2	383.44178	456.00
3	383.44178	456.00
4	383.44178	456.00
5	383.44178	456.00
...
50	383.44178	456.00
51	383.47316	456.00
...
98	383.47316	456.00
99	383.47316	456.00
100	383.47316	456.00
101	383.47316	456.00

As a result, the number of infected files= 288 and the Mean fitness = 383.4732.

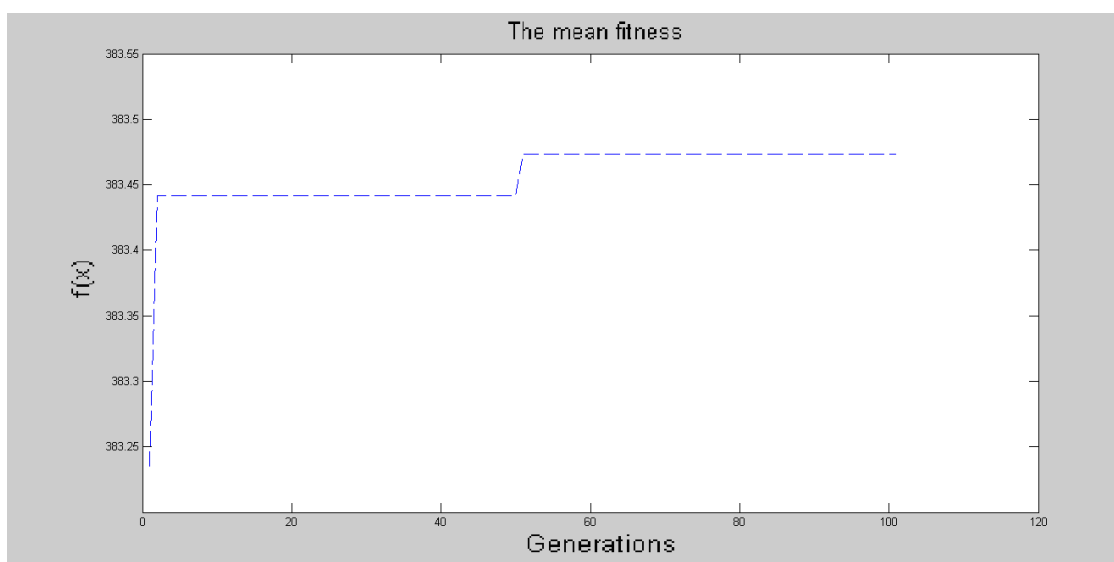


Figure 55: The Mean fitness of matching Sig4 run with 50% infected files

Table (28) and Figure (55) show that the Mean fitness increases by 0.20644 in the first iteration whereas it increases at the iteration number 50 by 0.03138. The Best fitness does not change; this explains the straight line in Figure (56). The number of detected files is 288 out of 300 (250+50) with a Detection rate of 96%, where the 250 infected files are in the original pool, and the 50 are from the new added pool. As a result this detection rate is accepted.

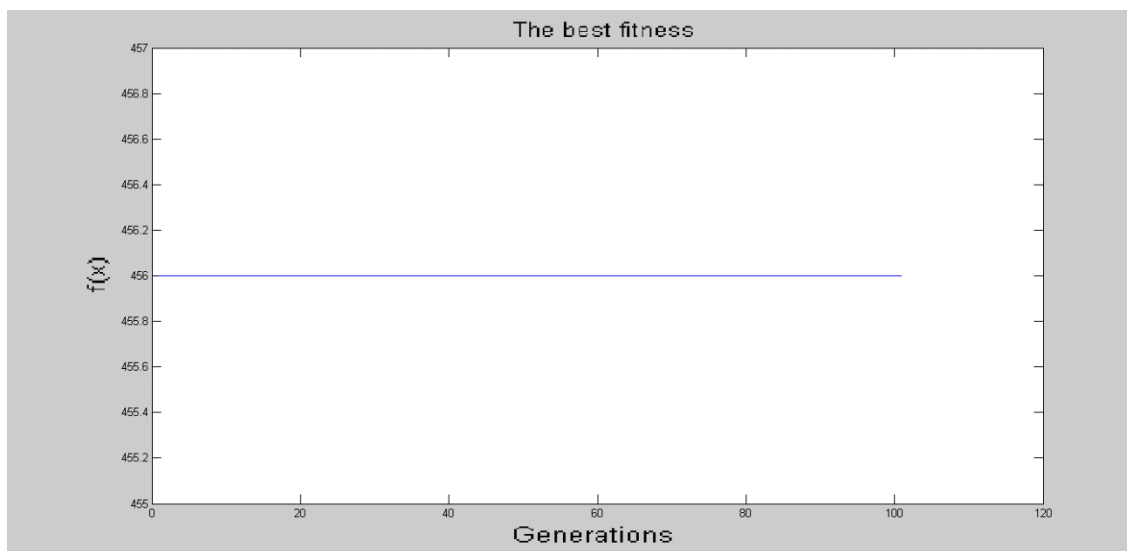


Figure 56: The Best fitness of matching Sig4 run with 50% infected files

The matching results of **Sig5** and **Sig8** with the files pool with 50% infected files are recapitulated in Table (25). At iteration number 50 new 100 files are added to the files' pool, with 50% of these files are infected.

Table 25: The summary of the matching of Sig5 and Sig8 with 50% infected files

Signatures pool	Matching Pool	Detected Files	Mean fitness	Best fitness	Detection Rate
Sig5	50%	288	485.4011	552.00	96%
Sig8	50%	288	1167.3765	1283.00	96%

In both matching runs the Mean fitness increases at two places; in the first iteration and iteration number 50. The Best fitness does not change. **Sig5** is presented in Appendix (B4).

The matching of **Sig1** with the files pool with 75% infected files. At iteration number 5 new 100 files are added to the files' pool, with 75% of these files are infected. The results are shown in Table (26).

Table 26: The results of the matching of Sig1 with 75% infected files

Iteration number	Mean fitness	Best fitness
1	226.66088	253.00
2	226.96526	284.00
3	226.96526	284.00
4	226.96526	284.00
5	226.96526	284.00
6	226.99669	290.00
...
97	226.99669	290.00
98	226.99669	290.00
99	226.99669	290.00
100	226.99669	290.00
101	226.99669	290.00

As a result, the number of infected files= 406 and the Mean fitness = 226.9967.

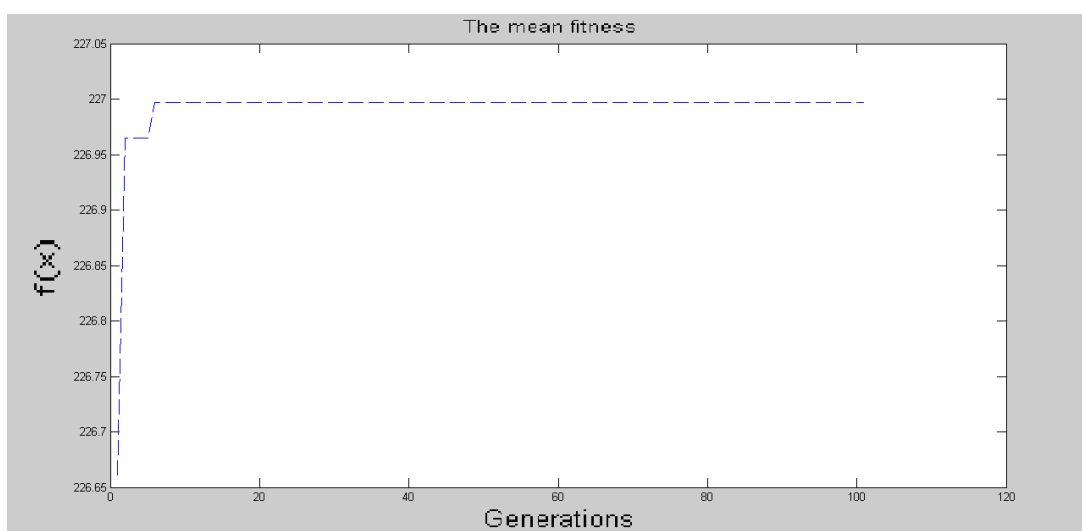


Figure 57: The Mean fitness of matching Sig1 run with 75% infected files

Table (26) and Figure (57) show that the Mean fitness increases by 0.30438 in the first iteration whereas it increases at the iteration number 5 by 0.03143.

The Best fitness increases at the first iteration by 31 whereas it increases at iteration number 5 by 6 as illustrated in Figure (58). The number of detected files is 406 out of 450 (375+75) with a Detection rate of 90.2%, where the 375 infected files are in the original pool, and the 75 are from the new added pool. As a result this detection rate is accepted.

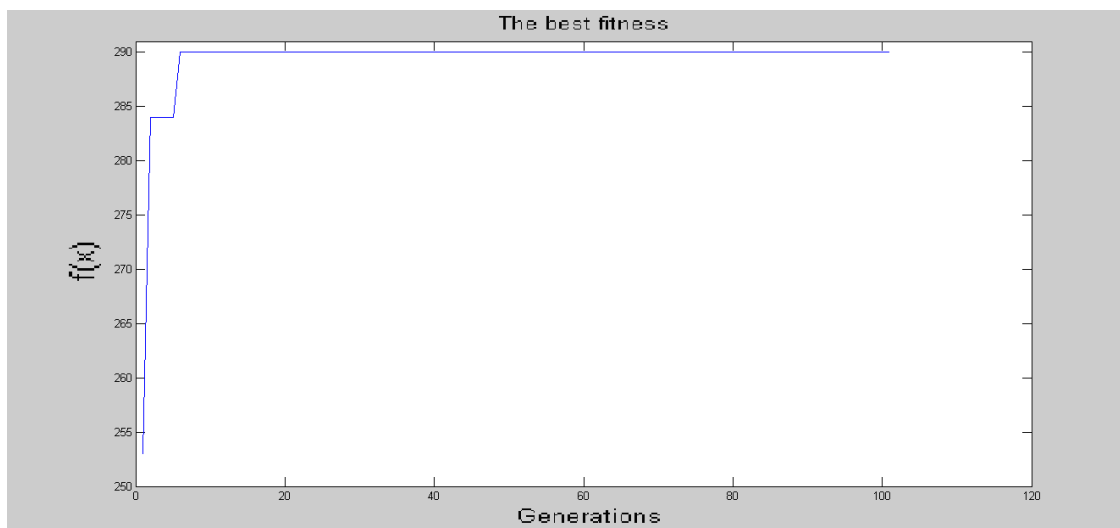


Figure 58: The Best fitness of matching Sig1 run with 75% infected files

The matching results of **Sig3**, **Sig6**, **Sig9** and **Sig12** with the files pool with 75% infected files are recapitulated in Table (27). At iteration number 5 new 100 files are added to the files' pool, with 75% of these files are infected.

Table 27: The summary of the matching of Sig3, Sig6, Sig9 and Sig12 with 75% infected files

Signatures pool	Matching Pool	Detected Files	Mean fitness	Best fitness	Detection Rate
Sig3	75%	406	253.7394	293.00	90.2%
Sig6	75%	406	585.6786	657.00	90.2%
Sig9	75%	406	1167.5172	1261.00	90.2%
Sig12	75%	406	225.6177	290.00	90.2%

In all the above mentioned matching runs the Mean fitness increases at two places; in the first iteration and iteration number 5. The Best fitness does not change except for Sig12, which is illustrated in Appendix (B5).

The matching of **Sig2** with the files pool with 100% infected files. At iteration number 50 new 100 files are added to the files' pool, with 100% of these files are infected. The results are shown in Table (28).

Table 28: The results of the matching of Sig2 with 100% infected files

Iteration number	Mean fitness	Best fitness
1	223.59950	251.00
2	224.00908	305.00
3	224.00908	305.00
4	224.00908	305.00
5	224.00908	305.00
...
50	224.00908	305.00
51	224.05698	312.00
...
97	224.05698	312.00
98	224.05698	312.00
99	224.05698	312.00
100	224.05698	312.00
101	224.05698	312.00

As a result, the number of infected files = 554 and the Mean fitness = 224.0570.

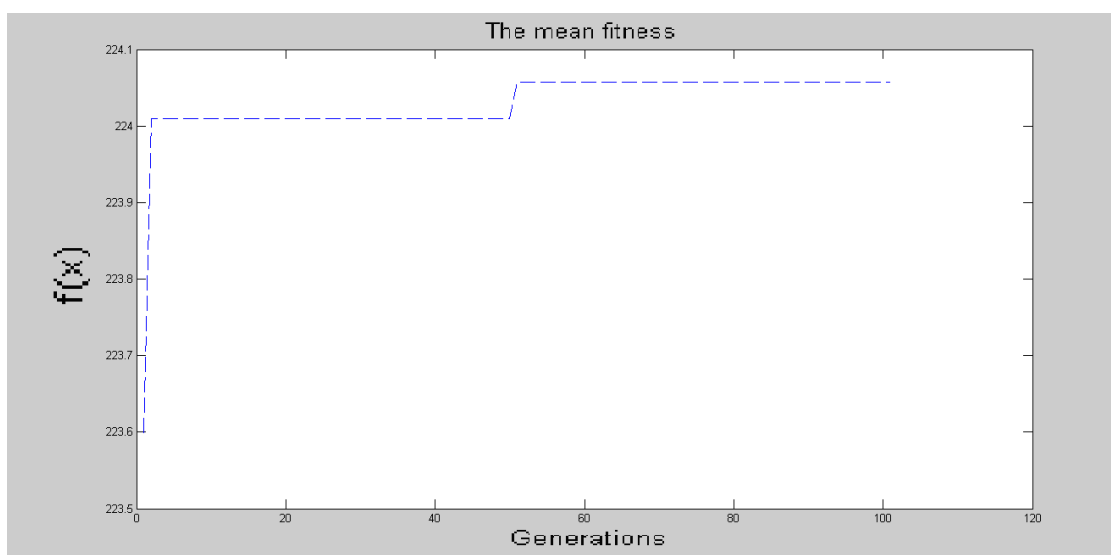


Figure 59: The Mean fitness of matching Sig2 run with 100% infected files

Table (28) and Figure (59) show that the Mean fitness increases by 0.40958 in the first iteration whereas it increases at the iteration number 50 by 0.04790.

The Best fitness increases at the first iteration by 54 whereas it increases at iteration number 50 by 7 as illustrated in Figure (60). The number of detected files is 554 out of 600 (500+100) with a Detection rate of 92.3%, where the 500 infected files are in the original pool, and the 100 are from the new added pool. As a result this detection rate is accepted.

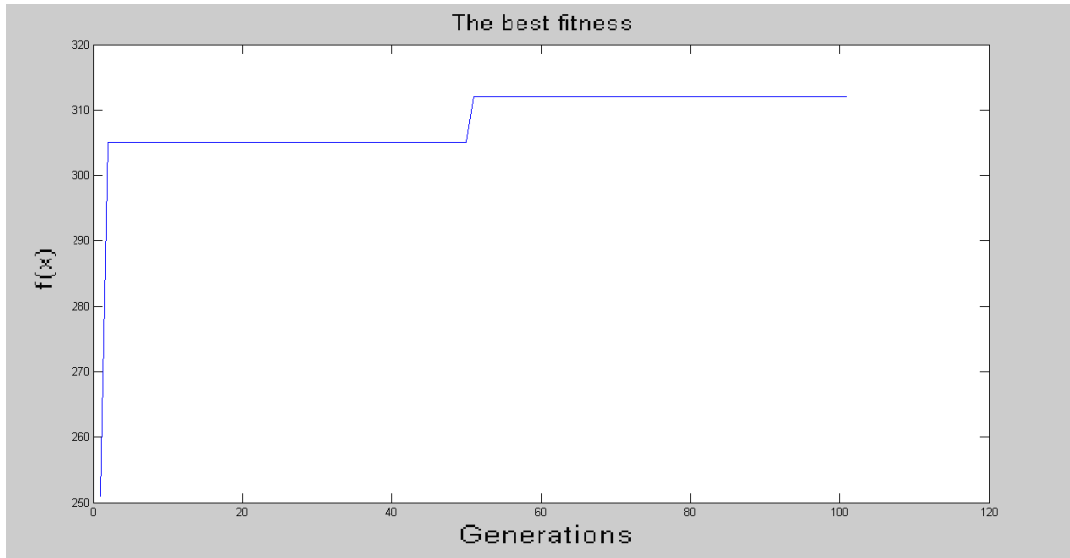


Figure 60: The Best fitness of matching Sig2 run with 100% infected files

The matching results of **Sig4**, **Sig7**, **Sig10**, **Sig11** and **Sig12** with the files pool with 100% infected files are summarized in Table (29). At iteration number 50 new 100 files are added to the files' pool, with 100% of these files infected.

Table 29: The summary of the matching of Sig4, Sig7, Sig10, Sig11 and Sig12 with 100% infected files

Signatures pool	Matching Pool	Detected Files	Mean fitness	Best fitness	Detection Rate
Sig4	100%	554	383.6928	456.00	92.3%
Sig7	100%	554	925.0413	1042.00	92.3%
Sig10	100%	554	557.3427	641.00	92.3%
Sig11	100%	554	786.9967	889.00	92.3%
Sig12	100%	554	225.7399	312.00	92.3%

In all the above mentioned matching runs the Mean fitness increases at two places; in the first iteration and iteration number 50. The Best fitness does not change except for Sig12, which is viewed in Appendix (B6).

4.2.1 The Matching Phase Analysis

The results of the runs of the matching phase, which includes 24 runs, are summarized in Tables (30) and (31).

Table (30) explains the Δ Best fitness (the change in the Best fitness), the Mean fitness and the Δ Mean fitness (the change in the Mean fitness).

- Δ Best fitness: in some cases of the matching the Best fitness changes, while in other cases it does not. The change in the Best fitness depends on the repetition of the signatures that have the higher fitness in the initial signatures' pool, where this repetition leads to increasing the value of the Best fitness. It is noticed in Table (30), that the change happens in 7 cases. The cases are Sig1, Sig2 and Sig12 with the matching pool with 25% infected files, Sig1 and Sig12 with the matching pool with 75% infected files, and Sig2 and Sig12 with the matching pool with 100% infected files.

The reason for the change in the Best fitness is that in the training phase for Sig1, Sig2 and Sig12, the training pool equals 5% and the *learning Gen* equals 100.

The meaning of having the training pool with 5% of infected files is that there are 25 infected files out of 500 files. In the detection process in the training phase, only 17 files are detected out of 25 files, because in the training phase, the VDC algorithm takes half of the signatures with the higher fitness to operate on them. (Based on the diligence of the researcher, the half of the signatures has been chosen to speed up the algorithm, and because matching is doing the actual detection of the viruses). The detection of the 17 files has caused the increase by δ on the fitness for only these 17 files, which have signatures with the higher fitness. The other reason for having an increase in the fitness is the Hypermutation process. In each generation the VDC algorithm chooses 11 mutated signatures with the higher fitness (after the Hypermutation process) and adds them to the signatures pool of the following generation. So when the *learning Gen* increases, the number of new signatures that have the higher fitness increases, which increases the Best fitness of the signature pool, and because the *learning Gen* of Sig1, Sig2 and Sig12 is 100, so the increase of the Best fitness is less than for Sig3 that has the *learning Gen* of 300. And though the training pool of Sig3 is 5%, the Best fitness does not change for Sig3 because its *learning Gen* =300, which leads to lifting up the ceiling of the Best fitness in the training phase, so when matching is done of this high ceil, it does not show any change in the Best fitness.

As for the rest of the cases of the matching runs ($24 - 7 = 17$ where 3 cases out of the 17 have the matching pool with 0% of infected files, to end up with 14 cases), the training pool of these 14 runs is either with 25% or 75% of infected files. When the training pool = 25%, the infected files are 125 files out of 500 files, the detection has been done on 103 files, which means adding δ to the fitness of 103 signatures (which are the signatures with the higher fitness within the half), so the increase in the Best fitness happens in the training phase, and when the matching is processed the Best fitness does not change. And when the training pool= 75%, the infected files are 375 files out of 500 files. The detection has been done on the average of 304 of the files, which are a large number of signatures that δ is added to them, and that leads to the increase in the Best fitness in the training phase, and when the matching is processed the Best fitness does not change.

Table 30: The matching results

Signatures Pool	Learning Gen	Pm	Fat	Training Files' pool	Matching Pool	Δ Best fitness	Mean fitness	Δ Mean fitness
Sig6	150	0.2	0.05	25%	5%	0	585.4639	0.01590
Sig7	100	0.2	0.05	75%	5%	0	924.6069	0.02312
Sig10	100	0.2	0.05	25%	5%	0	556.9083	0.02312
Sig11	100	0.2	0.10	75%	5%	0	786.5623	0.02313
Sig1	100	0.05	0.05	5%	25%	11	226.7783	0.11745
Sig2	100	0.1	0.05	5%	25%	11	223.7168	0.11726
Sig12	100	0.05	0.1	5%	25%	12	225.3997	0.11726
Sig4	100	0.05	0.05	25%	50%	0	383.4732	0.23782
Sig5	300	0.1	0.1	25%	50%	0	485.4011	0.08444
Sig8	150	0.1	0.1	75%	50%	0	1167.4	0.16354
Sig1	100	0.05	0.05	5%	75%	37	226.9967	0.33581
Sig3	300	0.05	0.1	5%	75%	0	253.7394	0.11902
Sig6	150	0.2	0.05	25%	75%	0	585.6786	0.23055
Sig9	300	0.05	0.1	75%	75%	0	1167.5	0.11903
Sig12	100	0.05	0.1	5%	75%	38	225.6177	0.33526
Sig2	100	0.1	0.05	5%	100%	61	224.057	0.45748
Sig4	100	0.05	0.05	25%	100%	0	383.6928	0.45748
Sig7	100	0.2	0.05	75%	100%	0	925.0413	0.45747
Sig10	100	0.2	0.05	25%	100%	0	557.3427	0.45747
Sig11	100	0.2	0.1	75%	100%	0	786.9967	0.45748
Sig12	100	0.05	0.1	5%	100%	60	225.7399	0.45747

- The Mean fitness and the Δ Mean fitness: Table (30) shows the change in the Mean fitness in all the cases of the matching. There are 5 variables that affect the Mean fitness or the Δ Mean fitness: matching pool, Pm , Fat , $Learning$ Gen and training pool.

1. Matching pool: When the number of infected files inside the matching pool increases, the Mean Fitness for that pool increases with a small value. For sig6, when the matching pool = 5%, the Mean Fitness = 585.4634, and when the matching pool = 75%, the Mean Fitness = 585.6786. The deviation is 0.2152. Another example is sig10, when the matching pool = 5%, the Mean Fitness = 556.9083, and when the matching pool = 100%, the Mean Fitness = 557.342. The deviation is 0.4344. It is noticed that both deviations are in small values. It is noticed that it is the main operative that affects the change in the Mean fitness, as when infected files increase inside the matching pool, the Δ Mean fitness is larger. For example: when the matching pool=5%, the Δ Mean fitness is between 0.01590 and 0.02313, while when the matching pool=100%, the Δ Mean fitness is between 0.45797 and 0.45748. As shown in the whole table the Δ Mean fitness ranges between 0.01590 as a lower value, when the matching pool =5%, and 0.45748 as a higher value when the matching pool =100%.

The reason for having the matching pool operative as the main effect on the Δ Mean fitness is that the detection of the infected files increases the fitness by δ for each infected file, and hence increases the Mean fitness.

2. Pm : when all the variables are fixed, and Pm is changed as in the following 2 cases:

1st: when the matching pool=25%, for Sig1 when $Pm=0.05$ and Sig2 when $Pm = 0.1$ the deviation between the Mean fitness of Sig1 and Sig2 equals 3.0165 in favor of Sig1, and the deviation between the Δ Mean fitness of Sig1 and the Δ Mean fitness of Sig2 equals 0.00019, where in Sig1 with the Pm of 0.05, the Δ Mean fitness is higher.

Signatures pool	Learning Gen	Pm	Fat	Training Files' pool	Matching Pool	Δ Best fitness	Mean fitness	Δ Mean fitness
Sig1	100	0.05	0.05	5%	25%	11	226.7783	0.11745
Sig2	100	0.1	0.05	5%	25%	11	223.7168	0.11726

2nd: when the matching pool=100%, for Sig4, when $Pm=0.05$, and Sig10 when $Pm = 0.2$, the deviation between the Mean fitness of Sig4 and Sig10 equals 173.7899 in favor of Sig10, and the deviation between the ΔMean fitness of Sig4 and the ΔMean fitness of Sig10 equals 0.00001, where in Sig4 with the Pm of 0.05, the ΔMean fitness is higher.

Signatures pool	Learning Gen	Pm	Fat	Training Files' pool	Matching Pool	Δ Best fitness	Mean fitness	Δ Mean fitness
Sig4	100	0.05	0.05	25%	100%	0	383.6928	0.45748
Sig10	100	0.2	0.05	25%	100%	0	557.3427	0.45747

3. *Fat*: when all the variables are fixed, and *Fat* is changed as in the following 3 cases:

1st: when the matching pool=5%, with the *Fat* of Sig7 =0.05, and the *Fat* of Sig11 = 0.1, the deviation between the Mean fitness of Sig7 and Sig11 equals 138.0446 in favor of Sig7, and the deviation between the ΔMean fitness of Sig7 and the ΔMean fitness of Sig11 equals 0.00001, where in Sig11 with the *Fat* = 0.1, the ΔMean fitness is higher.

Signatures pool	Learning Gen	Pm	Fat	Training Files' pool	Matching Pool	Δ Best fitness	Mean fitness	Δ Mean fitness
Sig7	100	0.2	0.05	75%	5%	0	924.6069	0.02312
Sig11	100	0.2	0.10	75%	5%	0	786.5623	0.02313

2nd: when the matching pool=75%, with the *Fat* of Sig1 =0.05, and the *Fat* of Sig12 = 0.1, the deviation between the Mean fitness of Sig1 and Sig12 equals 1.379 in favor of Sig1, and the deviation between the ΔMean fitness of Sig1 and the ΔMean fitness of Sig12 equals 0.00055, where in Sig1 with the *Fat* = 0.05, the ΔMean fitness is higher.

Signatures pool	Learning Gen	Pm	Fat	Training Files' pool	Matching Pool	Δ Best fitness	Mean fitness	Δ Mean fitness
Sig1	100	0.05	0.05	5%	75%	37	226.9967	0.33581
Sig12	100	0.05	0.1	5%	75%	38	225.6177	0.33526

3rd: when the matching pool=100%, with the *Fat* of Sig7 =0.05, and the *Fat* of Sig11 = 0.1, the deviation between the Mean fitness of Sig7 and Sig11 equals 138.0446 in favor of Sig7, and the deviation between the ΔMean fitness of Sig7 and the ΔMean fitness of Sig11 equals 0.00001, where in Sig11 with the *Fat* = 0.1, the ΔMean fitness is higher.

Signatures pool	Learning Gen	Pm	Fat	Training Files' pool	Matching Pool	Δ Best fitness	Mean fitness	Δ Mean fitness
Sig7	100	0.2	0.05	75%	100%	0	925.0413	0.45747
Sig11	100	0.2	0.1	75%	100%	0	786.9967	0.45748

* Note: the effect of these 2 variables (*Pm* and *Fat*) is discussed later after processing the GA.

4. Training pool: when all variables are fixed, and the change is in the training pool as in the following 3 cases:

1st: when the matching pool =5%, with the training pool of Sig7=75% and the training pool of Sig10=25%, the deviation between the Mean fitness of Sig7 and Sig10 equals 367.6986 in favor of Sig10, and the Δ Mean fitness does not change.

Signatures pool	Learning Gen	Pm	Fat	Training Files' pool	Matching Pool	Δ Best fitness	Mean fitness	Δ Mean fitness
Sig7	100	0.2	0.05	75%	5%	0	924.6069	0.02312
Sig10	100	0.2	0.05	25%	5%	0	556.9083	0.02312

2nd: when the matching pool =75%, with the training pool of Sig3=5% and the training pool of Sig9=75%, the deviation between the Mean fitness of Sig3 and Sig9 equals 913.7608 in favor of Sig9, and the deviation between the ΔMean fitness of Sig3 and the ΔMean fitness of Sig9=0.00001, where in Sig9 with the training pool of 75%, the ΔMean fitness is higher.

Signatures pool	Learning Gen	Pm	Fat	Training Files' pool	Matching Pool	Δ Best fitness	Mean fitness	Δ Mean fitness
Sig3	300	0.05	0.1	5%	75%	0	253.7394	0.11902
Sig9	300	0.05	0.1	75%	75%	0	1167.5	0.11903

3rd: when the matching pool =100%, with the training pool of Sig7=75% and the training pool of Sig10=25%, the deviation between the Mean fitness of Sig7 and Sig10 equals 367.6986 in favor of Sig7, and the ΔMean fitness does not change.

Signatures pool	Learning Gen	Pm	Fat	Training Files' pool	Matching Pool	Δ Best fitness	Mean fitness	Δ Mean fitness
Sig7	100	0.2	0.05	75%	100%	0	925.0413	0.45747
Sig10	100	0.2	0.05	25%	100%	0	557.3427	0.45747

During the training phase process, the training pool affects the fitness by increasing the Mean fitness, when the training phase produces the signatures' pools that are used in the matching phase. During the matching phase, the effect of the training pool does not appear on the ΔMean fitness, because it is already appearing in the first phase (training), but it affects the Mean fitness.

5. *Learning Gen*: when all variables are fixed, and the change is in the *learning Gen* as in the following 2 cases:

1st: when the matching pool =5%, with the *learning Gen* of Sig6=150, and the *learning Gen* of Sig10=100, the deviation between the Mean fitness of Sig6 and Sig10 equals 28.5556 in favor of Sig6, and the deviation between the ΔMean fitness of Sig6 and the ΔMean fitness of Sig10=0.00722, where in Sig10 with the *learning Gen* of 100, the ΔMean fitness is higher.

Signatures pool	Learning Gen	Pm	Fat	Training Files' pool	Matching Pool	Δ Best fitness	Mean fitness	Δ Mean fitness
Sig6	150	0.2	0.05	25%	5%	0	585.4639	0.01590
Sig10	100	0.2	0.05	25%	5%	0	556.9083	0.02312

2nd: when the matching pool =75%, with the *learning Gen* of Sig3=300, and the *learning Gen* of Sig12=100, the deviation between the Mean fitness of Sig3 and Sig12 equals 28.1217 in favor of Sig3, and the deviation between the ΔMean fitness of Sig3 and the ΔMean fitness of Sig12=0.21624, where in Sig12 with the *learning Gen* of 100, the ΔMean fitness is higher.

Signatures pool	Learning Gen	Pm	Fat	Training Files' pool	Matching Pool	Δ Best fitness	Mean fitness	Δ Mean fitness
Sig3	300	0.05	0.1	5%	75%	0	253.7394	0.11902
Sig12	100	0.05	0.1	5%	75%	38	225.6177	0.33526

It is noticed that when the *learning Gen* value is higher, the Mean fitness increases by a small value, and when the *learning Gen* value is less, the ΔMean fitness is higher significantly. The reason for that is when the *learning Gen* is high, the effect of Hypermutation is higher than the detection, which decreases the deviation in the fitness rate. Although the detection increases the fitness by δ ($=10$) each time, however, when the number of Hypermutation is large, it increases the fitness by 1 so many times, so the increase of the Hypermutation is larger.

One should keep in mind that even the Hypermutation process exists only in the training phase, and the main influence on the fitness in the matching phase is the detection, though Hypermutation affects the production of signatures' pools, which are used in the matching phase.

Table (31) shows the detection rate on the 24 matching runs, where in the case of 0% of infected files (all files are benign), the detection rate is 100% as it has detected zero number of infected files, and this is the false positive testing which is considered as a good result.

Table 31: The detection rate of the matching results

Matching pool	Signatures' pools	Detection rate
0%	Sig1, Sig5, Sig8	100%
5%	Sig6, Sig7, Sig10, Sig11	93.3%
25%	Sig1, Sig2, Sig12	94.7%
50%	Sig4, Sig5, Sig8	96%
75%	Sig1, Sig3, Sig6, Sig9, Sig12	90.2%
100%	Sig2, Sig4, Sig7, Sig10, Sig11, Sig12	92.3%
The Average of detection rate		94.4%

As for the detection rate of viruses for the rest of the files' pools (5%, 25%, 50%, 75% and 100% of infected files), it ranges between 90.2% and 94.7% with the average for the detection rate on all cases is 94.4%, which is considered as good from the researcher point of view.

The procedures that have been applied through the training and matching phases show that the use of the VDC algorithm is good in detecting the computer viruses, and hence this answers the first question of the research questions.

Chapter Five

The Optimization of the VDC Algorithm using the GA

This chapter includes the process of using the GA as an optimizer for the VDC algorithm. The results after optimizing the VDC algorithm are compared to the results of the standard VDC algorithm viewed at the previous chapter.

This part answers the second and third questions of the research questions, and they are: **"Are the AIS (VDC) and GA applicable for solving the problem of computer viruses detection?"** and **"Will the tuning process by GA improve the AIS (VDC) algorithm accuracy, or not?"**

Chapter five is divided into two sections: the optimized VDC algorithm based on GA, and the comparison between the results of the VDC algorithm, and the results of the optimized VDC algorithm based on GA.

5.1. The Optimized VDC Algorithm based on GA

The employed GA in this research is the genetic algorithm toolbox under MATLAB, where the VDC algorithm is called as the Fitness function for the GA. The VDC algorithm is pre-appended with the minus sign to maximize the problem. The inputs are the *Pm* and *Fat*, and the output is the Mean fitness.

The purpose of applying the GA is to find the best values of the *Pm* and *Fat*, to tune these values in order to get better optimized algorithm.

In this section, there are three processes:

1. GA optimization: the process to find the values of the *Pm* and *Fat* by using the GA.
2. GA training: to employ the values resulted from the first process to build the signatures' pools.
3. GA matching: to do testing for the signatures' pools, which resulted from the previous process.

5.1.1 GA Optimization

This process aims to find the values of Pm and Fat after executing the GA with the VDC algorithm; in order to compare the results of using these values with the results of chapter four.

There are two kinds of parameters for the GA; general parameters for the toolbox, and specific parameters. The specific parameters are the Pm and the Fat .

The general parameters, which are five, consist of the number of variables, the lower bound, the upper bound, the initial population type, and the GA generations as a stopping criterion.

The number of variables equals 2. These 2 are the specific parameters Pm and Fat .

The lower bound = [0.01, 0.02] is the lowest value for each of the two specific parameters, where the lower value for the Pm is 0.01, which represents the lowest rate of the Hypermutation probability. The lowest value for the Fat is 0.02, and this represents the number of elements per clone to be at least 2.

The upper bound = [1, 1] for the two specific parameters is 1, and it represents the higher value for both of them. The initial population type is double vector, which means real values. The GA generations ($GA\ Gen$) which is the number of times the GA is executed, which equals 10 for each run, and the total number of runs is four.

There are as well 3 specific changes on the VDC algorithm, to be able to use it with the GA. First on the number of elements to be cloned (Cloning), the second is the Files' pool used with the GA, and the third is the number of generations for the VDC algorithm, when it is called by the GA.

The number of elements to be cloned is the half size of the signatures' pool in the original algorithm, according to the equation 3.3. This number is changed to a fixed value equaling 50, to speed up the algorithm.

The files' pools that are used when performing the GA have the values 5%, 25%, 50% and 75% as illustrated in Table (32) in the column GA pool. The number of generations that the VDC algorithm is executed on is illustrated in Table (32) in the column $GA\ VDC\ Gen$.

The values $GA\ Gen=10$, $Cloning=50$ and $GA\ VDC\ Gen = (10, 20, 30)$ are chosen to speed up extracting the results. The researcher has conducted several experiments, and when using higher values, it has needed longer time but without getting any results, keeping in mind that the used computers are micro-computers. The researcher is not able to get a computer with high CPU properties enough to conduct these experiments with higher values.

Table 32: The GA Optimization Runs Specifications

Run	GA VDC Gen	GA pool	GA Gen
1	10	5%	10
2	20	50%	10
3	30	25%	10
4	10	75%	10

The **first** GA optimization run parameters are: the $GA\ VDC\ Gen=10$, the $GA\ pool = 5\%$ and the $GA\ Gen = 10$. The resulted $Pm = 0.636$, and the resulted $Fat = 0.935$, as shown in Figure (61).

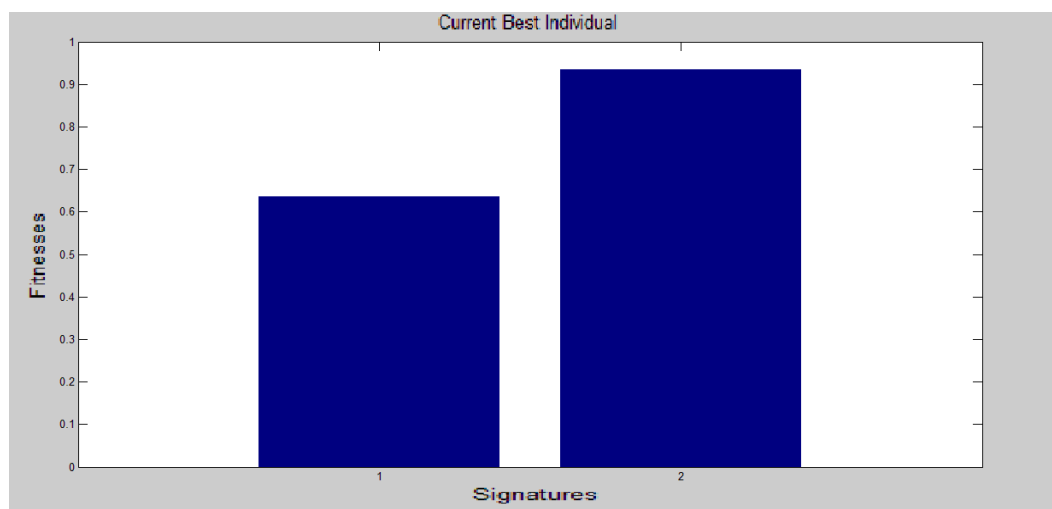


Figure 61: The Current Best Individual in the first GA optimization run

Figure (62) presents the Best and Mean fitness for each GA generation. The Best fitness is fixed for all generations, while the Mean fitness has changed with the highest value of 380.2162. As has been mentioned previously the GA looks for the minimization by default, that is why the minus sign is added to make it search for the maximization. So considering that these values are multiplied by the minus sign, the highest value is actually the lowest value in the figure.

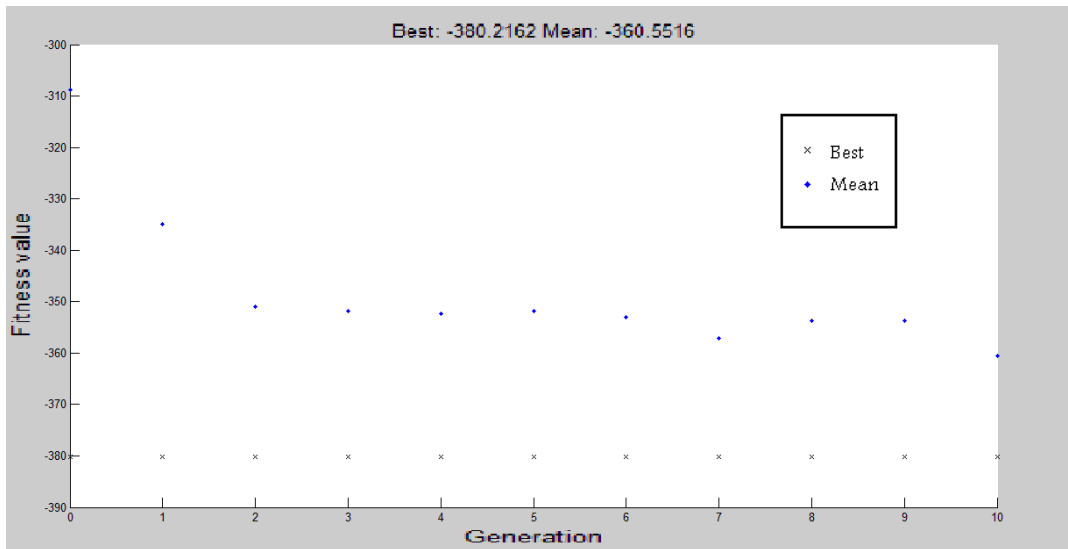


Figure 62: The Mean & Best Fitness in the first GA optimization run

The **second** GA optimization run parameters are: the *GA VDC Gen*=20, the *GA pool*= 50% and the *GA Gen* = 10. The resulted *Pm* = 0.96, and the resulted *Fat* = 1.0, as illustrated in Figure (63).

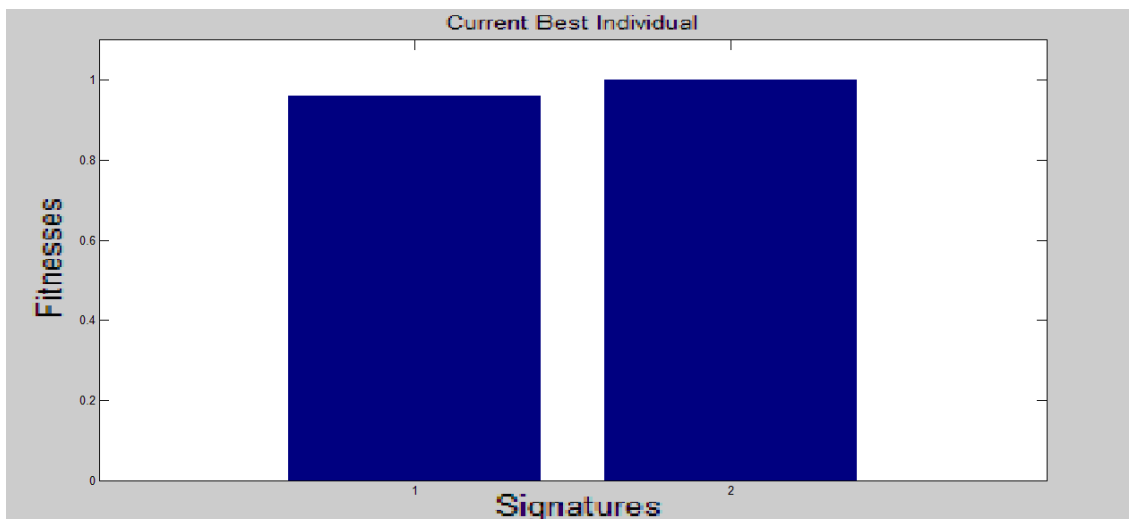


Figure 63: The Current Best Individual in the second GA optimization run

Figure (64) demonstrates the Best and Mean fitness for each GA generation. The Best fitness is fixed for all generations, while the Mean fitness has changed with the highest value of 2540.2172.

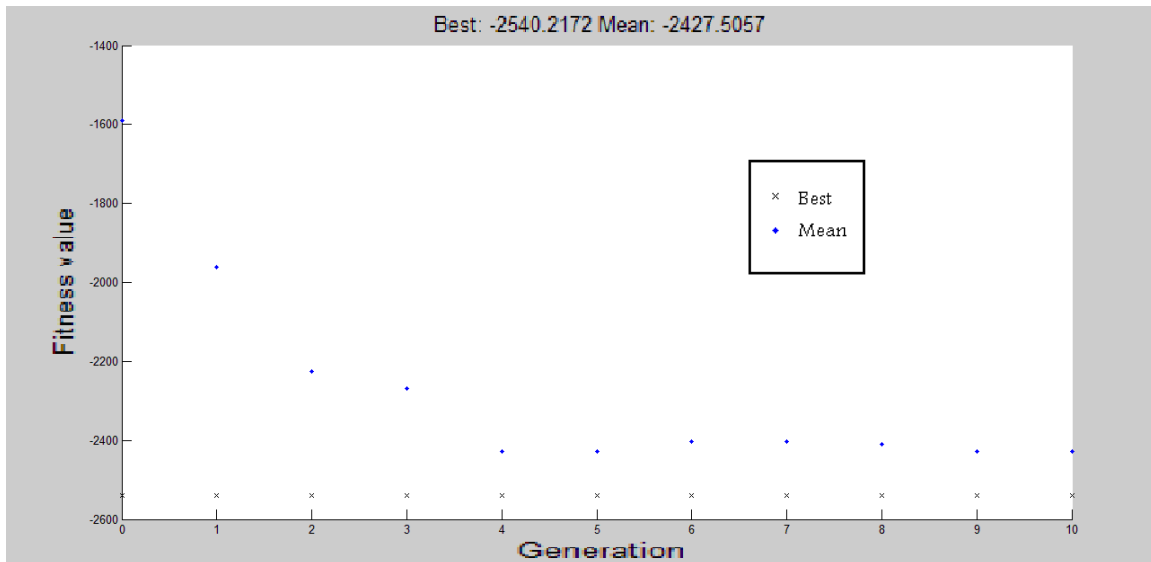


Figure 64: The Mean & Best Fitness in the second GA optimization run

The **third** GA optimization run parameters are: the *GA VDC Gen*=30, the *GA pool* = 25% and the *GA Gen* = 10. The resulted *Pm* = 0.65, and the resulted *Fat* = 0.96, as exhibited in Figure (65).

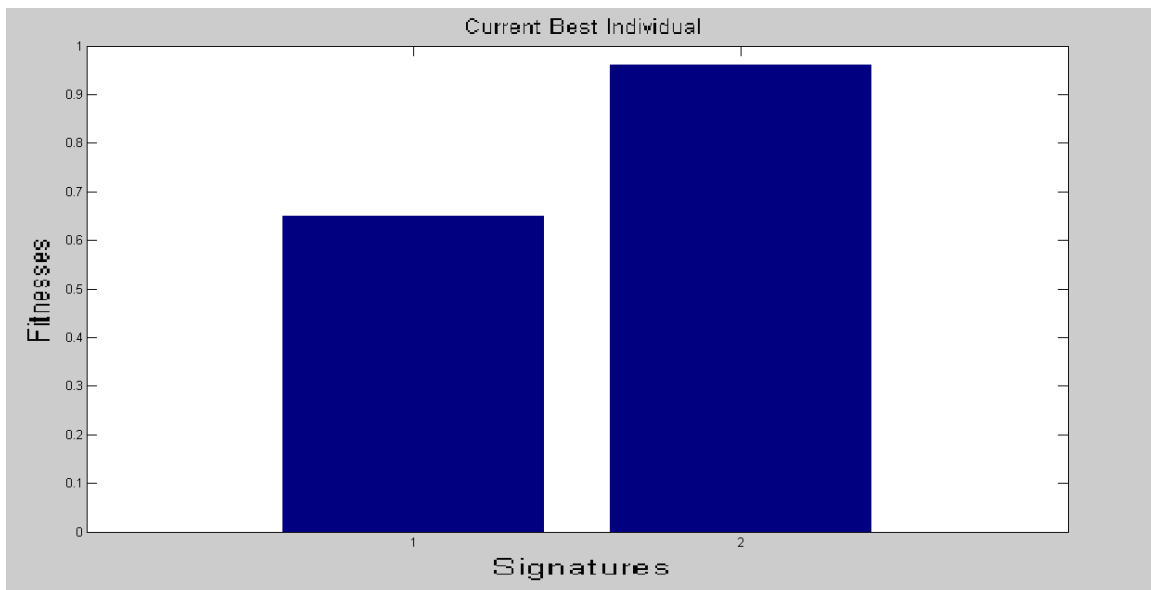


Figure 65: The Current Best Individual in the third GA optimization run

Figure (66) views the Best and Mean fitness for each GA generation. The Best fitness is fixed for all generations, while the Mean fitness has changed with the highest value of 1260.2085.

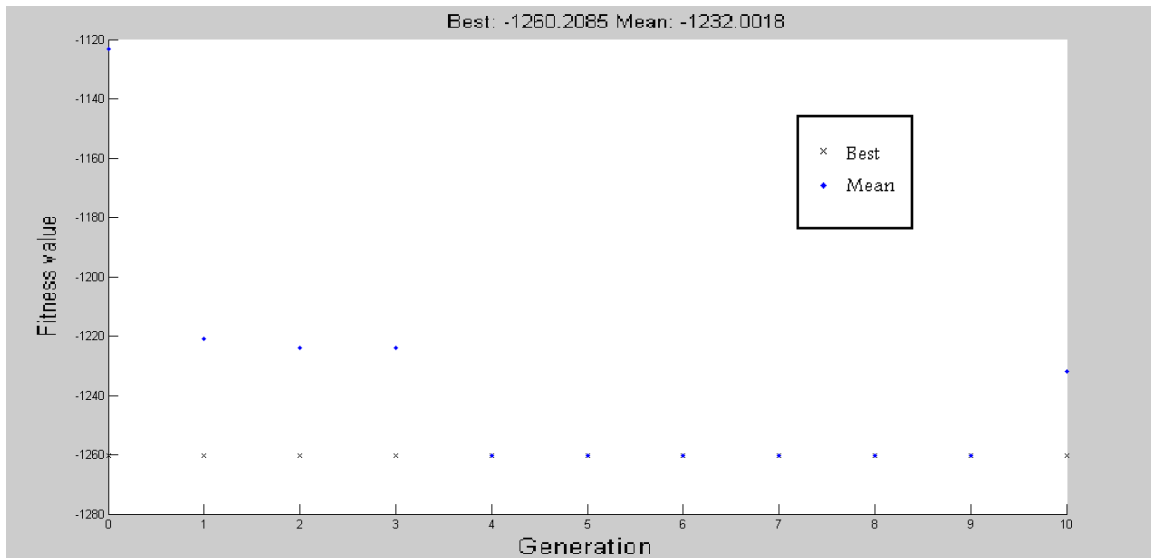


Figure 66: The Mean & Best Fitness in the third GA optimization run

The **forth** GA optimization run parameters are: the *GA VDC Gen*=10, the *GA pool* = 75% and the *GA Gen* = 10. The resulted *Pm* = 0.914, and the resulted *Fat* = 0.935, as shown in Figure (67).

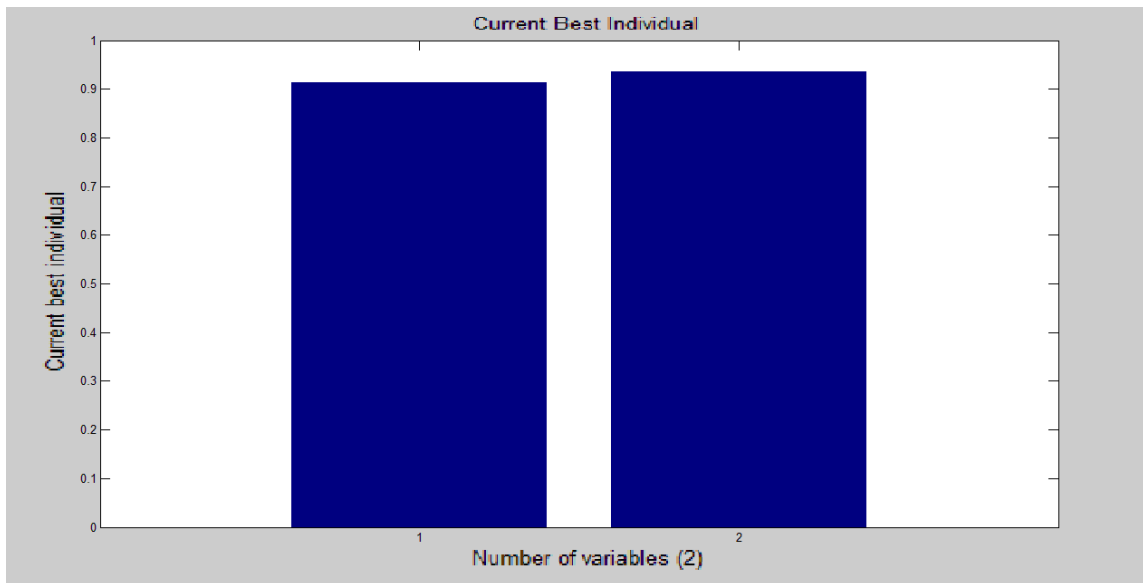


Figure 67: The Current Best Individual in the forth GA optimization run

Figure (68) presents the Best and Mean fitness for each GA generation. The Best fitness is fixed for all generations, while the Mean fitness has changed with the highest value of 3650.226.

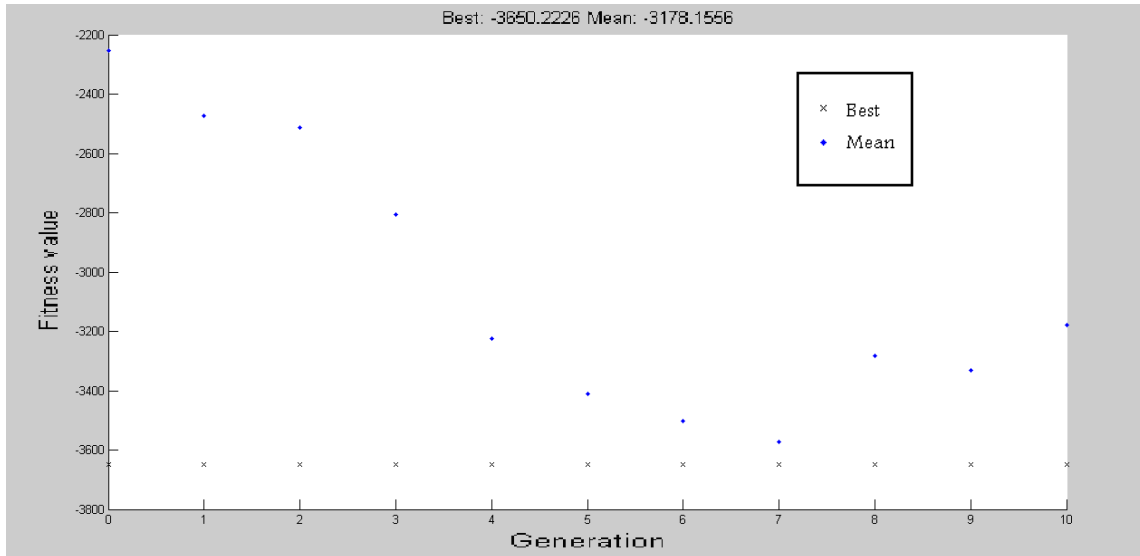


Figure 68: The Mean & Best Fitness in the fourth GA optimization run

After executing the 4 runs, *Pm* and *Fat* values have resulted, as illustrated in Table (33), and these values are used in the next process; the GA training.

Table 33: The GA optimization results

Run	<i>Pm</i>	<i>Fat</i>	GA objective function value
1	0.636	0.935	380.2162
2	0.96	1.0	2540.2172
3	0.65	0.96	1260.2085
4	0.914	0.935	3650.2226

5.1.2 GA Training

The GA training process includes the production of the signatures' pools, and uses the *Pm* and *Fat* which resulted in the GA optimization. For each GA optimization run, there is a GA training run, to end up having 4 training runs. The results of the first run are used to produce SigGA1, while the results of the second run are employed to create SigGA2, and the results of the third run are utilized to make SigGA3, whereas the results of the fourth run are exploited to construct SigGA4, as illustrated in Table (34).

Table 34: The GA training runs specifications

Run	GA signatures' pool	GA Learning Gen	GA Training pool	<i>Pm</i>	<i>Fat</i>	GA VDC Gen
1	SigGA1	100	5%	0.636	0.935	10
2	SigGA2	100	50%	0.96	1.0	20
3	SigGA3	100	25%	0.65	0.96	30
4	SigGA4	100	75%	0.914	0.935	10

The **first** GA training run parameters are: *GA Learning Gen*=100, *Pm*=0.636 and *Fat* = 0.935, and the files' pool is with 5% of infected files and the produced signatures' pool is saved as **SigGA1**. The results of the first run are shown in Table (35), where each iteration corresponds to the number of signatures in the signatures' pool. On the first iteration, the number of signatures value is 100 and, it is increased by 11 signatures or less in each iteration, to have at the last iteration 1200 signatures.

The selection threshold ranges between 0.6 and 1.0; this threshold determines the selection process stochastically.

The Mean Fitness increases at a rapid rate in the beginning, then the increase continues at a slower rate; at first it increases at a higher changing rate reaching 16.9, after that the rate increases to reach 14.57 in the second iteration, to become 35.89 in the third iteration, till it reaches around 0.74 in the last five iterations, as shown in Table (35).

While the Best Fitness increases quickly in the first iteration only, and later on the increase becomes slower; in the first iteration the changing rate is 163 for the Best Fitness, after that it increases by 1 in each iteration. Figure (69) represents the Mean Fitness and the Best Fitness.

Table 35: The first GA training run results

Iteration number	No of signatures	Selection threshold	Mean fitness	Best fitness
1	100	0.90	160.12	211.00
2	111	1.00	177.02	374.00
3	122	0.70	191.59	375.00
4	133	0.60	227.48	376.00
5	144	0.90	255.39	377.00
...
96	1145	0.80	441.50	468.00
97	1156	0.70	442.23	469.00
98	1167	0.80	443.01	470.00
99	1178	0.70	443.73	471.00
100	1189	1.00	444.51	472.00
101	1200	0.90	445.23	473.00

As a result, the number of detected infected files is 17 out of 25 (as it is the training phase, the researcher decides to take half the size of the signatures' pool as mentioned in section 3.1), and the Mean fitness = 445.2282.

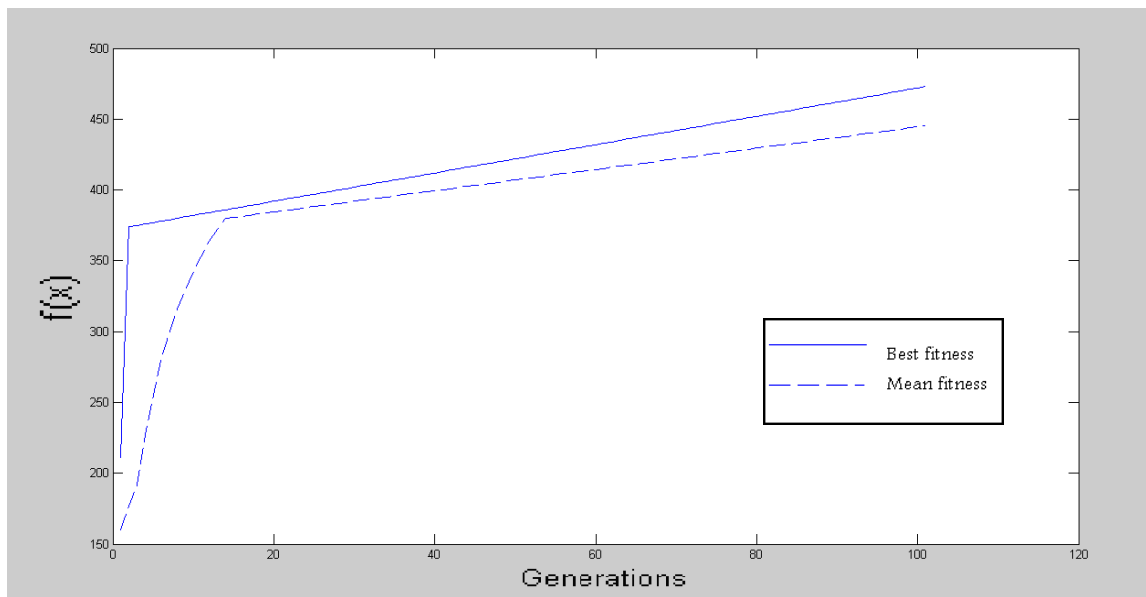


Figure 69: The first GA training run Mean Fitness & Best Fitness

The initial population is exhibited in Figure (70) in a solid line (The same initial population is demonstrated in Figure (24)), while the final population which includes the new signatures after Hypermutation is presented by dotted line. This figure shows that the fitness of the new mutated signatures is higher.

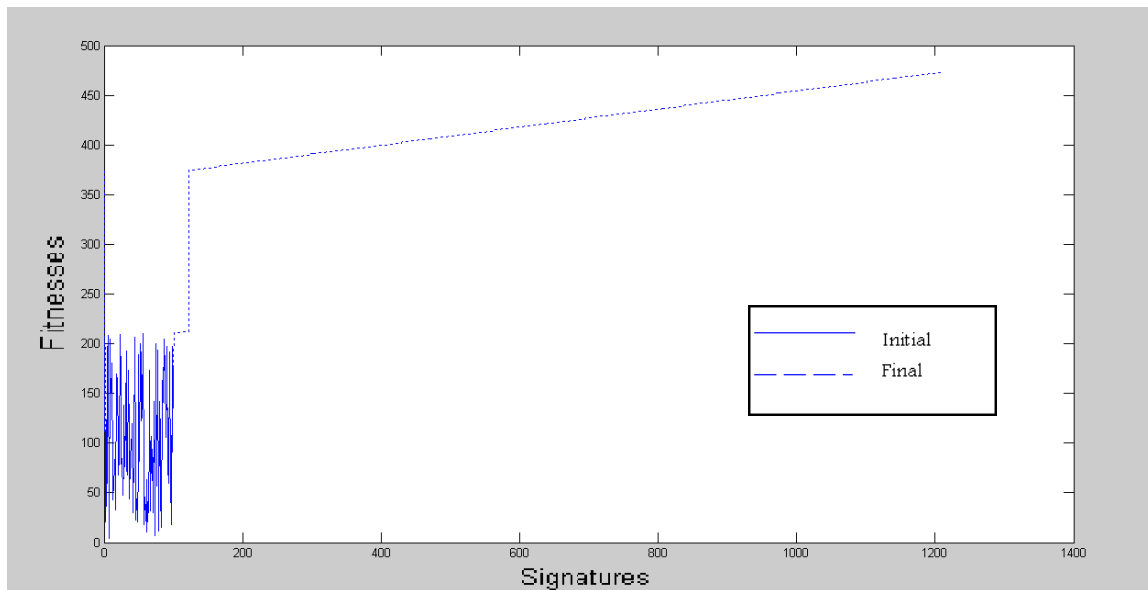


Figure 70: The first GA training run final population

The **second** GA training run parameters are: *GA Learning Gen*=100, *Pm*=0.96 and *Fat* = 1.0, and the files' pool with 50% of infected files and the produced signatures' pool is saved as **SigGA2**. The results of the second run are presented in Table (36), where each iteration corresponds to the number of signatures in the signatures' pool. On the first iteration, the number of signatures value is 100 and, it is increased by 11 signatures or less in each iteration, to have at the last iteration 1200 signatures.

The selection threshold ranges between 0.6 and 1.0; this threshold determines the selection process stochastically.

The Mean Fitness increases at a rapid rate in the beginning, then the increase continues at a slower rate; at first it increases at a higher changing rate reaching 17.33, after that the rate increases to reach 49.38 in the second iteration, to become 391.58 in the third iteration and so on, till it reaches around 0.74 in the last five iterations, as shown in Table (36).

While the Best Fitness increases quickly in the first iteration only, and later on the increase becomes slower; in the first iteration the changing rate is 2313 for the Best Fitness, after that it increases by 1 in each iteration. Figure (71) demonstrates the Mean Fitness and the Best Fitness.

Table 36: The second GA training run results

Iteration number	No of signatures	Selection threshold	Mean fitness	Best fitness
1	100	1.00	160.10	211.00
2	111	0.60	177.43	2524.00
3	122	1.00	226.81	2525.00
4	133	0.60	618.39	2526.00
5	144	0.80	942.18	2527.00
...		
96	1145	0.70	2591.51	2618.00
97	1156	0.80	2592.24	2619.00
98	1167	0.90	2593.00	2620.00
99	1178	0.80	2593.74	2621.00
100	1189	0.80	2594.51	2622.00
101	1200	0.80	2595.23	2623.00

As a result, the number of detected infected files is 232 out of 250, and the Mean fitness = 2595.2000.

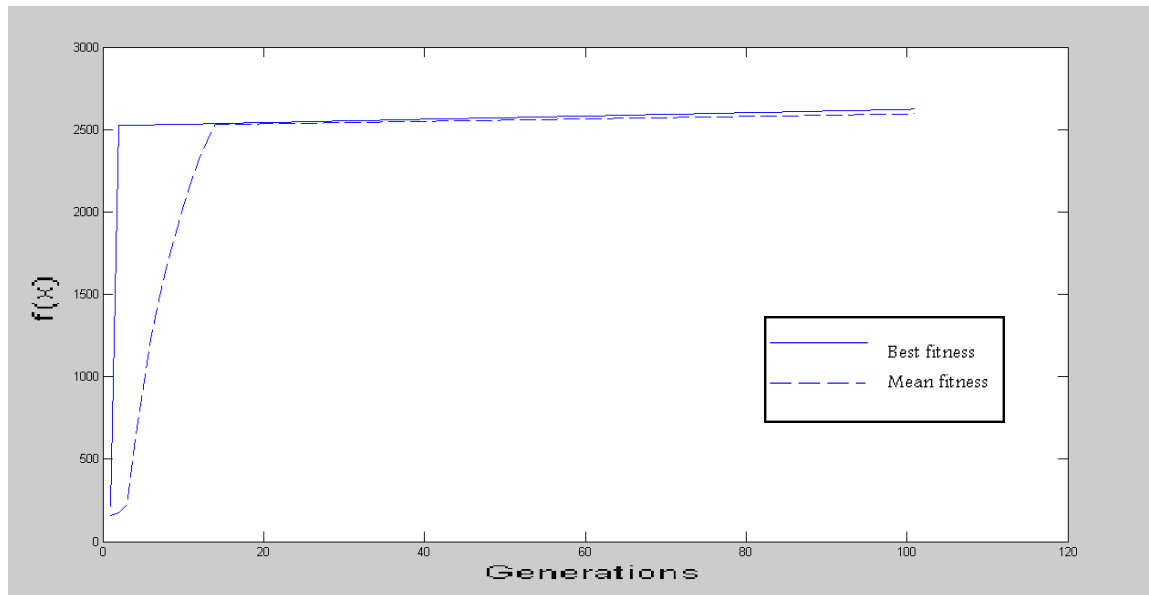


Figure 71: The second GA training run Mean Fitness & Best Fitness

The initial population is exhibited in Figure (72) in a solid line, while the final population which includes the new signatures after Hypermutation is presented by dotted line. This figure shows that the fitness of the new mutated signatures is higher.

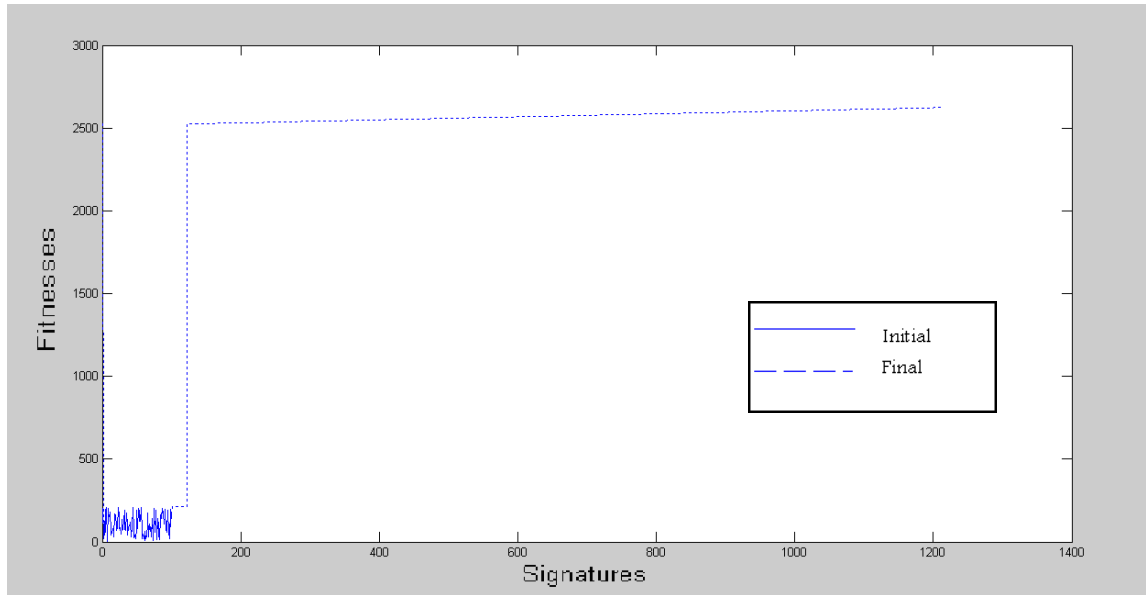


Figure 72: The second GA training run final population

The **third** GA training run parameters are: *GA Learning Gen*=100, *Pm*=0.65 and *Fat* = 0.96, and the files' pool is with 25% of infected files, and the produced signatures' pool is saved as **SigGA3**. The results of the third run are viewed in Table (37), where each iteration corresponds to the number of signatures in the signatures' pool. On the first iteration, the number of signatures value is 100 and, it is increased by 11 signatures or less in each iteration, to have the last iteration with 1200 signatures.

The selection threshold ranges between 0.6 and 1.0; this threshold determines the selection process stochastically.

The Mean Fitness increases at a rapid rate in the beginning, then the increase continues at a slower rate; at first it increases at a higher changing rate reaching 17.05, after that the rate increases to reach 26.88 in the second iteration, to become 161.62 in the third iteration, till it reaches around 0.74 in the last five iterations, as shown in Table (37).

While the Best Fitness increases quickly in the first iteration only, and later on the increase becomes slower; in the first iteration the changing rate is 923 for the Best Fitness, after that it increases by 1 in each iteration. Figure (73) illustrates the Mean Fitness and the Best Fitness.

Table 37: The third GA training run results

Iteration number	No of signatures	Selection threshold	Mean fitness	Best fitness
1	100	1.00	160.11	211.00
2	111	0.90	177.16	1134.00
3	122	0.60	204.04	1135.00
4	131	0.90	365.66	1136.00
5	142	0.60	498.16	1137.00
...
96	1145	0.60	1201.51	1228.00
97	1156	0.90	1202.23	1229.00
98	1167	0.80	1203.01	1230.00
99	1178	0.70	1203.73	1231.00
100	1189	1.00	1204.50	1232.00
101	1200	1.00	1205.23	1233.00

As a result, the number of detected infected files is 93 out of 125, and the Mean fitness = **1205.2000**.

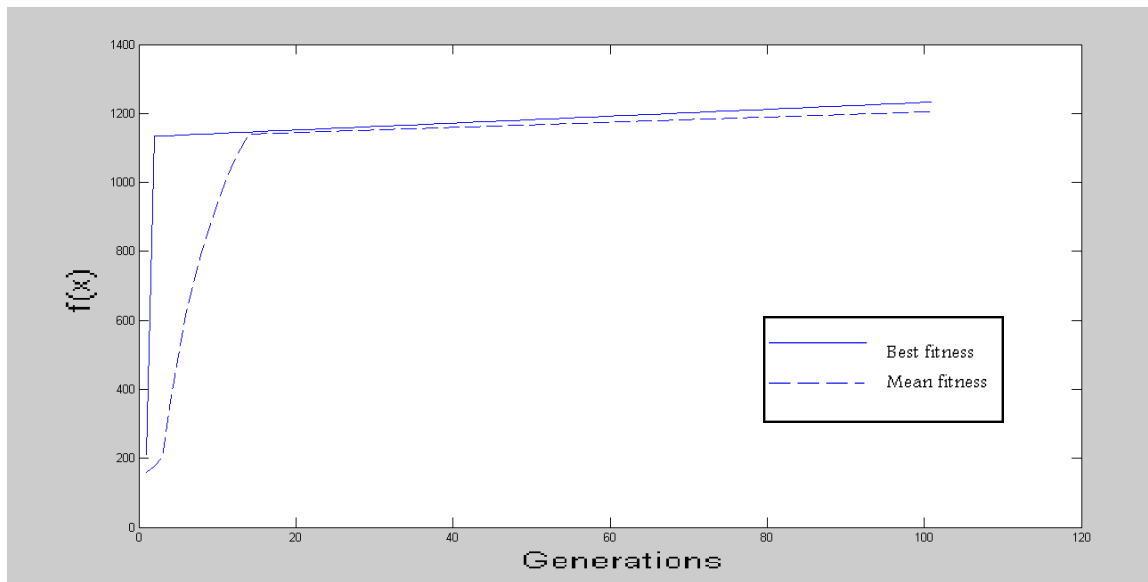


Figure 73: The third GA training run Mean Fitness & Best Fitness

The initial population is exhibited in Figure (74) in a solid line, while the final population which includes the new signatures after Hypermutation is presented by dotted line. This figure shows that the fitness of the new mutated signatures is higher.

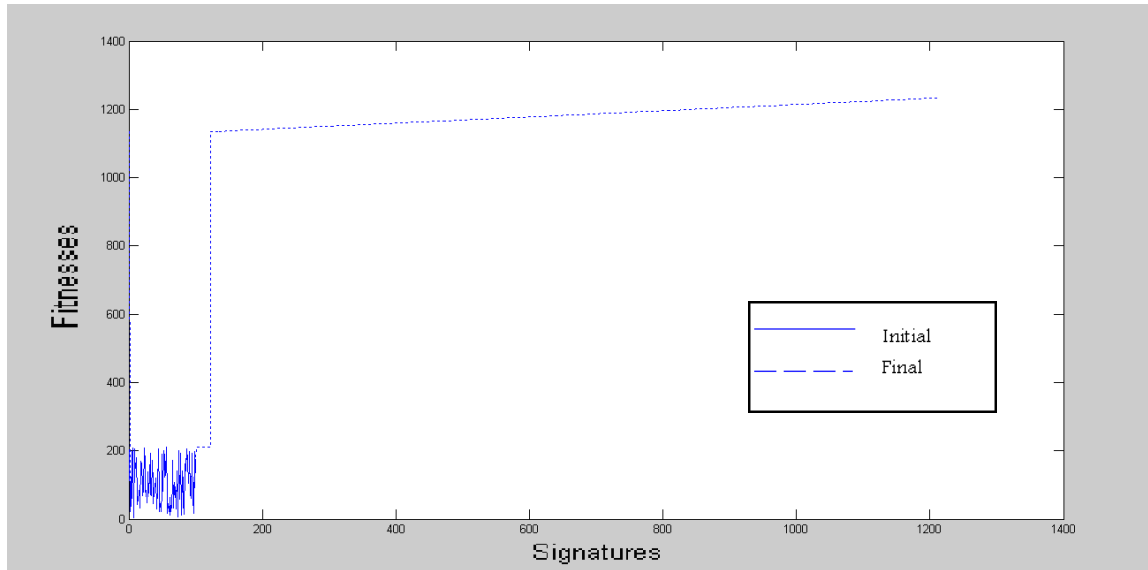


Figure 74: The third GA training run final population

The **forth** GA training run parameters are: *GA Learning Gen*=100, *Pm*=0.914 and *Fat* = 0.935, and the files' pool with 75% of infected files and the produced signatures' pool is saved as **SigGA4**. The results of the forth run are exhibited in Table (38), where each iteration corresponds to the number of signatures in the signatures' pool. On the first iteration, the number of signatures value is 100 and, it is increased by 11 signatures or less in each iteration, to have at the last iteration 1200 signatures.

The selection threshold ranges between 0.6 and 1.0; this threshold determines the selection process stochastically.

The Mean Fitness increases at a rapid rate in the beginning, then the increase continues at a slower rate; at first it increases at a higher changing rate reaching 17.42, after that the rate increases to reach 57.01 in the second iteration, to become 469.3 in the third iteration and so on, till it reaches around 0.74 in the last five iterations, as shown in Table (38).

While the Best Fitness increases quickly in the first iteration only, and later on the increase becomes slower; in the first iteration the changing rate is 2783 for the Best Fitness, after that it increases by 1 in each iteration. Figure (75) displays the Mean Fitness and the Best Fitness.

Table 38: The forth GA training run results

Iteration number	No of signatures	Selection threshold	Mean fitness	Best fitness
1	100	0.90	160.12	211.00
2	111	1.00	177.54	2994.00
3	122	0.70	234.55	2995.00
4	133	0.60	703.85	2996.00
5	144	0.90	1092.34	2997.00
...
96	1145	0.80	3061.51	3088.00
97	1156	0.70	3062.24	3089.00
98	1167	0.80	3063.01	3090.00
99	1178	0.70	3063.73	3091.00
100	1189	1.00	3064.51	3092.00
101	1200	0.90	3065.23	3093.00

As a result, the number of detected infected files is 279 out of 375, and the Mean fitness = 3065.200.

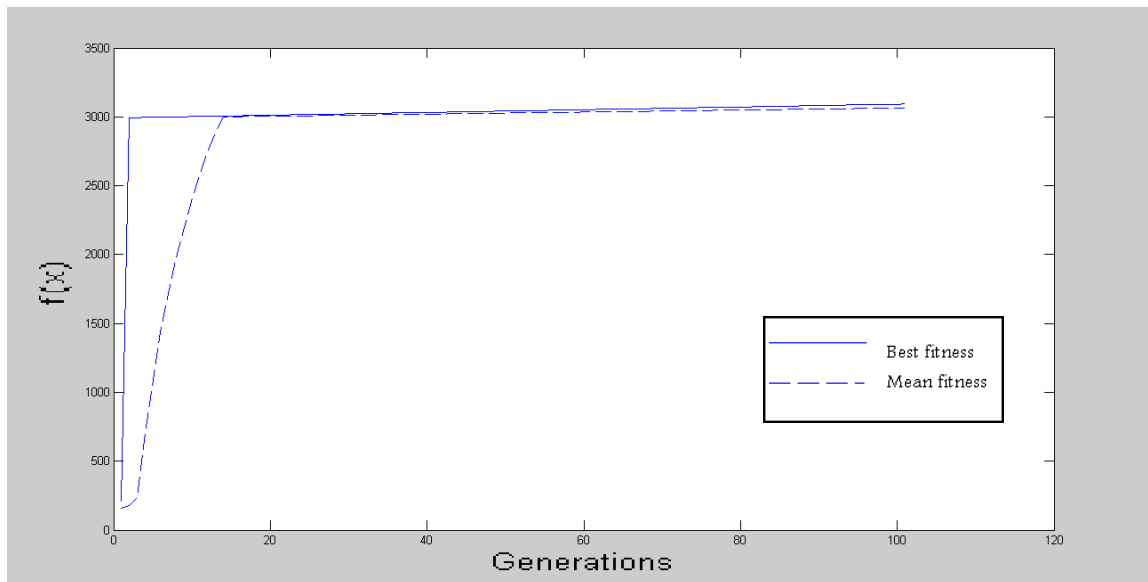


Figure 75: The forth GA training run Mean Fitness & Best Fitness

The initial population is exhibited in Figure (76) in a solid line, while the final population which includes the new signatures after Hypermutation is presented by dotted line. This figure shows that the fitness of the new mutated signatures is higher.

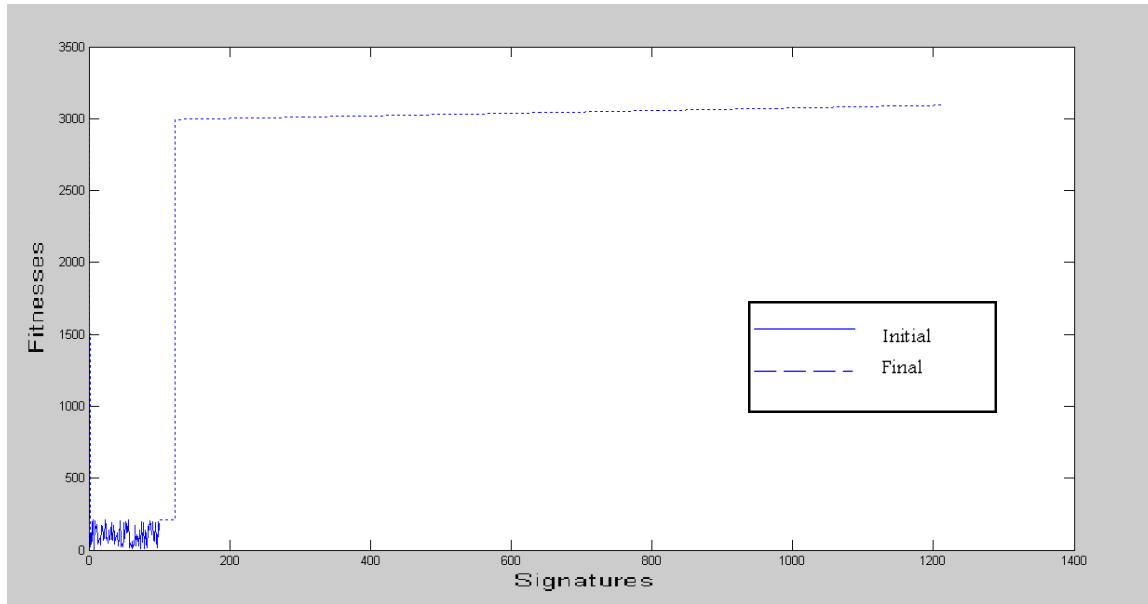


Figure 76: The forth GA training run final population

GA training Analysis:

After finalizing the GA Training process, which includes 4 runs, the results are recapitulated in Table (39). The table reveals the Mean fitness, which has resulted from defining the parameters values, *GA Learning Gen*, *Pm*, *Fat*, *GA VDC Gen*, and the GA Training pool.

The *learning Gen* has been set to equal 100 for all the training cases.

As mentioned previously, the *Pm* and *Fat* control the Hypermutation process (each Hypermutation affects the fitness by adding 1 at most). Where the *Pm* ranges between 0.636 and 0.96, and the *Fat* ranges between 0.935 and 1.0. These values have been gained from GA Optimization, and they are high, which increases the Hypermutation rate. It is recalled that the *GA VDC Gen* has affected the choosing of these values for *Pm* and *Fat*.

Table 39: The Summary of the GA training results

GA signatures' pool	GA Learning Gen	GA Training pool	Pm	Fat	GA VDC Gen	Mean fitness
SigGA1	100	5%	0.636	0.935	10	445.2282
SigGA2	100	50%	0.96	1.0	20	2595.2
SigGA3	100	25%	0.65	0.96	30	1205.2
SigGA4	100	75%	0.914	0.935	10	3065.2

The values of the GA Training pool are 5%, 25%, 50% and 75%, and these values are the same as used for the GA training pool in the previous step (GA Optimization), whenever the number of infected files increases at the training files' pool, the Mean fitness increases.

It is noticed in the results of the previous 4 runs that the number of signatures = 1200 for all of them. The reason for this is that the used *GA Learning Gen* is 100. In order to produce the signatures' pools (SigGA1 ... SigGA4) for the GA matching process, the GA training process has been performed. Consequently, further conclusions on the GA training process lack the space and time.

5.1.3 GA Matching

The GA matching checks the signatures' pools resulted in the GA training process where the number of runs is 10, according to Table (40).

Table 40: The GA matching runs specifications

GA Matching pool	GA signatures' pool
0%	SigGA2
5%	SigGA4
25%	SigGA1, SigGA2
50%	SigGA4
75%	SigGA1
100%	SigGA1, SigGA2, SigGA3, SigGA4

SigGA2 is tested with the files' pool with 0% infected files (all the files are benign). The results are figured in Table (41).

Table 41: The results of the matching of SigGA2 with 0% infected files

Iteration number	Mean fitness	Best fitness
1	2329.15277	2623.00
2	2329.15277	2623.00
3	2329.15277	2623.00
4	2329.15277	2623.00
5	2329.15277	2623.00
...
96	2329.15277	2623.00
97	2329.15277	2623.00
98	2329.15277	2623.00
99	2329.15277	2623.00
100	2329.15277	2623.00
101	2329.15277	2623.00

As a result, the number of infected files = 0 and the Mean fitness = 2329.2.

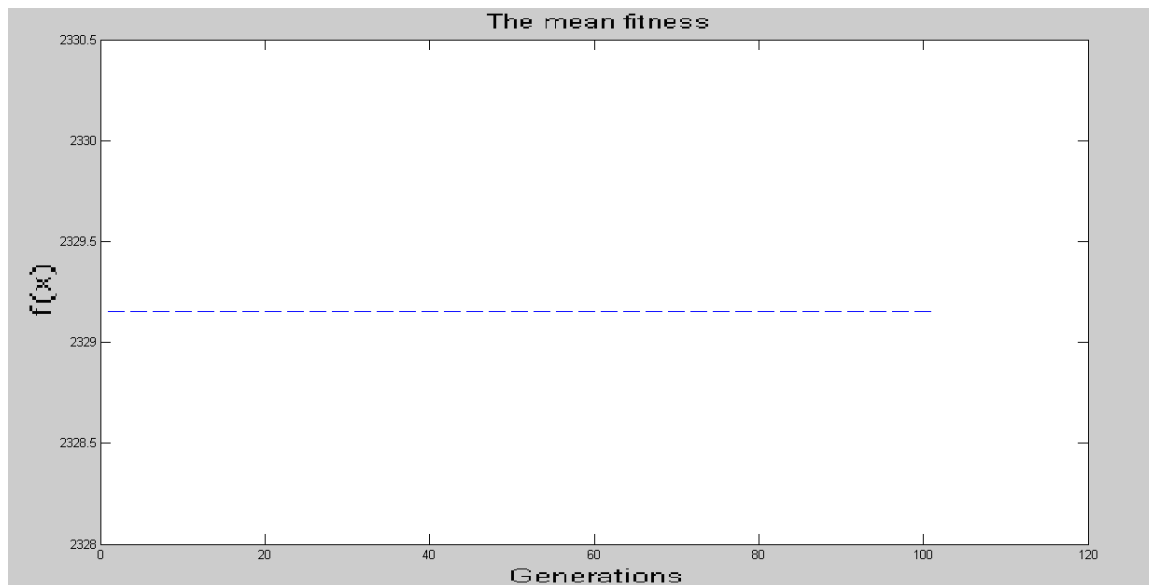


Figure 77: The Mean fitness of matching Sig1 run with 0% infected files

As illustrated in Table (41), Figures (77) and (78), none of the files are detected as infected files, so the Mean fitness and Best fitness do not change. This is due to the fact that all files are benign. Hence, the detection rate is 100%. The result properly reflects the reality, and it is an accepted result.

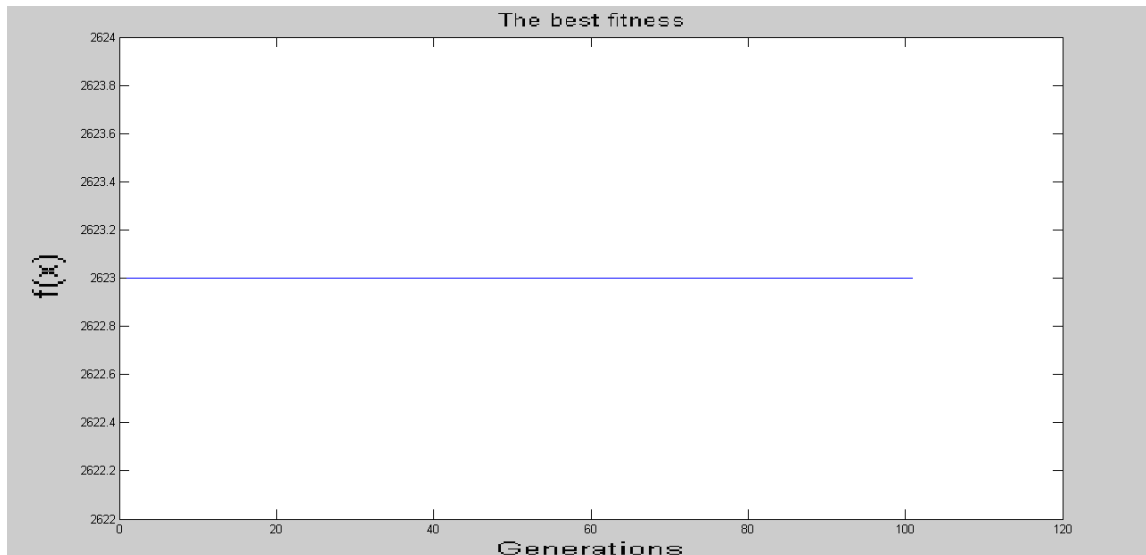


Figure 78: The Best fitness of matching Sig1 run with 0% infected files

SigGA4 is tested with the files pool with 5% infected files. At iteration number 5, new 100 files are added to the files' pool, with 5% of these files are infected. The results are shown in Table (42).

Table 42: The results of the matching of SigGA4 with 5% infected files

Iteration number	Mean fitness	Best fitness
1	2752.19158	3093.00
2	2752.21140	3093.00
3	2752.21140	3093.00
4	2752.21140	3093.00
5	2752.21140	3093.00
6	2752.21470	3093.00
...
97	2752.21470	3093.00
98	2752.21470	3093.00
99	2752.21470	3093.00
100	2752.21470	3093.00
101	2752.21470	3093.00

As a result, the number of infected files= 28 and the Mean fitness = 2752.2.

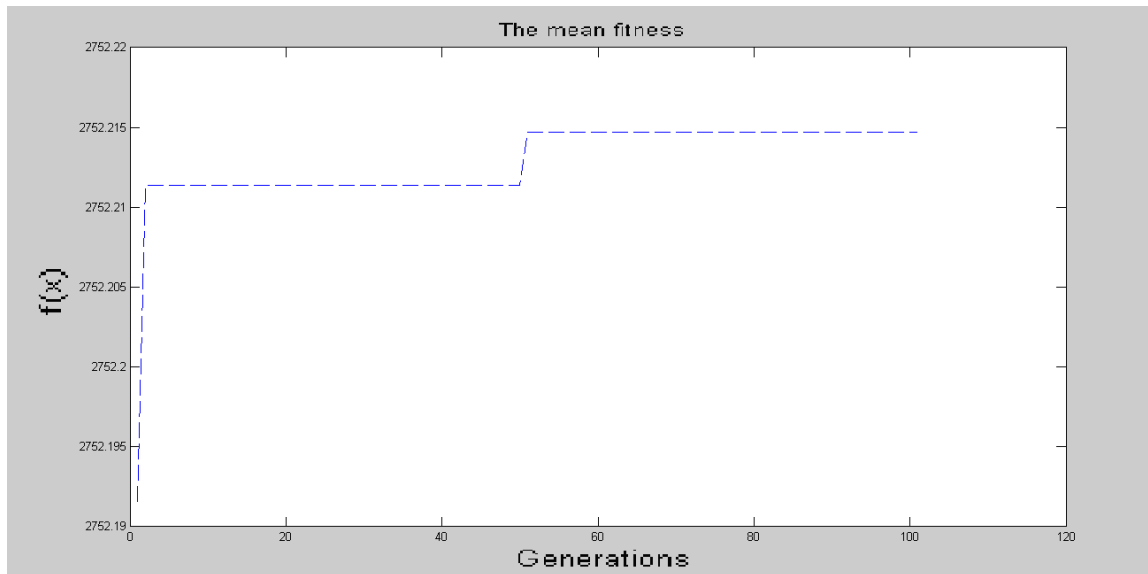


Figure 79: The Mean fitness of matching SigGA4 run with 5% infected files

Table (42) and Figure (79) show that the Mean fitness increases by 0.01982 in the first iteration, whereas it increases at the iteration number 50 by 0.0033.

The straight line in Figure (80) is for the Best fitness, which does not change. The number of detected files is 28 out of 30 (25+5) with a Detection rate of 93.3%, where the 25 infected files are in the original pool, and the 5 are from the new added pool. Hence this detection rate is accepted.

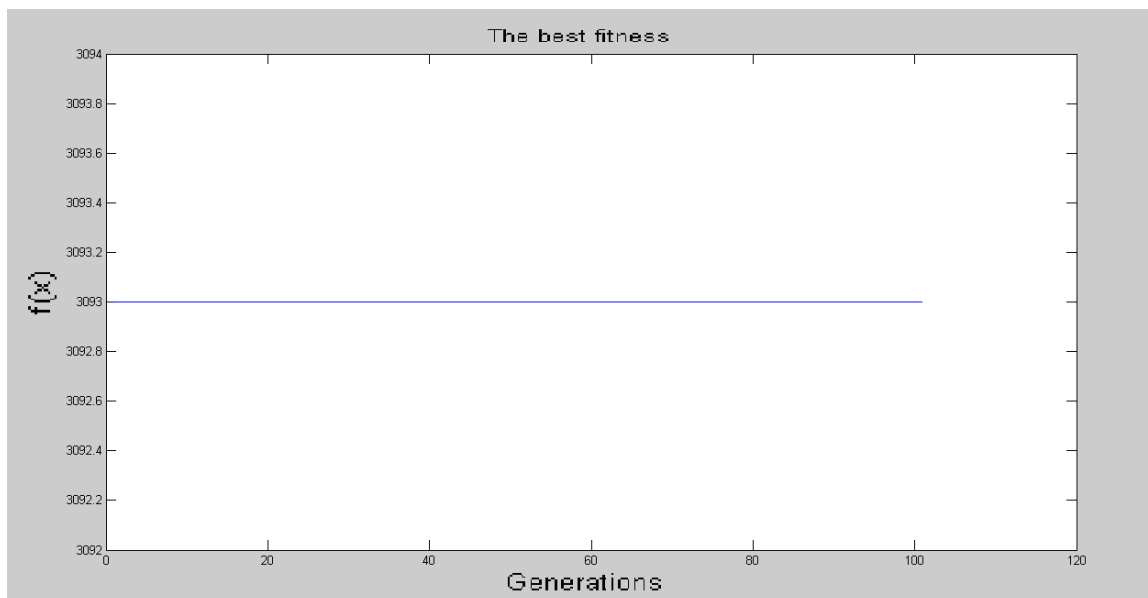


Figure 80: The Best fitness of matching SigGA4 run with 5% infected files

SigGA1 is tested with the files pool with 25% infected files. At iteration number 50, new 100 files are added to the files' pool, with 25% of these files infected. The results are shown in Table (43).

Table 43: The results of the matching of SigGA1 with 25% infected files

Iteration number	Mean fitness	Best fitness
1	393.97523	473.00
2	394.07762	473.00
3	394.07762	473.00
4	394.07762	473.00
5	394.07762	473.00
...
50	394.07762	473.00
51	394.09249	473.00
...
98	394.09249	473.00
99	394.09249	473.00
100	394.09249	473.00
101	394.09249	473.00

As a result, the number of infected files=142 and the Mean fitness= 394.0925.

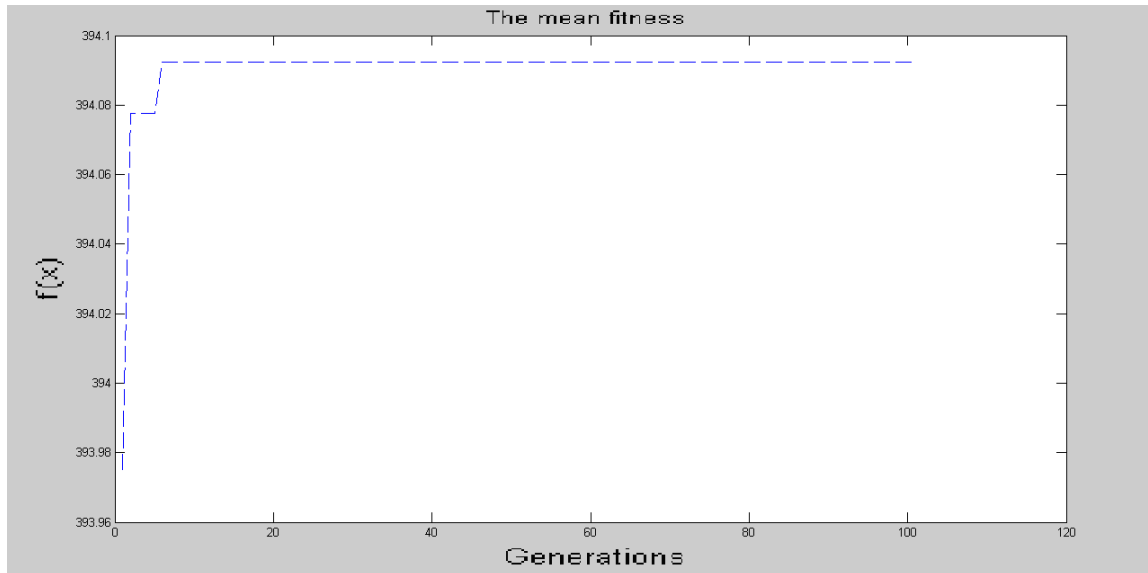


Figure 81: The Mean fitness of matching SigGA1 run with 25% infected files

Table (43) and Figure (81) show that the Mean fitness increases by 0.10239 in the first iteration whereas it increases at the iteration number 50 by 0.01487.

The Best fitness does not change; this explains the straight line in Figure (82). The number of detected files is 142 out of 150 (125+25) with a Detection rate of 94.7%, where the 125 infected files are in the original pool, and the 25 are from the new added pool. Hence this detection rate is accepted.

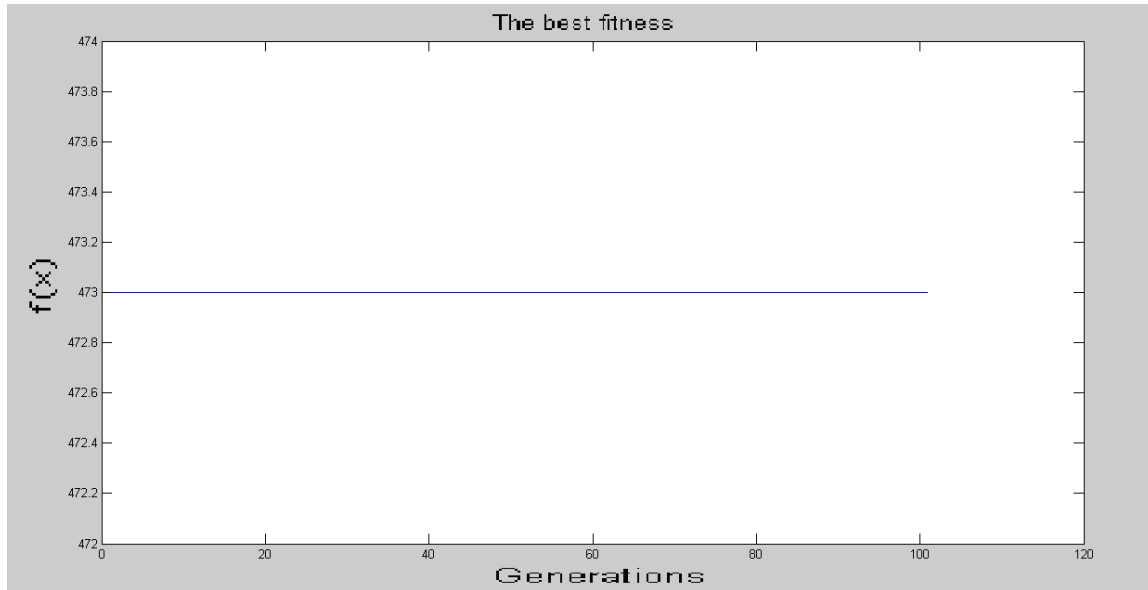


Figure 82: The Best fitness of matching SigGA1 run with 25% infected files

SigGA2 is tested with the files pool with 25% infected files. At iteration number 50, new 100 files are added to the files' pool, with 25% of these files infected. The results summary is shown in Table (44).

Table 44: The summary of the GA matching results of SigGA2 with 25% infected files

Signatures pool	Matching Pool	Detected Files	Mean fitness	Best fitness	Detection Rate
SigGA2	25%	142	2329.2700	263.00	94.7%

In the matching of SigGA2 the Mean fitness increases at two places; in the first iteration and iteration number 50. The Best fitness does not change. SigGA2 is illustrated in Appendix (C1).

The matching of **SigGA4** with the files pool with 50% infected files. At iteration number 50, new 100 files are added to the files' pool, with 50% of these files infected. The results are shown in Table (45).

Table 45: The results of the matching of SigGA4 with 50% infected files

Iteration number	Mean fitness	Best fitness
1	2752.19158	3093.00
2	2752.39802	3093.00
3	2752.39802	3093.00
4	2752.39802	3093.00
5		3093.00
...		...
50	2752.39802	3093.00
51	2752.42940	3093.00
...		...
98	2752.42940	3093.00
99	2752.42940	3093.00
100	2752.42940	3093.00
101	2752.42940	3093.00

As a result, the number of infected files=288 and the Mean fitness= 2752.4.

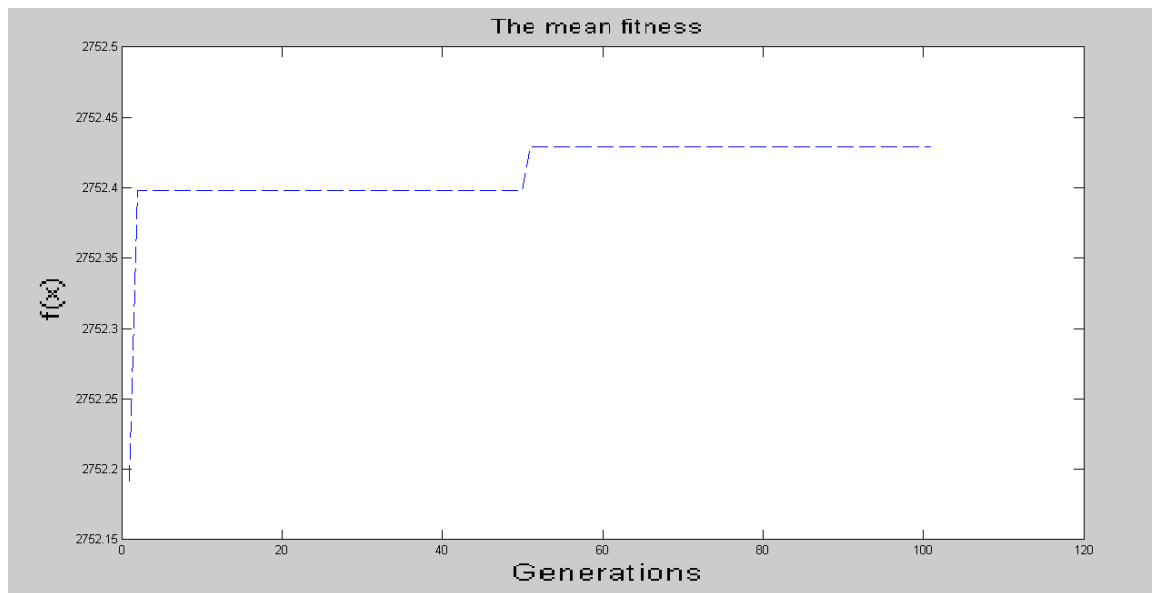


Figure 83: The Mean fitness of matching SigGA4 run with 50% infected files

Table (45) and Figure (85) show that the Mean fitness increases by 0.20644 in the first iteration whereas it increases at the iteration number 50 by 0.03138.

The Best fitness does not change; this explains the straight line in Figure (86). The number of detected files is 288 out of 300 (250+50) with a Detection rate of 96.0%, where the 250 infected files are in the original pool, and the 50 are from the new added pool. As a result this detection rate is accepted.

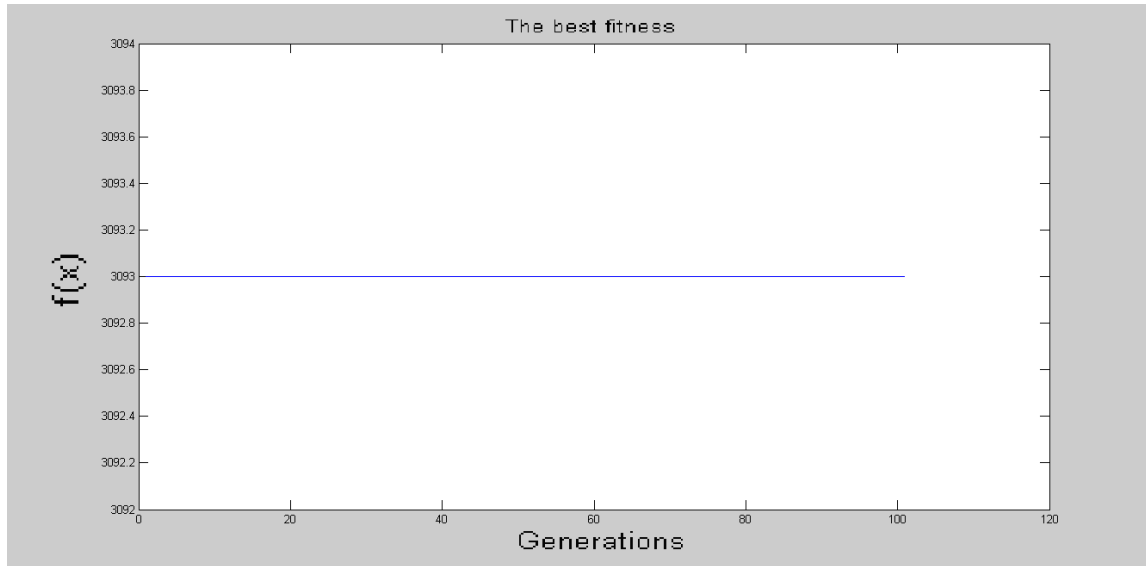


Figure 84: The Best fitness of matching SigGA4 run with 50% infected files

The matching of **SigGA1** with the files pool with 75% infected files. At iteration number 5, new 100 files are added to the files' pool, with 75% of these files infected. The results are shown in Table (46).

Table 46: The results of the matching of SigGA1 with 75% infected files

Iteration number	Mean fitness	Best fitness
1	393.97523	473.00
2	394.27911	473.00
3	394.27911	473.00
4	394.27911	473.00
5	394.27911	473.00
6	394.31049	473.00
...
97	394.31049	473.00
98	394.31049	473.00
99	394.31049	473.00
100	394.31049	473.00
101	394.31049	473.00

As a result, the number of infected files = 406 and the Mean fitness = 394.3105.

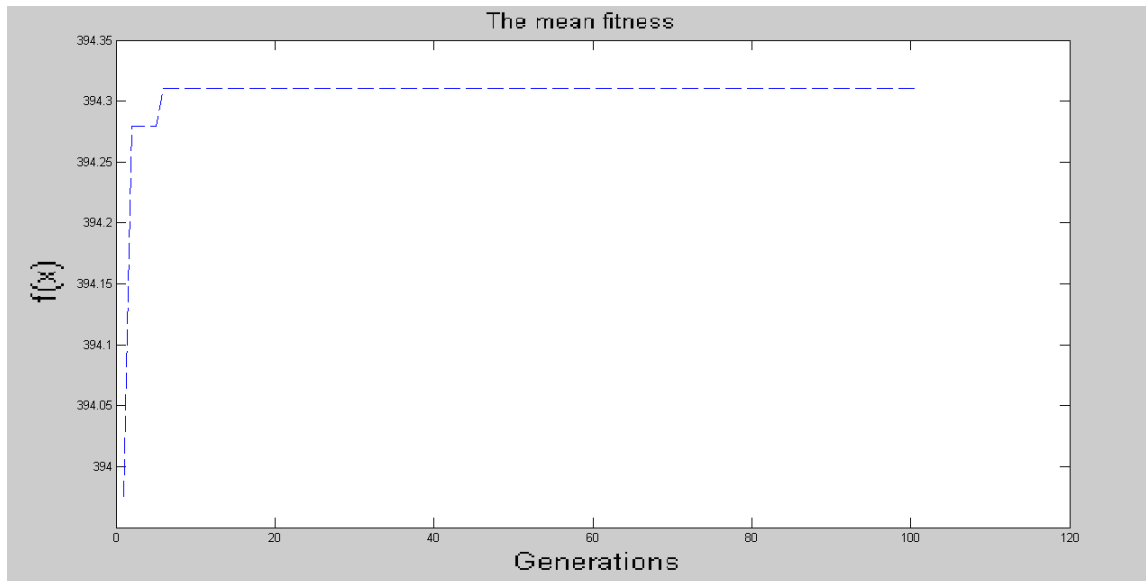


Figure 85: The Mean fitness of matching SigGA1 run with 75% infected files

Table (46) and Figure (87) show that the Mean fitness increases by 0.30388 in the first iteration whereas it increases at the iteration number 50 by 0.03138.

The Best fitness does not change; this explains the straight line in Figure (88). The number of detected files is 406 out of 450 (375+75) with a Detection rate of 90.2%, where the 375 infected files are in the original pool, and the 75 are from the new added pool. As a result this detection rate is accepted.



Figure 86: The Best fitness of matching SigGA1 run with 75% infected files

The matching of **SigGA1** with the files pool with 100% infected files. At iteration number 50, new 100 files are added to the files' pool, with 100% of these files infected. The results are shown in Table (47).

Table 47: The results of the matching of SigGA1 with 100% infected files

Iteration number	Mean fitness	Best fitness
1	393.97523	473.00
2	394.38481	473.00
3	394.38481	473.00
4	394.38481	473.00
5	394.38481	473.00
...
50	394.38481	473.00
51	394.43270	473.00
...
98	394.43270	473.00
99	394.43270	473.00
100	394.43270	473.00
101	394.43270	473.00

As a result, the number of infected files = 554 and the Mean fitness = 394.4327.

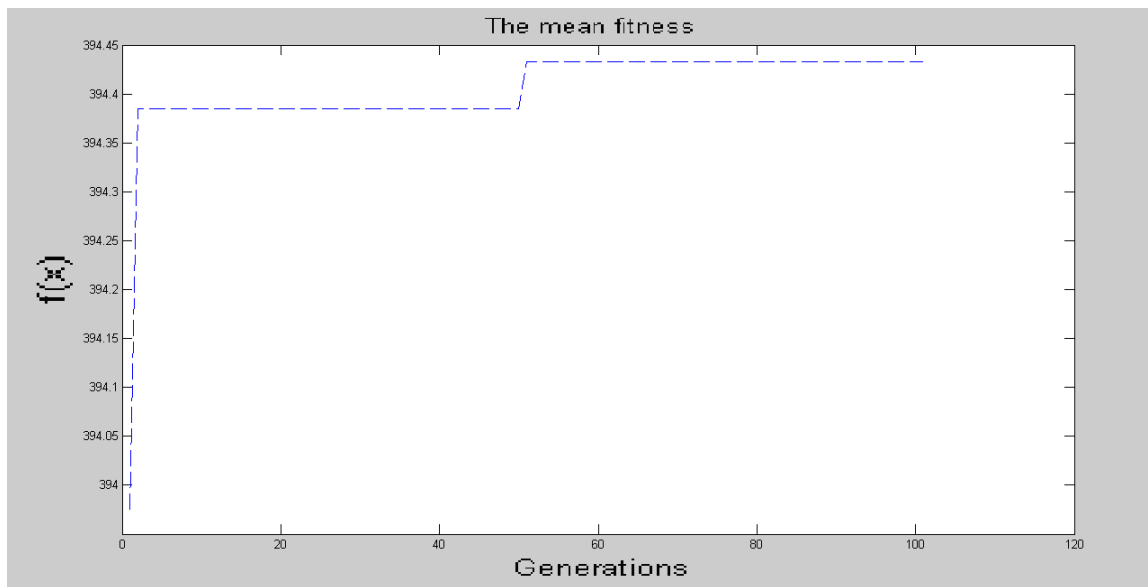


Figure 87: The Mean fitness of matching SigGA1 run with 100% infected files

Table (47) and Figure (89) show that the Mean fitness increases by 0.40958 in the first iteration whereas it increases at the iteration number 50 by 0.04789.

The Best fitness does not change; this explains the straight line in Figure (90). The number of detected files is 554 out of 600 (500+100) with a Detection rate of 92.3%, where the 500 infected files are in the original pool, and the 100 are from the new added pool. As a result this detection rate is accepted.

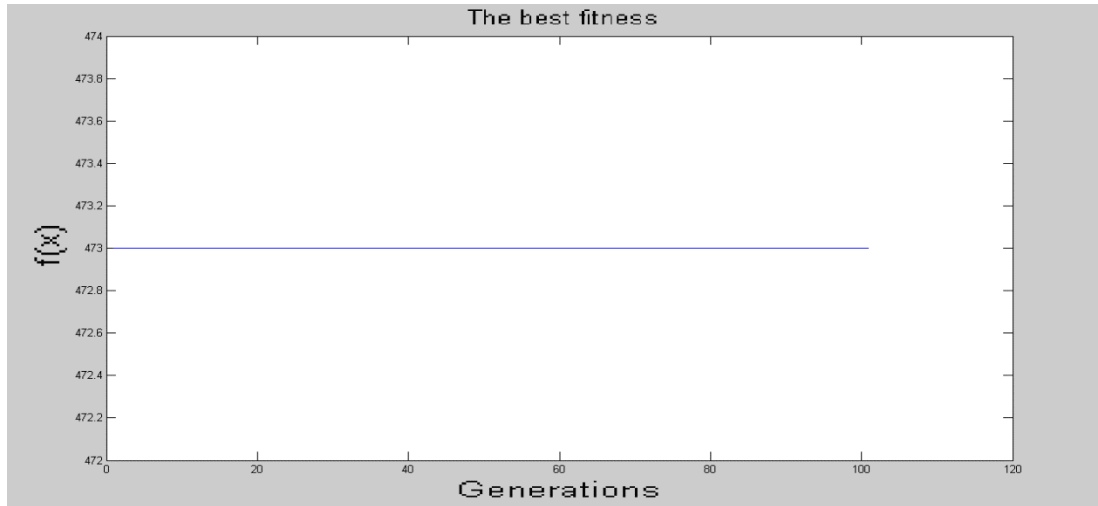


Figure 88: The Best fitness of matching SigGA1 run with 100% infected files

The matching results of **SigGA2**, **SigGA3** and **SigGA4** with the files pool with 100% infected files are recapitulated in Table (48). At iteration number 50, new 100 files are added to the files' pool, with 100% of these files infected.

Table 48: The summary of the GA matching of SigGA2, SigGA3 and SigGA4 with 100% infected files

Signatures pool	Matching Pool	Detected Files	Mean fitness	Best fitness	Detection Rate
SigGA2	100%	554	2329.6103	2623.00	92.3%
SigGA3	100%	554	1078.4955	1233.00	92.3%
SigGA4	100%	554	2752.6491	3093.00	92.3%

In the above mentioned matching runs the Mean fitness increases at two places; in the first iteration and iteration number 50 by 0.04789. The Best fitness does not change. SigGA3 is presented in Appendix (C2).

GA Matching Analysis:

The results of the GA matching process, which includes 10 runs, are summed up in tables (49) and (50). Table (49) is elaborated in regard to:

- Δ Best fitness:

In all cases, discrepancy dose not occur in the change in the Best fitness.

This is due to the resulted values for Pm and Fat , from the GA optimization, that are high, which has led to having high values for the Mean fitness and Best fitness (as has been described in section 4.2.1). Consequently, the Best fitness remains the same.

- The Mean fitness and the Δ Mean fitness:

The Mean Fitness and the Δ Mean fitness change, because the following 5 Variables: *Pm*, *Fat*, GA Training Pool, GA VDC Gen, and GA Matching Pool have changed. Hence the *GA Learning Gen* = 100 for all runs, it is not considered as a variable.

For the GA Matching pool variable, when the number of infected files increases inside the pool, the Mean Fitness increases by a small value. For SigGA4, when the matching pool = 5%, the Mean Fitness = 2752.2, and when the matching pool = 100%, the Mean Fitness = 2752.6 .The deviation is 0.4, which is considered as a small value.

Notably, the GA Matching pool is the main variable that affects the Δ Mean fitness. The reason for that is that when infected files increase in the GA Matching pool, the Δ Mean fitness increases, because each detection increases the fitness by δ .

As demonstrated in the whole table, the Δ Mean fitness ranges between 0.02312 as a lower value, when GA Matching pool = 5% and 0.45747 as a higher value, when the GA Matching pool = 100%.

As it is noticed in Table (49), even though there have been changes in the values of the GA Training pool, *Pm* and *Fat*, like when the GA Matching pools equals 100%, the Δ Mean fitness value does not change, but the Mean fitness changes.

Table 49: The GA Matching results

Signature Pool	GA Learning Gen	<i>Pm</i>	<i>Fat</i>	GA Training pool	GA VDC Gen	GA Matching Pool	Δ Best fitness	Mean fitness	Δ Mean fitness
SigGA4	100	0.914	0.935	75%	10	5%	0	2752.2	0.02312
SigGA2	100	0.96	1.0	50%	20	25%	0	2329.3	0.11725
SigGA1	100	0.636	0.935	5%	10	25%	0	394.0925	0.11726
SigGA4	100	0.914	0.935	75%	10	50%	0	2752.4	0.23782
SigGA1	100	0.636	0.935	5%	10	75%	0	394.3105	0.33526
SigGA1	100	0.636	0.935	5%	10	100%	0	394.4327	0.45747
SigGA2	100	0.96	1.0	50%	20	100%	0	2329.6	0.45747
SigGA3	100	0.65	0.96	25%	30	100%	0	1078.5	0.45747
SigGA4	100	0.914	0.935	75%	10	100%	0	2752.6	0.45747

The detection rate of the 10 runs appears in Table (50). The detection rate is

100% in the case of 0% infected files, as it has detected zero number of infected files. This is called the false positive testing, and is considered as a good result.

The rest of the files' pool (5%, 25%, 50%, 75% and 100% of infected files), have the detection rate that ranges between 90.2% and 96% with the average of all cases equals 94.4%. According to the researcher, this is considered as a good result, too.

The procedures that have been performed through the GA optimization, GA training and GA matching processes illustrate that the use of the VDC algorithm and GA are applicable for solving the problem of computer viruses detection. This conclusion answers the second question of the dissertation questions.

Table 50: The GA Detection Rate of the GA matching Results

GA Matching pool	Signatures' pools	Detection rate
0%	SigGA2	100%
5%	SigGA4	93.3%
25%	SigGA1, SigGA2	94.7%
50%	SigGA4	96%
75%	SigGA1	90.2%
100%	SigGA1, SigGA2, SigGA3, SigGA4	92.3%
The Average of detection rate		94.4%

5.2. The Comparison between the Standard VDC algorithm and the Optimized VDC algorithm based on GA

After Testing (Training and Matching processes) the standard VDC algorithm and the optimized VDC algorithm based on GA, and obtaining the results, this section presents a comparison between the results of each of them.

While comparing the training processes in both algorithms, Tables (17) and (39) must be noticed, the GA improves the Mean Fitness value enormously. When the training pool = 5% in the standard VDC, the Mean Fitness ranges between 242.2893 and 275.2660, while with the GA, the Mean Fitness equals 495.2282

. And when the training pool = 25%; in the standard VDC, the Mean Fitness ranges between 449.6050 and 642.7941, while with the GA, the Mean Fitness equals 1205.2. Finally, when the training pool = 75% in the standard VDC, the Mean Fitness value ranges between 838.7423 and 1265.9, while with the GA, the Mean Fitness equals 3065.2 .

The above results reveal that the GA has improved the values of the Mean Fitness enormously in the training phase. This is because the values of the *Pm* and *Fat* increase largely after using the GA.

The comparison of the matching processes in regard to the Mean fitness, the Δ Mean fitness and the Detection Speed. Table (51) summarizes the Mean fitness and the Δ Mean fitness for both algorithms.

Table 51: The comparison according to the Mean fitness and the Δ Mean fitness

Files pool	Signature pool	Mean fitness	Δ Mean fitness	GA Signature pool	Mean fitness	Δ Mean fitness
0%	Sig1	226.6609	0	SigGA2	2329.2	0
	Sig5	485.3166	0			
	Sig8	1167.2	0			
5%	Sig6	585.4639	0.01590	SigGA4	2752.2	0.02312
	Sig7	924.6069	0.02312			
	Sig10	556.9083	0.02312			
	Sig11	786.5623	0.02313			
25%	Sig1	226.7783	0.11745	SigGA1	394.0925	0.11726
	Sig2	223.7168	0.11726	SigGA2	2329.3	0.11725
	Sig12	225.3997	0.11726			
50%	Sig4	383.4732	0.23782	SigGA4	2752.4	0.23782
	Sig5	485.4011	0.08444			
	Sig8	1167.4	0.16354			
75%	Sig1	226.9967	0.33581	SigGA1	394.3105	0.33526
	Sig3	253.7394	0.11902			
	Sig6	585.6786	0.23055			
	Sig9	1167.5	0.11903			
	Sig12	225.6177	0.33526			
100 %	Sig2	224.057	0.45748	SigGA1	394.4327	0.45747
	Sig4	383.6928	0.45747	SigGA2	2329.6	0.45747
	Sig7	925.0413	0.45747	SigGA3	1078.5	0.45747
	Sig10	557.3427	0.45747	SigGA4	2752.6	0.45747
	Sig11	786.9967	0.45748			
	Sig12	225.7399	0.45747			

- When Matching pool = 5% there are 2 cases to compare:
 1. Sig7 and SigGA4: where in both of them the *learning Gen* = 100 and the Training pool =75%. They differ in the *Pm* and *Fat* values; where in Sig7 the

$P_m=0.2$ and the $Fat=0.05$, and in SigGA4 the $P_m=0.914$ and $Fat=0.935$. The mean Fitness in Sig7= 924.6069 and in SigGA4 = 2752.2, so it is clear that the GA improves the Mean Fitness. The Δ Mean fitness =0.02312 in both of them.

2. Sig11 and SigGA4: where in both of them the *learning Gen* = 100 and the Training pool =75%. They differ in the P_m and Fat values; where in Sig11 the $P_m=0.2$ and the $Fat=0.1$, and in SigGA4 the $P_m=0.914$ and $Fat=0.935$. The Mean Fitness in Sig11 = 786.5623 and in SigGA4 = 2752.2, so it is clear that the GA improves the Mean fitness. The Δ Mean fitness for Sig11 =0.02312, and for SigGA4= 0.02313. This Means the deviation =0.00001, where in Sig11, with $P_m =0.2$ and $Fat=0.1$, the Δ Mean fitness is higher.

- When Matching pool = 25% there are 3 cases to compare:

1. Sig1 and SigGA1: where in both of them the *learning Gen* = 100 and the Training pool =5%. They differ in the P_m and Fat values; where in Sig1 the $P_m =0.05$ and the $Fat=0.05$, and in SigGA1 the $P_m=0.636$ and $Fat=0.935$. The Mean Fitness in Sig1 = 226.782 and in SigGA1 = 394.0925, so it is clear that the GA improves the Mean fitness. The Δ Mean fitness for Sig1 =0.11745, and for SigGA1= 0.11726. This means the deviation =0.00019, where in Sig1, with $P_m =0.05$ and $Fat=0.05$, the Δ Mean fitness is higher.

2. Sig2 and SigGA1: where in both of them the *learning Gen* = 100 and the Training pool =5%. They differ in the P_m and Fat values; where in Sig2 the $P_m =0.1$ and the $Fat=0.05$, and in SigGA1 the $P_m=0.636$ and $Fat=0.935$. The Mean Fitness in Sig2 = 223.7168 and in SigGA1 = 394.0925, so it is clear that the GA improves the Mean fitness. The Δ Mean fitness =0.11726 in both of them.

3. Sig12 and SigGA1: where in both of them the *learning Gen* = 100 and the Training pool =5%. They differ in the P_m and Fat values; where in Sig12 the $P_m =0.05$ and the $Fat=0.1$, and in SigGA1 the $P_m=0.636$ and $Fat=0.935$. The Mean Fitness in Sig12 = 225.3997 and in SigGA1 = 394.0925, so it is clear that the GA improves the Mean fitness. The Δ Mean fitness =0.11726 in both of them.

- When Matching pool = 50% there are 1 case to compare:
Sig8 and SigGA4: where in both of them the Training pool =75%. They differ in the *Learning Gen*, *Pm* and *Fat* values; where in Sig8 the *Learning Gen* =150, the *Pm*=0.1 and the *Fat*=0.1, and in SigGA4 the *Learning Gen*=100, the *Pm*=0.914 and *Fat*=0.935. The Mean Fitness in Sig8 = 1167.4 and in SigGA4 = 2752.4, so it is clear that the GA improves the Mean fitness. The Δ Mean fitness for Sig8 =0.16354, and for SigGA4= 0.23782. This means the deviation =0.07428, where in SigGA4, with *Learning Gen*=100, *Pm* =0.05 and *Fat*=0.05, the Δ Mean fitness is higher.
- When Matching pool = 75% there are 2 cases to compare:
 1. Sig1 and SigGA1: where in both of them the training Gen = 100 and the Training pool =5%. They differ in the *Pm* and *Fat* values; where in Sig1 the *Pm* =0.05 and the *Fat*=0.05, and in SigGA1 the *Pm*=0.636 and *Fat*=0.935. The Mean Fitness in Sig1 = 226.9967 and in SigGA1 = 394.3105, so it is clear that the GA improves the Mean fitness. The Δ Mean fitness for Sig1 =0.33581, and for SigGA1= 0.33526. This means the deviation =0.00055, where in Sig1, with *Pm* =0.05 and *Fat*=0.05, the Δ Mean fitness is higher.
 2. Sig12 and SigGA1: where in both of them the *learning Gen* = 100 and the Training pool =5%. They differ in the *Pm* and *Fat* values; where in Sig12 the *Pm* =0.05 and the *Fat*=0.1, and in SigGA1 the *Pm*=0.636 and *Fat*=0.935. The Mean Fitness in Sig12 = 225.6177 and in SigGA1 = 394.3105, so it is clear that the GA improves the Mean fitness. The Δ Mean fitness =0.33526 in both of them.
- When Matching pool = 100% there are 6 cases to compare:
 1. Sig2 and SigGA1: where in both of them the *learning Gen* = 100 and the Training pool =5%. They differ in the *Pm* and *Fat* values; where in Sig2 the *Pm* =0.1 and the *Fat*=0.05, and in SigGA1 the *Pm*=0.636 and *Fat*=0.935. The Mean Fitness in Sig2 = 224.057 and in SigGA1 = 394.4327, so it is clear that the GA improves the Mean fitness. The Δ Mean fitness for Sig2 =0.45748, and for SigGA1= 0.45747.

This means the deviation =0.00001, where in Sig2, with $Pm = 0.05$ and $Fat=0.05$, the Δ Mean fitness is higher.

2. Sig12 and SigGA1: where in both of them the *learning Gen* = 100 and the Training pool =5%. They differ in the Pm and Fat values; where in Sig12 the $Pm = 0.05$ and the $Fat=0.1$, and in SigGA1 the $Pm=0.636$ and $Fat=0.935$. The Mean Fitness in Sig12 = 225.7399 and in SigGA1 = 394.4327, so it is clear that the GA improves the Mean fitness. The Δ Mean fitness =0.45747 in both of them.
3. Sig4 and SigGA3: where in both of them the *learning Gen* = 100 and the Training pool =25%. They differ in the Pm and Fat values; where in Sig4 the $Pm=0.05$ and the $Fat=0.05$, and in SigGA3 the $Pm=0.65$ and $Fat=0.96$. The Mean Fitness in Sig4 = 383.6928 and in SigGA3 = 1078.5, so it is clear that the GA improves the Mean fitness. The Δ Mean fitness =0.45747 in both of them.
4. Sig10 and SigGA3: where in both of them the *learning Gen* = 100 and the Training pool =25%. They differ in the Pm and Fat values; where in Sig10 the $Pm = 0.2$ and the $Fat=0.05$, and in SigGA3 the $Pm=0.65$ and $Fat=0.96$. The Mean Fitness in Sig10 = 557.3427 and in SigGA3 = 1078.5, so it is clear that the GA improves the Mean fitness. The Δ Mean fitness for Sig10 =0.45748, and for SigGA3= 0.45747. This means the deviation =0.00001, where in Sig10, with $Pm = 0.2$ and $Fat=0.05$, the Δ Mean fitness is higher.
5. Sig7 and SigGA4: where in both of them the *learning Gen* = 100 and the Training pool =75%. They differ in the Pm and Fat values; where in Sig7 the $Pm = 0.2$ and the $Fat=0.05$, and in SigGA4 the $Pm=0.914$ and $Fat=0.935$. The Mean Fitness in Sig7 = 925.0413 and in SigGA4 = 2752.6, so it is clear that the GA improves the Mean fitness. The Δ Mean fitness =0.45747 in both of them.
6. Sig11 and SigGA4: where in both of them the *learning Gen* = 100 and the Training pool =75%. They differ in the Pm and Fat values; where in Sig11 the $Pm = 0.2$ and the $Fat=0.1$, and in SigGA4 the $Pm=0.914$ and $Fat=0.935$.

The Mean Fitness in Sig11 = 786.9967 and in SigGA4 = 2752.6, so it is clear that the GA improves the Mean fitness. The Δ Mean fitness for Sig11 = 0.45748, and for SigGA4 = 0.45747. This means the deviation = 0.00001, where in Sig11, with $Pm = 0.2$ and $Fat = 0.1$, the Δ Mean fitness is higher.

After assessing 14 comparison cases of the results from the tables (30) and (49), The Mean fitness increases significantly, in the signatures' pool after applying the GA, more than with the standard VDC. In 7 cases, the Δ Mean Fitness dose not differ after getting the Fat and Pm from the GA optimization. In 6 cases the change is small in favor of the standard VDC algorithm, before performing the optimization using the GA. In one case only, the GA is better, not due to the using of GA, but because the *learning Gen* is different, where in Sig8 the *Learning Gen* = 150, and in SigGA4 the *Learning Gen* = 100.

Keeping in mind, that the values of Pm and Fat after the GA optimization are higher, which has led to increase the Mean Fitness values in the GA training process significantly. This differs from the suggested Fat and Pm in chapter 4, which are less.

When the Fat and Pm values are small, the Hypermutation rate decrease, while when their values are high, the Hypermutation rate increases, and this leads to increase the Mean Fitness in the training process and the matching process, but dose not improve the Δ Mean fitness in the matching phase.

The discussion above is summarized as following:

There have been 5 variables (*Learning Gen*, Pm , Fat , Training pool and matching pool) that are discussed in regard to their effects on the Mean fitness and Δ Mean Fitness, which are chosen because they can be measured.

- *Learning Gen*: when its value is higher, the Mean Fitness increases, in contrary to the Δ Mean fitness. When the *Learning Gen* value is less, the Δ Mean Fitness is higher in the matching phase.
- Training pool: it affects the Mean fitness, as when it increases, the Mean fitness increases. But the Δ Mean Fitness is not affected significantly.

- *Pm* and *Fat*: as they both control the Hypermutation process, their effect is combined together. When their values are small (as before using the GA in chapter four), the Hypermutation rate decreases, and thus the Mean fitness decreases. While when their values are high (as after using the GA in chapter five), the Hypermutation rate increases, and consequently the Mean fitness increases. But the increase in the Hypermutation rate does not increase the value of the Δ Mean Fitness.
- Matching pool: whenever the number of infected files increases inside the matching pool, the Mean fitness and the Δ Mean Fitness are higher. This variable reflects the actual reality and it can not be controlled nor interfered with.

The comparison of the Detection Speed is shown in Table (52). The table shows that the time consumed during the matching phase with the standard VDC algorithm is longer than with the GA. As when the matching pool =0%, the deviation between the standard VDC average time and the GA average time is 9244.839616 seconds. When the matching pool =5%, the deviation between the standard VDC average time and the GA average time is 574.582860 seconds. When the matching pool =25%, the deviation between the standard VDC average time and the GA average time is 3940.587127 seconds. When the matching pool =50%, the deviation between the standard VDC average time and the GA average time is 6018.846909 seconds. When the matching pool =75%, the deviation between the standard VDC average time and the GA average time is 2509.217777 seconds. And when the matching pool =100%, the deviation between the standard VDC average time and the GA average time is 0.228592 seconds.

The using of GA enhances the training process by improving the properties of the resulted signatures' pools, in regard to producing a higher Mean fitness for these signatures. So when the matching process is executed, the detection speed is better and faster. This is due to the fact that before applying the detection process in the matching, the signatures are sorted in descending order according to their fitness, which leads to having a faster detection. The experiments shown in Table (52) prove this result.

Knowing that, all runs for each matching pool are done on the same computer, for both the standard VDC and the GA together, in order to guarantee having the same properties and speed of the computer. For example, for the matching pool =0%, Sig1, Sig5, Sig8 and SigGA2 are tested on the same computer; Acer Laptop.

During the comparison between the tables (31) and (50), it is noticed that neither the detection rate nor the false positive change.

Consequently, the results have proven that using the GA as an optimizer for the VDC algorithm with the *Fat* and *Pm* as variables, improves the performance of the VDC algorithm, because the Mean fitness significantly increases.

This conclusion answers the third question of the dissertation questions.

The focus is on the detection process for its importance. If the detection finds any infected files, elimination can be done by deleting this file, or by using any of the suitable elimination methods.

Table 52: The Detection Speed summary

Matchi ng pool	Signat ure pool	Time in Second s	Avg Time	GA Signa ture pool	Time in Seconds	GA Avg Time	The Deviation (Avg Time – GA Avg Time)
0%	Sig1	14208.6 47688	22525.823 53	SigGA 2	13280.983 914	13280.983 914	9244.8396 16
	Sig5	35171.7 66852					
	Sig8	18197.0 56039					
5%	Sig6	9823.55 273	9508.9583 63	SigGA 4	8934.3755 03	8934.3755 03	574.58286 0
	Sig7	8345.79 6465					
	Sig10	8407.28 3349					
	Sig11	11459.2 00907					
25%	Sig1	10311.5 81502	10504.877 563	SigGA 1	6593.6054 29	6564.2904 36	3940.5871 27
	Sig2	10743.8 21194		SigGA 2	6534.9754 43		
	Sig12	10459.2 29993					
50%	Sig4	7103.76 6218	13387.121 866	SigGA 4	7368.2749 57	7368.2749 57	6018.8469 09
	Sig5	22787.7 65619					
	Sig8	10269.8 33760					
75%	Sig1	3165.08 4227	5655.0352 32	SigGA 1	3145.8174 55	3145.8174 55	2509.2177 77
	Sig3	8762.23 8266					
	Sig6	4431.98 4430					
	Sig9	8729.63 3644					
	Sig12	3186.23 5592					
100%	Sig2	6.36820 5	6.364439	SigGA 1	6.167821	6.135847	0.228592
	Sig4	6.37853 1		SigGA 2	6.093931		
	Sig7	6.35973 4		SigGA 3	6.143650		
	Sig10	6.38850 7		SigGA 4	6.137986		

	Sig11	6.47753 7					
	Sig12	6.21412 2					

Chapter Six

Conclusions and Future Work

The dissertation aims to develop an algorithm inspired from the AIS concepts to detect viruses. The new algorithm is called the VDC algorithm. And the needed viruses' signatures for the research are gained from the VX Heaven website.

The VDC algorithm is formed through three main steps; Cloning, Hypermutation and reselection stochastically. Within the step of reselection stochastically, the viruses' detection process exists, by doing the exact match between each of the viruses' signatures and the files.

A lot of experiments have been done to validate the algorithm through two phases: training and matching, where the needed variables are determined for each process.

The variables for the training phase are the *learning Gen*, *Fat*, *Pm*, and the training pool. This phase has produced the 12 signatures' pools, which have been tested within the second phase (the matching). The variable for the matching phase is the matching pool.

After that, the GA has been employed as an optimizer for the VDC algorithm through 3 processes; the optimization, which has produced 4 different values for the *Pm* and *Fat*. These values are used in the Training process to generate 4 signatures' pools, which are validated in the matching process.

The results of the VDC algorithm and the optimized VDC algorithm based on GA are discussed, in chapters four and five and recapitulated in the first section. Then a comparison with the Related Work is summarized in the second section. After that, the Limitations are described in the third section. Finally, the recommendations for future work are listed.

6.1. Conclusions

1. In the VDC algorithm, the following affects the fitness of the signatures by increasing it when they are increased: the number of generations, the number of the infected files inside the files' pool and the Hypermutation rate during the training phase.

2. Employing the GA to optimize the VDC algorithm, improves the Detection Speed of the VDC algorithm, by increasing the Mean fitness, which leads the algorithm to be faster in detecting viruses.
3. Regarding the average detection rate, it is 94.4% and the false positive is 0%. These rates are considered good, and they do not change with the use of the GA, on the contrary, they are confirmed.
4. The results of the dissertation clarify the ability of using the VDC algorithm to detect viruses.

6.2. Comparison with Related Works

This dissertation has agreed with the studies of Castro and Zuben (2000), Castro and Zuben (2002), Castro and Timmis (2002), Yang (2006) and Liu (2006) in regard to addressing the AIS subject in general.

Castro and Zuben (2002) suggested the CLONALG algorithm. This algorithm has been used in this research, but there has been a difference, as they employed the CLONALG in Machine Learning, Pattern Recognition and Optimization Problems, while here it has been used in Virus Detection.

The study of Castro and Timmis (2002) dealt with Clonal Selection Algorithm same as in this research, but their study dealt as well with the Negative Selection and the Immune Network in the pattern recognition.

Yang (2006) had the same method in applying AIS with the GA, but varied in using it in the Dynamic Environments.

The study of Liu et al (2006) was of the same opinion in employing the Clonal Selection, but they used three different methods for mutation, while this research has applied only one method.

This research has disagreed with the study of Castro and Zuben (2000) in regard to proposing the Immune Network Model.

This work has agreed with the studies of Kolter and Maloof (2006), Perda et al (2007) and Al-Daoud et al (2009) in addressing the issue of virus detection, but differed in the method used for detection. As for Kotler, he employed the machine

learning method with the detection rate of 98% and the false positive of 5%. Despite the fact that, Perda employed the Semantic based detectors, and Al-Daoud applied the ALCFG in detecting the metamorphic viruses, which were generated by NGVCK0.03 and VCL32.

This study agreed with the studies of Forrest (1994), Kephart (1994), Edge et al (2006), Unterleitner (2008) and Yu et al (2009) in concentrating on the AIS with virus detection, but deviated from them, in applying the Negative Selection Algorithm. This research employed the Clonal Selection Algorithm. Note that Yu et al (2009) had the detection rate of 97%, and the false positive of 3.6%, and also enclosed a list of detection rates for antivirus companies which were: Eset NOD32 = 94%, Kaspersky = 88%, Panda 2008 = 67%, KV 2008 = 55% and Kingsoft = 44%. Consequently, the results of this research (i.e. detection rate of 94.4% and the false positive of 0%) are considered good and acceptable.

6.3. Limitations

The main limitation that faced this work is getting virus signatures. The researcher contacted the antivirus companies, whose addresses were known to her. Unfortunately they refused to respond, except Eset NOD32 which apologized for not providing such information, because it was classified as confidential. After prolonged research and asking experts in the field, the researcher found the VX Heaven website, which provides such information to the public, for the purpose of providing help to research in this field.

The other limitation is the difficulties in reaching the results in chapter four, due to the large number of runs; (12) training runs, and (24) matching runs. Each run of the training phase has taken around 20 hours, and each run in the matching phase has taken approximately 2.85 hours. These are only the documented runs; the researcher conducted many experiments that exceeded the documented number by double.

Regarding chapter five, which is about the GA with the VDC algorithm, the researcher performed several experiments before starting the documentation. For example: when running the GA optimization process with a number of generations

equaling 100, it took 17 consecutive days without reaching any results. It should be in mind that when the number of generations was 5, it took 3 consecutive days. To get the results in chapter five, GA optimization was executed 4 times, and each run took around 5 continuous days. Then each run of the GA optimization with one training task, has taken around 20 hours. After that, the ten times matching of the resulted GA training, took around 1.82 hours. Knowing that, the researcher used 5 micro-computers with the best available properties that she was able to get. These computers are 2 Acer laptops (Intel® Core™ 2 Duo CPU T6400 @ 2.00GHz and 3GB RAM), an HP laptop (Intel® Core™ 2 Duo CPU T5600 @ 1.83GHz and 2.5GB RAM), an Acer PC (Intel® Core™ 2 Duo CPU E7300 @ 2.67GHz and 2GB RAM), and the fifth is a Magic PC (Intel® Core™ 2 Duo CPU E7300 @ 2.80GHz and 4GB RAM).

The researcher tried to access computers with better properties from several universities and the Royal Scientific Society, but unfortunately, she was denied access to these computers due to the presence of confidential private information for these institutions.

6.4. Future Work

After accomplishing this research, and getting the results, the researcher recommended the following:

1. In the beginning the VDC algorithm used the initial fitness of the signatures as random numbers. It is suggested that the Data Mining (the process of extracting patterns from data, and transforming this data into information) in categorizing the viruses according to their wide spread. To have their initial fitness depending on the prevalence of the virus be applied.
2. The VDC algorithm employed the exact match between signatures and files. It is recommended that different matching methods be applied. Such as Euclidean Distance, Manhattan Distance or Hamming Distance.
3. Adding the Negative Selection Algorithm to the VDC algorithm, so that it would be possible to distinguish between Self and Non-self in regard to the existing files and later the detected infected files.

4. The use of different methods of mutation; such as Gauss Mutation, Cauchy Mutation or Mean Mutation.
5. To apply the VDC algorithm to different types of malware.
6. To conduct studies on viruses' elimination with different suitable methods.

The References

- [Aickelin,2004] Aickelin, U. (2004). "**Artificial Immune Systems (AIS) – A New Paradigm for Heuristic Decision Making**", The University of Nottingham, Nottingham, NG8 1BB, United Kingdom.
- [Al-daoud,2009] Al daoud, E., Al-Shbail, A. and Al-Smadi, A. (2009). "**Detecting Metamorphic viruses by using Arbitrary Length of Control Flow Graphs and Nodes Alignment**". Special Issue on ICIT 2009 Conference - Bioinformatics and Image. Ubiquitous Computing and Communication Journal UbiCC Journal – Volume 4 No. 3.
- [Castro,1999] Castro,L. and Zuben, F.(1999). "**Artificial Immune Systems: Part I – Basic Theory And Applications**". Technical Report, TR – DCA 01/99.
- [Castro,2000a] Castro,L. and Zuben, F.(2000). "**An Evolutionary Immune Network for Data Clustering**". In Proceedings of the IEEE Computer Society Press, SBRN'00 (Brazilian Symposium on Neural Networks), vol. 1, pp. 84-89, Rio de Janeiro/RJ, 22-25.
- [Castro,2001] Castro, L. and Zuben, F. (2001). "**aiNet: An Artificial Immune Network for Data Analysis**". Idea Group Publishing, USA.
- [Castro,2002a] Castro,L. and Timmis, J. (2002). "**Artificial Immune Systems: A Novel Paradigm to Pattern Recognition**". SOCO-2002, University of Paisley, UK, pp. 67-84.
- [Castro,2002b] Castro, L. and Zuben, F. (2002). "**Learning and Optimization Using the Clonal Selection Principle**". IEEE, vol. 6, n. 3, pp. 239-251.

- [Chess,2000] Chess, D. M., and White, S. R. (2000). "**An Undetectable Computer Virus**". IBM Thomas J. Watson Research Center. New York, USA.
- [Cohen,1984] Cohen, F. (1984). "**Computer Viruses - Theory and Experiments**". Published in Computers and Security, Vol. 6, pp. 22-35.
- [Coppin,2004] Coppin, B. (2004). "**Artificial Intelligence Illuminated**". Jones and Bartlett Publishers.
- [D'haeseleer,1996] D'haeseleer, P., Forrest, S. and Helman, P. (1996). "**An Immunological Approach to Change Detection: Algorithms, Analysis and Implications**". IEEE Symposium on Security and Privacy.
- [Edge,2006] Edge, K., Lamont, G. and Raines, R. (2006) "**A Retrovirus Inspired Algorithm for Virus Detection & Optimization**". Wright-Patterson AFB, Dayton, OH USA 45433. GECCO'06, Washington, USA.
- [EPSRC,2008] EPSRC (2008). Engineering and Physical Sciences Research Council. available: <http://www.artificial-immune-systems.org/index.shtml> Accessed at 10/3/2008.
- [Forrest,1994] Forrest, S., Perelson, A., Allen, L. and Cherukuri, R. (1994). "**Self-Nonself Discrimination in a Computer**". In Proceedings of IEEE Symposium on Research in Security and Privacy.
- [Goldberg,1989] Goldberg, D. (1989). "**Genetic Algorithms in search, Optimization, and Machine Learning**". Addison-Wesley Publishing Company.

- [Garrett,2005] Garrett, S. (2005). "**How Do We Evaluate Artificial Immune Systems?**" University of Wales, Aberystwyth, Wales, SY23 3DB. UK. 145-178.
- [Greeger,2010] Creeger, M. (2010). "**The battle is bigger than most of us realize: CTO Roundtable: Malware Defense**". Article development led by queue.acm.org, April 2010 | vol. 53 | no. 4 | communications of the ACM.
- [Gregory,2004] Gregory, P. (2004). "**Computer Viruses for Dummies**". Wiley Publishing Inc. USA.
- [Hang,2005] Hang, X. and Dai, H. (2005). "**Applying both Positive and Negative Selection to Supervised Learning for Anomaly Detection**". GECCO'05, Washington, D.C., USA. ACM 1-59593-010-8/05/0006.
- [Harley,2001] Harley, D., Slade, R. and Gattiker, U. (2001) "**Virus Revealed**". McGraw-Hill companies. USA.
- [Kephart,1994a] Kephart, J. (1994). "**A Biologically Inspired Immune System for computers**". Published in Artificial Life IV, Proceedings of the Fourth International Workshop on Synthesis and Simulation of Living Systems, Rodney A. Brooks and Pattie Maes, eds., MIT Press, Cambridge, Massachusetts, pp. 130-139.
- [Kephart,1994b] Kephart, J., and Arnold, W. (1994). "**Automatic Extraction of Computer Virus Signatures**". In Proceedings of tech 4th Virus Bulletin International Conference, R. Ford, ed., Virus Bulletin Ltd., Abingdon, England, pp. 178-184.

- [Kephart,1994c] Kephart, J., Arnold, W., Chess, D., and White, S. (1994). **Automatic Immune System For Computers and Computer Networks**. International Business Machines Corporation, Armonk, N.Y. Appl.No.: 4,872.
- [Kephart,1997] Kephart, J., Sorkin, G., Swimmer, M. and White, S. (1997). **"Blueprint for a Computer Immune System"**. IBM Thomas J. Watson Research Center. This paper was originally presented at the Virus Bulletin International Conference in San Francisco, California, USA.
- [Kolter,2004] Kolter, J. Z., and Maloof, M. A. (2004). **"Learning to Detect Malicious Executables in the Wild"**. ACM SIGKDD, available: <http://doi.acm.org/10.1145/1014052.1014105>. Accessed at 25/2/2008.
- [Kolter,2006] Kolter, J. Z., and Maloof, M. A. (2006). **"Learning to Detect and Classify Malicious Executables in the Wild"**. Journal of Machine Learning Research 7.
- [Landesman,2008] Landesman, M. (2008). **"What is a Virus Signature? "**. available: <http://antivirus.about.com/od/whatisavirus/a/virussignature.htm>. Accessed at 30/12/2008.
- [Liu,2006] Liu, R., Chen, L., and Wang, S. (2006). **"Immune Clonal Strategies Based on Three Mutation Methods"**. (Eds.): ICNC 2006, Part II, LNCS 4222, pp. 114 – 121. Springer-Verlag Berlin Heidelberg.
- [Mitchell,1996] Mitchell, M. (1996). **"An Introduction to Genetic Algorithms"**. A Bradford Book, The MIT Press: England.

- [Okamoto,1999a] Okamoto, T. & Ishida, Y. (1999), "**A Distributed Approach to Computer Virus Detection and Neutralization by Autonomous and Heterogeneous Agents**", In Proc of the ISADS'99, pp. 328-331.
- [Okamoto,1999b] Okamoto, T. & Ishida, Y. (1999), "**Multiagent Approach Against Computer Virus: An Immunity- Based System**", In Proc. of the AROB'99, pp. 69-72.
- [Pietzowski,2006] Pietzowski, A., Trumler, W. and Ungerer, T. (2006). "**An Artificial Immune System and its Integration into an Organic Middleware for Self-Protection**". GECCO'06, Seattle, Washington, USA. ACM 1-59593-186-4/06/0007.
- [Perda,2007] Preda, M. D., Christodorescu, M., Jha, S., and Debray, S. (2007). "**A Semantics-Based Approach to Malware Detection**". POPL'07, Nice, France. ACM.
- [Schmauder,2000] Schmauder, P.(2000). "**Virus Proof**". Prima Publishing, Jawsa media group. USA.
- [Secure, 2008] Secure Computing Corporation. (2008). "**Virus Signature Solutions from Secure Computing**". Accessed at 25/12/2008.
- [Stalling, 2007] Stalling, W. (2007). "**Network Security Essentials: Applications and Standards**". Third Edition. Person Education, Inc. USA.

- [Szor, 2005] Szor, P. (2005). **"The Art of Computer Virus Research and Defense"**. Addison Wesley Professional. USA.
- [TopTen, 2008] TopTenReviews. (2008). **"2008 Anti-virus Software Report"**. Available: <http://anti-virus-software-review.toptenreviews.com/> Accessed at 25/2/2008.
- [Unterleitner,2008] Unterleitner, M. (2008). **"Computer Immune System for Intrusion and Virus Detection: Adaptive Detection Mechanisms and their Implementation"**. VMD Verlag Dr Muller Aktiengesellschaft & Co. Germany.
- [VXHeaven,2010] VX Heaven (2010). Available: <http://vx.netlux.org/>. Accessed at 1/1/2010.
- [Wikipedia, 2010] Wikimedia Foundation (2010, Jun) Neural network, Wikipedia, the free encyclopedia (online), available: http://en.wikipedia.org/wiki/Main_Page Accessed at 26/6/2010.
- [Yang, 2006] Yang, S. (2006). **"A Comparative Study of Immune System Based Genetic Algorithms in Dynamic Environments"**. GECCO'06, Seattle, Washington, USA.

- [Yu, 2009] Yu, Z., Tao, L. and Renchao, Q. (2009). "**Unknown Computer Virus Detection Inspired by Immunity**". ISSN 1673-9418 CODEN JKYTA8, Journal of Frontiers of Computer Science and Technology.

Open Source References

- [Castro,2000b] Castro, L., and Zuben, F. (2000a). CLONALG. MATLAB® Artificial Immune Systems. Available: <ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/lnunes/demo.zip> Accessed at 12/03/2008.
- [Kelsey,2003] Kelsey, J. (2003) B-cell algorithm (BCA) C++ code: A clonal selection optimization algorithm. Available: <http://www.artificial-immune-systems.org/algorithms.shtml#clonal-alg> Accessed at 12/03/2008.
- [Watkins,2002] Watkins, A. (2002). AIRS: Artificial Immune Recognition System. C++ source code. Available: <http://www.artificial-immune-systems.org/algorithms.shtml#clonal-alg> Accessed at 12/03/2008.

Appendixes

Appendix A:

The First Training table results

Table A.1: The first Training table results

Iteration number	No of signatures	Selection threshold	Mean fitness	Best fitness
1	100	0.90	160.14	210.00
2	111	0.90	170.11	235.00
3	122	1.00	176.39	236.00
4	131	0.60	181.64	236.00
5	142	0.70	186.43	236.00
6	153	0.90	190.87	236.00
7	164	0.80	194.13	236.00
8	175	0.60	197.20	236.00
9	186	0.80	198.66	236.00
10	197	0.70	200.86	236.00
11	208	1.00	202.53	236.00
12	219	1.00	204.15	236.00
13	230	0.90	205.33	236.00
14	241	0.70	207.20	237.00
15	252	1.00	208.47	237.00
16	263	0.60	209.96	237.00
17	274	0.60	211.41	238.00
18	285	0.60	212.87	239.00
19	296	0.70	214.47	240.00
20	307	0.70	216.19	240.00
21	318	0.60	217.55	240.00
22	329	0.70	218.98	240.00
23	340	0.70	220.46	240.00
24	351	0.80	221.71	241.00
25	362	0.80	223.09	241.00
26	373	0.90	224.48	242.00
27	384	0.70	225.69	242.00
28	395	0.90	226.88	242.00
29	406	0.90	227.93	242.00
30	417	0.80	229.03	242.00
31	428	0.80	230.01	242.00
32	439	0.70	231.07	243.00
33	450	0.80	231.98	243.00
34	461	0.70	232.91	243.00
35	472	0.80	233.68	243.00
36	483	0.80	234.53	243.00
37	494	0.70	235.26	244.00
38	505	1.00	236.04	244.00
39	516	0.80	236.55	244.00

40	527	0.60	237.03	244.00
41	538	1.00	237.49	245.00
42	549	0.80	237.95	245.00
43	560	0.70	238.36	245.00
44	571	0.80	238.77	245.00
45	582	0.60	239.14	245.00
46	593	0.70	239.53	245.00
47	604	0.90	239.80	245.00
48	615	0.70	240.02	245.00
49	626	0.60	240.23	245.00
50	637	0.80	240.45	246.00
51	648	0.70	240.64	246.00
52	659	0.80	240.81	246.00
53	670	0.60	240.99	246.00
54	681	1.00	241.18	246.00
55	692	0.80	241.35	246.00
56	703	0.80	241.53	247.00
57	714	0.70	241.67	247.00
58	725	1.00	241.83	247.00
59	736	0.70	241.97	247.00
60	747	0.70	242.12	247.00
61	758	0.70	242.26	247.00
62	769	0.60	242.41	247.00
63	780	0.60	242.53	247.00
64	791	1.00	242.68	247.00
65	802	0.70	242.81	247.00
66	813	0.80	242.95	248.00
67	824	0.60	243.09	248.00
68	835	0.60	243.23	249.00
69	846	0.60	243.36	249.00
70	857	0.70	243.51	249.00
71	868	0.80	243.63	249.00
72	879	0.90	243.75	249.00
73	890	0.80	243.88	249.00
74	901	0.90	244.01	249.00
75	912	0.70	244.13	249.00
76	923	1.00	244.26	249.00
77	934	0.80	244.37	249.00
78	945	0.60	244.50	250.00
79	956	1.00	244.64	250.00
80	967	0.70	244.76	250.00
81	978	0.80	244.88	250.00
82	989	0.60	244.99	251.00
83	1000	1.00	245.11	251.00
84	1011	0.80	245.23	251.00
85	1022	0.70	245.34	252.00
86	1033	1.00	245.48	252.00
87	1044	0.60	245.59	252.00

88	1055	0.60	245.69	252.00
89	1066	0.90	245.79	252.00
90	1077	0.90	245.90	252.00
91	1088	0.90	246.02	252.00
92	1099	0.60	246.14	252.00
93	1110	1.00	246.26	252.00
94	1121	0.90	246.39	252.00
95	1132	0.90	246.49	253.00
96	1143	0.90	246.61	253.00
97	1154	1.00	246.72	253.00
98	1165	0.70	246.82	253.00
99	1176	0.90	246.92	253.00
100	1187	0.70	247.03	253.00
101	1198	1.00	247.15	253.00

Appendix B: Parts of the Matching Results

B1: The matching of Sig7 with 5% infected files

The matching of **Sig7** with the files pool with 5% infected files. At iteration number 5 new 100 files are added to the files' pool, with 5% of these files are infected. The results are shown in Table (B.1).

Table AAA.1: The results of the matching of Sig7 with 5% infected files

Iteration number	Mean fitness	Best fitness
1	924.58382	1042.00
2	924.60363	1042.00
3	924.60363	1042.00
4	924.60363	1042.00
5	924.60363	1042.00
6	924.60694	1042.00
...
97	924.60694	1042.00
98	924.60694	1042.00
99	924.60694	1042.00
100	924.60694	1042.00
101	924.60694	1042.00

As a result, the number of infected files = 28 and the Mean fitness = 924.6069.

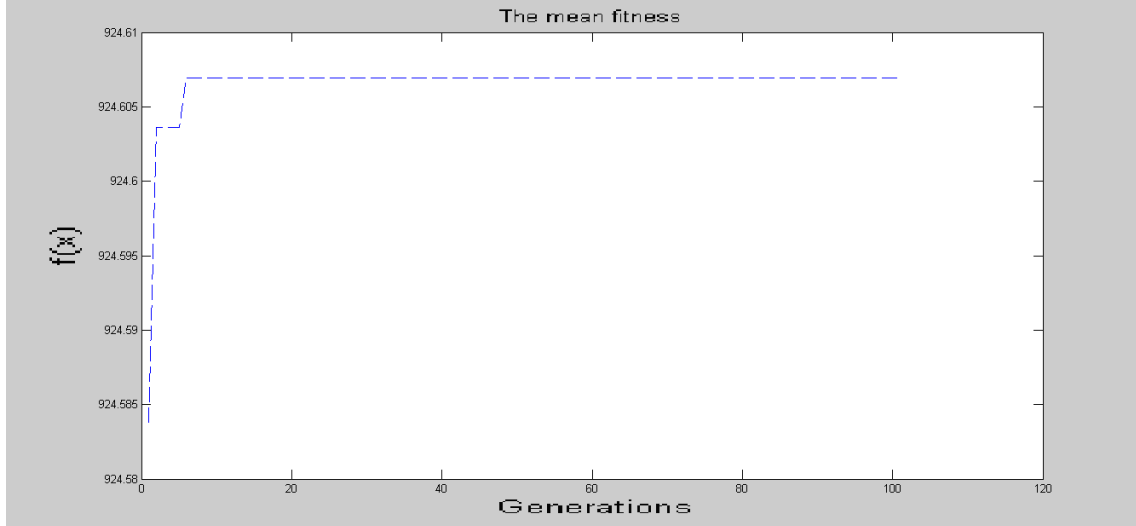


Figure i: The Mean fitness of matching Sig7 run with 5% infected files

Table (B.1) and Figure (i) show that the Mean fitness increases by 0.01981 in the first iteration whereas it increases in the iteration number 5 by 0.00331.

The Best fitness does not change; this explains the straight line in Figure (ii). The number of detected files is 28 out of 30 (25+5) with a Detection rate of 93.3%, where the 25 infected files are in the original pool, and the 5 are from the new added pool. As a result this detection rate is accepted.

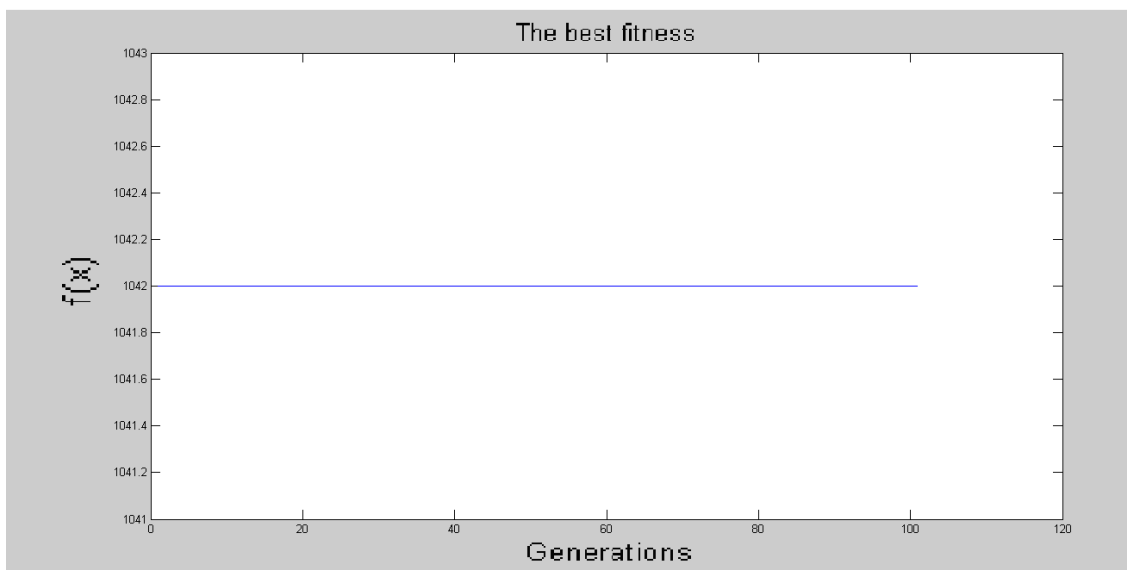


Figure ii: The Best fitness of matching Sig7 run with 5% infected files

B2: The matching of Sig2 with 25% infected files

The matching of **Sig2** with the files pool with 25% infected files. At iteration number 50 new 100 files are added to the files' pool, with 25% of these files are infected. The results are shown in Table (B.2).

Table B.2: The results of the matching of Sig2 with 25% infected files

Iteration number	Mean fitness	Best fitness
1	223.59950	251.00
2	223.70190	263.00
3	223.70190	263.00
4	223.70190	263.00
5	223.70190	263.00
...
50	223.70190	263.00
51	223.71676	264.00
...
98	223.71676	264.00
99	223.71676	264.00
100	223.71676	264.00
101	223.71676	264.00

As a result, the number of infected files= 142 and the Mean fitness = 223.7168.

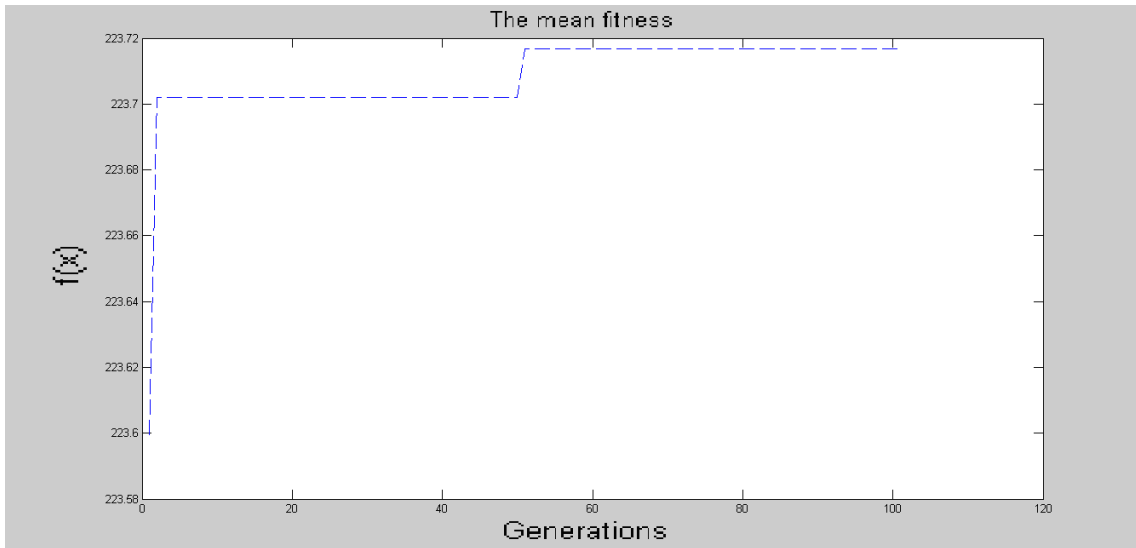


Figure iii: The Mean fitness of matching Sig2 run with 25% infected files

Table (B.2) and Figure (iii) show that the Mean fitness increases by 0.10240 in the first iteration whereas it increases in the iteration number 50 by 0.01486.

The Best fitness increases in the first iteration by 12 whereas it increases at iteration number 50 by 1 as illustrated in Figure (iv). The number of detected files is 142 out of 150 (125+25) with a Detection rate of 94.7%, where the 125 infected files are in the original pool, and the 25 are from the new added pool. As a result this detection rate is accepted.

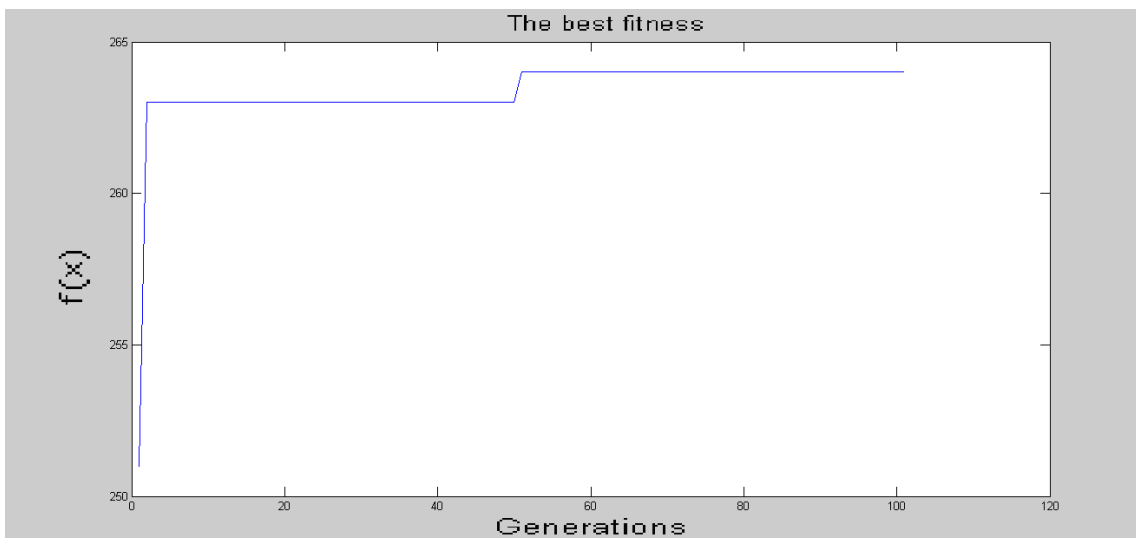


Figure iv: The Best fitness of matching Sig2 run with 25% infected files

B3: The matching of Sig12 with 25% infected files

The matching of **Sig12** with the files pool with 25% infected files. At iteration number 50 new 100 files are added to the files' pool, with 25% of these files are infected. The results are shown in Table (B.3).

Table B.3: The results of the matching of Sig12 with 25% infected files

Iteration number	Mean fitness	Best fitness
1	225.28241	252.00
2	225.38481	263.00
3	225.38481	263.00
4	225.38481	263.00
5	225.38481	263.00
...
50	225.38481	263.00
51	225.39967	264.00
...
98	225.39967	264.00
99	225.39967	264.00
100	225.39967	264.00
101	225.39967	264.00

As a result, the number of infected files= 142 and the Mean fitness = 225.3997.

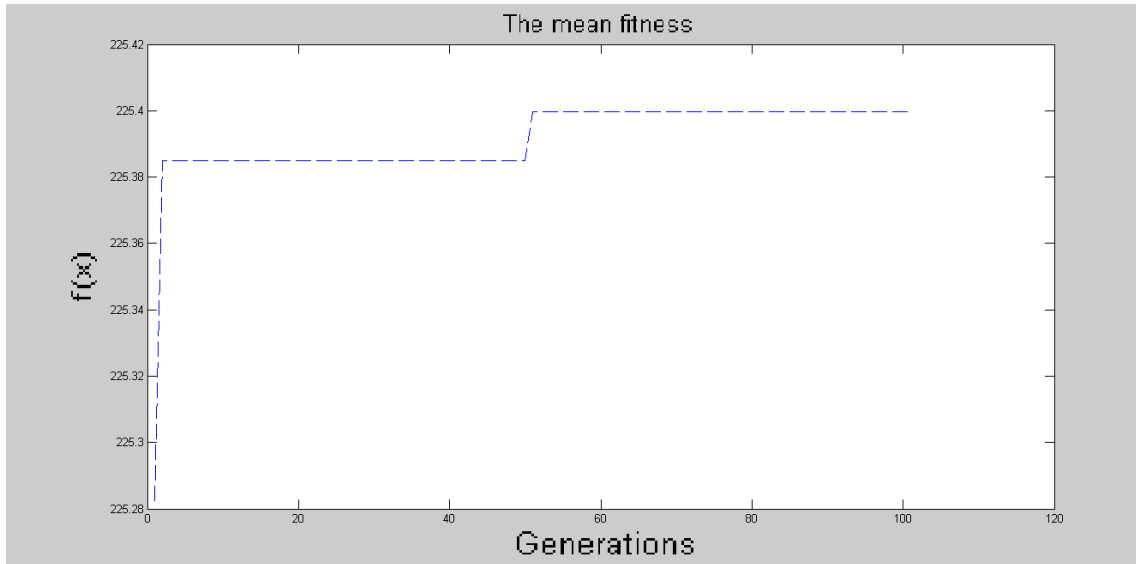


Figure v: The Mean fitness of matching Sig12 run with 25% infected files

Table (B.3) and Figure (v) show that the Mean fitness increases by 0.10240 in the first iteration whereas it increases in the iteration number 50 by 0.01486.

The Best fitness increases in the first iteration by 11 whereas it increases at iteration number 50 by 1 as illustrated in Figure (vi). The number of detected files is 142 out of 150 (125+25) with a Detection rate of 94.7%, where the 125 infected files are in the original pool, and the 25 are from the new added pool. As a result this detection rate is accepted.

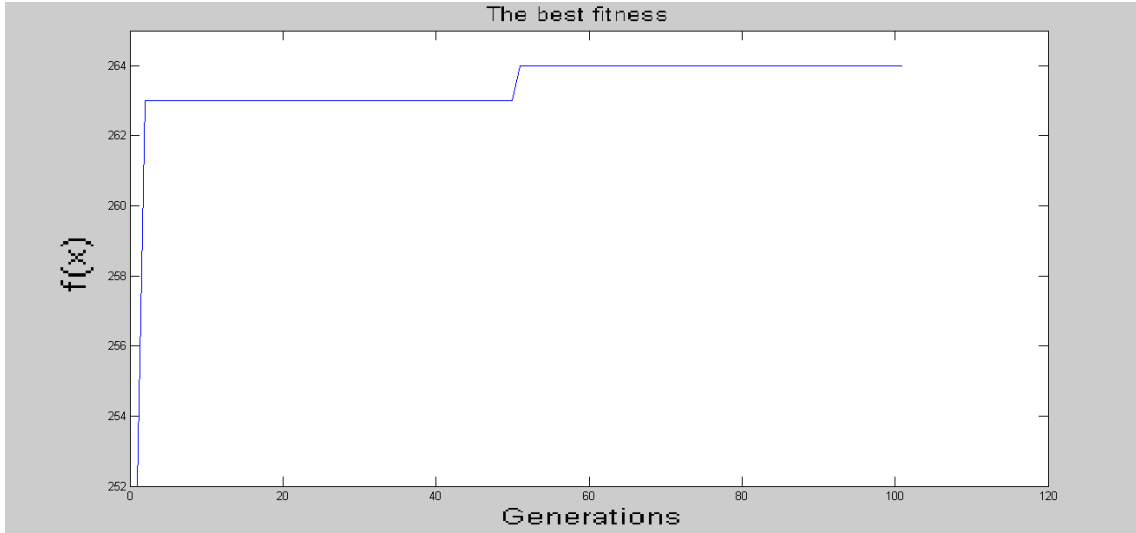


Figure vi: The Best fitness of matching Sig12 run with 25% infected files

B4: The matching of Sig5 with 50% infected files

The matching of **Sig5** with the files pool with 50% infected files. At iteration number 50 new 100 files are added to the files' pool, with 50% of these files are infected. The results are shown in Table (B.4).

Table B.4: The results of the matching of Sig5 with 50% infected files

Iteration number	Mean fitness	Best fitness
1	485.31662	552.00
2	485.38991	552.00
3	485.38991	552.00
4	485.38991	552.00
5	485.38991	552.00
...
50	485.38991	552.00
51	485.40106	552.00
...
98	485.40106	552.00
99	485.40106	552.00
100	485.40106	552.00
101	485.40106	552.00

As a result, the number of infected files= 288 and the Mean fitness = 485.4011.

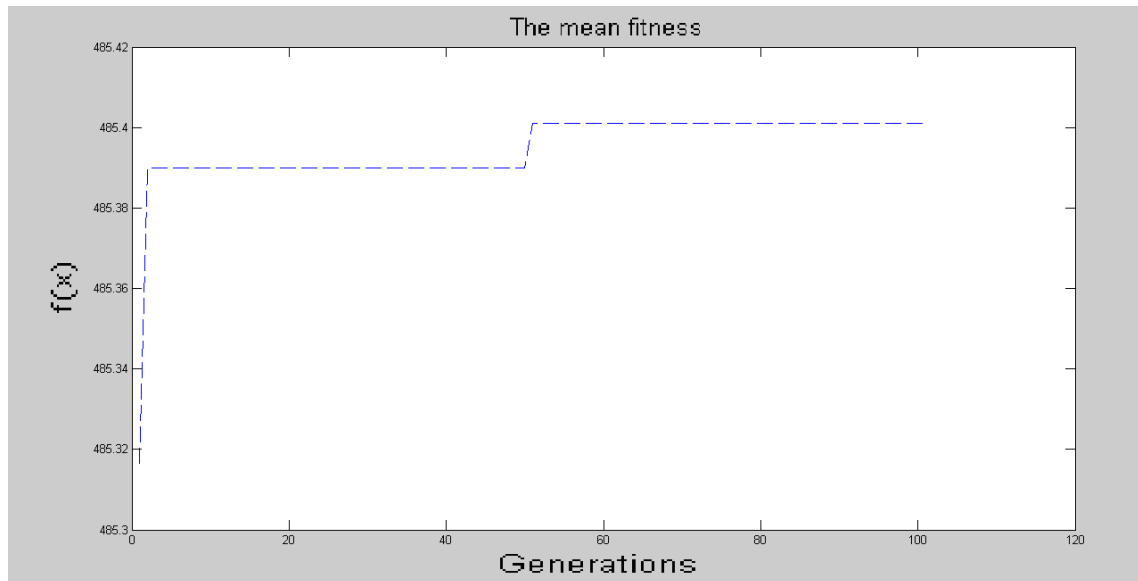


Figure vii: The Mean fitness of matching Sig5 run with 50% infected files

Table (B.4) and Figure (vii) show that the Mean fitness increases by 0.07329 in the first iteration whereas it increases in the iteration number 50 by 0.01115.

The Best fitness does not change; this explains the straight line in Figure (viii). The number of detected files is 288 out of 300 (250+50) with a Detection rate of 96%, where the 250 infected files are in the original pool, and the 50 are from the new added pool. As a result this detection rate is accepted.

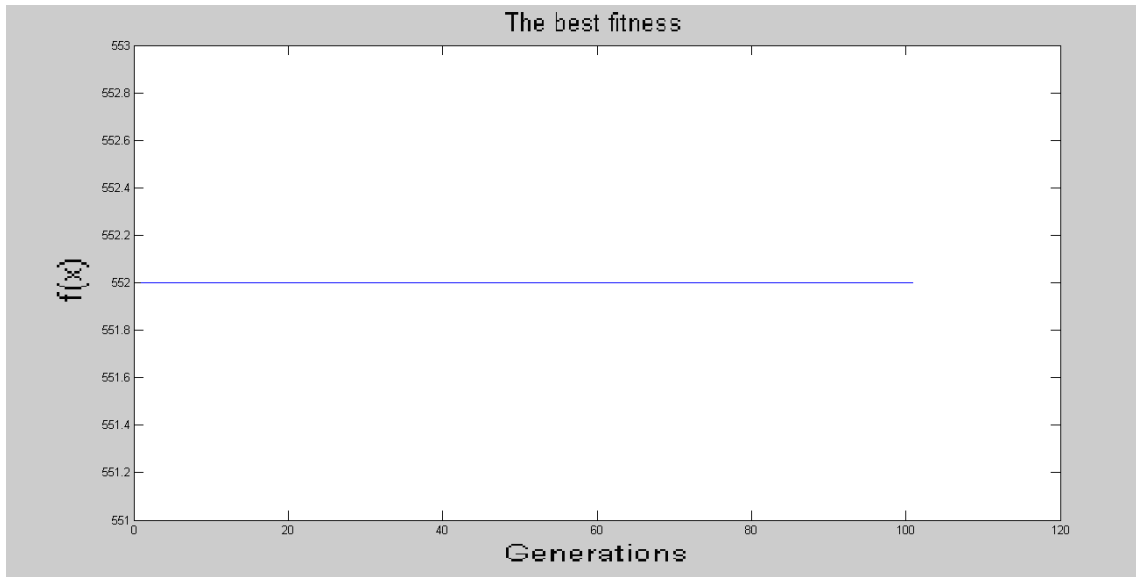


Figure viii: The Best fitness of matching Sig5 run with 50% infected files

B5: The matching of Sig12 with 75% infected files

The matching of **Sig12** with the files pool with 75% infected files. At iteration number 5 new 100 files are added to the files' pool, with 75% of these files are infected. The results are shown in Table (B.5).

Table B.5: The results of the matching of Sig12 with 75% infected files

Iteration number	Mean fitness	Best fitness
1	225.28241	252.00
2	225.58629	284.00
3	225.58629	284.00
4	225.58629	284.00
5	225.58629	284.00
6	225.61767	290.00
...
97	225.61767	290.00
98	225.61767	290.00
99	225.61767	290.00
100	225.61767	290.00
101	225.61767	290.00

As a result, the number of infected files = 406 and the Mean fitness = 225.6177.

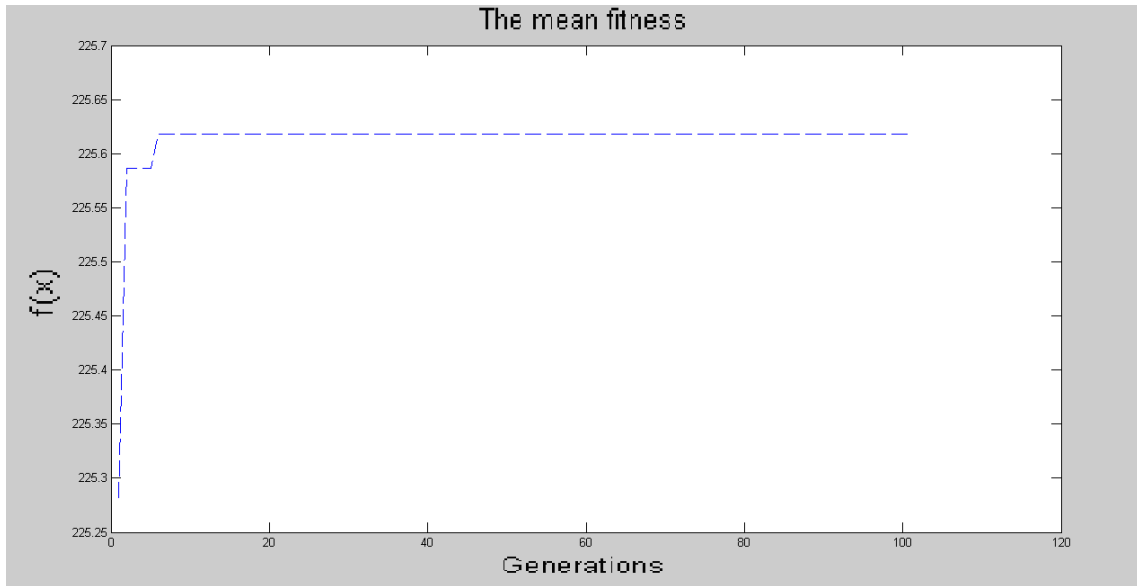


Figure ix: The Mean fitness of matching Sig12 run with 75% infected files

Table (B.5) and Figure (ix) show that the Mean fitness increases by 0.30388 in the first iteration whereas it increases in the iteration number 5 by 0.03138.

The Best fitness increases in the first iteration by 32 whereas it increases at iteration number 5 by 6 as illustrated in Figure (x). The number of detected files is 406 out of 450 (375+75) with a Detection rate of 90.2%, where the 375 infected files are in the original pool, and the 75 are from the new added pool. As a result this detection rate is accepted.

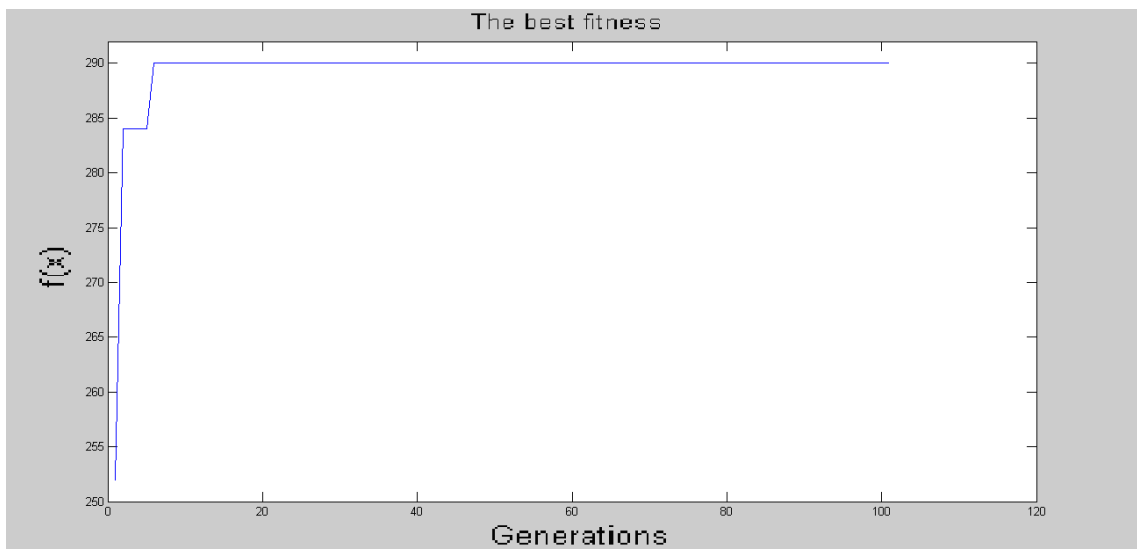


Figure x: The Best fitness of matching Sig12 run with 75% infected files

B6: The matching of Sig12 on 100% infected files

The matching of **Sig12** with the files pool with 100% infected files. At iteration number 50 new 100 files are added to the files' pool, with 100% of these files infected. The results are shown in Table (B.6).

Table B.6: The results of the matching of Sig12 with 100% infected files

Iteration number	Mean fitness	Best fitness
1	225.28241	252.00
2	225.69199	305.00
3	225.69199	305.00
4	225.69199	305.00
5	225.69199	305.00
...
50	225.69199	305.00
51	225.73988	312.00
...
98	225.73988	312.00
99	225.73988	312.00
100	225.73988	312.00
101	225.73988	312.00

As a result, the number of infected files = 554 and the Mean fitness = 225.7399.

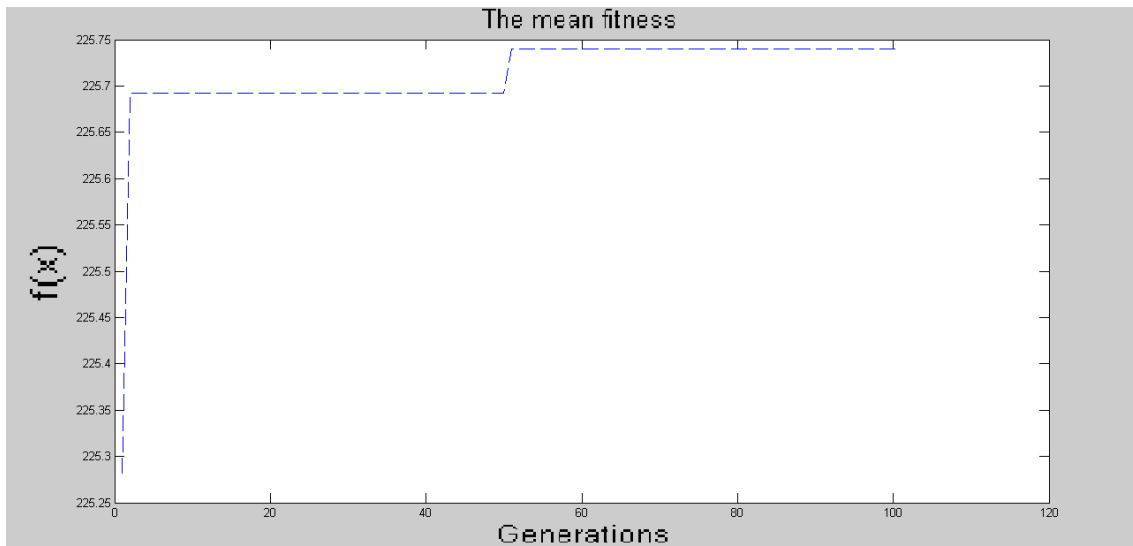


Figure xi: The Mean fitness of matching Sig12 run with 100% infected files

Table (B.6) and Figure (xi) show that the Mean fitness increases by 0.40958 in the first iteration whereas it increases in the iteration number 50 by 0.04789. The Best fitness increases in the first iteration by 53 whereas it increases at iteration number 50 by 7 as illustrated in Figure (xii). The number of detected files is 554 out of 600 (500+100) with a Detection rate of 92.3%, where the 500 infected files are in the original pool, and the 100 are from the new added pool. As a result this detection rate is accepted.

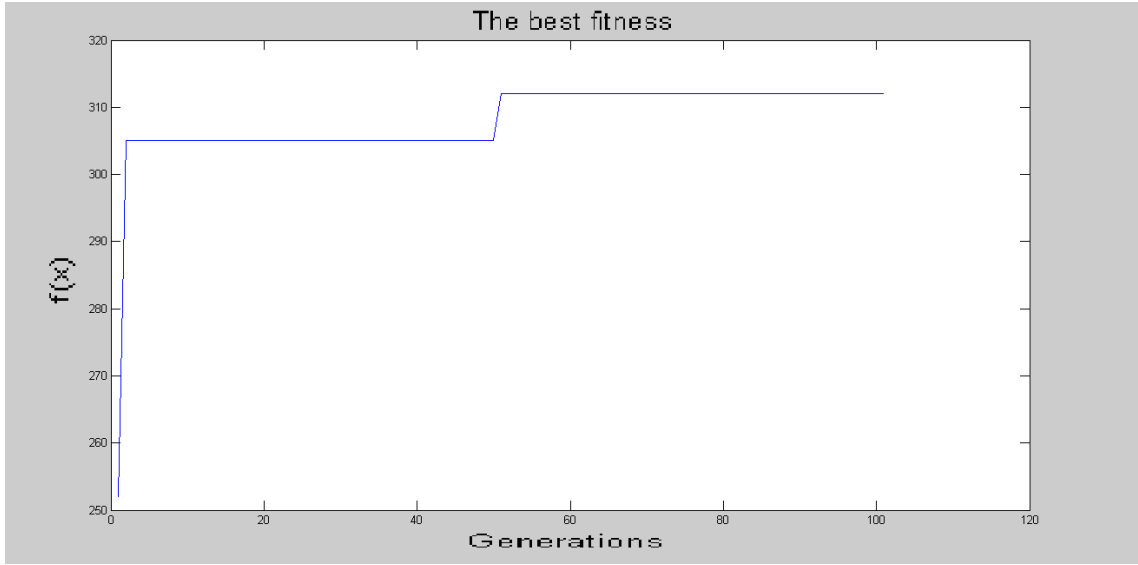


Figure xii: The Best fitness of matching Sig12 run with 100% infected files

Appendix C: Parts of the GA Matching Results

C1: The GA matching of SigGA2 with 25% infected files

SigGA2 is tested with the files pool with 25% infected files. At iteration number 50 new 100 files are added to the files' pool, with 25% of these files infected. The results are shown in Table (C.1).

Table C.1: The results of the matching of SigGA2 with 25% infected files

Iteration number	Mean fitness	Best fitness
1	2329.15277	2623.00
2	2329.25516	2623.00
3	2329.25516	2623.00
4	2329.25516	2623.00
5	2329.25516	2623.00
...
50	2329.25516	2623.00
51	2329.27002	2623.00
...
98	2329.27002	2623.00
99	2329.27002	2623.00
100	2329.27002	2623.00
101	2329.27002	2623.00

As a result, the number of infected files=142 and the Mean fitness= 2329.3.

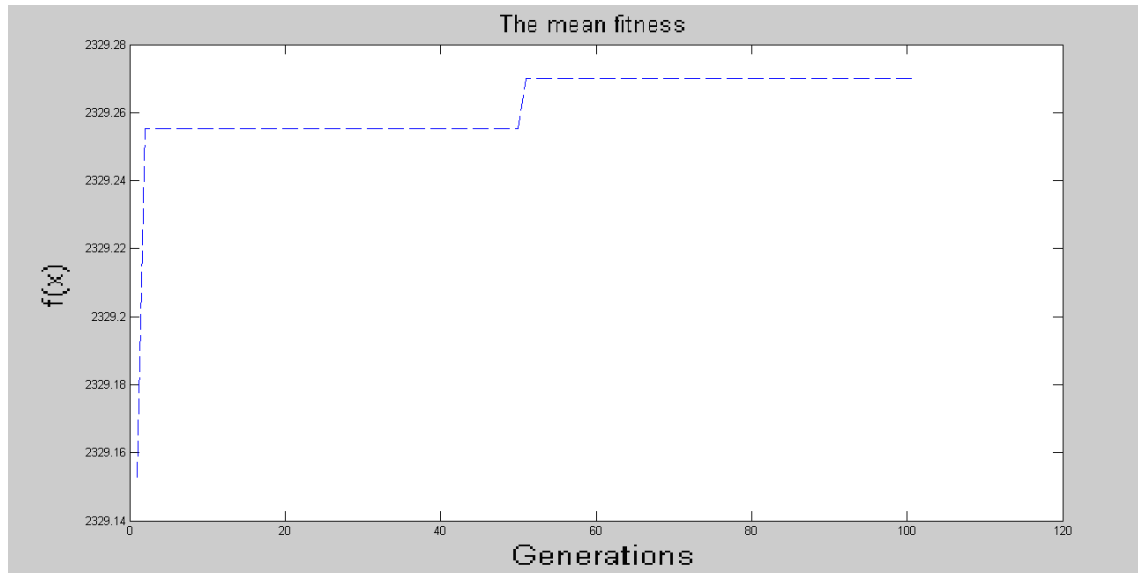


Figure xiii: The Mean fitness of matching SigGA2 run with 25% infected files

Table (C.1) and Figure (xiii) show that the Mean fitness increases by 0.10239 in the first iteration whereas it increases in the iteration number 50 by 0.01486.

The Best fitness does not change; this explains the straight line in Figure (xiv).

The number of detected files is 142 out of 150 (125+25) with a Detection rate of 94.7%, where the 125 infected files are in the original pool, and the 25 are from the new added pool. As a result this detection rate is accepted.

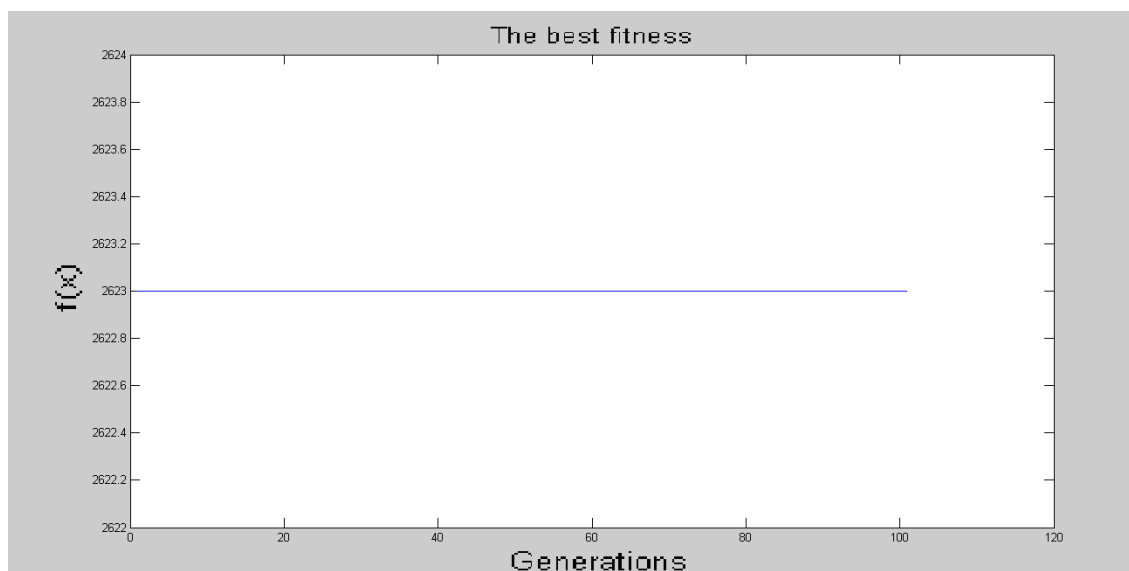


Figure xiv: The Best fitness of matching SigGA2 run with 25% infected files

C2: The GA matching of SigGA3 with 100% infected files

SigGA3 is tested with the files pool with 100% infected files. At iteration number 50 new 100 files are added to the files' pool, with 100% of these files infected. The results are shown in Table (C.2).

Table C.2: The results of the matching of SigGA3 with 100% infected files

Iteration number	Mean fitness	Best fitness
1	1078.03799	1233.00
2	1078.44756	1233.00
3	1078.44756	1233.00
4	1078.44756	1233.00
5	1078.44756	1233.00
...
50	1078.44756	1233.00
51	1078.49546	1233.00
...
98	1078.49546	1233.00
99	1078.49546	1233.00
100	1078.49546	1233.00
101	1078.49546	1233.00

As a result, the number of infected files=554 and the Mean fitness= 1078.5.

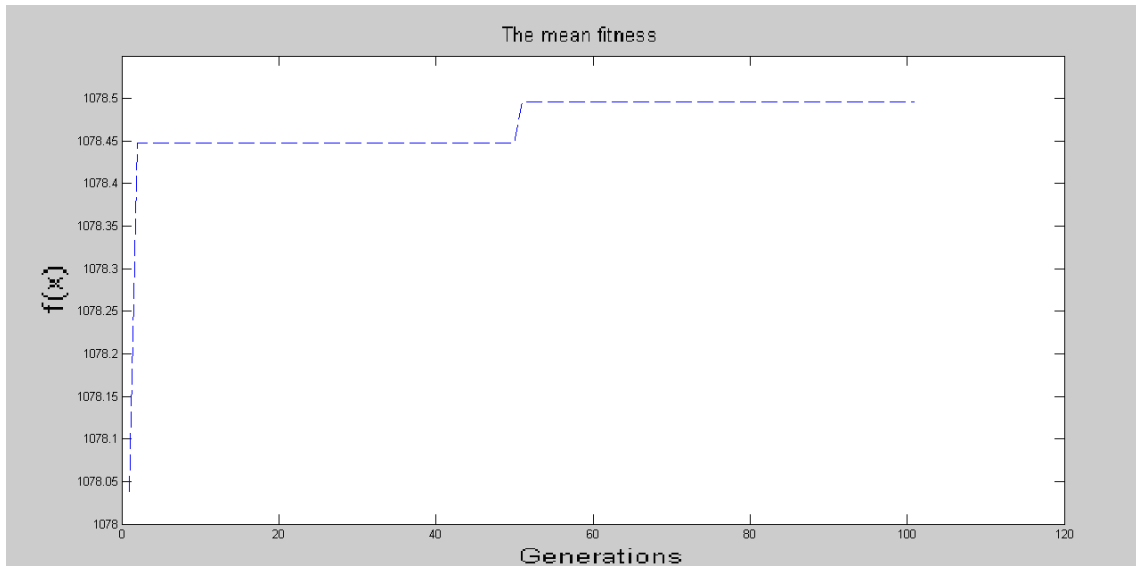


Figure xv: The Mean fitness of matching SigGA3 run with 100% infected files

Table (C.2) and Figure (xv) show that the Mean fitness increases by 0.40957 in the first iteration whereas it increases in the iteration number 50 by 0.04790. The Best fitness does not change; this explains the straight line in Figure (xvi). The number of detected files is 554 out of 600 (500+100) with a Detection rate of 92.3%, where the 500 infected files are in the original pool, and the 100 are from the new added pool. Hence this detection rate is accepted.

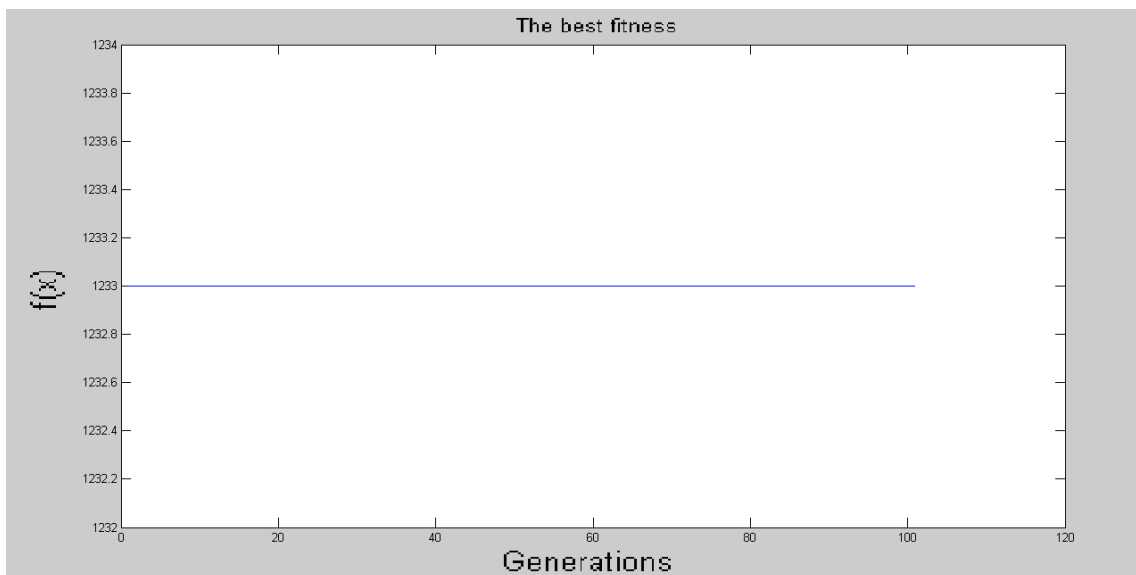


Figure xvi: The Best fitness of matching SigGA3 run with 100% infected files

Appendix D: Malware Types

There are several types of malware according to Stalling's categorization, with the addition of the spyware and the adware as they are classified as other types of malware without a harmful intent:

- **Virus:** a computer program written by a person that attaches itself to a program, and propagates copies of itself to other programs, and infect any computer without the permission or knowledge of the user. The virus can spread when its host is taken to the target computer either by being sent over a network or the Internet, or carried on a removable medium such as a floppy disk, CD, DVD, or USB drive.
- **Worm:** is a self-contained program (or set of programs) that spread from computer to computer, wreaking havoc on everything they touch. Unlike viruses, worms do not need to attach themselves to a host program. They are stand alone application that have their own population system, and spread fast through the internet by scanning for computers that have vulnerabilities. A worm usually executes itself automatically on a remote machine without any extra help from a user, but some worms such as mailer or mass-mailer worms may need the help of a user to be executed. Another difference between worms and viruses is that worms always cause harm to the network, if only by consuming bandwidth, whereas viruses almost always corrupt or modify files on a targeted computer.
- **Trojan horse:** A program that contains unexpected additional functionality, that appears to the users to interest and entice them to run the program to perform a desirable function, but in fact it facilitates unauthorized access to the computer system. Trojan horses are different from worms and viruses as they are not self-replicating, and they require interaction with a hacker to fulfill their purpose. In other cases, hackers leave behind Trojanized versions of real tools to camouflage their activities on a computer, so they can retrace their steps and perform malicious activities to the compromised system later.

- **Logic bomb:** is a routine or set of routines that are activated when a particular set of conditions occur to trigger the activation of these routines. logic bomb may be a component of a virus or Trojan
- **Backdoor (trapdoor):** a devious little program that allows secret access without user permission. It may take the form of an installed program, or a modification to an existing program or hardware device. The backdoor virus enters a system secretly by bypassing normal authentication, to let other users across the internet have unrestricted remote access to the infected system. The backdoor program uses two sets of files to perform its processing: files that reside on the infected PC (which becomes the server PC for remote access) and files that reside on the client PCs that accesses the system from across the net.
- **Exploits:** a hacker can construct specially coded messages that exploit one of the vulnerabilities that may be discovered in the programs that receive messages over the network. Depending on the actual use of the exploit code, the exploitation may be malicious as the hacker takes the control of a victim computer, and can cause this computer to do anything the hacker wants to do, even send these coded messages to other computers. The severity of the threat depends on the intention of the hacker. White hat hackers create a form of exploit code for penetration "pen" testing. The goal of the exploit code is to run a program on a remote, networked system automatically, or provide other forms of more highly privileged access to the target system.
- **Downloaders:** a malicious program that installs other items on a machine that is under attack. Usually, a downloader is sent in e-mails, and when it is executed (sometimes aided with the help of an exploit), it downloads malicious content from a Web site and then extracts and runs its content.

- **Auto-rooter:** is a malicious hacker tools that use a collection of exploits to break into a specified target machines remotely, to gain root on the machine. As a result the hacker gains administrative privileges to the machine.
- **Kit (virus generator):** a set of tools used to generate new viruses automatically. Virus writers developed kits, as the Virus Creation Laboratory (VCL) or PSMPC generators which are menu-based applications. With such tools, even novice users are able to develop harmful computer viruses without much background knowledge.
- **Spammer programs:** are used to send unwanted messages to Instant Messaging groups, newsgroups, or any kind of mobile device in forms of e-mail or cell phone SMS. The primary motivation of spammers is to make money by generating traffic to Web sites, and usually spam messages are used to implement phishing attacks.
- **Flooders:** are used by hackers to attack networked computer systems with a large volume of traffic to carry out a Denial of Service (DoS) attack.
- **Key-loggers:** is a small program that captures and records keystrokes and mouse movement on a compromised system, in an attempt to learn the bank account numbers, credit card numbers, and other sensitive information that the user does not want strangers to know about. For that matter, many viruses attempt to install key loggers on the user computer.
- **Rootkit:** a set of tools used by hackers after breaking into a computer system with exploits, and gaining root-level access by installing modified versions of common tools or trojanized system applications. Such programs can include monitoring utilities and system processes gimmicked so that they do not draw attention to illegitimate processes. They can also include utilities that are modified to enable the hacker to escalate account privileges, or to hide other component files.

- **Spyware:** spyware programs are secretly installed on the computers, to collect information about the users without their knowledge. These programs can collect various types of personal information, such as visited sites and internet surfing habits, and can also interfere with the user's control of the computer such as installing additional software and redirecting Web browser activity. They can change the computer's settings as well, in addition to secretly monitoring the user's computing.
- **Adware:** advertising-supported software is any software package which automatically plays, displays, or downloads advertisements to a computer after installing the software on it, or while the application is being used. Some types of adware are also spyware [Schmauder, 2000, Harley, 2001, Gregory, 2004, and Stalling, 2007] .

Appendix E

Antivirus Software Generations

According to Stalling (2007) there are four generations of antivirus software, and Szor (2005) added on them:

First generation: simple scanners

There are two types of simple scanners: a scanner that requires a virus signature to identify a virus which has essentially the same structure and bit pattern in all copies. Such scanners are limited to the detection of known viruses. The other type maintains a record of the length of programs, and looked for changes in length. Szor described antivirus scanners as simple programs that look for sequences of bytes extracted from computer viruses in files and in memory to detect them. This is, of course, one of the most popular methods to detect computer viruses, and it is reasonably effective. Nowadays, state-of-the-art antivirus software uses a lot more smart features to detect complex viruses, which cannot be handled using first-generation scanners alone. And he described many techniques of virus detection as first generation. This research mentions the following: the string scanning, Wildcards, Mismatches, Hashing, Bookmarks and Top-and-tail scanning; String scanning technique is the simplest approach to detect computer viruses. It uses an extracted sequence of bytes (strings) that is typical of the virus but not likely to be found in clean programs. The sequences extracted from the computer viruses are then organized in databases, which the virus scanning engines use to search systematically predefined areas of files and system areas to detect the viruses in the limited time allowed for the scanning. Wildcards technique is often supported by simple scanners. Typically, a wildcard is allowed to skip bytes or byte ranges, and some scanners also allow regular expressions. For example on the wildcards: "0400 B801 020E 07BB ??02 %3 33C9 8BD1 419C", where (??) means to ignore this byte, and (%3 33) means to try to match 33 in any of the following 3 positions and if matched continue.

Mismatches technique means mismatching in strings. It allows N number of bytes in the string to be any value, regardless of their position in the string.

Generic detection technique scans for several or all known variants of a family of computer viruses using a simple string. When more than one variant of a computer virus is known, the set of variants is compared to find common areas of code. A simple search string is selected that is available in as many variants as possible. Typically, a generic string contains both wildcards and mismatches. Hashing technique is a common term for techniques that speed up searching algorithms. Hashing might be done on the first byte or 16-bit and 32-bit words of the scan string. This allows further bytes to contain wildcards. Virus researchers can control hashing even better by being selective about what start bytes the string contains. For example, it is a good idea to avoid first bytes that are common in normal files, such as zeros. With further efforts, the researcher can select strings that typically start with the same common bytes, reducing the number of necessary matches.

Bookmarks technique (also called check bytes) is a simple way to guarantee more accurate detections and disinfections. Usually, a distance in bytes between the start of the virus body (often called the zero byte of the body) and the detection string is calculated and stored separately in the virus detection record.

Top-and-tail scanning technique is used to speed up virus detection by scanning only the top and the tail of a file, rather than the entire file.

Second generation: heuristic scanners

The scanner uses heuristic rule to search for probable virus infection. There are two approaches: the first one looks for fragments of a code that are often associated with viruses.

The second approach is integrity checking; for each program a checksum can be appended, then if a virus infects a program this checksum is changed. But Szor described more details about the techniques of this generation. The most important techniques are: Smart scanning, Skeleton detection, nearly exact identification, exact identification, Heuristic analysis, and Integrity checking.

Smart scanning skips instructions like No Operation (NOP) in the host program, and does not store such instructions in the virus signature.

Skeleton detection is especially useful in detecting macro virus families. Rather than selecting a simple string or a checksum of the set of macros, the scanner parses the macro statements line to line and drops all nonessential statements, as well as the aforementioned white spaces.

Nearly exact identification is based on the use of a checksum (such as a CRC32) range that is selected from the virus body. Typically, a disinfection-specific area of the virus body is chosen and the checksum of the bytes in that range is calculated. The advantage of this method is better accuracy. This is because a longer area of the virus body can be selected, and the relevant information can be still stored without overloading the antivirus database: the number of bytes to be stored in the database is often the same for a large range and a smaller one. Obviously, this is not the case with strings because the longer strings consume more disk space and memory.

Unlike nearly exact identification, which uses the checksum of a single range of constant bytes in the virus body, exact identification uses as many ranges as necessary to calculate a checksum of all constant bits of the virus body.

Heuristic analysis has proved to be a successful way to detect new viruses. The biggest disadvantage of heuristic analyzer based scanners is that they often find false positives, which is not cost-effective for users. In some ways, however, the heuristic analyzer is a real benefit. Heuristics are closely related to a good understanding of the actual infection techniques of a particular virus type. Different virus types require completely different rules on which the heuristic analyzer logic can be built. The usual method of binary heuristics is to emulate the program execution and look for suspicious code combinations.

Integrity checking is a generic method that detects and prevents changes to the file and other executable objects based on their integrity. For example, on-demand integrity checkers can calculate the checksums of each file using some known algorithm, such as a simple CRC32. Indeed, even simple CRC algorithms work effectively by changing the generator polynomial. On-demand integrity

checkers use a checksum database that is created on the protected system or elsewhere, such as an online location. This database is used each time the integrity checker is run to see whether any object is new on the system, or whether any objects have changed checksums. The detection of new and changed objects is clearly the easiest way to find out possible virus infections, and other system compromises.

Heuristics have evolved much over the last decade. Heuristic detection does not identify viruses specifically, but extracts features of viruses and detects classes of computer viruses generically.

Third generation: Activity Traps

The activity traps programs are memory-resident programs that identify a virus by its actions rather than its structure in an infected program. Such programs have the advantage that it is not necessary to develop signatures and heuristics for a wide range of viruses. Rather, it is necessary only to identify the small set of actions that indicate an infection is being attempted and then to intervene. Szor mentioned the Behavior Blocking as another set of systems that attempt to block virus infections based on application behavior. One of the first antivirus solutions, FluShot, belongs to this class of computer virus protection. For example, if an application opens another executable for write access, the blocker may display a warning asking for the user's permission to grant the write access. Unfortunately, such low-level events can generate too many warnings and therefore often become less acceptable to users than integrity checkers. Furthermore, the behavior of each class of computer virus can be significantly different, and the number of behavioral patterns that can cause infections is infinite. Behavior-blocking systems are not useless; they still work effectively against large classes of computer viruses. In fact, they can be implemented using heuristic methods. Heuristics can reduce the false positives by providing better understanding of the attack.

Fourth-generation: full-featured protection

The full featured protection products are packages consisting of a variety of antivirus techniques used in conjunction. These include scanning and activity trap components.

In addition, such a package includes access control capability, which limits the ability of viruses to penetrate a system and then limits the ability of a virus to update files in order to pass on the infection.

Szor mentioned more techniques like: Algorithmic Scanning Methods (Filtering and The X-RAY Method), code emulation, Metamorphic Virus Detection Examples (Geometric Detection, Disassembling Techniques and Using Emulators for Tracing), Heuristic Analysis Using Neural Networks, Inoculation and Sand-boxing.

Algorithmic scanning is a set of hard-coded detection routines that are typically released with the core engine code. There are two methods that deserve to be described; Filtering and The X-RAY Method.

The filtering technique is increasingly used in second-generation scanners. The idea behind filtering is that viruses typically infect only a subset of known object types. This gives the scanner an advantage, which reduces the number of string matches the scanner must perform.

Algorithmic scanning relies strongly on filters. Because such detections are more expensive in terms of performance, algorithmic detection needs to introduce good filtering. A filter can be anything that is virus-specific: the type of the executable, the identifier marks of the virus in the header of the scanned object, suspicious code section characteristics or code section names, and so on. Unfortunately, some viruses give little opportunity for filtering.

X-RAY scanning takes advantage of all single-encryption methods and performs these on selected areas of files, such as top, tail. Thus the scanner can still use simple strings to detect encrypted and even some difficult polymorphic viruses.

The scanning process is a bit slower, but the technique is general and therefore useful. For example: the virus uses a constant XOR encryption method with a randomly selected byte as a key stored in the virus.

Code emulation is an extremely powerful virus detection technique. A virtual machine is implemented to simulate the CPU and memory management systems to mimic the code execution. Thus malicious code is simulated in the virtual machine of the scanner, and no actual virus code is executed by the real processor.

Metamorphic Virus Detection Examples; there is a level of metamorphosis beyond which no reasonable number of strings can be used to detect the code that it contains. At that point, other techniques must be used, such as examination of the file structure or the code stream, or analysis of the code's behavior. To detect a metamorphic virus perfectly, a detection routine must be written which can regenerate the essential instruction set of the virus body from the actual instance of the infection. Other products use shortcuts to try to solve the problem, but such shortcuts often lead to an unacceptable number of false positives. Some useful techniques are introduced; Geometric Detection, Disassembling Techniques and Using Emulators for Tracing.

Geometric detection is the virus-detection technique based on alterations that a virus has made to the file structure. File viruses often rely on a virus infection marker to detect already infected files and avoid multiple infections. Such identifier can be useful to the scanner in combination with the other infection-induced geometric changes to the file. This makes geometric detection more reliable, but the risk of the false positives only decreases; it never disappears.

Disassembling Techniques are techniques used to separate or take apart the stream of code into individual instructions. This is useful for detecting viruses that insert garbage instructions between their core instructions. Simple string searching cannot be used for such viruses because instructions can be quite long, and there is a possibility that a string can appear "inside" an instruction, rather than being the instruction itself

.Using Emulators for Tracing is very useful for working with viruses because it allows virus code to execute in an environment in which it cannot escape. Code that runs in an emulator can be examined periodically, or when particular instructions are executed. If used properly, emulators are still very useful in detecting metamorphic viruses.

Heuristic Analysis Using Neural Networks; in general, a trained neural network seems to be overkill for detecting a single virus because of the amount of data and computations required. Even a well-optimized neural network scanner can decrease overall scanning performance by about 5%. Thus it is more interesting that neural networks can be applied to heuristic computer virus detection. Neural network based heuristics depend on a good training set. With more 32-bit Windows viruses in the training set, the automatically trained heuristic produces slightly better results. In practice, neural network heuristics are very effective against closely related variants of viruses that are used in the original training set. They also yield good results against new families of computer viruses that are similar enough to the feature set of known viruses in the training set. It is also important to select n-grams of the virus from the entire virus body. Some antivirus vendors are attempted to train neural networks with n-grams selected from emulated instructions of the virus body. However, looping virus code can often generate instruction sets (n-grams) similar to normal programs, yielding an unacceptable false positive ratio.

Inoculation software adds the marker of the viruses to objects, preventing infections, because the virus will believe that all objects are already infected. Unfortunately, this solution has some drawbacks: each virus has a different marker (or no marker at all), and overused inoculation can impair the effectiveness of the virus detection and disinfection.

Sand-boxing solutions introduce cages, "virtual subsystems" of the actual operating system. The idea is to let the un-trusted programs run on a virtual machine that has access to the same information to which the user has access on the local machine, but only has access to a copy of the information within the cage. On the virtual system, the new un-trusted program, such as a computer

virus, is able to read files that are "on the real system," even read the Registry keys and so on, but its networking capabilities are reduced. And when it attempts to make any changes, it makes them in the replica of information within the cage. Thus the virus is free to do anything it wants, but this happens in a cage instead of in the real system. When the application finishes execution, the file and Registry changes can be thrown away and malicious-looking actions can be logged.

