A GENETIC-BASED INTELLIGENT INTRUSION DETECTION SYSTEM

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

HALİL ÖZBEY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

SEPTEMBER 2005

Approval of the Graduate School of Natural and Applied Sciences

_____
Prof. Dr. Canan ÖZGEN
Director

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

_____
Prof. Dr. Çağlar GÜVEN
Head of Department

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

_____
Assoc. Prof. Dr. Tayyar ŞEN
Supervisor

Examining Committee Members

Assoc. Prof. Dr. Levent KANDİLLER (METU,IE) _____

Assoc. Prof. Dr. Tayyar ŞEN (METU,IE) _____

Assoc. Prof. Dr. Yasemin SERİN (METU,IE) _____

Dr. Ayten TÜRKCAN (METU,IE) _____

Prof. Dr. Ayşe KİPER (METU,CENG) _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Halil Özbey

Signature :

ABSTRACT

A GENETIC-BASED INTELLIGENT INTRUSION DETECTION SYSTEM

ÖZBEY, Halil

M.Sc., Department of Industrial Engineering

Supervisor: Assoc. Prof. Dr. Tayyar ŞEN

September 2005, 139 pages

In this study we address the problem of detecting new types of intrusions to computer systems which cannot be handled by widely implemented knowledge-based mechanisms. The solutions offered by behavior-based prototypes either suffer low accuracy and low completeness or require use data eplaining abnormal behavior which actually is not available. Our aim is to develop an algorithm which can produce a satisfactory model of the target system's behavior in the absence of negative data.

First, we design and develop an intelligent and behavior-based detection mechanism using genetic-based machine learning techniques with subsidies in the Bucket Brigade Algorithm [8]. It classifies the possible system states to be normal and abnormal and interprets the abnormal state observations as evidences for the presence of an intrusion.

Next we provide another algorithm which focuses on capturing normal behavior of the target system to detect intrusions again by identifying anomalies. A compact and highly complete rule set is generated by continuously inserting observed states as rules into the rule set and combining similar rule pairs in each step.

Experiments conducted using the KDD-99 data set have produced fairly good results for both of the algorihtms.

Keywords: Intrusion Detection, Genetic Algorithms, Machine Learning

ÖZ

GENETİK TABANLI AKILLI BİR SALDIRI TESPİT SISTEMİ

ÖZBEY, Halil

Yüksek Lisans, Endüstri Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Tayyar ŞEN

Eylül 2005, 139 sayfa

Bu çalışmada bilgisayar sistemlerinde yeni tip saldırıların tespit edilmesi sorunu ele alınmıştır. Yaygın olarak kullanılan bilgi-tabanlı mekanizmalar bu soruna karşı çaresiz kalmaktadır. Davranış tabanlı prototipler ya hassasiyet ve bütünlük sorunları yaşamakta ya da gerçekte elde olmayan anormal durum verisine gerek duymaktadır. Amacımız hedef sistem için negatif veri kullanmadan tatminkar bir davranış modeli üretebilecek bir algoritma geliştirmektir.

İlk olarak Kova Birliği Algoritması'na [8] teşvikler eklenmiş genetik tabanlı makine öğrenimi teknikleri kullanan davranış tabanlı akıllı bir algoritma geliştirilmiştir. Bu algoritma olası durumları normal ve anormal olmak üzere iki sınıfa ayırıp anormal gözlemleri bir saldırının varlığının kanıtları olarak yorumlamaktadır.

Daha sonra normal davranışı kavramaya odaklanıp yine saldırıları anormal durumları saptayarak algılayan bir model sunulmaktadır. Normal durum verilerinin sürekli olarak eklenip her adımda benzer kural ikililerinin birleştirilmesi ile az yer kaplayan ama yüksek derecede bütünlük arz eden bir kural kümesi elde edilmektedir.

KDD-99 veri kümesi ile yapılan testlerde oldukça iyi sonuçlar alınmıştır.

Anahtar Kelimeler: Saldırı Tespiti, Genetik Algoritmalar, Makine Öğrenimi.

To My Family

ACKNOWLEDGEMENTS

First of all I would like to present deepest gratitude to my thesis supervisor Assoc. Prof. Dr. Tayyar Şen for his guidance and timely help. It is a great pleasure to study under his supervision.

Most special thanks go to every member of my family for their endless love support and encouragements. Without their support this thesis would not be possible.

I would like to thank my dear friend Bora Kat for listening my boring explanations about every detail of this work and handing bright ideas in return. I also wish to thank my friends Oğuz Solyalı, Öncü Akyıldız and Tahir Fidan for continuous and intimate moral support.

I want to express my appreciation to my collaborator Alptekin Çakırcalı for making time out of things for me. His presence meant chance of focusing on this study just in time I need.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Intrusion detection is the act of detecting non-permitted, inappropriate, illegal use of computer resources. It is a highly complicated and ambiguous issue due to wide variety in types and specifications of the computer systems, profiles of the possible intruders, and forms of gaining access.

In today's fast evolving world, needs for computing facilities are expanding rapidly. Fast development in information and computing technologies offers various solutions to all types of needs, ranging from the simplest to the most complicated ones. The solutions come in the form of high-tech hardware such as faster processing chips, network elements communicating faster with astonishing instruments; as well as in the form of software like more functional and more attractive operating systems or package programs. However, the large scales of the projects and the limited time offered by competitive markets exert excessive pressure on developers. The result is systems with many bugs and vulnerabilities down to both hardware and software.

In 1980 the first suggestion to use audit trails for systems security by Anderson [3] did not receive much attention. On those days, the idea of computer security was limited to setting security permissions on sensitive data and requiring authorization for accessing systems. Neither the importance of bugs and vulnerabilities nor the need for intrusion detection systems was recognized until the Internet (Morris) Worm crashed more than 6000 computers connected to internet and paralyzed the internet for five days in 1988. Since then it has been admitted that no professional system can be built without bugs and vulnerabilities, and intrusion detection has to be a part of the systems security issue.

Intrusions are almost always harmful to the users and administrators of the target resources. The damages may occur in several ways such as:

- Loss of critical information and data.
- Theft of confidential information and data.
- Damage on hardware.
- Production or service downtime.
- Loss of reputation.
- Negative effect on customer relations.

In 2005 E-Crime Watch™ Survey [11], conducted among 819 security executives and law enforcement personnel, by *CSO* magazine in cooperation with the United States Secret Service and the Carnegie Mellon University Software Engineering Institute's CERT® Coordination Center, "Respondents report an average loss of $506,670 per organization due to e-crimes". Intrusions deteriorate the productivity and efficiency of computer systems.

There have been several approaches to the intrusion detection problem with different detection mechanisms using various data mining techniques, and detection system architectures. Most of the commercial intrusion detection systems use knowledge-based detection mechanisms in which the accumulated knowledge from the previous experiences is used to detect the intrusions. Expert Systems, Signature Analysis, Colored Petri Nets, State Transition Analysis are among the techniques used to implement knowledge-based mechanisms. Relatively low CPU power needs and high accuracy in detecting known attack types are the two prevalent features of the knowledge-based systems leading them to be the choice of producers.

However knowledge-based systems must be updated by an external source regularly and new intrusion types pose considerably high risks on targets protected by these systems since they can detect only the previously known intrusion types and have nothing to do with a new type of attack. That is why new virus threats are most effective on the very first day of their appearance. A research by Moore [27]

revealed that “On July 19, 2001 more than 359,000 computers were infected with the Code-Red (CRv2) worm in less than 14 hours” doubling the number of infected computers every 37 minutes. Figure 1 shows the number of infected hosts by CRv2 in UTC time. Worse than that, according to Moore et al. [28] Sapphire worm (also called SQL Slammer) “was the fastest computer worm in history. As it began spreading throughout the Internet, it doubled in size every 8.5 seconds. It infected more than 90 percent of vulnerable hosts within 10 minutes.” Its distribution after 30 minutes of its release can be seen in Figure 2.

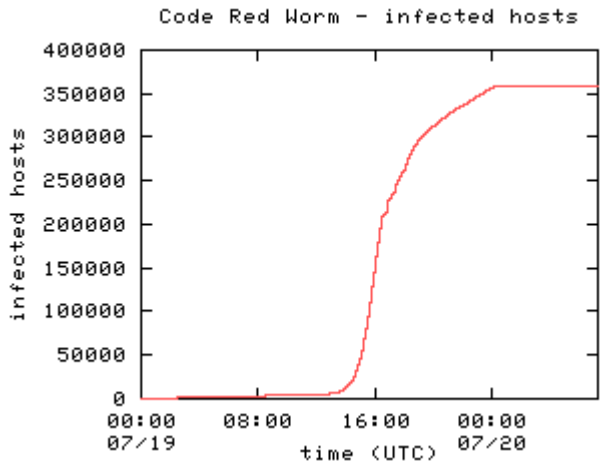

Figure 1 Number of Infected Hosts by CRv2 over UTC Time.

A solution to the problem caused by the new attack types is offered by the behavior-based intrusion detection mechanisms. These type of systems concentrate on the behavior of the target systems, usually the system states or the state transitions are classified to be normal and abnormal. Assuming that intrusions trigger anomalies in the target systems, the deviations from the normal behavior are identified to indicate presence of an intrusion.

Performance of behavior-based systems does not depend on whether the attack is a new type or a known one because they detect the intrusion attempts by modeling

and monitoring the behavior of the target system. However, even in simplest systems capturing the behavior is not an easy task. The large number of system parameters that can be used to model the system behavior result in huge numbers of possible system states to be classified and therefore substantial CPU power is required. On the other hand, limitations on CPU power may lead to high rates of false alarms. Thus, the efficiency of the tools used to model the system behavior is important. Expert Systems, Statistics, Neural Networks, User Intention Identification and Computer Immunology are among the tools that have been used for behavior-based intrusion detection.

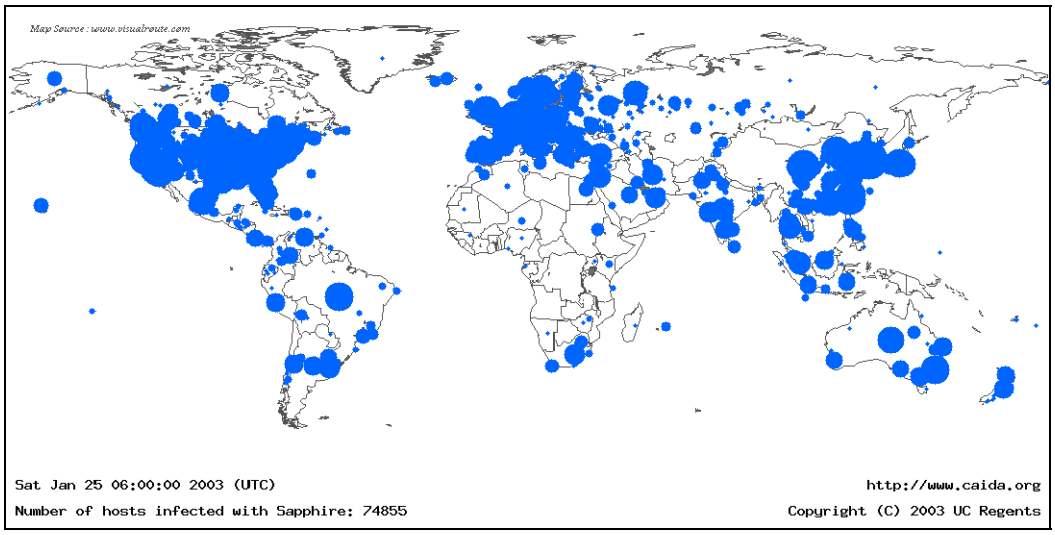

Figure 2 Geographic Spread of Sapphire in First 30 Minutes after Release.

In a recent work, Dasgupta and Gonzales [3] successfully used Genetic Based Machine Learning also called Classifier Systems with Bucket Brigade Algorithm as a tool for modeling the target system behavior. However their system needs data representing both positive (normal) states and negative (abnormal) states. In their study they assume that they have high level knowledge (i.e. exactly which states are normal and which are not) and that they can use both positive and negative states in the training period. Although it is easy to obtain the normal states from a

target system in operation it is almost impossible to obtain abnormal states especially the ones which will occur during an unknown attack type. In designing our first model we propose using subsidies in the Bucket Brigade Algorithm to modifying the model of Dasgupta and Gonzales [3] so that it can work in the absence of negative data.

Another way to deal with the huge number of states while using effective modeling tools is concentrating on only the normal behavior and reducing the possible number of normal states. It is possible to plant a host in a network with no specific purpose, for nobody's use but as a honey trap for intruders. For these hosts, many parameters become fixed since such systems normally do nothing, therefore the number and diversity of the normal states is relatively low. This gives the chance of producing rule set representations that cover the normal states with high completeness. Then a state is identified to be normal if it is covered by the rule set and abnormal otherwise. Our second model builds a rule set that represents the normal states with high completeness and using minimum possible number of rules.

To sum up, we build two genetic based intrusion detection mechanisms. The first one uses Classifiers with Bucket Brigade Algorithm. Subsidies used to promote the rules representing negative cases as a novel practice. The second model uses not all but some principles of genetic algorithm such as binary state representation and combination of rules to obtain better ones. In order to simulate the mechanisms, codes for both models have been developed and have been tested wıth KDD-99 data set for performance evaluation.

In Chapter 2, literature on intrusion detection, genetic algorithms and Genetic based machine learning is provided. In Chapter 3, the two intrusion detection mechanisms are presented including the operational details. We present and discuss the computational results in Chapter 4.

In Chapter 5 we discuss the conclusions and directions for further research.

CHAPTER 2

LITERATURE REVIEW

In this section first we outline the previous approaches to intrusion detection (ID), next we explain the genetic algorithms (GA's) and genetic based machine learning (GBML), and finally we describe the use of GBML in ID.

2.1 INTRUSION DETECTION SYSTEMS

2.1.1 Overview

Intrusion Detection Systems (IDS) are the software tools acting like the burglar alarms for a computer system which may be a host (server, workstation, mainframe), or a whole system composed of several hosts and an interconnecting network (Local Area Network [LAN], Wide Area Network [WAN], Virtual Private Network [VPN]). After all security measures have been taken, computers systems may and usually do contain backdoors for the penetration of the intruders due to the bugs and vulnerabilities residing in. IDS monitor and protect a *target system* by identifying presence of intrusions and ceasing them if possible.

An overall picture of an IDS may be seen in Figure 3. A *sensor* or a *probe* is the component, which collects the valuable data for detection mechanism, in direct interaction with the target system, by issuing system commands or continuously reviewing the system logs. An *audit trail* is a compact piece of information about the monitored parameters of the target system. The *detection mechanism* is an algorithm or a set of algorithms combined with data storage and mining tools used to perform the actual detection and is the most important component in an IDS. The performance of an IDS usually refer to the performance of the detection mechanism In the next three sections we outline the detection mechanisms, data mining tools and the performance measures used for the IDS in the literature.

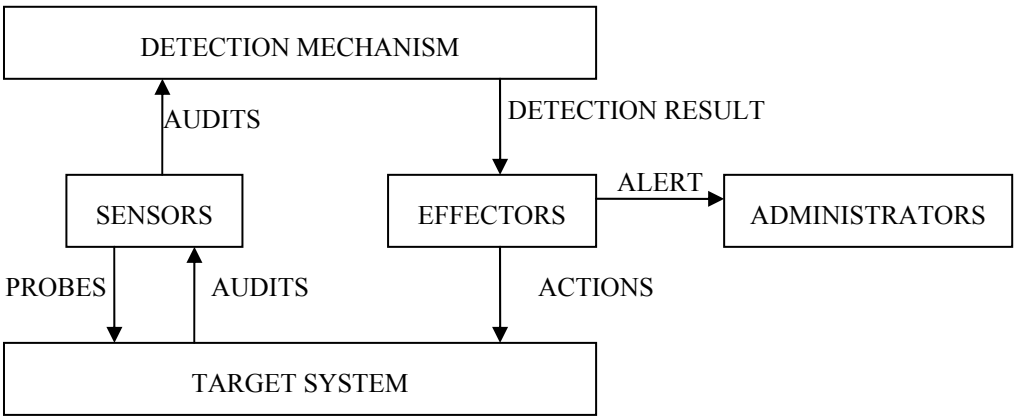

Figure 3 An overall picture of an Intrusion Detection System.

2.1.2 Intrusion Detection Mechanisms

The detection mechanism is like the brain of an IDS. The data mining tools to store, search and retrieve the related data and algorithms to recognize monitored conditions to identify an intrusion are considered within the detection mechanism.

IDSs can be divided into two major categories according to their detection mechanisms: *knowledge-based* systems and *behavior-based* systems. In knowledge-based systems, methods used by the intruders in the previously identified intrusion events are profiled according to their distinguishing properties. These profiles are stored in databases in form of definition signatures or transition patterns or command sequences. The audit trails of modified files, monitored parameters of the system, the successive commands issued by users, are continuously compared to those profiles. Best examples are the antivirus software.

These systems are very accurate (i.e. they have very low false alarm rates) and they achieve very high completeness (i.e. they detect almost all attacks). However their profile databases have to be updated by an external source regularly and they are very weak against new types of attacks since the new attack type has not been profiled at the moment of intrusion. Expert systems [14], [19], [25]; Signature Analysis [2], Petri Nets [24], and State Transition Analysis [29], [20] techniques

have been utilized to implement the knowledge based intrusion detection mechanism and they will be briefly explained later in this section.

In behavior-based systems, on the other hand, the operational behavior of the target system is regarded as the reference for detection rather than attack profiles. Several parameters of the target system at various levels are observed for quite a long period of time to construct a model to explain the behavior of the target system. This model is then used to classify the system behavior to be either normal or abnormal. The deviations from the normal behavior are interpreted to indicate presence of an intrusion. The underlying assumption for these systems is that an intrusion would cause anomalies in the system behavior. Hence the behavior of the system has to be defined carefully so that it can reveal important details.

The behavior of the target system may be defined in terms of system states, in case of state-based systems where the transition-based system define the system behavior using transitions between states. A system state is a specific combination of all monitored parameters. Although the type and number system parameters to be monitored vary in different systems, we present a list offered by Dasgupta and Gonzales [3].

- User level.
  - Type of users and user privileges.
  - Login/logout period and location.
  - Access of resources.
  - Type of software/program use.
  - Type of commands use.
- System level.
  - Cumulative per user CPU usage.
  - Usage of real and virtual memory.
  - Amount of swap space currently available.
  - Amount of free memory
  - I/O and disk usage.

- Process level.
  - The number of processes and their types.
  - Relationship among processes.
  - Time elapsed since the beginning of the process.
  - Current state of the process (running, blocked, waiting)
  - Percentages of process times (user process time, system process time, idle time)
- Packet (network) level.
  - Number of connections and connection status. (established, time_wait, close_wait)
  - Average number of packets sent and received.
  - Type of connection (remote/local)
  - Protocol and the port used.

The ability to detect new types of attacks is the prevalent advantage of behavior-based systems. These systems do not use profiles obtained from the previous experiences, unlike the knowledge-based systems. Therefore lack of knowledge about novel attacks does not impose any burden. All attacks can be detected by these systems as long as the attacks cause deviations from the normal behavior of the system.

Characteristic high false alarm rates are the main drawbacks of these systems. The complexity in modeling the system behavior arises due to the large number of possible system states which require relatively high computational power and significant memory space.

As for the techniques used in behavior based intrusion detection, statistics [17], [18], [21], [22]; expert systems [34], [10]; neural networks [13], [31]; user intension identification [32], [33] may be counted.

2.1.3 Data Mining Techniques Used in IDS

Expert systems are mainly used for knowledge based intrusion detection [14], [19], [25] by setting up rule sets through extraction of expertise knowledge on previous

attacks. Also it is possible to attach additional features to them. Garvey and Lunt [14] have designed an expert system for intrusion detection using model based reasoning. Expert systems never fail to test the known cases however it may be difficult to detect new types of attacks, the rule sets may grow enormously large and they may require substantial CPU power. Debar et al. [7] state that "Owing to the processor speed issue, expert system shells are used only in prototypes. Main use of expert systems in behavior based systems is policy-based usage profiles, however, statistical methods prove better, when large amount of information is present.

Signature analysis [2] has been used only in knowledge based systems. They are very similar to expert systems but the attacks are identified as some repetitive patterns in the system logs, network packets and ID sensors, thus the rule sets are significantly simplified. This technique has been implemented in many commercial packages due to its processing performance however inability to define new attack types is the main weakness for this model as well.

Colored Petri Nets [24] have also been used for some sort of signature analysis. Their simplicity and generality in representation is advantageous but matching a complex signature against an audit trail may become computationally very expensive.

Porras and Kemmerer [29] have proposed State Transition Analysis which defines an attack as a series of state transitions and operates by detecting special series. This idea has been first implemented in UNIX by Ilgun [20] and to other systems later on.

Statistics is widely used in behavior based systems to define normal system behavior. Login, logout times, session durations are the most common statistical measures. However many statistical models are too simple to represent the overall system behavior. Recently some complex models have been developed and put in production by Javitz and Valdes [21] and Javitz et al. [22].

Neural Networks have been used in behavior based systems by Gallinari et al.[13]

and Sarle [31]. Neural networks emulate the way neurons work and although it has not been fully explained how they work, they can be used for a way of machine learning. After a Neural Network learns the normal behaviors of the actors in the system they can be used to predict normal behavior and detect intrusions. Their advantage over statistics is that they can easily grasp (learn) non-linear relationships. On the other hand, they are computationally intensive and not very common.

User Intention Identification has been developed for the SECURE-NET project [33] by Spyrou and Darzentas. The high level tasks by the users are defined as the actions lists and when actions of a user do not resemble any of the action lists then an alarm is raised.

Computer Immunology is another technique developed by Forest el al. [12] which focuses on the normal behavior of system services rather than the user actions. A good sample of audits representing the appropriate system behavior is collected first then a list of good system call series is extracted. Then an intrusion alarm is raised when an unexpected sequence is met. These systems potentially have very low false alarm rates; however they provide no protection against intrusions, which are caused by configuration errors in system services.

Intrusion detection systems may also be categorized according to their *behavior on detection,* the system is *active* if it takes an evasive action to stop the intruder on detection and *passive* if it just generates an alert; according to their *audit source location*, which may be the *system log files*, *network packages*, *application log files* and *IDS sensor alerts*; according to their *detection paradigm* the system is *state based* if it monitors the state of the systems and it is called *transition based* if it monitors the transitions between system states.

2.1.4 Efficiency Measures for IDS

Measuring the efficiency of the IDSs is a rather complicated issue due to the diversity of the target systems and protection needs. Significance of the measures may differ along with the conditions and purposes; moreover different measures

may be required. Here we present three basic measures proposed by Porras and Valdes [30].

First, *accuracy* is the ability to raise true alarms, which may also be defined as the ratio of true ones to all alarms. As the ratio of false alarms increases, accuracy declines. The second, *completeness* is the ability to detect all intrusions which may be measured by the ratio of detected ones in all intrusions. As the number of the non-detected intrusions increase the completeness declines. However it is almost impossible to evaluate the completeness for a real system outside the laboratory because the absolute knowledge of all attacks is unavailable and the number of undetected intrusions cannot be easily determined.

Accuracy and completeness are complementary to each other hence must be used together, using one without the other may lead to very improper conclusions. In case a system raises an alarm to only one audit in thousands and if it is true then the system is hundred percent accurate but totally incomplete and on the other extreme a system raising an alarm for everything would be hundred percent complete since it has raised an alarm for every intrusion as well as every normal audit.

The third measure *performance* is the ability to perform detection tasks quickly which is be measured by rate of processing the audits. As the number of audits processed per unit CPU power increases, the performance of the system rises. An ID system must have relatively high performance so that it can be used in real time.

2.2 GENETIC ALGORITHMS

Genetic Algorithms (GAs) are adaptive search and optimization algorithms which imitate the mechanisms of biological evolution. After the pioneering works of John Holland in the 60's, GAs has been widely studied, experimented and applied in many fields in engineering world. GAs not only provide alternative methods for solving problems, but also consistently outperform other traditional methods in most of those problems. Many of the real world problems that are involved finding

optimal parameters might prove difficult for traditional methods but ideal for GAs. However, because of its outstanding performance in optimization, GAs have been wrongly regarded as a function optimizer. In fact, there are many ways to view genetic algorithms. Perhaps most users come to GAs looking for a problem solver, but this is a restrictive view. In the next section Genetic Based Machine Learning is outlined.

Like the mechanism the glossary used in GA's is taken from the field of biology.

- An *individual* is a candidate solution represented by a chromosome.
- A *chromosome* is a string in usually binary form and represents the candidate solution according to a predefined syntax like a vector of variables.
- A *gene* is a variable in the chromosome or the space set aside in the chromosome for the storage of the variable which is interpreted according to the syntax
- The *locus* of a gene is the place of the space of the variable.
- An *allele* is a possible value for a variable.
- *Crossover* is a procedure in which new candidate solutions are produced from the existing ones by mating parent chromosomes to reproduce the offspring.
- *Mutation* is random alteration in a chromosome, so that the diversity in the population is increased.
- *Fitness* or the strength of a chromosome is the measure of the quality of the solution represented.

The basic idea behind the GAs is that a pool of many chromosomes can be evolved to contain the optimal solution. In consecutive iterations the quality of candidate solutions can be augmented by applying the principles of biological evolution such as survival of the fittest, elimination of the weak, and reproduction of the better. Hopefully the optimal or a close to optimal solution will be included in the population after a number of iterations.

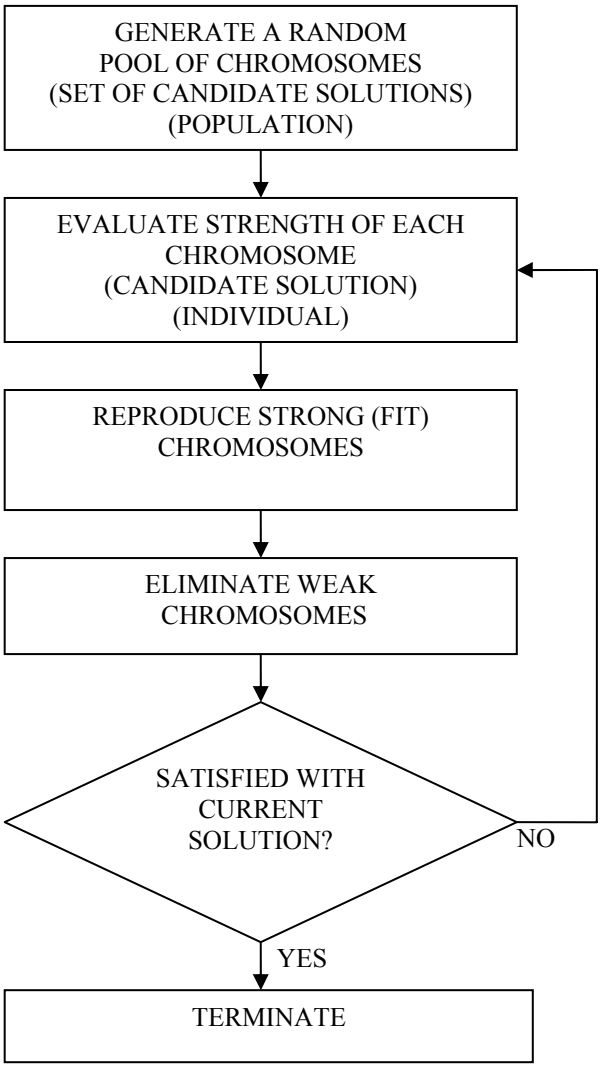

Figure 4 General view of genetic algorithms operation.

General operation of the genetic algorithms is available in Figure 4. First task is to create the initial population. Then the algorithm operates in an iterative manner. In each iteration, strength of each chromosome is evaluated using a fitness function. The weak ones are replaced by new chromosomes which have been reproduced from the fittest ones using the genetic operators such as cross over and mutation. After sufficient number of iterations optimal or near-to-optimal solutions are obtained.

To obtain successful results from the implementation of the genetic algorithms, four elements of the genetic algorithm must be clearly identified.

Syntax of a chromosome defines the way the variables are organized, the way values of a variable are coded and interpreted. Candidate solutions to each problem can be represented by a number of different manners. Binary Representation, Gray Encoding, Diploid Binary Encoding, Permutation Representation, Random Key Representation and domain specific representations have been used up to date. Usually the nature of the genetic operators in the algorithms is highly dependent on the syntax. In particular problems, the syntax of the chromosome may be the key feature determining the performance of the algorithm.

The *fitness function* is used to evaluate the quality of a candidate solution. In many cases it is the same thing as the objective function in Linear Programming formulation of the problem; however the result of the function may be scaled linearly or exponentially. Also it may be the result obtained from an evaluation procedure rather than a simple function.

Creation procedure for the initial population may be randomly generating the individuals or individuals may be obtained by hybridizing the possible values of variables so that the possible values of the variables are covered as much as possible.

The *genetic operators* to be used are usually the key components of the genetic algorithm. Four operations to be performed are parent selection, crossover, mutation and replacement. The vital dilemma while establishing the characteristics

of the operators is the balance between exploration and exploitation. That is the algorithm must continuously widen the area it has searched by generating individuals with diverse characteristics so that it is not stuck to inferior solutions, however at the same time it must focus on the candidate solution which has relatively better qualities so that it can obtain better results in short time. Several alternatives have been proposed for each operation and operator.

Selecting continuously strong *parents* may hinder exploration and cause premature convergence of the population meaning losing its diversity before exploring sufficiently large part of the problem domain. However increasing the randomness in parent selection may turn the algorithm into simple random search. Fitness Proportionate Selection, Ranking Selection, Tournament Selection, Truncation Selection, and Gene Pool Recombination are among the alternatives.

The *crossover operator* is probably the most important operator of the genetic algorithm. If it can reproduce new individuals with high quality features while preserving the diversity it may guarantee the success. One Point Crossover, Multi-Point Crossover, Uniform Crossover, PMX, Edge Recombination and other problem specific crossover techniques have been used.

Mutation is used to increase the diversity in the pool. It serves as a source of missing characteristics when used in small ratios but also may cause damage to the high quality genes and may cause serious problems especially when stability of the pool is important. Changing the value of randomly chosen bit, Interchange of two genes, Random permutation of elements between two points have been used.

Elimination of the weak individuals is obligatory to create for new individuals since the population can not continuously grow. The vital dilemma of exploration and exploitation is on charge as in other parts. Obtaining better individuals – the main objective – is impossible without eliminating the weak; however eliminating weak may also mean loss of diversity. Also the stability of the pool may be an important property as in case of the classifier systems. Replace all Policy, Elitist Policy and Steady-State Replacement are some of the adopted strategies.

Another point of decision is when to terminate the algorithm. The run may be terminated after a predetermined number of generations or after the same pool is observed for a predetermined number of steps or when the population is converged. Details of simple GAs can be found in [15] and [26] among a number of such references.

2.3 GENETIC-BASED MACHINE LEARNING (GBML) AND CLASSIFIER SYSTEMS

In this section we outline the most common GBML approach; the Classifier Systems with Bucket Brigade Algorithm emphasizing the four important elements of the genetic algorithms. A Classifier System consists of three main components

1) Rule and message system (RMS)
2) Apportionment of Credit System (ACS)
3) Genetic Operators.

Rule and message system defines the syntax and interpretation of a chromosome. The Apportionment of the Credit System serves as a fitness measure working jointly with the RMS. When any input is received from the environment, the RMS is used to determine the corresponding output. In the same time ACS is used to recalculate the fitness of the rules that have taken part in the determination of the output. After several iterations with the RMS-ACS cycle, when the strengths of the rules get differentiated the weak rules are eliminated and children of strong rules are produced using the genetic operators. General operation of a GBML system can be seen in Figure 5.

The knowledge is regarded as the ability to give the correct output to any input, which in turn may be defined as the ability to classify the input in correct output classes. If a system can classify all input into correct classes of output then the task of responding correctly to an input will be reduced to determining the output class of the input. Thus a genetic-based machine learning system is called a classifier system.

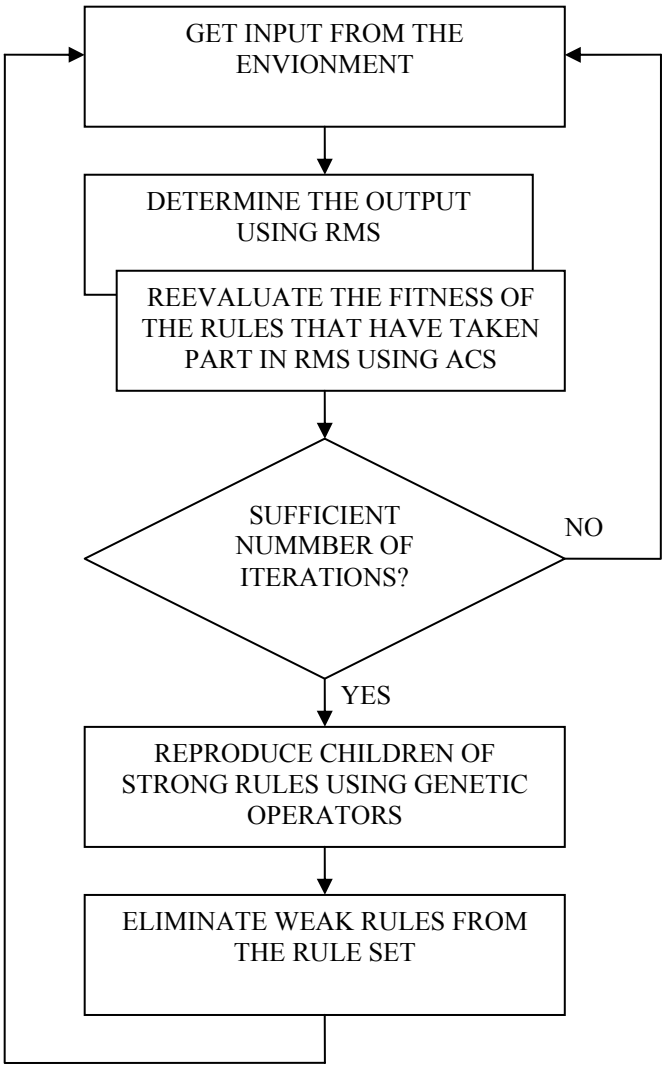

Figure 5 General GBML Operation.

The knowledge, learned by a classifier system is stored by the rules in rule set. Each rule in the rule set defines an output class and determines the set of inputs belonging to that class. Therefore a rule is sometimes called a classifier. A rule may also be called a chromosome since it is represented by binary strings and treated as the subject of the operations in the genetic algorithm where the rule set may be viewed as the pool of chromosomes.

The rule storage performs the functions of the long term memory. It knows the correct response to an input by the classes defined by the rules. When a new chromosome is created a new class or some new classes may be created so that some new knowledge is learned. Throughout the employment of the algorithm the rules may gain more strength as long as they lead to correct classifications, similar to reinforcement learning of human and animals. When the rules become weak after repeated failures in giving correct response the rules leading to those classifications get eliminated and incorrect information is disposed as in case of forgetting things.

A chromosome (also called a rule or a classifier) is a simple binary or ternary string which represents the message and action to be fired when a condition is matched. A rule may be one of the various forms but the most common form is <condition>:<message>:<action>. Also it may reduce to <condition>:<action>, or <condition>:<message> when the message and action parts are identical. A rule is interpreted as “if <condition> then <message> or <action>

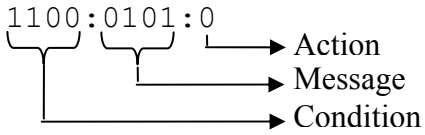

Figure 6 A rule with binary string.

Let there be a system with *n* parameters and each parameter have 2^l levels. Let R be a rule for this system and C be its condition, M its message and A its action. Let b_{ij} a bit i.e. $b_{ij} \in \{0,1\}$. Let the condition C be combination of n meaningful parts $c_1, c_2, \ldots, c_n$ each of which is an *l* bit string i.e. $c_i = b_{i1}b_{i2}\ldots b_{ij}\ldots b_{il}$ representing a level of a parameter of the environment. Then the condition C ($=c_1c_2\ldots c_n$) would be an *nl* bit string which may be interpreted as "level of p_i is c_i for each i". The message part is in exactly same form with a binary condition. However the message part is usually not meaningful for interpretation. It serves to the formation of chains in the messaging system which provides means for storage of complex knowledge. The action part is also a binary string representing the output to the environment, the length of action part may vary according to the number of outputs to the environment. A sample rule can be seen in Figure 6 The digits in the condition part may also include '#' character which means match both '0' and '1'. In this type of rules the condition part is a ternary string rather than a binary one. A rule ternary string may be seen in Figure7.

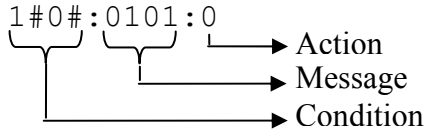

Figure 7 A rule with ternary string.

As stated before the rules are kept in a rule-storage. When an input is received from the environment as the information to be classified it is put in the message board. Only rules whose condition part matches the current message in the message board can respond. When multiple matching rules exist an auction is held. Every matching rule has to bid an amount which is computed using an expression. The bid amount is deducted from the strength of the rule, total bid is added to the strength of the activating rule which may also be the environment when the

activating message is received from the environment as the input to the system. The winner rule fires its message to the message board. Then the cycle is repeated until there is no matching rule for a message. When there is no matching rule for a message the rule which has fired the non-matching message fires its action. The action is produced. The term action is used to denote the output to the environment. The Rule and Messaging System outlined in this paragraph has been shown to work as a complete production system. An important point to be noted about the Rule and Messaging System is that it provides means for storage of complex information in simple rules by formation of rule chains via their messages posted in the message board.

The Apportion of Credit Algorithm serves as a means to evaluate the fitness of the rules. When the action is sent out to the environment a negative or positive payoff is received from the environment. The Apportionment of Credit System is utilized to apportion the payoff to the rules involved in the production of the action as the name implies. When an auction is held, sum of all bids are paid to the rule which has fired the message. Thus a rule can gain strength when it fires its message and its message is matched by other rules. When a rule fires its action the payoff from the environment is paid directly to the rule which has fired the action. Thus a rule can also gain strength when it fires a correct action. This happens when its message is not matched by other rules so that it can fire its message. As more and more inputs are received, the rule chains which produce correct actions gain more strength while the chains producing incorrect actions become weaker. The apportionment of credit System works like a service based economy. The strength of the environment is treated like the strength of any rule. All the rules and the environment start with same strength value and the strengths of the rules.

The Rule and Message System and the Apportionment of Credit System constitute the Bucket Brigade Algorithm which is very valuable due to three main reasons. First the basic functionality of a learning system is to respond to an input, BBA provides Rule and Message system for this function. Second, a learning system must store complex information; the chain structure in use of rules provides means

for storing complex information in very simple rules. Third, lack of an immediate fitness function for the rules would be detrimental for the Genetic Algorithm but the Apportionment of Credit Systems helps us in the calculation of strengths of the rules.

In operation, only after several steps with the Rule and Message system and Apportionment of Credit system, the strengths of the rules diversifies and one can distinguish the strong rules from the weak ones. Therefore the genetic operators are applied to eliminate the weak rules and produce children of stronger rules in return once after several steps unlike many implementations of the genetic algorithms where genetic operators are applied at each iteration.

Sum of strength of all rules in the rule set is constant in a cycle because the Apportionment of Credit System does not add subtract any strength to the rule set it just rearranges the strength among the rules. This causes one of the difficulties in implementation of classifier systems which is the problem of inactive rules. Condition parts of some rules do not match any input from the environment or the messages from other rules. These rules do not gain any additional strength however they do not lose any strength either. Since the total amount of strength in the rule set is constant these rules get stuck in the middle of the rule set when the rules are ranked according to their strengths. They do nothing useful for the production of a correct output to any input and they occupy valuable space in the rule storage. Solution to this problem is use life tax. At each step very small amount of strength is deducted from each rule so that a rule gets eliminated after sufficient number of steps unless it gains strength by involving in the production of correct outputs.

Also for the interested readers Genetic-Based Machine Learning and Classifier systems is explained in detail in [8] and [15].

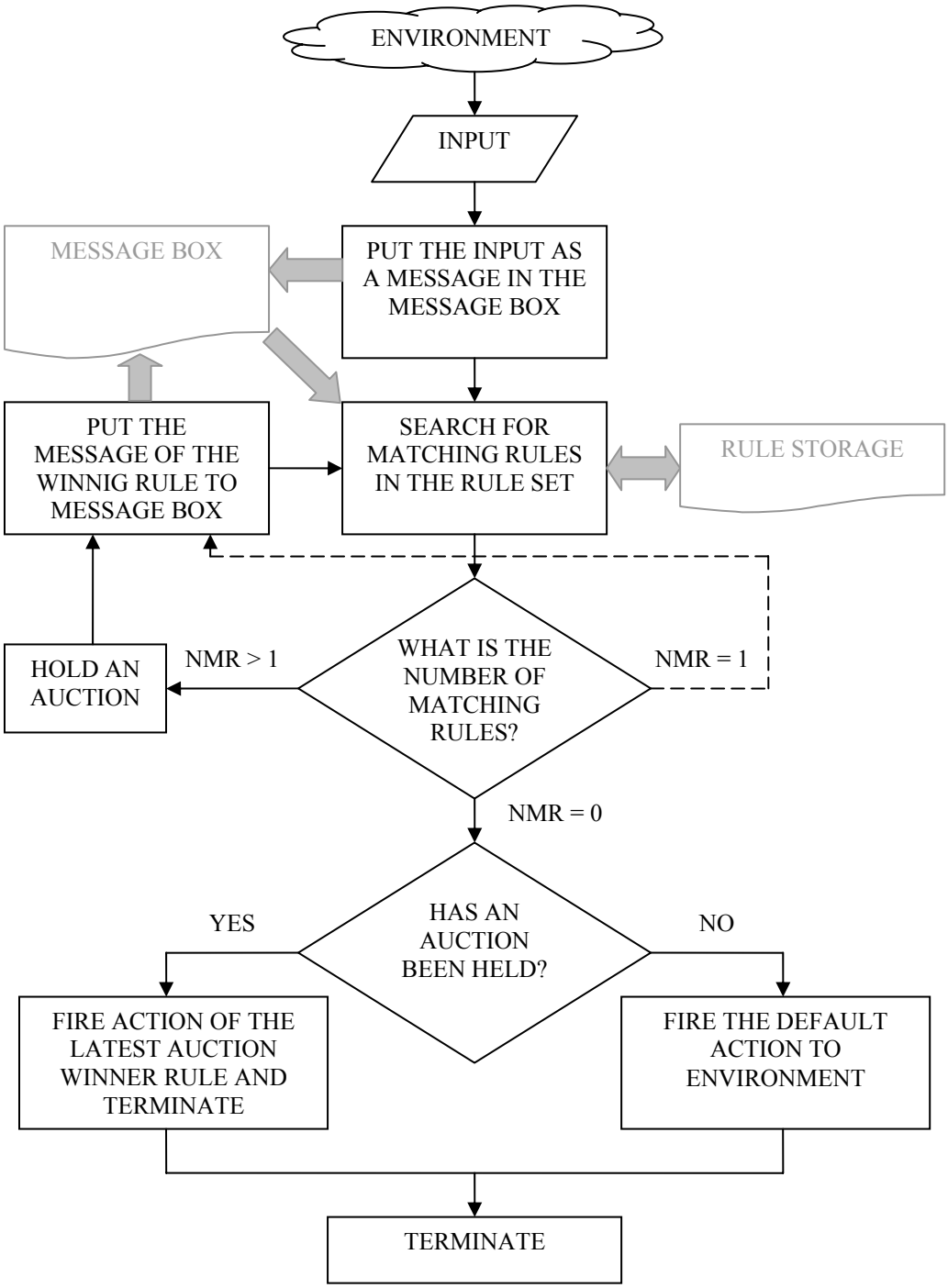

Figure 8 Rule and Messaging system.

2.4 GBML IN ID

Dasgupta and Gonzales [3] used GBML for intrusion detection in their work for the first time. They model the behavior of target system based on the system states. The chromosomes of the genetic algorithms can be regarded as a vector of independent variables; the state definition for the target system may combine a diverse set of parameters. Therefore they monitor the target system at multiple levels.

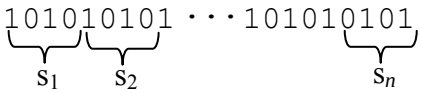

Figure 9 A System State.

When all data for n such parameters collected from the target system the data can be fused into a binary sting representing state of the target S system with n parameters (s_1, s_2, … , s_n) as in Figure 9.

After representing the system state as a binary string, they create rules for the classifier system in form of <condition>:<message>:<action> as described in the previous section. Then Rule and Messaging System is used to produce the decision of the system based on the current Rule Set. The Effectors use the system commands to apply the decision if necessary. The Apportion of Credit System reevaluates the strength of the rules involved in the decision of the most recent action in accordance with the correctness of the action produced and payoff received. The Genetic Operators are used to eliminate weak and reproduce strong rules periodically. Flow chart for such an ID system using GBML is available in Figure 10.

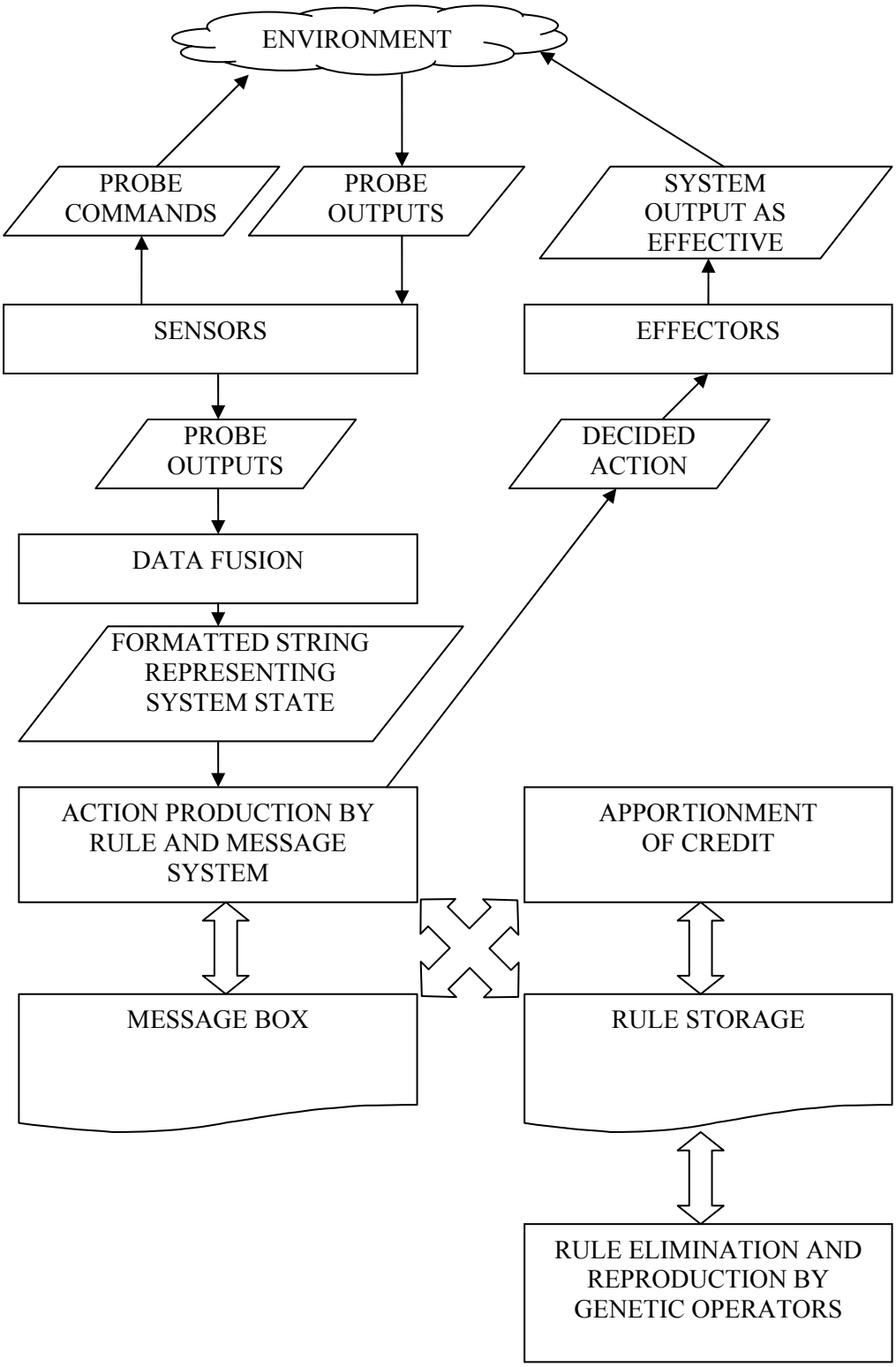

Figure 10 A Genetic-Based Intrusion Detection System.

They assumed that they had the high level knowledge on the system data and used a randomly generated data to train the system. After a certain training period they have observed a substantial increase in the performance of the intrusion detection system. One of their key assumptions is that they may use both positive and negative samples which are critical for the training of their system. They could do that since they used generated data based on their high level knowledge assumption. However this is not the reality for an actual system. Although it is possible to obtain the positive data in almost every system, the negative it is almost impossible to obtain the negative.

CHAPTER 3

INTRUSION DETECTION MODELS

In this section we clarify two genetic based models for intrusion detection. Our main aim is to design a learning system that can complete training in the absence of negative data. In the first model we develop a Michigan style classifier system starting with the model of Dasgupta and Gonzales [3] and improving their system by the use of subsidies so that only positive data (data representing the normal states of the target system) are sufficient for the training of the intrusion detection system. Next we develop a second model focusing only on the positive cases. Using not all but only some principles of the genetic based machine learning, it is possible to design an intrusion detection system which focuses on mastering only the normal cases and detecting everything else as indications in intrusion. The second model achieves 100% percent accuracy and completeness at the cost of additional need for CPU time in training period.

While building the models – following the general approach in the literature – we focus on the detection mechanism and place the components like the sensors and effectors out of our scope. We assume that the data are ready in the form of binary strings representing the system states and ready to be processed by the classifier system.

3.1 INTRUSION DETECTOR A

Intrusion Detector A (ID-A) is an intelligent ID system. When it is supplied with the data representing the normal states it can build up a classifier rule set from scratch and distinguish the normal and abnormal system states using it.

System states are vectors of several variables in binary representation. An instance of system states with two variables both with four possible values (`00`, `01`, `10`, `01`) may be seen in Figure 11 .

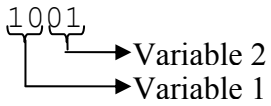

Figure 11 A Four-Bit System State.

A classifier rule for ID-A is a string consisting of three parts. First, the condition part is a string containing only three characters i.e. ‘0’, ‘1’ and ‘x’ meaning either ‘0’ or ‘1’. It is interpreted as “if the state is …” and may be exactly same as a state like the one in Figure 11. In this case it matches only one state. By using whatever symbol ‘x’ it may be used to match multiple states. A condition matching both 1001 and 1011 states may be seen in Figure 12.

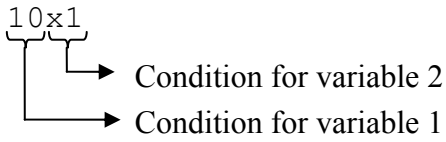

Figure 12 A four bit condition.

Second the message part of a rule is similar to a system state string value, it may contain only ‘0’ and ‘1’. It does not have any direct interpretation, but is necessary for the rule and messaging algorithm for the formation of rule chains. This will be clarified later in this section. An important point to be noted is the length of the system state, where the condition part and the message part are be equal.

Third, the action part is of any length (one bit in our case) binary string representing possible output of the system meaning that it represents the classes

into which the input is classified. When a rule has the right to fire its action it becomes the output of the system. In our system we have just two possible output types (two classes) for any possible input. First is “normal”, meaning no intrusion is present (represented by 0) and second is “abnormal” meaning an intrusion is present (represented by 1).

When the pieces are put together, a rule with four bit condition, four bit message and one bit action would look like the one in Figure 13.

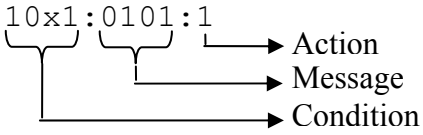

Figure 13 A Four-Bit Classifier Rule.

In Table 1 we present how the output is generated in a classifier system which is a very simplified version of the one used in ID-A. In the first step a random rule set with four rules for four bit system states has been generated. Strength of each rule (SoR) and the strength of environment (SoE) is defaulted to 1.00. And the message box (MB) is empty. In step two an input (`1001`) has been inputted from the environment and has been put into the message box. In step three, the rule set has been searched and condition of rule 2 (`10x1`) and condition of rule 3 (`100x`) has been found to be matching to the system state in the message box (`1001`). An auction will be held to determine which rule will be firing its message. In step four, the two rules have bid for the auction. The bid amount is ten percent of the strength of the rule for rules and the bids have been paid to the owner of the activating message which is the environment in this case. In step five, the winner of the auction is announced to be rule 2. The tie has been broken randomly since the bids of the two rules were equal. Rule 2 has fired its message (`1010`) and the message has been placed to the message board. In step six, the rule set has been

searched for matching conditions to the new message and condition of rule 4 (`101x`) has been found to be matching to the message in the message board (`1010`). In step seven, rule 4 has paid its bid to the owner of the activating message which is rule 2. Although rule 4 is the only matching the message an implicit auction has been held for the payment of the bid to the activator. In step eight, rule 4 has fired its message as the winner of the auction and the message has been placed in the message board. In step nine, the rule set has been searched for a rule whose condition matches the latest message. Since there is no message matching to the message the owner of the message (rule 4) is allowed to fire its action (‘0’ meaning the input represented a normal system state) as the output of the system to the environment. In step ten, rule 4 has been awarded with the ten percent of the strength of the environment since the output has been found to be correct. If the output had been incorrect rule 4 would be penalized by deducting ten percent of its strength to be added to the environment.

Now the system is ready for receiving another message representing a state of the target system. After several steps from the environment are generated, the rules and the rule chains producing correct outputs will gain more strength while the rules and the chains producing incorrect ones will lose their strengths. This is the basis for the fitness evaluation.

However the inactive rules may pose a problem as stated before. To overcome this problem some very little amount of strength (1/1000 of strength of a rule) is deducted from each rule as the life tax.

A negative rule, describing one or more negative states but no positive ones, is said to be a strong one since it would produce only correct outputs; the reverse is true for a strong positive rule. If the training data set included both positive and negative data both strong positive rules and strong negative rules would be gaining strength producing correct outputs to the inputs of their types. However in the absence of the negative data strong negative rules will not match to any positive data. Throughout the iterations they will get weakened due to the life tax and will finally get eliminated but this is not desired. In order to deal with this, we use subsidies for negative rules.

Table 1 Output Generation in a Four-Bit Classifier System with Four Rules.

	Rules	SoR	MB	SoE
1	0x0x: 0011:1	1.00		1.00
	10x1: 1010:0	1.00		
	100x: 0110:1	1.00		
	101x: 1100:0	1.00		

	Rules	SoR	MB	SoE
6	0x0x: 0011:1	1.00	1010	1.20
	10x1: 1010:0	0.90		
	100x: 0110:1	0.90		
	101x: 1100:0	1.00		

	Rules	SoR	MB	SoE
2	0x0x: 0011:1	1.00	**1001**	1.00
	10x1: 1010:0	1.00		
	100x: 0110:1	1.00		
	101x: 1100:0	1.00		

	Rules	SoR	MB	SoE
7	0x0x: 0011:1	1.00	1010	1.20
	10x1: 1010:0	**1.00**		
	100x: 0110:1	0.90		
	101x: 1100:0	**0.90**		

	Rules	SoR	MB	SoE
3	0x0x: 0011:1	1.00	1001	1.00
	10x1: 1010:0	1.00		
	100x: 0110:1	1.00		
	101x: 1100:0	1.00		

	Rules	SoR	MB	SoE
8	0x0x: 0011:1	1.00	**1100**	1.20
	10x1: 1010:0	1.00		
	100x: 0110:1	0.90		
	101x: **1100:**0	0.90		

	Rules	SoR	MB	SoE
4	0x0x: 0011:1	1.00	1001	**1.20**
	10x1: 1010:0	**0.90**		
	100x: 0110:1	**0.90**		
	101x: 1100:0	1.00		

	Rules	SoR	MB	SoE
9	0x0x: 0011:1	1.00	1100	1.20
	10x1: 1010:0	1.00		
	100x: 0110:1	0.90		
	101x: 1100:**0**	0.90		

	Rules	SoR	MB	SoE
5	0x0x: 0011:1	1.00	**1010**	1.20
	10x1: **1010:**0	0.90		
	100x: 0110:1	0.90		
	101x: 1100:0	1.00		

	Rules	SoR	MB	SoE
10	0x0x: 0011:1	1.00	**1010**	1.08
	10x1: 1010:0	1.00		
	100x: 0110:1	0.90		
	101x: 1100:0	1.02		

In each step life tax is collected only from the rules with positive action (rules firing '0' as the action, meaning the input state is a positive [normal] state) and the collected tax is distributed to the rules with negative action (rules firing '1' as the action, meaning the input state is a negative [abnormal] state).

To explain how the effect of using life tax and subsidies in fitness evaluation of rules, in Table 2, we present a categorization of the rules according to the type of states they describe and the actual actions they contain coded in their action parts.

When data available are describing only the positive states, the rules describing positive states (left column in Table 2) will be matching to the input, from these the rules with positive actions will be producing true positive outputs and these rules should be kept in the rule set. These rules gain strength each time they produce a true output and achieve higher ranks and they are naturally kept in. On the other hand, those rules describing positive states but containing negative actions will be matching to the input but will be producing false negative output therefore these rules will be losing strength each time they get activated.

Table 2 A Classification of Rules in the Rule Set.

	Describing Positive States	**Describing Negative States**
Positive Action	■ Strong positive rule. ■ Matching to the positive data. ■ Producing true positive output. ■ Should be kept in the rule set. ✓ Promoted naturally by the algoritm.	■ Weak positive rule. ■ Not matching to the positive data. ■ Not producing output. ■ Should be removed from the rule set. ✓ Demoted by the life tax.
Negative Action	■ Weak positive rule. ■ Matching to the positive data. ■ Producing false positive output. ■ Should be removed from the rule set. ✓ Demoted naturally by the algoritm.	■ Strong negaitive rule. ■ Not matching to the positive data. ■ Not producing any output. ■ Should be kept in the rule set. ✓ Promoted by subsidies.

Since negative states are not included in the input data, the rules describing negative states will not be matching to the input hence will not be producing any output (right column in Table 2). Therefore the Rule and Messaging System and Apportionment of Credit System mechanisms of the Bucket Brigade Algorithm cannot either strengthen or weaken these rules. Hence their strength must be evaluated by other means. A rule describing negative states but containing positive action is a weak rule and its strength must be decreased. This is done by life tax.

Life tax is collected only from the rules containing positive action. The rules describing negative data and having positive action is a strong rule. They do not match to the positive input and so not lose any strength since they do not produce any incorrect output. Their strengths are increased by the subsidies. In each turn the total life tax collected from the rules with positive action is distributed equally among those with negative action.

To the best of our knowledge, the use of subsidies in such a classifier system to promote the rules which describe a class of input for which the data are missing is novel to this study.

Up to now we have explained the syntax of the chromosomes and the fitness evaluation which are the critical components of the genetic based machine learning algorithms. Now we wrap up this section by giving the details of the population initialization and the genetic operators used in the algorithm.

The initial population is created randomly. Creation of a rule is as follows. For each bit (character) in a rule a random number is generated using Prime modulus multiplicative linear congruential generator. For the message and action parts the probabilities of selecting one and selecting zero as the next character are equal and sum up to one. When generating the condition part it is necessary allocate some probability to the match any character 'x' while probabilities of selecting one and zero is still equal and summing up to 1. The probability of selecting the next character P(x) is calculated according to the following formula, where n(states) is the number of all possible states (i.e. 2^{10} = 1024 when ten bit states are used), n(rules) is the number of rules used in the rule set, and n(bits) is number of bits in a system state.

$$P(x) = \frac{\log_2\left(\frac{n(states)}{n(rules)}\right)}{n(bits)} \tag{3.1}$$

If incidentally the generated rule happens to be same as one of the existing rules it is simply discarded.

After a predetermined steps (usually n(states) for small problems) with the Bucket Brigade Algorithm the weak rules are eliminated and new rules are reproduced. We want a strong and stable rule set and to preserve the stability each time we eliminate and reproduce only one rule. The weakest rule is eliminated from the rule set. The parents are selected randomly to avoid premature conversion. The two parents are mated using single point crossover. The new rule is added to the rule set as well as its parents. Also every rule is mutated with 1/1000 probability in each step.

In the test period no genetic operations takes place, only production of output is conducted in which bids are announced but not paid, no life tax is collected and no subsidy is paid.

3.2 INTRUSION DETECTOR B

Intrusion Detector B (ID-B) is another intrusion detection system. It is similar to ID-A in that it uses the same state representation for the target system, and it uses the idea of combining existing rules to obtain more useful ones. However it is not a complete genetic based machine learning algorithm and the rule combination is not a genetic operator like crossover.

As stated before it is not reasonable to assume that we may have data for both the normal and abnormal cases to be used in the training period. Therefore our main assumption is that in the training period data only for normal cases will be available to the ID system. Basic idea in building ID-B is that, if a system can produce a compact representation of the normal states with high completeness (i.e. covering almost all of the normal states) in the training period, then in the test period the system can classify the states as normal if they are covered by the rule set and abnormal otherwise.

ID-B uses the same state representation with the ID-A which has been explained in the previous section. A sample state is available in Figure 9 on page 28. The rules used by ID-B and the generation mechanism are simpler compared to the ID-A. Actually ID-B uses only the condition parts of the rules of ID-A. The rule set is a

set of conditions matching the states. The rules represent only the normal states so a state is said to be normal if the state matches any of the rules; and abnormal otherwise. To be able to successfully detect all anomalies and achieve low false alarm rates, the states encountered in the training period must be exhaustive (i.e. all possible normal states should be contained in the training data).

In the training period, since all inputs from the environment represent normal cases, in each step the input message representing a normal state is inserted in the rule set. By this way we assure that we do not miss any normal state even if the ID-B encounters a normal state only once in the training period. However if we insert each state as a rule to rule set and keep it as it is, we would need a substantial amount of memory space to store the rule set and full CPU power to search the rule set. Therefore after the insertion of a rule, the rule set is searched to find out if it is possible to compact it. If it is possible to compact the rule set, it is compacted until no more compacting is possible in anyway. Also the rule set is further reduced before a test.

Table 3 Combination of two rules.

	Rule	States Represented
First Rule	0x00	0000 0100
Second Rule	0x10	0010 0110
Combined Rule	0xx0	0000 0100 0010 0110

By compacting the rule set, we mean combining every possible pair of rules into a single rule. Two rules with all bits identical but one, where one rule contains ‘0’ in the non-identical bit and the other has ‘1’ can be combined by keeping all identical

bits and putting an ‘x’ in place of the non-identical one. ‘0x10’ and ‘0x00’ can be combined into ‘0xx0’ as in Table 3.

Reducing the rule set denotes removing the rules which represent a subset of the states represented by another rule. In reduction the less general rules are removed and more general rules are kept. A rule is said to be more general as it contains more ‘x’ characters. In Table 4 a rule set containing three rules have been reduced to contain only one rule while representing the same set of states.

Table 4 Reduction in rule set.

	Rule Set	States Represented
Non-Reduced	0xx0 **01x0** **0010**	0000 0100 0010 0110
Reduced	0xx0	0000 0100 0010 0110

In the first step of Table 5, “0001” has been inputted as the message representing the state of the target system and has been inserted in the rule set. There is no possibility of compacting the rule set since there is only one rule in it. In the second step, “0000” has been inputted and placed in the rule set. In the third step, the rule set has been searched and it the two rules “0000” and “0001” has been combined into “000x”. In the step four, state “0000”, in step five “0101” and in step six “0100” have been inputted and have been placed in the rule set. In step seven, “0100” and “0101” have been combined into “010x”, and in step eight “000x” and “010x” has been combined into “0x0x”. In step nine, the reduction is illustrated. The rule “0000” is contained by the rule “0x0x”. The more general rule is kept and the less general is removed to reduce the data set hence “0000” is removed.

Table 5 ID-B Operation in training period.

	Input Message	Rule set
1	0001	0001
2	0000	0000 0001
3		000x
4	0000	0000 000x
5	0101	0000 000x 0101
6	0100	0000 000x 0100 0101
7		0000 000x 010x
8		0000 0x0x
9		0x0x

CHAPTER 4

COMPUTATIONAL EXPERIMENTS

4.1 EFFICIENCY MEASURES

A discussion of the efficiency measures for the intrusion detection systems have been presented in Section 2.1.4. We use accuracy, completeness and performance to evaluate our system. Accuracy is the ability to raise true alarms rather than false ones, measured by the ratio of true alarms to all alarms. Completeness is the ability to detect all intrusions, measured by the ratio of the true intrusion alarms to all intrusions. Performance is the ability to perform detection tasks quickly. The training and testing times can be used to evaluate the training performance and test performance respectively.

4.2 CONDUCT OF EXPERIMENTS

Shared data sets are used to evaluate the efficiency of the IDS. These data sets are usually of two parts; the training data and the test data. Training data mainly contains exhaustive information about the normal operation of the target system. Behavior-based systems may use it for building a model in different ways using various techniques such as:

- Parameter estimation in systems employing statistical techniques
- Rule extraction in systems employing expert systems
- Training periods of learning systems

Practically training data are not needed for the knowledge-based systems since they use profiles of the previous intrusions rather than the target system behavior. Although the training data may also contain data for known attack types these can be easily removed especially if the focus is on detection of new attack types. The test data are similar to the training data however they are usually smaller in size

and contain data for new intrusion types in addition to the known intrusion types and normal data.

One of such shared data sets is the KDD-99 data set [23], which has been used in The Third International Knowledge Discovery and Data Mining Tools Competition. The competition was held in conjunction with The Fifth International Conference on Knowledge Discovery and Data Mining. The focus of the conference and the competition was intrusion detection.

This data set is based on a previous work by Defense Advanced Research Projects Agency (DARPA) and Massachusetts Institute of Technology (MIT) Lincoln Laboratories. In 1998, for the DARPA Intrusion Detection Evaluation Program, MIT Lincoln Labs set up an environment which simulated a typical US Air Force LAN and operated the network for nine weeks as if it was a true network but peppered with some attacks.

For the KDD-99 data set, raw data for the first seven weeks was processed into five million network connections to be used as the training data. The raw data for the final two weeks was converted into two million connections to be used as the test data. The training data included data for 24 attack types which were assumed to be known, and the test data contained 14 additional attack types assumed to be novel. School of Information and Computer Sciences at University of California, Irvine has published a reduced version called the 10 percent data set as well as the full data. This is the primary data set used in recent work in the field. To evaluate the performance of our models and to compare them to the related studies we perform computational experiments with the 10 percent version of the KDD-99 dataset.

As stated before our main aim is to develop detection algorithms which can operate in the absence of negative data in the training period. Therefore, we remove the data for the intrusion attempts in the training data, before beginning the experiments.

In the experiments a 21-bit state definition is used which is given in Appendix A. The first row in the Table 9 shows the number and the location of the bits allocated for a variable. The name of the variable is located in the second row. The following rows illustrate the bit streams used in that location and their meaning. For example the first 5 bits of the state definition is dedicated for the service used in a connection and "00000" stands for "auth" (i.e. authentication service). The abbreviations used in Table 9 are the standard abbreviations used in KDD-99 Data Set. The normal data in the training part of the KDD-99 Data Set and the whole test data have been converted to a training input which can be used directly by the ID-A and ID-B algorithms.

The only parameter used by ID-B algorithm is the number of bits in the state definition which has a fixed value of 21. Therefore it has been run only once. The result of this experiment is available in Table 7. On the other hand, ID-A algorithm has seven parameters which are:

- **NR** : Number of rules in the rule set (Initial population size)
- **NS** : Number of steps between two iterations of genetic operations.
- **NE** : Number of rules (chromosomes) eliminated in an iteration.
- **BR** : Bid rate.
- **RPR** : Reward / penalty rate.
- **TR** : Life tax rate.
- **MR** : Mutation rate.

Table 6 Factor Levels

NR	NS	NE	BR	RPR	TR	MR
128	400	1	0.0003	0.00030000	0.00000001	0.00001000
416	1000	4	0.001	0.00100000	0.0000001	0.00005000
480	2400		0.008	0.008	0.000001	

For testing ID-A, first, single factor tests are performed where only one parameter is changed while all other parameters are fixed. The results of these tests are available in Appendix C. Then a full factorial design is used with the levels of parameters shown in Table 6. The results of these tests can be seen in Appendix D. The experiment runs have been performed on an IBM PC compatible computer with a P-IV 1.6 GHz processor and 256 Megabytes of memory. Pseudo random number series have been used in all experiments which are obtained using Linear Conruential Generators. All tests are replicable.

In order to find out the effects of the parameters and their interaction on the performance of the algorithm Analysis of Variance (ANOVA) is used. For these analyses General Linear Model option of Minitab 13.30 is utilized. The ANOVA tables, normality and residual vs. fit plots, main effect plots and interaction plots are available in Appendix B.

4.3 RESULTS

In this section we discuss the results of the experiments with KDD-99 data set. The best result obtained from the experiments with ID-A is available in Table 7 and the result obtained from the experiment from ID-B is available in Table 8.

Table 7 The best result obtained from ID-A

NB	NR	NS	NE	BR	RPR	TR	MR	% Acc	% Comp	TrT	TeT
21	416	1000	4	0.001	0.0003	0.000001	0.00005	95.6	89.7	39.0	62.0

Table 8 Result of ID-B

TrT	TeT	% Acc	% Comp
618.4	16.2	95.3	99.3

Both of the models have produced fairly good results. Although some recently published studies slightly outperform our results, an important point to be noted here is that those systems use both knowledge-based and behavior-based models in a hybrid manner where our system consists of only a behavior based model hence such a comparison is not fair. Knowledge-based systems detect known attack types with 100 percent accuracy and completeness and the debate is for the unknown attack types. Both of our models can be integrated to knowledge-based systems making superior results possible.

As stated in the previous section, we have conducted several tests to understand the effects of the parameters and their relation on the performance of the ID-A algorithm. Results of these tests are available in Appendix D as well as the ANOVA analyses in Appendix C. The first thing to be noted from these analyses is that the variation left to error terms is relatively high for all performance measure both in models for main factors and in models containing the interaction. This shows that the changes in the main factors and in the interaction terms are insufficient to explain the variation in the performance measures. However it is still possible to derive some conclusions both from the single factor tests and the ANOVA tables.

Table 18 and Table 19 clearly shows that increasing Mutation Rate and increasing Number of Rules to be Elimininated in each genetic operation step deteriorates both Accuracy and Completeness, In Table 24 it is possible to observe increasing tax rate increases the completeness and limits the accuracy.

Although the variance left to error term is high we also observe that the NS, NE, BR, RPR and the NS-NE, BR-RPR interactions are the significant factors for accuracy. The significant factors for completeness are NR, NS, NE, BR, RPR and NR-BR and NR-RPR interactions.

The most significant factor exlaining the training and testing times (TrT and TeT) is the number of rules (NR) in the rule set (initial population size). This is also intuitively correct because the complexity of the algorithm depends on NR.

One can easily observe that ID-A can achieve very high accuracy and completeness ratios, however the good performance is not steady. This is because the idea of using subsidies conflicts with one of the underlying principles in the Bucket Brigade Algorithm, the rule chains. The algorithm uses the rules in chains to produce the output for an input. However the subsidies are distributed on an individual basis. This may and usually do break the chain structure causing instability and low performance. In the production of an output, all of the activated rules fire their messages, and none of them fire its action except the last activated rule. Therefore action of the rules in a chain is not so important except for the last rule in the chain. However the subsidies are distributed to all rules according to their actions. This causes an improper assessment of the fitness of the rules, instability in the rule set leading to poor results.

ID-B achieves very high accuracy completeness values accompanied with very high training performance. That is because it is somewhat easier to build compact and complete rule set in the absence of negative data especially when the training data are guarantied to be completely normal. The drawback of this algorithm is the low training performance (i.e. long training time) due to the compacting task performed in each step of the training. Complexity of compacting is $O(n^2)$ and is likely to trigger long training period for real systems, however this is in a way tolerable especially if the mechanism is installed on a honey trap (a machine with no specific duty but serving as an attractive target for intruders). A honey trap would have a reduced number of normal states meaning smaller sized problem and configuration changes are unlikely to occur in such systems that is frequent trainings are not expected.

CHAPTER 5

CONCLUSION

In this study we have focused on the design of two behavior-based intrusion detection algorithms for detection of novel attacks when only normal data are available for training.

First we provide a model which uses genetic-based machine learning algorithms as the detection mechanism. This model is a modified version of the model provided by Dasgupta and Gonzales. The idea of using subsidies in the Bucket Brigade Algorithm is original to this study. Our main observation is that for appropriate conditions subsidies may provide means for the creation of strong negative rules, hence allow deduction of knowledge about the abnormal behavior. However, the interference of subsidies with the chain structure of the rules causes a turbulent environment in the rule set where stability is necessary to attain high accuracy and completeness values.

Next we provide another algorithm which uses the same state representation with the previous model to implement an original approach. This mechanism builds up a very compact and relatively complete rule set representing the normal behavior by inserting the normal states into the rule set as rules and combining similar rule pairs when possible. It can attain high accuracy and completeness values using minimal CPU power in the test period. However it requires substantial run time in the training period.

Both models have been coded in C and tested using the 10 percent version of the KDD-99 data set which is the primary data set used to evaluate the intrusion detection tools. The results of the tests have been used in Analysis of Variance (ANOVA) to determine the significance of the factors.

We have observed that the first algorithm can produce rule sets that can detect the intrusion attempts very accurately and completely; however, the performance is not steady and may change considerably between similar parameter sets. Also in ANOVA analyses we have observed that the substantial share of the variance is left to error term meaning that the factors (i.e. the parameters used in the test setup) are not significant in explaining the change in the outcome. This may also be interpreted as an evidence of the turbulent environment in the rule set due to the conflict between the idea of using subsidies and the principle of rule chains.

The second algorithm has only one parameter, the number of bits in the state definition which is constant throughout the experiments. Hence it has been run only once and demonstrated fairly high accuracy and completeness. However this algorithm is very sensitive on the "absence of negative data in training" assumption. If this algorithm is exposed to negative data in the training period the performance will rapidly decline.

Adapting Pitt style classifiers in the first algorithm may be a sound precaution to avoid instability problems although the advantage of using rule chains will be lost but the simplicity may provide better results. Another factor which may be used to increase and stabilize the performance may be a fuzzy control system which may be used in further studies.

REFERENCES

[1] J. P. Anderson. Computer Security Threat Monitoring and Surveillance. Technical Report. James P. Anderson Co., Fort Washington, PA., April 1980.

[2] Cisco Systems. Cisco Secure Intrusion Detection System (NetRanger) Overview. Available at http://www.cisco.com/univercd/cc/td/doc/product/iaabu/csids/csids2/220ug/overview.htm. September 2005.

[3] W. H. Chen, S. H. Hsu and H.-P. Shen, Application of SVM and ANN for intrusion detection, *Computers & Operations Research*, Volume 32, Issue 10, October 2005, Pages 2617-2634.

[4] D. Dasgupta and F. A. Gonzales. An Intelligent Decision Support System for Intrusion Detection and Response. In *Lecture Notes in Computer Science (publisher: Springer-Verlag) as the proceedings of International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security (MMM-ACNS)*, May 21-23, 2001, St. Petersburg, Russia.

[5] H. Debar, M. Dacier, M. Nassehi, and A. Wespi. Fixed vs. variable-length patterns for detecting suspicious process behavior. In Jean-Jacques Quisquater, Yves Deswarte, Catherine Meadows, and Dieter Gollmann, editors, *Computer Security - ESORICS 98*, Vol. 1485 of Lecture Notes in Computer Science, pages 1-16. Springer, 1998.

[6] H. Debar, M. Dacier, and A. Wespi. Reference audit information generation for intrusion detection systems. In Reinhard Posch and György Papp, editors, *Information Systems Security, Proceedings of the 14th International Information Security Conference IFIP SEC'98*, pages 405-417, Vienna, Austria and Budapest, Hungary, 1998. Chapman & Hall.

[7] H. Debar, M. Dacier and A. Wespi. A Revised Taxonomy for Intrusion-Detection Systems. Technical Report. RZ 3176 (#93222) 10/25/99. IBM Research, Zurich Research Office. 1999.

[8] B. de Boer. Classifier Systems: A useful approach to machine learning? Masters thesis. Leiden University. 1994.

[9] D. E. Denning. An Intrusion-Detection Model. *IEEE Transactions on Software Engineering*, 13(2):222-232, 1987.

[10] C. Dowell and P. Ramstedt. The ComputerWatch data reduction tool. *In Proceedings of the 13th National Computer Security Conference*, pages 99-108, Washington, DC, October 1990.

[11] Chief Security Officer Magazine. E-Crime Watch™ Survey. Available at http://www2.csoonline.com/info/release.html?CID=5429. September 2005.

[12] S. Forrest, S. A. Hofmeyr, and A. Somayaji. Computer immunology. *Communications of the ACM*, 40(10):88-96, October 1997.

[13] P. Gallinari, S. Thiria, and F. Fogelman-Soulie. Multilayer perceptrons and data analysis. In *Proceedings of the IEEE Annual International Conference on Neural Networks (ICNN88)*, Vol. I, pages 391-399, San Diego, CA, July 1988.

[14] T. Garvey and T. Lunt. Model-based intrusion detection. In *Proceedings of the 14th National Computer Security Conference*, pages 372-385, October 1991.

[15] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley. 1989.

[16] F. A. Gonzales, D. Dasgupta and R. Kozma. Combining Negative Selection and Classification Techniques for Anomaly Detection. *Journal IEEE Transactions on Evolutionary Computation*. 6:281-291. 2002.

[17] P. Helman and G. Liepins. Statistical foundations of audit trail analysis for the detection of computer misuse. *IEEE Transactions on Software Engineering*, 19:886-901, September 1993.

[18] P. Helman, G. Liepins, and W. Richards. Foundations of intrusion detection. In *Proceedings of the Fifth Computer Security Foundations Workshop*, pages 114-120, Franconic, NH, June 1992.

[19] N. Habra, B. Le Charlier, A. Mounji, and I. Mathieu. Asax: Software architecture and rule-based language for universal audit trail analysis. In Y. Deswarte, G. Eizenberg, and J.-J. Quisquater, editors, *Proceedings of the Second European Symposium on Research in Computer Security (ESORICS)*, Toulouse, France, November 1992, Vol. 648 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, Germany/

[20] K. Ilgun. Ustat: A real-time intrusion detection system for UNIX. In *Proceedings of the 1993 IEEE Symposium on Research in Security and Privacy*, pages 16-28, Oakland, CA, May 1993.

[21] H. S. Javitz and A. Valdes. The SRI IDES statistical anomaly detector. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 316-326, May 1991.

[22] H. S. Javitz, A. Valdez, T. F. Lunt, A. Tamaru, M. Tyson, and J. Lowrance. Next generation intrusion detection expert system (NIDES) –1. Statistical algorithms rationale –2. Rationale for proposed resolver. Technical Report A016 – Rationales, SRI International, 333 Ravenswood Avenue, Menlo Park, CA, March 1993.

[23] University of California, Irvine School of Information and Computer Sciences. KDD Cup 1999 Data. Available at http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. September 2005 .

[24] S. Kumar and E. Spafford. A pattern matching model for misuse intrusion detection. In *Proceedings of the 17th National Computer Security Conference*, pages 11-21, October 1994.

[25] T. F. Lunt and R. Jagannathan. A prototype real-time intrusion-detection expert system. In *Proceedings of the 1988 Symposium on Security and Privacy*, pages 59-66, Oakland, CA, April 1988.

[26] Z. Michalewics. Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag. Third Edition. 1996.

[27] D. Moore. The Spread of Code Red Worm (CRv2). Available at http://www.caida.org/analysis/security/code-red/coderedv2_analysis.xml. September 2005.

[28] D. Moore V. Paxson S. Savage C. Shannon S. Staniford and N. Weaver. The Spread of the Sapphire/Slammer Worm. Available at http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html, September 2005.

[29] P. A. Porras and R. Kemmerer. Penetration state transition analysis - A rule-based intrusion detection approach. In *Proceedings of the Eighth Annual Computer Security Applications Conference*, pages 220-229. IEEE Computer Society Press, November 1992.

[30] P. A. Porras and A. Valdes. Live traffic analysis of TCP/IP gateways. In *Proceedings of the 1998 ISOC Symposium on Network and Distributed System Security (NDSS'98)*, San Diego, CA, March 1998. Internet Society

[31] W. S. Sarle. Neural networks and statistical models. In *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, April, 1994, pages 1538-1550, Cary, NC, April 1994. SAS Institute.

[32] P. Spirakis, S. Katsikas, D. Gritzalis, F. Allegre, J. Darzentas, C. Gigante, D. Karagiannis, P. Kess, H. Putkonen, and T. Spyrou. SECURENET: A network-oriented intelligent intrusion prevention and detection system. *Network Security Journal*, 1(1), November 1994.

[33] T. Spyrou and J. Darzentas. Intention modeling: Approximating computer user intentions for detection and prediction of intrusions. In S.K. Katsikas and D. Gritzalis, editors, *Information Systems Security*, pages 319-335, Samos, Greece, May 1996. Chapman & Hall.

[34] H. S. Vaccaro and G. E. Liepins. Detection of anomalous computer session activity. In *Proceedings of the 1989 IEEE Symposium on Research in Security and Privacy*, pages 280-289, 1989.

[35] D. Vincenzetti and M. Cotrozzi. Atp - Anti tampering program. In *Proceedings of the Fourth USENIX Security Symposium*, pages 79-89, Santa Clara, CA, October 1993.

APPENDIX A

State Definition

Table 9 State Definition

5 Bits (1-5)		4 Bits (6-9)		2 Bits (10-11)	
service		flag		src_bytes	
Bit Stream	Meaning	Bit Stream	Meaning	Bit Stream	Meaning
00000	auth	0000	OTH	00	src_bytes < 20
00001	domain	0001	REJ	01	20 ≤ src_bytes < 500
00010	domain_u	0010	RSTO	10	500 ≤ src_bytes < 1250
00011	eco_i	0011	RSTOS0	11	1250 ≤ src_bytes
00100	ecr_i	0100	RSTR		
00101	finger	0101	S0		
00110	ftp	0110	S1		
00111	ftp_data	0111	S2		
01000	http	1000	S3		
01001	icmp	1001	SF		
01010	IRC	1111	o/w		
01011	link				
01100	ntp_u				
01101	other				
01110	pop_3				
01111	private				
10000	red_i				
10001	remote_job				
10010	shell				
10011	smtp				
10100	ssh				
10101	telnet				
10110	tftp_u				
10111	time				
11000	tim_i				
11001	urh_i				
11010	urp_i				
11011	X11				
11111	o/w				

Table 9 State Definition (Continued)

1 Bit (12)		2 Bits (13-14)	
src_bytes		dst_bytes	
Bit Stream	Meaning	Bit Stream	Meaning
0	src_bytes ≠ 2599	00	dst_bytes < 125
1	src_bytes = 2599	01	125 ≤ dst_bytes < 150
		10	150 ≤ dst_bytes < 250
		11	250 ≤ dst_bytes

Table 9 State Definition (Continued)

1 Bit (15)		1 Bit (16)		1 Bits (17)	
hot		count		srv_count	
Bit Stream	Meaning	Bit Stream	Meaning	Bit Stream	Meaning
0	hot < 1	0	count < 5	0	srv_count < 4
1	hot ≥ 1	1	count ≥ 5	1	srv_count ≥ 4

Table 9 State Definition (Continued)

2 Bits (18-19)	
dst_host_count	
Bit Stream	Meaning
00	dst_host_count < 40
01	40 ≤ dst_host_count < 200
11	200 ≤ dst_host_count

Table 9 State Definition (Continued)

2 Bits (20-21)	
dst_host_same_src_port_rate	
Bit Stream	Meaning
00	dst_host_same_src_port_rate < 0.15
01	0.15 ≤ dst_host_same_src_port_rate < 0.80
10	0.80 ≤ dst_host_same_src_port_rate < 0.95
11	0.95 ≤ dst_host_same_src_port_rate

APPENDIX B

ANOVA Analyses

Table 10 ANOVA Analysis for Main Effects on Accuracy

Analysis of Variance for AC, using Adjusted SS for Tests

Source	DF	Seq SS	Adj SS	Adj MS	F	P
NR	2	143.7	143.7	71.8	0.16	0.856
NS	2	2348.8	2348.8	1174.4	2.54	0.079
NE	1	3870.4	3870.4	3870.4	8.38	0.004
BR	2	2150.7	2150.7	1075.4	2.33	0.098
RPR	2	27790.3	27790.3	13895.2	30.08	0.000
TR	2	790.3	790.3	395.2	0.86	0.425
MR	1	602.4	602.4	602.4	1.30	0.254
Error	959	443057.3	443057.3	462.0		
Total	971	480753.9				

Table 11 ANOVA Analysis for Significant Main Effects and Interactions on Accuracy

Analysis of Variance for AC, using Adjusted SS for Tests

Source	DF	Seq SS	Adj SS	Adj MS	F	P
NS	2	2348.8	2348.8	1174.4	2.60	0.075
NE	1	3870.4	3870.4	3870.4	8.58	0.003
BR	2	2150.7	2150.7	1075.4	2.38	0.093
RPR	2	27790.3	27790.3	13895.2	30.79	0.000
NS*NE	2	4120.4	4120.4	2060.2	4.57	0.011
NS*BR	4	1992.6	1992.6	498.1	1.10	0.353
NS*RPR	4	932.0	932.0	233.0	0.52	0.724
NE*BR	2	761.6	761.6	380.8	0.84	0.430
NE*RPR	2	14.3	14.3	7.2	0.02	0.984
BR*RPR	4	9883.0	9883.0	2470.7	5.48	0.000
Error	946	426889.8	426889.8	451.3		
Total	971	480753.9				

Table 12 ANOVA Analysis Main Effects on Completeness

Analysis of Variance for CO, using Adjusted SS for Tests

Source	DF	Seq SS	Adj SS	Adj MS	F	P
NR	2	7149.8	7149.8	3574.9	5.58	0.004
NS	2	5618.3	5618.3	2809.1	4.38	0.013
NE	1	2749.2	2749.2	2749.2	4.29	0.039
BR	2	8682.2	8682.2	4341.1	6.77	0.001
RPR	2	28812.1	28812.1	14406.0	22.48	0.000
TR	2	2174.5	2174.5	1087.2	1.70	0.184
MR	1	193.5	193.5	193.5	0.30	0.583
Error	959	614598.7	614598.7	640.9		
Total	971	669978.4				

Table 13 ANOVA Analysis for Significant Main Effects and Interactions on Completeness

Analysis of Variance for CO, using Adjusted SS for Tests

Source	DF	Seq SS	Adj SS	Adj MS	F	P
NR	2	7149.8	7149.8	3574.9	5.76	0.003
NS	2	5618.3	5618.3	2809.1	4.53	0.011
NE	1	2749.2	2749.2	2749.2	4.43	0.036
BR	2	8682.2	8682.2	4341.1	7.00	0.001
RPR	2	28812.1	28812.1	14406.0	23.21	0.000
NR*NS	4	1222.7	1222.7	305.7	0.49	0.741
NR*NE	2	1614.5	1614.5	807.2	1.30	0.273
NR*BR	4	3805.2	3805.2	951.3	1.53	0.190
NR*RPR	4	10025.8	10025.8	2506.4	4.04	0.003
NS*NE	2	6802.8	6802.8	3401.4	5.48	0.004
NS*BR	4	4784.6	4784.6	1196.1	1.93	0.104
NS*RPR	4	562.7	562.7	140.7	0.23	0.924
NE*BR	2	1172.9	1172.9	586.4	0.95	0.389
NE*RPR	2	146.0	146.0	73.0	0.12	0.889
BR*RPR	4	9716.7	9716.7	2429.2	3.91	0.004
Error	930	577112.9	577112.9	620.6		
Total	971	669978.4				

Table 14 ANOVA Analysis Main Effects on Training Performance

Analysis of Variance for TrT, using Adjusted SS for Tests

Source	DF	Seq SS	Adj SS	Adj MS	F	P
NR	2	376650	376650	188325	9208.52	0.000
NS	2	129	129	65	3.16	0.043
NE	1	260	260	260	12.69	0.000
BR	2	276	276	138	6.76	0.001
RPR	2	19	19	9	0.46	0.629
TR	2	83	83	41	2.03	0.133
MR	1	7	7	7	0.35	0.556
Error	959	19613	19613	20		
Total	971	397037				

Table 15 ANOVA Analysis for Significant Main Effects and Interactions on Training Performance

Analysis of Variance for TrT, using Adjusted SS for Tests

Source	DF	Seq SS	Adj SS	Adj MS	F	P
NR	2	376650	376650	188325	9659.69	0.000
NS	2	129	129	65	3.32	0.037
NE	1	260	260	260	13.31	0.000
BR	2	276	276	138	7.09	0.001
NR*NS	4	197	197	49	2.53	0.039
NR*NE	2	88	88	44	2.26	0.105
NR*BR	4	613	613	153	7.86	0.000
NS*NE	2	11	11	5	0.27	0.760
NS*BR	4	282	282	70	3.61	0.006
NE*BR	2	88	88	44	2.25	0.106
Error	946	18443	18443	19		
Total	971	397037				

Table 16 ANOVA Analysis Main Effects on Test Performance

Analysis of Variance for TeT, using Adjusted SS for Tests

Source	DF	Seq SS	Adj SS	Adj MS	F	P
NR	2	3198550	3198550	1599275	1332.64	0.000
NS	2	19071	19071	9536	7.95	0.000
NE	1	9795	9795	9795	8.16	0.004
BR	2	2638	2638	1319	1.10	0.334
RPR	2	5549	5549	2774	2.31	0.100
TR	2	1027	1027	514	0.43	0.652
MR	1	1649	1649	1649	1.37	0.241
Error	959	1150877	1150877	1200		
Total	971	4389156				

Table 17 ANOVA Analysis for Significant Main Effects and Interactions on Test Performance

Analysis of Variance for TeT, using Adjusted SS for Tests

Source	DF	Seq SS	Adj SS	Adj MS	F	P
NR	2	3198550	3198550	1599275	1328.74	0.000
NS	2	19071	19071	9536	7.92	0.000
NE	1	9795	9795	9795	8.14	0.004
NR*NS	4	5666	5666	1416	1.18	0.319
NR*NE	2	1520	1520	760	0.63	0.532
NS*NE	2	1499	1499	749	0.62	0.537
Error	958	1153055	1153055	1204		
Total	971	4389156				

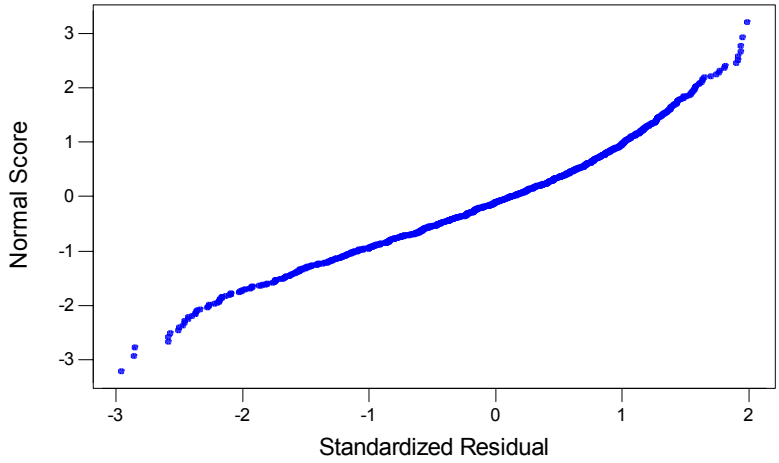

Figure 14 Normal Probability Plot of ANOVA Analysis for Main Effects on Accuracy

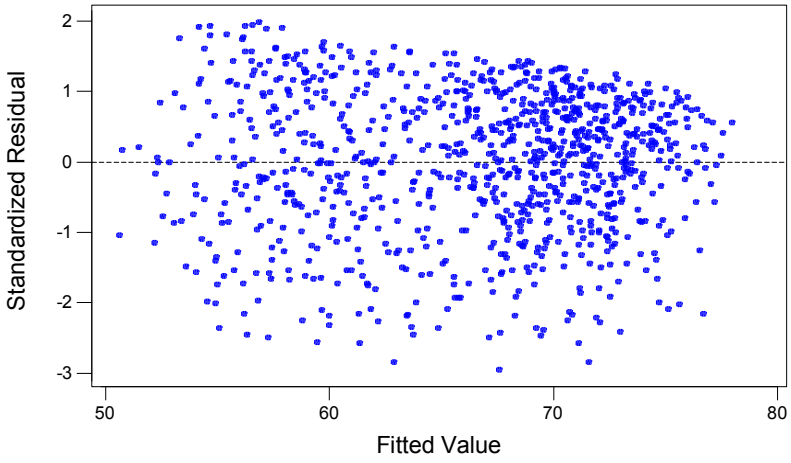

Figure 15 Residuals vs Fit Plot of ANOVA Analysis for Main Effects on Accuracy

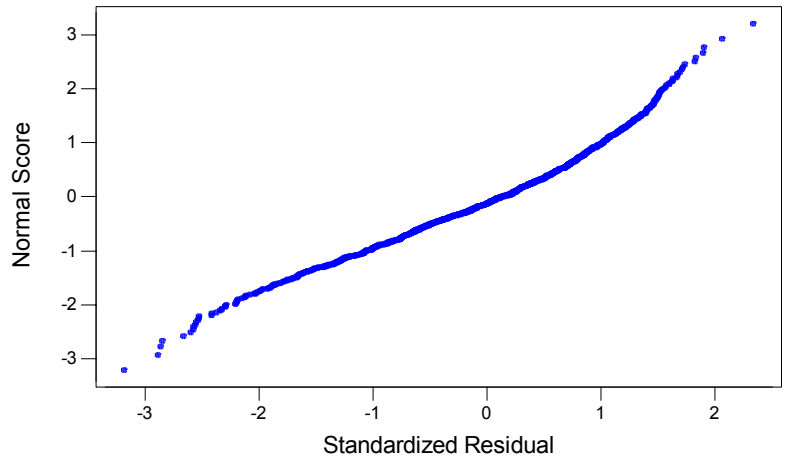

Figure 16 Normal Probability Plot of ANOVA Analysis for Significant Main Effects and Interactions on Accuracy

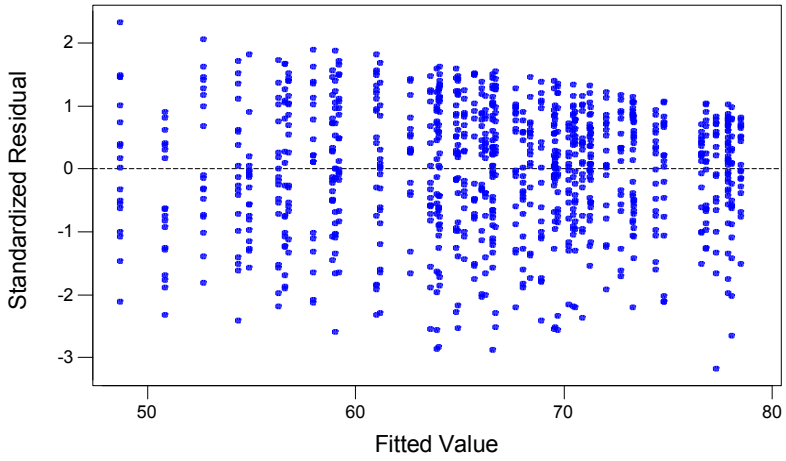

Figure 17 Residuals vs Fit Plot of ANOVA Analysis for Significant Main Effects and Interactions on Accuracy

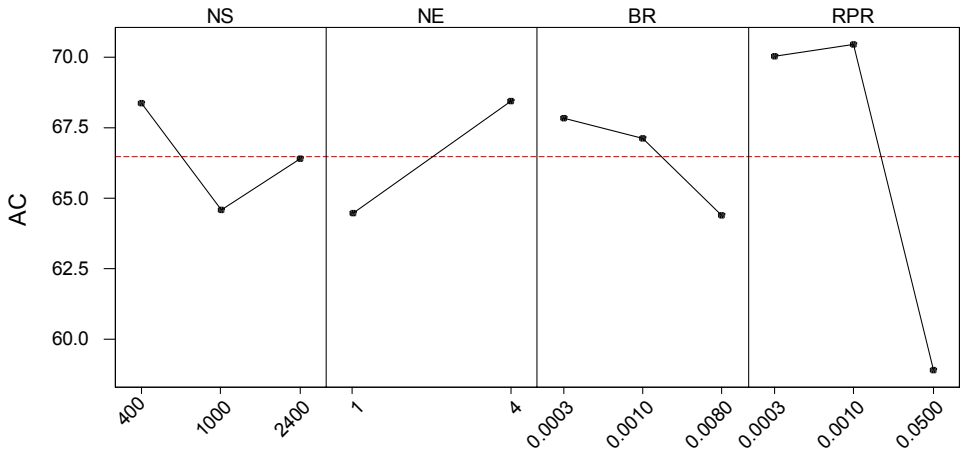

Figure 18 Plot of Significant Main Effects on Accuracy

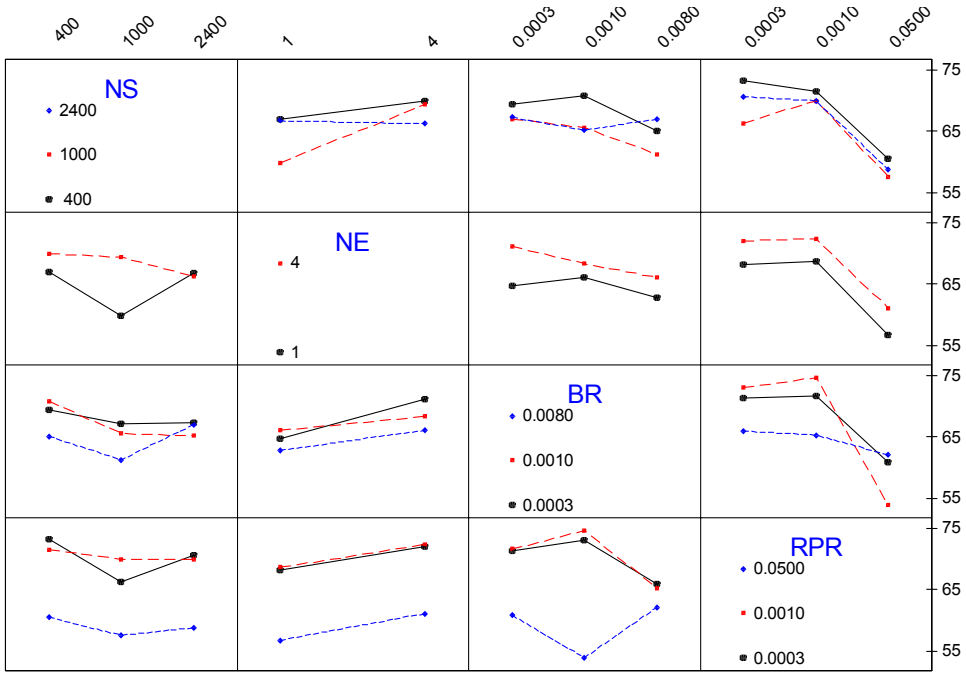

Figure 19 Plot of Effects of Interactions on Accuracy

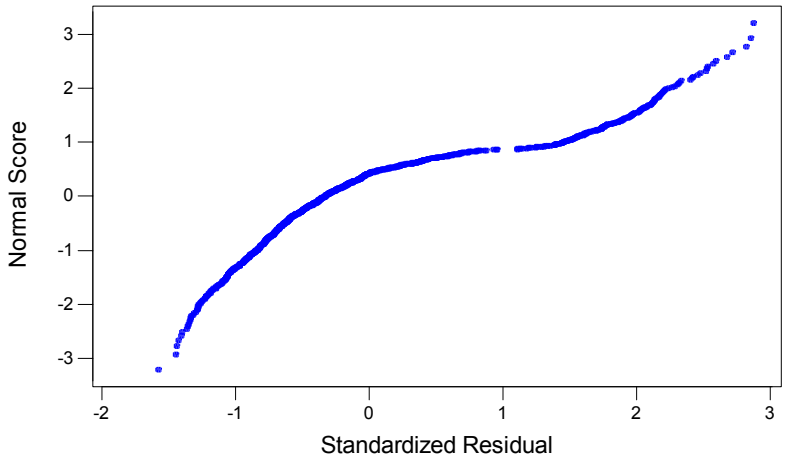

Figure 20 Normal Probability Plot of ANOVA Analysis for Main Effects on Completeness

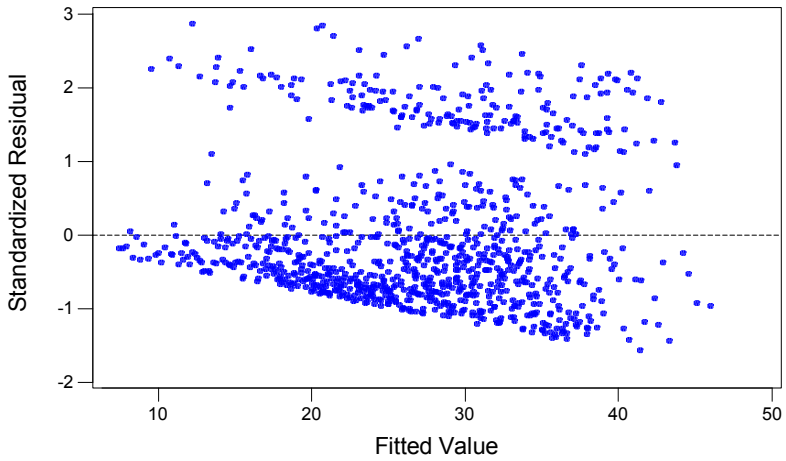

Figure 21 Residuals vs Fit Plot of ANOVA Analysis for Main Effects on Completeness

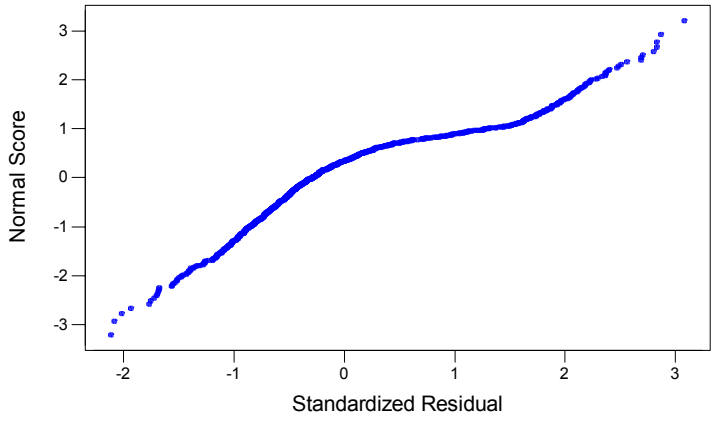

Figure 22 Normal Probability Plot of ANOVA Analysis for Significant Main Effects and Interactions on Completeness

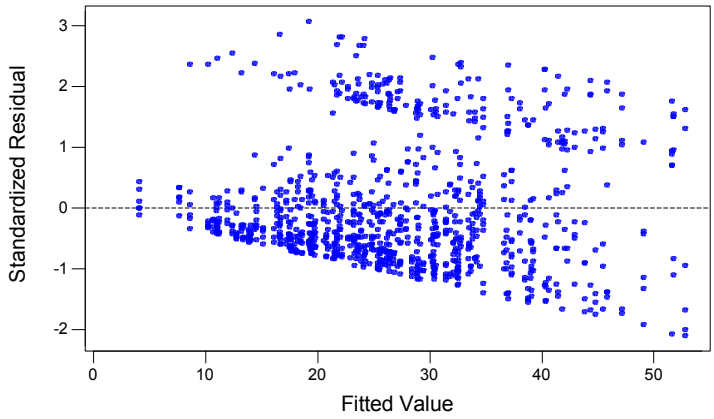

Figure 23 Residuals vs Fit Plot of ANOVA Analysis for Significant Main Effects and Interactions on Accuracy

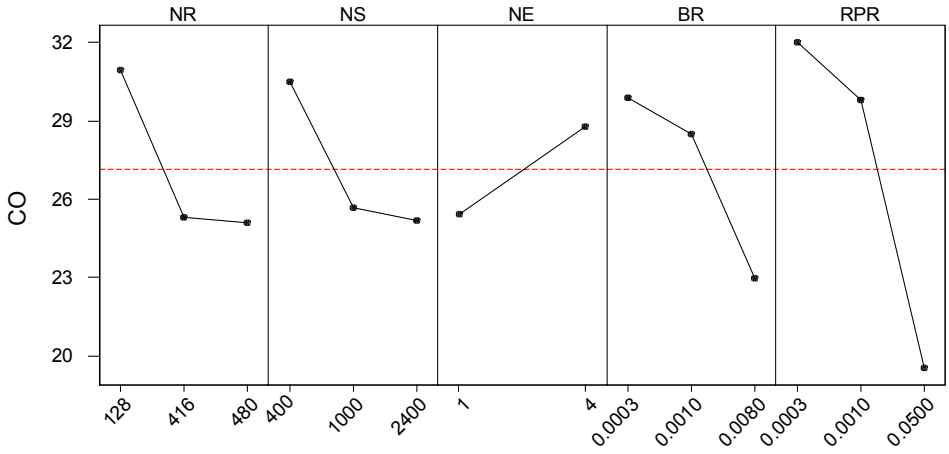

Figure 24 Plot of Significant Main Effects on Completeness

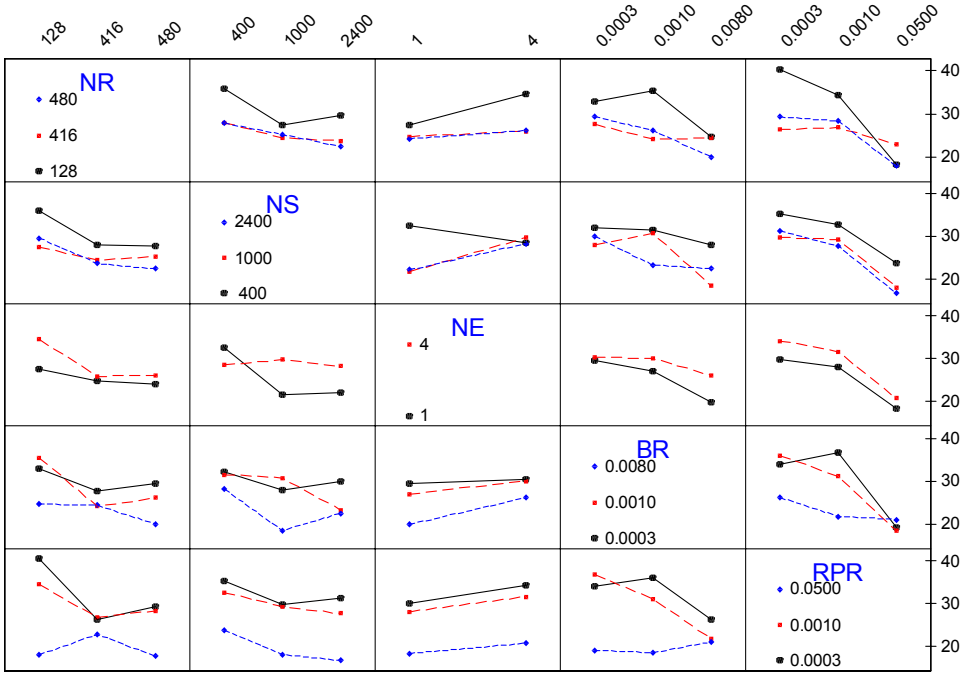

Figure 25 Plot of Effects of Interactions on Completeness

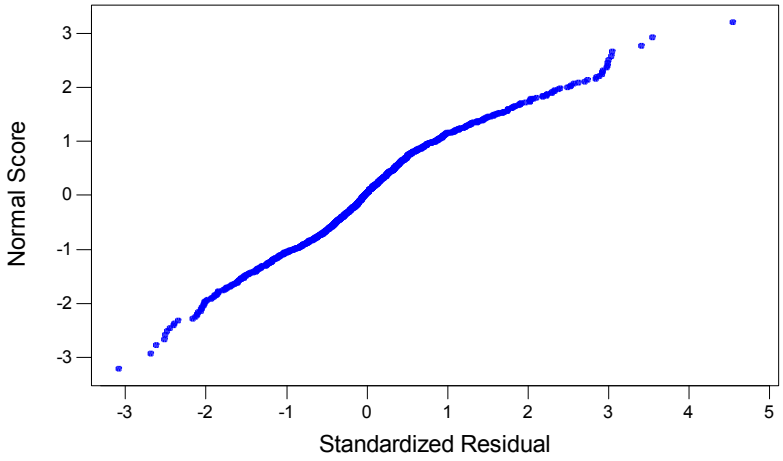

Figure 26 Normal Probability Plot of ANOVA Analysis for Main Effects on Training Performance

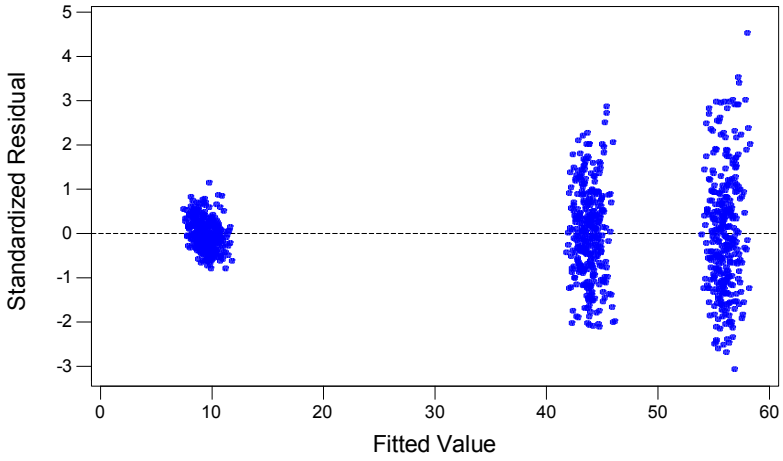

Figure 27 Residuals vs Fit Plot of ANOVA Analysis for Main Effects on Training Performance

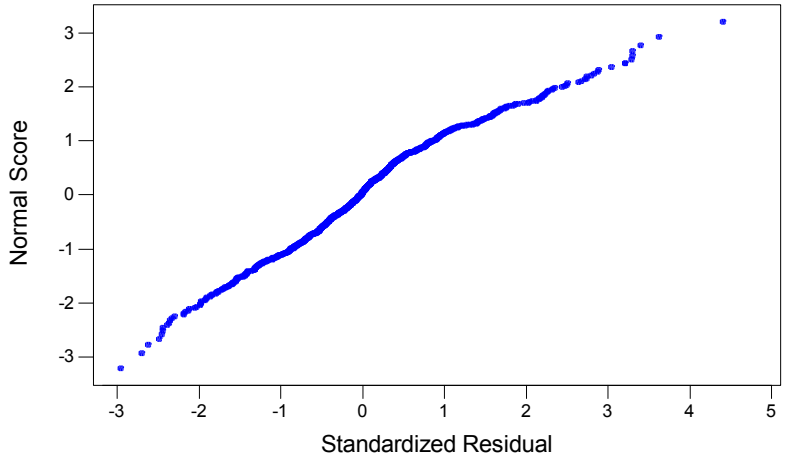

Figure 28 Normal Probability Plot of ANOVA Analysis for Significant Main Effects and Interactions on Training Performance

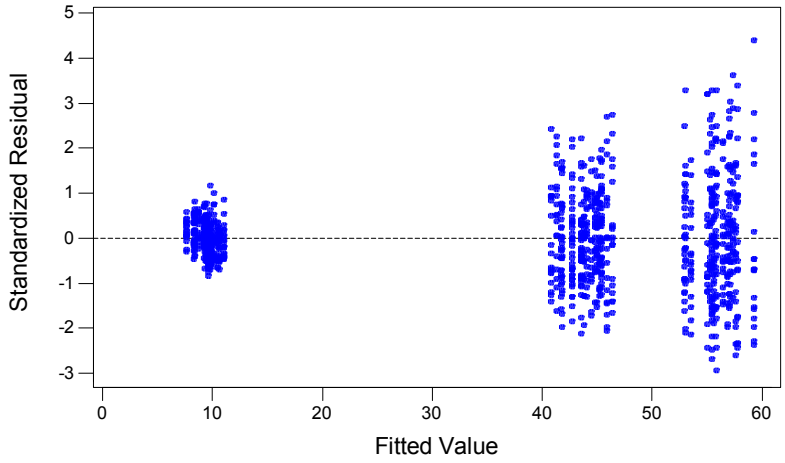

Figure 29 Residuals vs Fit Plot of ANOVA Analysis for Significant Main Effects and Interactions on Training Performance

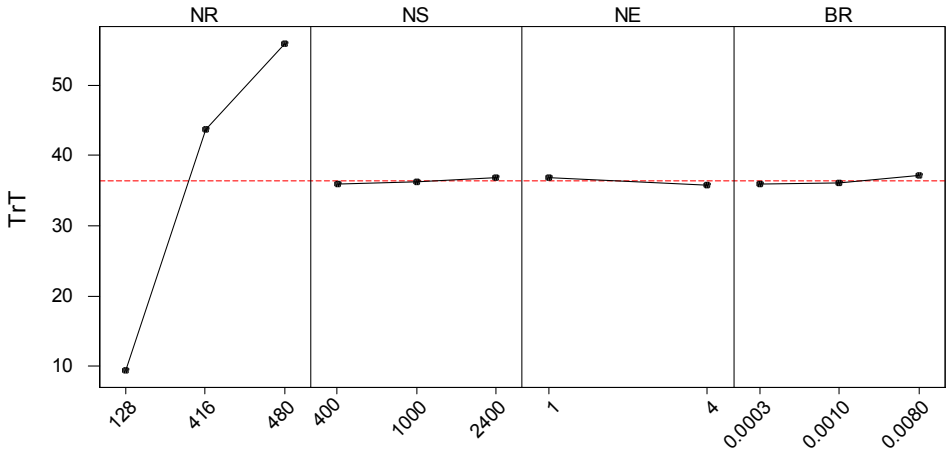

Figure 30 Plot of Significant Main Effects on Training Time

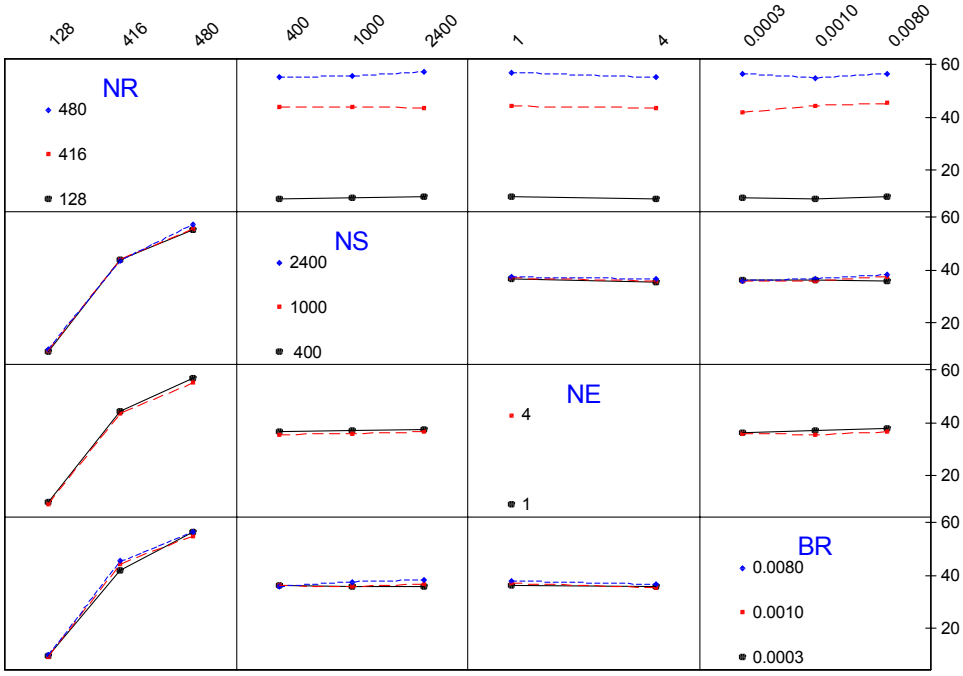

Figure 31 Plot of Effects of Interactions on Training Time

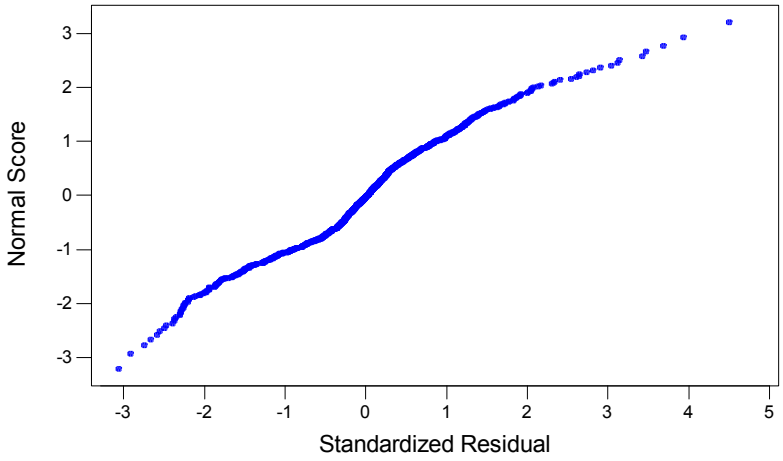

Figure 32 Normal Probability Plot of ANOVA Analysis for Main Effects on Test Performance

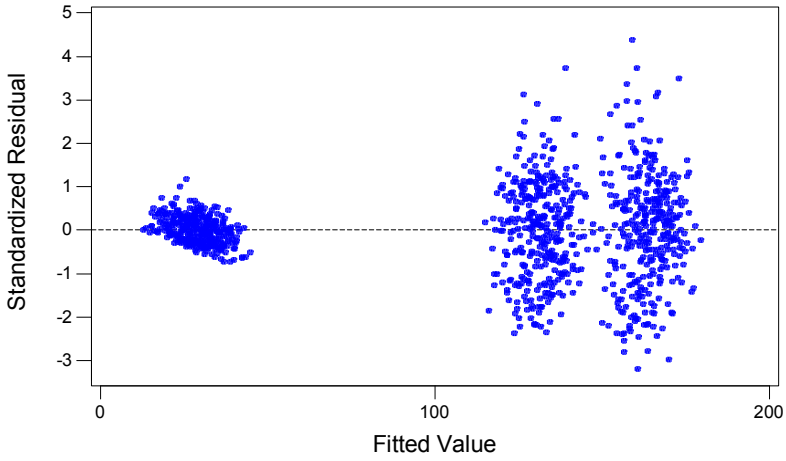

Figure 33 Residuals vs Fit Plot of ANOVA Analysis for Main Effects on Test Performance

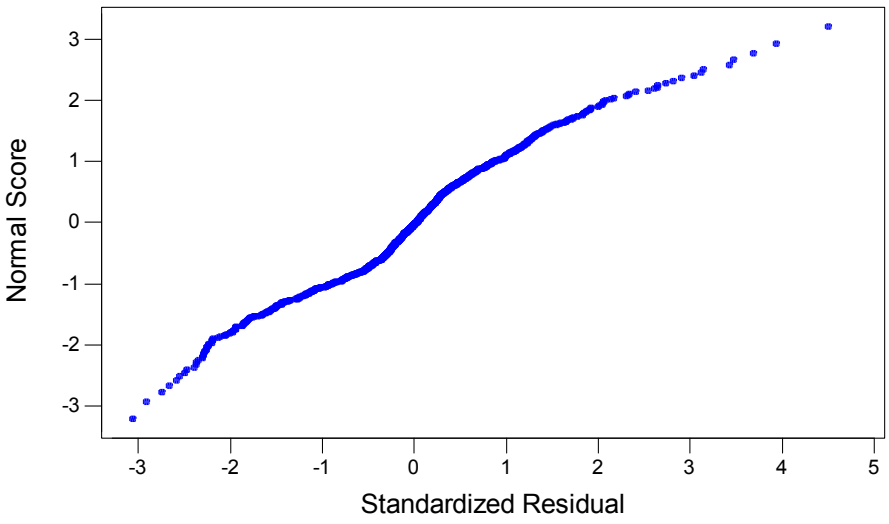

Figure 34 Normal Probability Plot of ANOVA Analysis for Significant Main Effects and Interactions on Test Performance

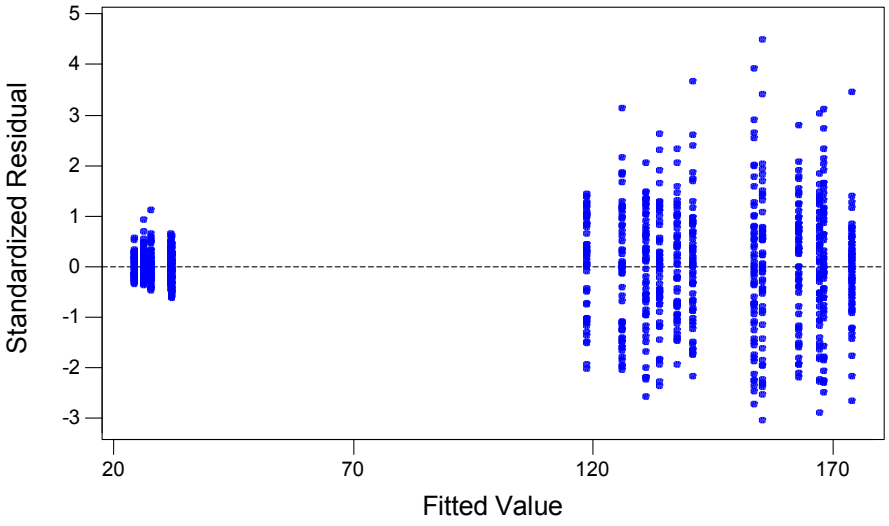

Figure 35 Residuals vs Fit Plot of ANOVA Analysis for Significant Main Effects and Interactions on Test Performance

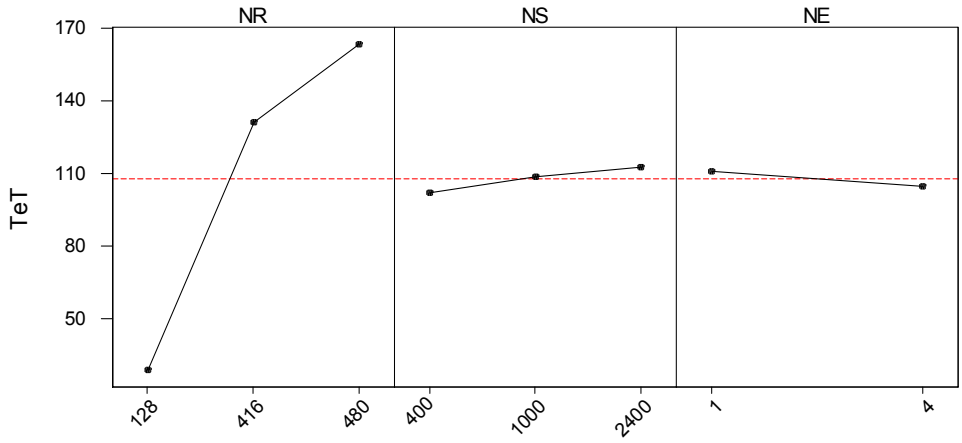

Figure 36 Plot of Significant Main Effects on Test Performance

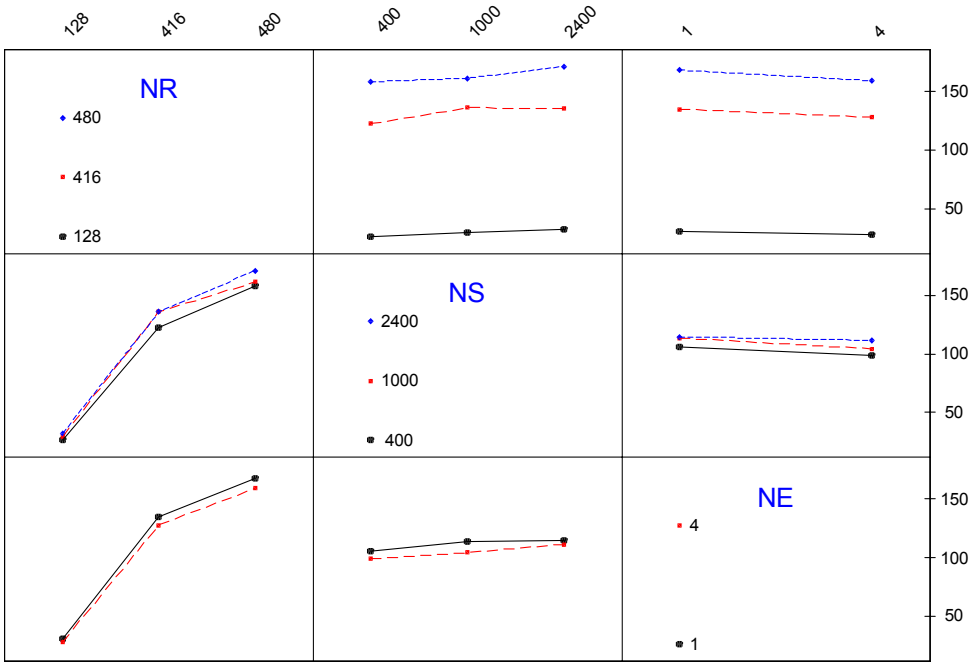

Figure 37 Plot of Effects of Interactions on Test Performance

APPENDIX C

Single Factor Test Results

Table 18 Test Results for changing MR, where NR = 128, NS = 100, NE = 1, BR = 0.008 RPR = 0.001, TR = 0.00000001

MR	% Acc	% Comp	TrT	TeT
0.0000000005	92.4	97.1	6.5	14.0
0.000000001	92.4	97.1	6.5	13.8
0.000000005	92.4	97.1	6.5	14.0
0.00000001	92.4	97.1	6.6	14.0
0.00000005	92.4	97.1	6.7	14.1
0.0000001	92.4	97.1	6.9	14.3
0.0000005	92.4	97.1	6.9	14.2
0.000001	92.4	97.1	7.0	14.3
0.000005	92.3	97.1	7.0	14.4
0.00001	93.0	99.3	7.1	12.0
0.00005	90.3	95.0	7.5	12.8
0.0001	56.2	9.1	8.2	29.3
0.0005	59.3	24.2	7.8	28.0
0.001	13.7	2.0	9.2	44.4
0.005	86.3	92.4	7.8	13.3
0.01	74.1	31.1	8.1	22.5
0.05	85.3	99.1	5.5	10.2

Table 19 Test Results for changing NE, where NR = 128, NS = 100, BR = 0.008 RPR = 0.001, TR = 0.00000001, MR = 0.00001

NE	% Acc	% Comp	TrT	TeT
1	93.1	96.4	8.5	11.5
2	94.4	22.0	8.6	35.2
3	82.0	11.9	7.7	24.7
4	92.0	93.3	8.5	16.0
5	83.2	71.8	10.7	22.3
6	22.7	0.6	11.9	30.4
7	96.5	75.5	9.2	22.4
8	65.0	3.1	10.8	32.5
9	71.7	12.5	12.9	43.7
10	47.3	3.4	9.8	22.9

Table 20 Test Results for changing NR, where NS = 100, NE = 1, BR = 0.001, RPR = 0.001, TR = 0.00000001, MR = 0.00001

NR	% Acc	% Comp	TrT	TeT
64	96.3	68.1	3.6	9.5
96	95.7	68.0	7.0	14.2
128	93.1	96.4	8.5	11.4
160	87.2	15.3	12.0	42.6
192	54.3	5.9	14.1	42.6
224	95.4	25.3	20.8	53.7
256	85.1	3.4	20.4	81.8
288	74.5	8.6	29.6	98.7
320	71.7	18.9	30.1	73.4
352	72.9	15.9	37.6	126.4
384	70.5	21.2	38.5	135.4
416	97.1	84.5	52.8	65.5
448	92.6	25.7	63.2	181.5
480	91.1	72.1	77.6	78.7
512	89.2	34.7	51.8	179.8

Table 21 Test Results for changing NS, where NR = 128, NE = 1, BR = 0.001 RPR = 0.001, TR = 0.00000001, MR = 0.00001

NS	% Acc	% Comp	TrT	TeT
200	98.7	81.5	8.9	20.0
400	98.4	67.3	7.4	20.8
600	94.0	39.1	10.4	45.4
800	60.4	17.8	6.9	37.9
1000	81.6	72.7	8.2	20.2
1200	14.3	2.8	8.2	19.3
1400	18.5	4.3	8.3	20.8
1600	31.8	1.3	9.5	26.2
1800	70.2	0.6	7.1	34.3
2000	72.0	12.2	9.8	51.6
2200	13.0	2.6	10.7	30.8
2400	91.7	69.9	8.1	29.8
2600	34.7	8.5	7.7	23.0
2800	30.3	6.0	8.4	27.1
3000	48.2	6.4	8.8	37.3

Table 22 Test Results for changing BR, where NR = 128, NS = 100, NE = 1, RPR = 0.001, TR = 0.00000001, MR = 0.00001

BR	% Acc	% Comp	TrT	TeT
0.0001	89.2	2.1	10.8	33.6
0.0002	65.9	7.2	10.4	50.3
0.0003	84.7	92.0	8.4	13.2
0.0004	36.9	8.0	10.9	27.8
0.0005	97.7	68.4	9.3	22.5
0.0006	74.8	8.7	8.5	23.6
0.0007	96.4	69.1	9.4	18.6
0.0008	93.3	72.8	10.7	16.2
0.0009	78.9	40.5	9.0	36.7
0.001	93.1	96.4	9.1	11.9
0.002	60.0	30.4	12.3	16.9
0.003	84.5	66.4	8.5	20.2
0.004	65.9	18.9	12.3	32.6
0.005	67.5	10.9	13.2	27.1
0.006	91.6	20.5	9.6	26.2
0.007	79.5	23.3	10.5	25.2
0.008	93.0	99.3	6.5	11.5
0.009	94.6	95.9	9.1	14.3
0.01	74.7	27.8	8.7	20.1
0.02	95.3	71.9	9.5	13.8
0.03	37.8	11.1	8.0	24.9
0.04	97.5	69.3	8.6	17.7
0.05	39.1	8.7	10.0	18.5
0.06	79.2	70.6	8.2	12.9
0.07	37.2	12.8	8.7	24.8
0.08	76.1	76.4	9.4	16.2
0.09	76.4	75.8	9.9	15.8
0.1	44.8	10.7	8.2	30.3

Table 23 Test Results for changing RPR, where NR = 128, NS = 100, NE = 1, BR = 0.008, TR = 0.00000001, MR = 0.00001

RPR	% Acc	% Comp	TrT	TeT
0.0001	88.3	76.5	8.0	18.5
0.0002	68.5	4.7	9.0	24.4
0.0003	81.3	92.2	6.8	12.0
0.0004	82.7	13.6	10.2	25.8
0.0005	68.9	28.0	7.3	31.6
0.0006	75.2	4.4	9.4	22.3
0.0007	75.6	6.3	10.2	43.5
0.0008	84.5	30.3	10.1	29.6
0.0009	83.0	70.2	10.9	18.9
0.001	93.0	99.3	7.0	12.0
0.002	73.3	29.8	10.2	29.9
0.003	73.8	8.1	11.7	30.9
0.004	62.8	1.1	11.7	26.4
0.005	82.3	28.8	9.9	33.8
0.006	71.5	29.6	9.6	35.2
0.007	66.3	46.5	8.4	29.8
0.008	30.5	4.6	10.0	27.1
0.009	94.6	14.1	10.0	25.4
0.01	94.5	71.6	10.9	16.9
0.02	98.8	72.9	10.7	18.7
0.03	12.8	1.5	8.2	50.2
0.04	89.9	2.8	10.8	27.3
0.05	94.5	94.8	11.0	13.2
0.06	89.8	28.9	11.2	33.3
0.07	85.5	67.1	9.1	18.4
0.08	94.9	72.8	11.3	14.7
0.09	19.7	2.0	12.3	55.1
0.1	90.9	69.0	9.5	18.5

Table 24 Test Results for changing TR, where NR = 128, NS = 100, NE = 1, BR = 0.001, RPR = 0.001, MR = 0.00001

TR	% Acc	% Comp	TrT	TeT
0.00000000	63.5	8.9	7.1	29.8
0.00000001	63.5	8.9	7.3	29.7
0.00000001	93.0	99.3	6.5	11.6
0.00000005	80.8	31.9	8.6	27.6
0.00000010	86.1	78.8	7.2	12.9
0.00000050	79.2	27.5	11.7	38.2
0.00000100	84.2	90.7	8.2	14.2
0.00000500	84.5	27.7	10.2	23.8
0.00001000	73.3	35.6	7.7	22.5
0.00005000	77.4	35.8	8.1	34.7
0.00010000	80.5	100.0	5.2	10.0
0.00050000	80.5	100.0	7.8	15.2
0.00100000	80.5	100.0	6.6	11.2
0.00500000	80.5	100.0	7.4	10.5

APPENDIX D

Full Factorial Test Results

NR : Number of rules in the rule set (Initial population size)

NS : Number of steps between two iterations of genetic operations.

NE : Number of rules (chromosomes) eliminated in an iteration.

BR : Bid rate.

RPR : Reward / penalty rate.

TR : Life tax rate.

MR : Mutation rate.

% Acc : Percent accuracy.

% Comp : Percent completeness.

TrT : Training time.

Tet : Testing time.

Table 25 Test Results for ID-A, where NR= 128, NS= 400, NE= 1, BR= 0.0003.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	91.9	10.0	11.0	66.0
0.0003	0.00000001	0.00005	83.3	55.1	10.9	39.1
0.0003	0.0000001	0.00001	83.3	92.9	7.0	12.4
0.0003	0.0000001	0.00005	84.7	96.5	7.2	12.5
0.0003	0.000001	0.00001	60.1	11.9	8.1	23.6
0.0003	0.000001	0.00005	60.1	11.9	8.1	23.7
0.001	0.00000001	0.00001	16.7	1.8	8.5	21.9
0.001	0.00000001	0.00005	73.6	10.1	11.2	46.3
0.001	0.0000001	0.00001	78.9	72.5	7.0	13.0
0.001	0.0000001	0.00005	52.2	6.9	8.1	28.0
0.001	0.000001	0.00001	75.7	27.7	14.9	43.5
0.001	0.000001	0.00005	90.9	76.6	9.3	17.4
0.05	0.00000001	0.00001	98.8	11.2	9.7	38.6
0.05	0.00000001	0.00005	4.6	0.2	9.3	29.8
0.05	0.0000001	0.00001	59.2	10.6	9.3	30.1
0.05	0.0000001	0.00005	81.2	68.0	9.7	19.9
0.05	0.000001	0.00001	37.0	4.9	9.0	30.3
0.05	0.000001	0.00005	44.8	9.6	11.1	44.0

Table 26 Test Results for ID-A, where NR= 128, NS= 400, NE= 1, BR= 0.001.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	66.4	0.8	11.5	35.5
0.0003	0.00000001	0.00005	93.8	73.4	10.8	18.7
0.0003	0.0000001	0.00001	92.4	68.7	11.9	22.4
0.0003	0.0000001	0.00005	89.0	74.6	11.8	20.3
0.0003	0.000001	0.00001	89.8	68.7	8.1	27.4
0.0003	0.000001	0.00005	81.9	94.8	7.5	11.6
0.001	0.00000001	0.00001	98.4	67.3	7.3	20.9
0.001	0.00000001	0.00005	98.9	73.1	6.8	17.7
0.001	0.0000001	0.00001	79.2	32.1	7.1	18.5
0.001	0.0000001	0.00005	63.4	37.4	6.0	45.2
0.001	0.000001	0.00001	58.6	30.8	6.3	19.2
0.001	0.000001	0.00005	56.5	12.1	6.9	28.8
0.05	0.00000001	0.00001	52.3	6.1	9.4	38.1
0.05	0.00000001	0.00005	45.2	6.0	9.5	27.9
0.05	0.0000001	0.00001	10.5	0.8	7.8	32.5
0.05	0.0000001	0.00005	23.4	6.6	7.4	26.3
0.05	0.000001	0.00001	85.1	67.5	9.7	18.8
0.05	0.000001	0.00005	78.1	25.6	9.1	27.6

Table 27 Test Results for ID-A, where NR= 128, NS= 400, NE= 1, BR= 0.008.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	82.6	94.7	8.7	22.2
0.0003	0.00000001	0.00005	13.8	2.8	8.4	36.5
0.0003	0.0000001	0.00001	94.4	71.1	11.6	16.9
0.0003	0.0000001	0.00005	93.4	67.9	10.7	25.5
0.0003	0.000001	0.00001	94.8	84.6	8.7	14.8
0.0003	0.000001	0.00005	18.7	0.3	9.1	33.6
0.001	0.00000001	0.00001	87.9	1.0	11.0	32.5
0.001	0.00000001	0.00005	51.1	3.9	11.3	33.8
0.001	0.0000001	0.00001	4.4	0.3	9.3	30.4
0.001	0.0000001	0.00005	37.7	5.8	8.8	21.8
0.001	0.000001	0.00001	90.1	71.8	10.4	18.4
0.001	0.000001	0.00005	45.2	16.7	8.0	29.0
0.05	0.00000001	0.00001	88.9	70.0	11.3	18.0
0.05	0.00000001	0.00005	54.8	4.2	10.5	26.2
0.05	0.0000001	0.00001	71.5	0.3	11.2	50.1
0.05	0.0000001	0.00005	92.5	89.7	7.7	13.6
0.05	0.000001	0.00001	87.4	66.6	10.1	25.3
0.05	0.000001	0.00005	80.3	1.3	10.6	48.2

Table 28 Test Results for ID-A, where NR= 128, NS= 400, NE= 4, BR= 0.0003.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	63.8	1.8	8.9	26.4
0.0003	0.00000001	0.00005	85.3	74.7	8.5	15.9
0.0003	0.0000001	0.00001	44.8	7.0	8.9	18.7
0.0003	0.0000001	0.00005	91.8	37.9	10.0	32.6
0.0003	0.000001	0.00001	81.9	21.7	8.2	19.2
0.0003	0.000001	0.00005	81.9	21.7	8.2	19.1
0.001	0.00000001	0.00001	97.2	88.0	9.3	18.2
0.001	0.00000001	0.00005	97.5	68.9	9.1	18.7
0.001	0.0000001	0.00001	89.4	25.8	8.5	19.2
0.001	0.0000001	0.00005	90.5	57.0	7.6	34.2
0.001	0.000001	0.00001	77.4	33.3	8.5	26.3
0.001	0.000001	0.00005	89.0	67.7	7.7	15.3
0.05	0.00000001	0.00001	61.2	2.7	10.1	27.8
0.05	0.00000001	0.00005	86.3	26.5	9.1	18.7
0.05	0.0000001	0.00001	17.0	3.1	8.5	30.4
0.05	0.0000001	0.00005	82.7	33.7	10.1	30.7
0.05	0.000001	0.00001	80.2	25.7	8.7	18.7
0.05	0.000001	0.00005	98.5	8.7	10.5	22.6

Table 29 Test Results for ID-A, where NR= 128, NS= 400, NE= 4, BR= 0.001.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	89.2	24.7	10.5	30.7
0.0003	0.00000001	0.00005	92.4	89.6	11.9	12.5
0.0003	0.0000001	0.00001	92.9	88.7	7.5	13.8
0.0003	0.0000001	0.00005	93.4	88.4	9.2	15.3
0.0003	0.000001	0.00001	78.2	75.2	7.1	43.6
0.0003	0.000001	0.00005	95.7	31.3	8.2	33.2
0.001	0.00000001	0.00001	94.7	65.7	9.8	17.2
0.001	0.00000001	0.00005	95.6	73.6	11.2	18.6
0.001	0.0000001	0.00001	90.7	50.8	8.8	28.7
0.001	0.0000001	0.00005	92.5	90.3	9.0	13.4
0.001	0.000001	0.00001	92.3	25.9	8.8	34.5
0.001	0.000001	0.00005	76.3	20.7	9.1	29.6
0.05	0.00000001	0.00001	62.7	26.3	10.8	29.8
0.05	0.00000001	0.00005	62.7	26.3	10.8	29.8
0.05	0.0000001	0.00001	92.1	18.9	6.6	19.5
0.05	0.0000001	0.00005	87.2	9.8	6.6	21.5
0.05	0.000001	0.00001	79.7	78.2	8.3	16.2
0.05	0.000001	0.00005	14.4	1.9	8.0	33.2

Table 30 Test Results for ID-A, where NR= 128, NS= 400, NE= 4, BR= 0.008.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	18.4	3.0	6.5	18.9
0.0003	0.00000001	0.00005	78.0	80.7	6.4	13.0
0.0003	0.0000001	0.00001	32.6	2.6	8.4	26.3
0.0003	0.0000001	0.00005	47.9	10.3	7.9	18.6
0.0003	0.000001	0.00001	46.5	8.5	9.1	26.8
0.0003	0.000001	0.00005	76.0	33.2	7.9	15.3
0.001	0.00000001	0.00001	73.6	2.5	9.6	42.0
0.001	0.00000001	0.00005	75.7	4.9	8.7	31.0
0.001	0.0000001	0.00001	59.1	25.7	9.5	36.0
0.001	0.0000001	0.00005	91.2	78.7	7.9	16.5
0.001	0.000001	0.00001	23.4	3.8	8.1	33.4
0.001	0.000001	0.00005	93.1	25.6	9.2	24.2
0.05	0.00000001	0.00001	46.5	1.1	7.4	34.0
0.05	0.00000001	0.00005	10.2	0.6	8.3	28.7
0.05	0.0000001	0.00001	54.1	23.0	10.2	21.0
0.05	0.0000001	0.00005	56.7	7.2	7.7	23.7
0.05	0.000001	0.00001	89.3	27.2	8.8	23.5
0.05	0.000001	0.00005	89.9	66.9	9.2	15.7

Test Results for ID-A, where NR= 128, NS= 1000, NE= 1, BR= 0.0003.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	35.1	9.0	10.2	29.4
0.0003	0.00000001	0.00005	73.6	21.5	12.3	45.9
0.0003	0.0000001	0.00001	92.8	70.1	10.2	16.1
0.0003	0.0000001	0.00005	92.0	72.6	11.2	17.5
0.0003	0.000001	0.00001	27.7	8.2	7.3	20.9
0.0003	0.000001	0.00005	74.2	20.9	9.0	34.3
0.001	0.00000001	0.00001	24.2	4.3	9.5	29.2
0.001	0.00000001	0.00005	31.4	7.1	8.6	32.3
0.001	0.0000001	0.00001	72.3	38.9	9.0	31.6
0.001	0.0000001	0.00005	36.8	1.7	8.6	30.3
0.001	0.000001	0.00001	50.5	13.4	8.3	36.1
0.001	0.000001	0.00005	76.3	8.6	8.8	25.5
0.05	0.00000001	0.00001	20.5	4.4	10.8	27.4
0.05	0.00000001	0.00005	62.3	5.8	9.5	35.9
0.05	0.0000001	0.00001	86.4	66.3	9.3	19.9
0.05	0.0000001	0.00005	3.8	0.2	9.3	35.8
0.05	0.000001	0.00001	55.6	15.1	9.6	33.5
0.05	0.000001	0.00005	33.8	10.0	9.5	22.7

Table 31 Test Results for ID-A, where NR= 128, NS= 1000, NE= 1, BR= 0.001.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	55.9	12.2	8.7	28.0
0.0003	0.00000001	0.00005	87.7	44.7	8.9	26.0
0.0003	0.0000001	0.00001	85.2	82.7	11.2	24.0
0.0003	0.0000001	0.00005	33.5	7.3	8.9	28.2
0.0003	0.000001	0.00001	97.8	66.0	9.5	17.4
0.0003	0.000001	0.00005	97.0	67.9	9.2	14.7
0.001	0.00000001	0.00001	81.6	72.7	8.2	20.1
0.001	0.00000001	0.00005	50.4	13.9	7.8	40.6
0.001	0.0000001	0.00001	80.2	70.2	7.6	13.6
0.001	0.0000001	0.00005	79.0	68.8	9.6	13.2
0.001	0.000001	0.00001	60.1	31.7	7.7	46.4
0.001	0.000001	0.00005	43.2	13.7	7.3	40.1
0.05	0.00000001	0.00001	35.5	6.3	8.2	20.5
0.05	0.00000001	0.00005	26.1	6.9	8.2	19.9
0.05	0.0000001	0.00001	17.8	0.4	12.9	33.6
0.05	0.0000001	0.00005	57.2	23.0	9.6	27.9
0.05	0.000001	0.00001	37.2	5.6	9.9	24.2
0.05	0.000001	0.00005	52.3	3.3	11.2	24.7

Table 32 Test Results for ID-A, where NR= 128, NS= 1000, NE= 1, BR= 0.008.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	91.8	13.8	10.6	41.5
0.0003	0.00000001	0.00005	78.8	23.7	11.1	29.7
0.0003	0.0000001	0.00001	73.5	19.1	10.9	28.0
0.0003	0.0000001	0.00005	40.6	2.5	11.5	51.3
0.0003	0.000001	0.00001	21.1	4.2	10.5	24.6
0.0003	0.000001	0.00005	91.7	20.3	11.3	38.1
0.001	0.00000001	0.00001	41.8	15.1	8.4	38.4
0.001	0.00000001	0.00005	57.6	1.3	8.5	39.4
0.001	0.0000001	0.00001	58.0	0.8	10.4	24.8
0.001	0.0000001	0.00005	95.3	13.3	9.5	53.2
0.001	0.000001	0.00001	82.8	6.9	7.6	21.0
0.001	0.000001	0.00005	24.7	5.6	10.3	40.6
0.05	0.00000001	0.00001	28.3	4.1	8.5	29.9
0.05	0.00000001	0.00005	27.7	4.1	8.3	31.6
0.05	0.0000001	0.00001	48.7	11.6	10.7	26.8
0.05	0.0000001	0.00005	59.3	14.5	9.9	41.7
0.05	0.000001	0.00001	43.0	6.8	11.3	33.3
0.05	0.000001	0.00005	30.9	1.2	9.7	39.3

Table 33 Test Results for ID-A, where NR=128, NS=1000, NE=4, BR=0.0003.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	64.5	5.9	7.3	36.0
0.0003	0.00000001	0.00005	54.0	7.6	7.7	29.7
0.0003	0.0000001	0.00001	94.0	68.0	9.8	18.2
0.0003	0.0000001	0.00005	86.5	94.6	9.3	13.8
0.0003	0.000001	0.00001	81.2	35.8	10.4	36.4
0.0003	0.000001	0.00005	89.1	72.4	9.9	17.0
0.001	0.00000001	0.00001	59.7	0.9	10.1	30.4
0.001	0.00000001	0.00005	85.6	70.4	10.2	19.1
0.001	0.0000001	0.00001	96.4	68.6	11.6	23.7
0.001	0.0000001	0.00005	99.4	65.5	10.3	14.6
0.001	0.000001	0.00001	94.5	68.0	7.1	24.9
0.001	0.000001	0.00005	71.3	47.8	7.7	29.2
0.05	0.00000001	0.00001	87.8	2.0	10.8	31.6
0.05	0.00000001	0.00005	84.6	2.1	10.4	33.2
0.05	0.0000001	0.00001	94.7	24.2	10.9	21.0
0.05	0.0000001	0.00005	83.2	69.6	10.0	15.6
0.05	0.000001	0.00001	42.5	5.1	11.5	36.1
0.05	0.000001	0.00005	71.5	16.3	11.2	31.7

Table 34 Test Results for ID-A, where NR= 128, NS= 1000, NE= 4, BR= 0.001.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	43.5	1.3	10.6	32.1
0.0003	0.00000001	0.00005	62.3	11.6	10.7	34.7
0.0003	0.0000001	0.00001	90.8	84.7	7.5	18.5
0.0003	0.0000001	0.00005	53.7	3.7	8.7	58.5
0.0003	0.000001	0.00001	84.3	92.5	8.0	17.2
0.0003	0.000001	0.00005	82.1	29.7	9.8	27.5
0.001	0.00000001	0.00001	35.5	4.6	10.4	45.4
0.001	0.00000001	0.00005	89.0	69.4	9.8	18.1
0.001	0.0000001	0.00001	98.9	30.8	9.7	30.6
0.001	0.0000001	0.00005	94.6	77.5	9.8	19.5
0.001	0.000001	0.00001	80.3	76.0	7.8	16.1
0.001	0.000001	0.00005	55.1	21.9	7.7	42.4
0.05	0.00000001	0.00001	78.7	74.9	6.2	16.2
0.05	0.00000001	0.00005	86.3	67.8	7.1	17.5
0.05	0.0000001	0.00001	86.3	13.9	9.9	27.2
0.05	0.0000001	0.00005	75.1	22.0	7.9	23.1
0.05	0.000001	0.00001	33.2	5.9	9.0	35.3
0.05	0.000001	0.00005	45.5	1.8	8.8	25.5

Table 35 Test Results for ID-A, where NR= 128, NS= 1000, NE= 4, BR= 0.008.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	57.9	11.4	9.3	33.1
0.0003	0.00000001	0.00005	42.6	9.9	9.3	32.0
0.0003	0.0000001	0.00001	91.7	90.3	10.2	17.0
0.0003	0.0000001	0.00005	84.0	15.3	10.0	42.6
0.0003	0.000001	0.00001	79.8	39.8	6.7	28.9
0.0003	0.000001	0.00005	56.3	6.7	9.8	25.2
0.001	0.00000001	0.00001	81.2	9.5	7.2	22.5
0.001	0.00000001	0.00005	88.7	20.1	10.5	35.2
0.001	0.0000001	0.00001	49.3	2.8	10.4	41.9
0.001	0.0000001	0.00005	94.8	28.6	10.9	28.2
0.001	0.000001	0.00001	21.6	4.8	9.7	50.1
0.001	0.000001	0.00005	80.1	75.7	9.6	16.3
0.05	0.00000001	0.00001	29.6	6.7	9.9	24.8
0.05	0.00000001	0.00005	10.0	1.8	9.0	40.5
0.05	0.0000001	0.00001	97.6	65.5	9.4	16.6
0.05	0.0000001	0.00005	83.3	9.7	8.8	22.6
0.05	0.000001	0.00001	22.8	1.4	9.1	39.2
0.05	0.000001	0.00005	49.1	6.8	9.0	37.6

Table 36 Test Results for ID-A, where NR=128, NS= 2400, NE=1, BR=0.0003.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	62.2	24.7	9.0	43.4
0.0003	0.00000001	0.00005	83.2	25.2	9.4	33.1
0.0003	0.0000001	0.00001	42.8	14.3	10.3	34.5
0.0003	0.0000001	0.00005	45.9	15.7	10.3	36.9
0.0003	0.000001	0.00001	72.1	3.3	10.9	33.6
0.0003	0.000001	0.00005	25.1	5.9	9.8	37.1
0.001	0.00000001	0.00001	89.3	67.4	9.7	21.0
0.001	0.00000001	0.00005	99.2	66.5	10.2	24.9
0.001	0.0000001	0.00001	81.5	34.0	11.4	27.5
0.001	0.0000001	0.00005	96.7	73.9	10.8	20.2
0.001	0.000001	0.00001	84.6	67.8	10.0	21.8
0.001	0.000001	0.00005	16.0	2.2	9.7	34.8
0.05	0.00000001	0.00001	28.4	6.2	11.2	41.3
0.05	0.00000001	0.00005	54.3	19.7	11.2	48.1
0.05	0.0000001	0.00001	74.3	5.0	11.0	40.5
0.05	0.0000001	0.00005	83.7	20.2	10.5	39.7
0.05	0.000001	0.00001	89.8	67.6	10.5	16.5
0.05	0.000001	0.00005	65.2	3.9	10.6	38.8

Table 37 Test Results for ID-A, where NR= 128, NS= 2400, NE= 1, BR= 0.001.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	89.6	72.3	8.0	24.8
0.0003	0.00000001	0.00005	38.8	1.9	9.7	40.4
0.0003	0.0000001	0.00001	80.0	72.3	12.4	23.5
0.0003	0.0000001	0.00005	37.0	6.4	10.0	41.9
0.0003	0.000001	0.00001	65.7	2.4	11.8	28.4
0.0003	0.000001	0.00005	91.5	0.8	9.8	46.0
0.001	0.00000001	0.00001	91.7	69.9	8.1	30.0
0.001	0.00000001	0.00005	27.2	7.4	8.2	37.9
0.001	0.0000001	0.00001	55.5	22.1	7.7	44.6
0.001	0.0000001	0.00005	97.5	68.3	9.8	25.2
0.001	0.000001	0.00001	56.2	20.6	13.4	42.1
0.001	0.000001	0.00005	95.5	52.6	14.5	37.9
0.05	0.00000001	0.00001	36.8	2.1	13.4	24.8
0.05	0.00000001	0.00005	45.8	4.9	10.8	27.6
0.05	0.0000001	0.00001	14.7	2.8	8.6	29.0
0.05	0.0000001	0.00005	83.4	2.5	10.1	31.5
0.05	0.000001	0.00001	23.5	3.8	9.8	25.5
0.05	0.000001	0.00005	87.0	68.1	11.6	22.2

Table 38 Test Results for ID-A, where NR= 128, NS= 2400, NE= 1, BR= 0.008.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	97.3	66.2	9.3	19.3
0.0003	0.00000001	0.00005	69.9	8.7	9.1	49.3
0.0003	0.0000001	0.00001	98.2	65.9	10.1	20.8
0.0003	0.0000001	0.00005	75.5	1.0	9.6	44.1
0.0003	0.000001	0.00001	95.3	66.6	10.2	20.7
0.0003	0.000001	0.00005	95.3	66.6	10.2	20.7
0.001	0.00000001	0.00001	60.6	4.7	11.9	36.2
0.001	0.00000001	0.00005	28.3	2.1	11.3	34.6
0.001	0.0000001	0.00001	68.6	2.4	13.4	44.7
0.001	0.0000001	0.00005	41.3	0.3	14.8	54.4
0.001	0.000001	0.00001	45.4	0.4	9.0	32.3
0.001	0.000001	0.00005	34.3	4.9	12.3	40.7
0.05	0.00000001	0.00001	70.4	0.9	9.2	33.5
0.05	0.00000001	0.00005	47.1	5.6	10.9	35.4
0.05	0.0000001	0.00001	44.2	6.1	9.5	37.9
0.05	0.0000001	0.00005	69.9	14.7	11.2	52.8
0.05	0.000001	0.00001	73.0	5.2	10.7	26.5
0.05	0.000001	0.00005	19.2	2.8	11.8	49.5

Table 39 Test Results for ID-A, where NR=128, NS=2400, NE=4, BR=0.0003.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	87.1	72.5	9.9	15.2
0.0003	0.00000001	0.00005	95.8	74.5	10.1	16.5
0.0003	0.0000001	0.00001	46.8	6.2	8.4	28.4
0.0003	0.0000001	0.00005	97.9	87.4	8.3	17.1
0.0003	0.000001	0.00001	85.3	93.0	7.4	12.0
0.0003	0.000001	0.00005	31.9	4.5	8.1	34.8
0.001	0.00000001	0.00001	99.1	66.9	11.0	21.0
0.001	0.00000001	0.00005	86.3	90.2	8.8	14.1
0.001	0.0000001	0.00001	61.7	13.3	10.1	39.7
0.001	0.0000001	0.00005	39.0	3.0	9.3	20.5
0.001	0.000001	0.00001	82.2	95.7	6.8	11.5
0.001	0.000001	0.00005	89.3	89.8	7.1	18.6
0.05	0.00000001	0.00001	59.4	2.2	10.0	30.9
0.05	0.00000001	0.00005	26.7	6.8	10.1	47.3
0.05	0.0000001	0.00001	61.5	4.1	9.4	25.2
0.05	0.0000001	0.00005	36.2	0.9	10.8	33.6
0.05	0.000001	0.00001	61.9	12.6	12.6	37.5
0.05	0.000001	0.00005	96.8	15.9	11.2	41.1

Table 40 Test Results for ID-A, where NR=128, NS=2400, NE=4, BR=0.001.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	98.8	39.8	9.0	36.6
0.0003	0.00000001	0.00005	24.5	2.0	9.1	35.0
0.0003	0.0000001	0.00001	89.0	38.5	8.5	44.9
0.0003	0.0000001	0.00005	88.4	21.1	8.6	44.5
0.0003	0.000001	0.00001	48.4	16.4	9.6	44.0
0.0003	0.000001	0.00005	91.2	75.4	9.0	15.1
0.001	0.00000001	0.00001	50.9	9.4	8.0	39.7
0.001	0.00000001	0.00005	79.7	73.8	8.6	23.6
0.001	0.0000001	0.00001	49.8	17.0	8.4	41.7
0.001	0.0000001	0.00005	69.1	35.7	8.6	34.5
0.001	0.000001	0.00001	92.4	10.4	8.4	47.5
0.001	0.000001	0.00005	64.8	38.8	6.3	27.6
0.05	0.00000001	0.00001	33.8	8.9	10.3	35.3
0.05	0.00000001	0.00005	57.3	4.3	12.7	39.9
0.05	0.0000001	0.00001	37.4	7.3	9.3	22.7
0.05	0.0000001	0.00005	59.0	22.0	8.6	29.3
0.05	0.000001	0.00001	15.2	0.3	10.5	35.1
0.05	0.000001	0.00005	2.1	0.4	12.3	30.6

Table 41 Test Results for ID-A, where NR= 128, NS= 2400, NE= 4, BR= 0.008.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	79.2	67.7	8.7	18.8
0.0003	0.00000001	0.00005	31.8	9.2	8.5	37.4
0.0003	0.0000001	0.00001	80.7	96.2	7.6	10.6
0.0003	0.0000001	0.00005	73.2	28.3	7.6	33.4
0.0003	0.000001	0.00001	81.5	87.9	8.6	13.9
0.0003	0.000001	0.00005	80.7	96.0	8.2	15.1
0.001	0.00000001	0.00001	47.9	6.4	11.5	40.3
0.001	0.00000001	0.00005	65.7	11.0	10.5	23.4
0.001	0.0000001	0.00001	43.3	8.7	12.0	41.7
0.001	0.0000001	0.00005	53.0	7.4	10.8	38.6
0.001	0.000001	0.00001	98.3	67.9	10.5	24.8
0.001	0.000001	0.00005	88.3	19.4	9.6	23.6
0.05	0.00000001	0.00001	69.4	21.0	10.2	35.8
0.05	0.00000001	0.00005	45.7	5.9	11.9	30.7
0.05	0.0000001	0.00001	55.1	2.3	11.5	48.3
0.05	0.0000001	0.00005	64.5	3.4	10.9	39.2
0.05	0.000001	0.00001	91.3	91.2	10.0	44.5
0.05	0.000001	0.00005	91.9	73.0	8.8	15.4

Table 42 Test Results for ID-A, where NR= 416, NS= 400, NE= 1, BR= 0.0003.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	60.0	18.6	38.2	154.5
0.0003	0.00000001	0.00005	72.7	50.3	38.0	121.3
0.0003	0.0000001	0.00001	58.5	27.1	39.7	158.6
0.0003	0.0000001	0.00005	81.0	44.8	52.3	85.9
0.0003	0.000001	0.00001	76.2	52.9	34.6	143.3
0.0003	0.000001	0.00005	67.8	35.8	42.3	165.4
0.001	0.00000001	0.00001	81.8	77.8	45.5	76.6
0.001	0.00000001	0.00005	86.1	80.5	36.1	77.1
0.001	0.0000001	0.00001	87.6	20.6	46.7	127.1
0.001	0.0000001	0.00005	65.6	34.4	39.7	125.2
0.001	0.000001	0.00001	70.3	28.0	38.7	129.3
0.001	0.000001	0.00005	70.2	33.2	37.0	190.6
0.05	0.00000001	0.00001	30.7	5.5	51.6	124.2
0.05	0.00000001	0.00005	59.3	25.0	43.7	87.6
0.05	0.0000001	0.00001	24.3	2.5	47.6	121.8
0.05	0.0000001	0.00005	88.5	70.0	48.9	82.9
0.05	0.000001	0.00001	40.8	8.6	41.2	162.4
0.05	0.000001	0.00005	85.2	43.0	37.0	112.0

Table 43 Test Results for ID-A, where NR= 416, NS= 400, NE= 1, BR= 0.001.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	63.6	4.5	52.8	83.6
0.0003	0.00000001	0.00005	75.5	7.1	50.0	149.7
0.0003	0.0000001	0.00001	88.7	76.7	48.3	58.2
0.0003	0.0000001	0.00005	10.6	1.3	45.1	88.5
0.0003	0.000001	0.00001	42.6	8.1	47.3	128.4
0.0003	0.000001	0.00005	56.3	20.3	40.6	146.8
0.001	0.00000001	0.00001	95.8	69.6	50.1	70.6
0.001	0.00000001	0.00005	63.6	10.7	45.2	128.2
0.001	0.0000001	0.00001	76.7	29.4	42.0	61.7
0.001	0.0000001	0.00005	50.6	10.5	42.7	102.6
0.001	0.000001	0.00001	79.7	34.8	39.9	170.7
0.001	0.000001	0.00005	62.8	17.3	43.0	167.1
0.05	0.00000001	0.00001	72.4	4.5	53.0	167.7
0.05	0.00000001	0.00005	56.6	8.9	54.0	153.4
0.05	0.0000001	0.00001	24.2	3.1	46.6	134.6
0.05	0.0000001	0.00005	53.6	8.4	49.7	183.8
0.05	0.000001	0.00001	80.3	23.1	43.6	136.7
0.05	0.000001	0.00005	92.8	77.7	44.4	74.3

Table 44 Test Results for ID-A, where NR= 416, NS= 400, NE= 1, BR= 0.008.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	87.3	2.7	48.9	106.1
0.0003	0.00000001	0.00005	78.2	25.0	51.0	74.2
0.0003	0.0000001	0.00001	77.0	64.9	45.9	234.3
0.0003	0.0000001	0.00005	99.5	15.3	48.4	165.2
0.0003	0.000001	0.00001	73.0	11.0	48.4	189.4
0.0003	0.000001	0.00005	96.8	68.9	46.0	82.4
0.001	0.00000001	0.00001	78.8	71.1	41.7	55.2
0.001	0.00000001	0.00005	76.6	73.8	46.5	70.8
0.001	0.0000001	0.00001	65.0	3.4	42.4	159.9
0.001	0.0000001	0.00005	41.7	14.3	41.8	189.0
0.001	0.000001	0.00001	74.6	18.0	38.7	124.2
0.001	0.000001	0.00005	76.3	22.4	38.2	137.9
0.05	0.00000001	0.00001	12.2	0.6	44.0	87.9
0.05	0.00000001	0.00005	93.4	74.7	43.4	57.3
0.05	0.0000001	0.00001	94.0	67.3	43.3	68.6
0.05	0.0000001	0.00005	69.7	29.9	44.0	200.9
0.05	0.000001	0.00001	84.6	71.3	47.3	64.4
0.05	0.000001	0.00005	22.4	4.3	48.8	125.9

Table 45 Test Results for ID-A, where NR= 416, NS= 400, NE= 4, BR= 0.0003.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	85.8	18.0	39.1	129.1
0.0003	0.00000001	0.00005	84.3	23.9	38.7	168.2
0.0003	0.0000001	0.00001	55.4	13.3	40.8	162.6
0.0003	0.0000001	0.00005	80.9	73.4	39.9	71.5
0.0003	0.000001	0.00001	79.2	31.2	44.5	151.8
0.0003	0.000001	0.00005	75.1	8.1	44.3	153.5
0.001	0.00000001	0.00001	75.6	49.7	40.3	93.8
0.001	0.00000001	0.00005	52.7	7.4	44.1	134.0
0.001	0.0000001	0.00001	30.5	4.3	42.7	154.4
0.001	0.0000001	0.00005	30.5	4.3	42.7	154.3
0.001	0.000001	0.00001	32.4	6.4	39.7	131.3
0.001	0.000001	0.00005	65.1	30.1	39.2	147.5
0.05	0.00000001	0.00001	95.3	40.5	48.6	141.5
0.05	0.00000001	0.00005	75.6	37.3	42.0	101.9
0.05	0.0000001	0.00001	70.0	5.5	44.8	159.2
0.05	0.0000001	0.00005	75.6	23.5	46.2	80.1
0.05	0.000001	0.00001	83.3	94.3	42.6	80.2
0.05	0.000001	0.00005	70.9	26.2	43.2	115.6

Table 46 Test Results for ID-A, where NR= 416, NS= 400, NE= 4, BR= 0.001.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	62.6	7.3	48.0	128.1
0.0003	0.00000001	0.00005	80.1	4.4	45.6	102.1
0.0003	0.0000001	0.00001	70.5	28.0	48.4	66.5
0.0003	0.0000001	0.00005	95.0	8.3	42.1	119.1
0.0003	0.000001	0.00001	46.8	5.8	44.5	93.1
0.0003	0.000001	0.00005	72.6	16.3	44.2	118.2
0.001	0.00000001	0.00001	90.6	27.1	48.7	161.2
0.001	0.00000001	0.00005	80.9	7.7	45.3	131.8
0.001	0.0000001	0.00001	92.5	17.7	44.8	166.6
0.001	0.0000001	0.00005	86.6	89.3	38.4	48.6
0.001	0.000001	0.00001	84.1	28.8	47.1	134.1
0.001	0.000001	0.00005	49.7	20.7	39.3	129.8
0.05	0.00000001	0.00001	34.8	4.5	46.9	145.9
0.05	0.00000001	0.00005	81.8	18.3	48.2	153.2
0.05	0.0000001	0.00001	97.9	67.9	40.6	82.0
0.05	0.0000001	0.00005	30.5	5.3	39.9	128.0
0.05	0.000001	0.00001	74.7	31.3	41.3	152.9
0.05	0.000001	0.00005	60.3	7.9	43.0	124.9

Table 47 Test Results for ID-A, where NR= 416, NS= 400, NE= 4, BR= 0.008.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	45.8	5.4	38.2	74.1
0.0003	0.00000001	0.00005	31.3	2.9	48.6	121.0
0.0003	0.0000001	0.00001	91.3	15.9	43.8	78.2
0.0003	0.0000001	0.00005	98.2	72.6	41.6	82.6
0.0003	0.000001	0.00001	64.3	36.3	35.4	83.5
0.0003	0.000001	0.00005	91.9	72.0	42.3	52.1
0.001	0.00000001	0.00001	60.9	14.9	45.2	123.1
0.001	0.00000001	0.00005	95.3	19.2	47.4	103.6
0.001	0.0000001	0.00001	71.0	12.2	50.9	159.8
0.001	0.0000001	0.00005	42.2	14.4	38.2	162.0
0.001	0.000001	0.00001	84.6	68.7	39.9	67.5
0.001	0.000001	0.00005	59.4	2.4	41.5	166.3
0.05	0.00000001	0.00001	59.2	5.8	44.7	119.8
0.05	0.00000001	0.00005	86.1	25.3	45.2	126.8
0.05	0.0000001	0.00001	77.9	29.0	43.8	83.1
0.05	0.0000001	0.00005	24.1	1.5	44.3	155.9
0.05	0.000001	0.00001	51.4	3.3	45.9	148.0
0.05	0.000001	0.00005	51.4	3.3	45.9	148.0

Table 48 Test Results for ID-A, where NR=416, NS=1000, NE=1, BR=0.0003.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	69.4	35.5	37.2	174.4
0.0003	0.00000001	0.00005	54.1	20.4	41.6	122.6
0.0003	0.0000001	0.00001	54.9	17.5	43.5	174.5
0.0003	0.0000001	0.00005	79.9	22.2	41.9	132.5
0.0003	0.000001	0.00001	68.4	24.8	40.8	145.8
0.0003	0.000001	0.00005	58.1	16.1	37.9	110.3
0.001	0.00000001	0.00001	90.8	67.6	38.1	163.7
0.001	0.00000001	0.00005	58.1	15.7	36.0	103.3
0.001	0.0000001	0.00001	70.0	36.2	38.6	136.9
0.001	0.0000001	0.00005	56.9	19.7	39.8	153.4
0.001	0.000001	0.00001	84.0	53.1	48.5	171.1
0.001	0.000001	0.00005	72.7	20.3	44.2	117.4
0.05	0.00000001	0.00001	78.0	26.1	38.6	128.5
0.05	0.00000001	0.00005	22.7	4.8	40.5	81.0
0.05	0.0000001	0.00001	47.2	15.9	42.7	150.3
0.05	0.0000001	0.00005	39.5	12.9	42.8	142.5
0.05	0.000001	0.00001	48.9	21.3	34.8	162.5
0.05	0.000001	0.00005	67.6	39.8	45.1	200.4

Table 49 Test Results for ID-A, where NR= 416, NS= 1000, NE= 1, BR= 0.001.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	50.6	16.6	38.6	117.9
0.0003	0.00000001	0.00005	82.5	77.9	45.9	101.3
0.0003	0.0000001	0.00001	29.1	8.6	41.5	83.7
0.0003	0.0000001	0.00005	49.8	11.4	42.9	141.6
0.0003	0.000001	0.00001	76.6	23.9	46.4	185.3
0.0003	0.000001	0.00005	97.3	68.9	41.9	65.4
0.001	0.00000001	0.00001	69.8	6.2	48.2	115.8
0.001	0.00000001	0.00005	65.6	30.5	43.6	170.0
0.001	0.0000001	0.00001	75.4	8.6	46.8	129.5
0.001	0.0000001	0.00005	86.8	13.9	47.6	147.4
0.001	0.000001	0.00001	62.0	8.2	43.6	151.8
0.001	0.000001	0.00005	84.1	11.6	44.9	97.7
0.05	0.00000001	0.00001	70.1	11.4	48.6	199.4
0.05	0.00000001	0.00005	56.4	13.3	46.8	159.3
0.05	0.0000001	0.00001	4.5	0.7	44.5	142.1
0.05	0.0000001	0.00005	64.4	22.2	49.1	180.7
0.05	0.000001	0.00001	80.2	13.3	46.2	171.3
0.05	0.000001	0.00005	97.8	32.7	51.4	153.9

Table 50 Test Results for ID-A, where NR= 416, NS= 1000, NE= 1, BR= 0.008.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	17.2	0.9	47.7	111.8
0.0003	0.00000001	0.00005	30.9	2.7	41.8	140.7
0.0003	0.0000001	0.00001	16.8	1.1	46.3	105.7
0.0003	0.0000001	0.00005	34.0	4.0	45.8	230.7
0.0003	0.000001	0.00001	51.2	7.0	40.3	135.7
0.0003	0.000001	0.00005	30.7	3.2	47.3	110.5
0.001	0.00000001	0.00001	59.1	17.6	40.3	128.7
0.001	0.00000001	0.00005	90.8	72.7	39.2	84.1
0.001	0.0000001	0.00001	77.3	35.7	51.9	267.7
0.001	0.0000001	0.00005	79.6	18.4	45.8	155.9
0.001	0.000001	0.00001	69.5	7.3	41.1	144.2
0.001	0.000001	0.00005	69.4	5.8	42.5	169.5
0.05	0.00000001	0.00001	54.4	9.5	53.4	80.6
0.05	0.00000001	0.00005	53.7	5.6	54.1	164.5
0.05	0.0000001	0.00001	52.2	3.8	47.5	223.7
0.05	0.0000001	0.00005	38.1	3.9	46.5	89.1
0.05	0.000001	0.00001	52.8	8.7	58.4	205.0
0.05	0.000001	0.00005	43.2	3.9	56.6	87.5

Table 51 Test Results for ID-A, where NR=416, NS=1000, NE=4, BR=0.0003.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	43.2	13.5	37.5	129.1
0.0003	0.00000001	0.00005	87.5	42.0	38.0	111.8
0.0003	0.0000001	0.00001	60.5	23.9	36.8	141.6
0.0003	0.0000001	0.00005	58.9	18.2	36.5	148.1
0.0003	0.000001	0.00001	79.5	29.4	39.9	153.1
0.0003	0.000001	0.00005	92.6	41.8	40.1	98.2
0.001	0.00000001	0.00001	88.5	22.5	45.1	166.8
0.001	0.00000001	0.00005	93.6	25.0	41.9	173.6
0.001	0.0000001	0.00001	74.6	14.3	40.7	177.2
0.001	0.0000001	0.00005	36.5	12.2	34.5	115.4
0.001	0.000001	0.00001	79.4	30.3	48.1	169.5
0.001	0.000001	0.00005	81.4	52.0	48.6	144.2
0.05	0.00000001	0.00001	71.6	24.2	44.8	137.5
0.05	0.00000001	0.00005	51.6	11.8	41.3	118.8
0.05	0.0000001	0.00001	79.3	35.4	49.2	176.8
0.05	0.0000001	0.00005	6.1	0.9	33.2	106.4
0.05	0.000001	0.00001	63.8	28.4	44.7	55.3
0.05	0.000001	0.00005	89.0	3.3	48.7	124.1

Table 52 Test Results for ID-A, where NR= 416, NS= 1000, NE= 4, BR= 0.001.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	50.9	17.3	40.0	99.0
0.0003	0.00000001	0.00005	85.2	91.4	42.5	42.0
0.0003	0.0000001	0.00001	89.1	82.9	50.5	72.6
0.0003	0.0000001	0.00005	58.5	19.9	46.9	156.2
0.0003	0.000001	0.00001	89.2	20.8	41.4	152.3
0.0003	0.000001	0.00005	95.6	89.7	39.0	62.0
0.001	0.00000001	0.00001	65.7	22.8	41.4	172.4
0.001	0.00000001	0.00005	95.0	40.9	44.1	143.4
0.001	0.0000001	0.00001	22.3	4.1	42.1	111.9
0.001	0.0000001	0.00005	66.0	30.2	38.1	112.0
0.001	0.000001	0.00001	59.8	5.0	44.0	83.2
0.001	0.000001	0.00005	67.4	41.3	53.3	135.2
0.05	0.00000001	0.00001	68.4	8.8	41.5	175.3
0.05	0.00000001	0.00005	42.9	8.2	42.4	108.5
0.05	0.0000001	0.00001	29.0	5.2	43.8	140.4
0.05	0.0000001	0.00005	71.4	25.6	45.3	166.9
0.05	0.000001	0.00001	89.0	92.5	39.7	54.1
0.05	0.000001	0.00005	62.2	24.7	39.9	151.7

Table 53 Test Results for ID-A, where NR= 416, NS= 1000, NE= 4, BR= 0.008.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	71.4	29.4	43.4	108.0
0.0003	0.00000001	0.00005	77.0	6.6	51.4	101.5
0.0003	0.0000001	0.00001	88.6	67.0	50.5	97.7
0.0003	0.0000001	0.00005	61.2	11.7	47.3	140.8
0.0003	0.000001	0.00001	44.1	12.1	47.5	201.8
0.0003	0.000001	0.00005	83.3	75.4	42.0	120.7
0.001	0.00000001	0.00001	86.6	45.2	50.1	181.3
0.001	0.00000001	0.00005	39.8	12.1	40.9	79.5
0.001	0.0000001	0.00001	89.2	67.3	47.0	90.2
0.001	0.0000001	0.00005	40.7	10.1	41.5	134.8
0.001	0.000001	0.00001	65.3	21.7	45.3	158.5
0.001	0.000001	0.00005	66.6	28.7	42.7	154.2
0.05	0.00000001	0.00001	85.0	5.2	48.6	178.1
0.05	0.00000001	0.00005	79.2	5.8	41.9	182.3
0.05	0.0000001	0.00001	3.9	0.5	46.8	136.2
0.05	0.0000001	0.00005	61.3	22.3	43.4	162.6
0.05	0.000001	0.00001	52.5	13.0	50.2	87.4
0.05	0.000001	0.00005	89.8	79.8	44.8	55.3

Table 54 Test Results for ID-A, where NR=416, NS= 2400, NE=1, BR=0.0003.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	69.7	27.3	37.7	159.4
0.0003	0.00000001	0.00005	53.8	16.2	37.4	146.4
0.0003	0.0000001	0.00001	44.4	13.8	34.6	110.2
0.0003	0.0000001	0.00005	74.6	51.6	40.1	176.9
0.0003	0.000001	0.00001	84.7	46.6	45.7	152.4
0.0003	0.000001	0.00005	85.6	46.5	44.9	153.7
0.001	0.00000001	0.00001	67.4	29.0	35.5	173.1
0.001	0.00000001	0.00005	70.3	36.9	36.5	192.0
0.001	0.0000001	0.00001	70.8	39.4	39.3	144.9
0.001	0.0000001	0.00005	66.7	26.7	44.8	139.5
0.001	0.000001	0.00001	55.1	10.8	43.0	130.1
0.001	0.000001	0.00005	42.8	14.8	35.3	100.4
0.05	0.00000001	0.00001	74.4	4.4	44.9	91.1
0.05	0.00000001	0.00005	92.0	34.1	44.9	105.3
0.05	0.0000001	0.00001	66.1	12.3	37.9	112.4
0.05	0.0000001	0.00005	75.8	5.7	37.5	113.1
0.05	0.000001	0.00001	48.7	15.4	51.4	123.0
0.05	0.000001	0.00005	48.4	19.2	44.4	104.6

Table 55 Test Results for ID-A, where NR= 416, NS= 2400, NE= 1, BR= 0.001.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	58.6	26.8	44.2	150.2
0.0003	0.00000001	0.00005	64.2	36.5	47.5	163.8
0.0003	0.0000001	0.00001	46.0	11.9	40.8	171.9
0.0003	0.0000001	0.00005	84.7	25.7	48.1	133.8
0.0003	0.000001	0.00001	97.6	87.7	50.9	70.9
0.0003	0.000001	0.00005	61.0	17.9	48.9	111.3
0.001	0.00000001	0.00001	91.4	5.1	41.6	110.4
0.001	0.00000001	0.00005	79.7	24.7	44.2	157.3
0.001	0.0000001	0.00001	92.7	28.7	40.2	171.5
0.001	0.0000001	0.00005	90.1	18.9	40.8	178.8
0.001	0.000001	0.00001	69.1	6.7	47.5	150.9
0.001	0.000001	0.00005	96.0	26.7	43.1	167.3
0.05	0.00000001	0.00001	42.7	11.3	36.9	187.7
0.05	0.00000001	0.00005	50.8	14.2	42.3	166.7
0.05	0.0000001	0.00001	37.5	5.7	40.5	217.7
0.05	0.0000001	0.00005	73.6	18.7	37.3	182.9
0.05	0.000001	0.00001	31.5	7.0	48.6	142.6
0.05	0.000001	0.00005	77.4	13.0	52.1	129.5

Table 56 Test Results for ID-A, where NR= 416, NS= 2400, NE= 1, BR= 0.008.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	55.3	1.1	49.5	89.2
0.0003	0.00000001	0.00005	20.6	1.3	57.7	208.2
0.0003	0.0000001	0.00001	67.3	6.4	45.8	139.0
0.0003	0.0000001	0.00005	74.4	9.1	43.5	151.3
0.0003	0.000001	0.00001	68.0	10.0	39.6	103.7
0.0003	0.000001	0.00005	56.2	9.1	39.5	129.0
0.001	0.00000001	0.00001	29.7	3.5	38.4	158.2
0.001	0.00000001	0.00005	87.3	72.5	45.9	100.7
0.001	0.0000001	0.00001	77.7	13.5	42.0	173.0
0.001	0.0000001	0.00005	63.5	1.2	40.8	180.6
0.001	0.000001	0.00001	80.3	21.3	37.2	129.8
0.001	0.000001	0.00005	57.8	6.5	36.9	120.6
0.05	0.00000001	0.00001	35.9	2.9	49.8	111.1
0.05	0.00000001	0.00005	62.0	8.0	44.8	113.6
0.05	0.0000001	0.00001	95.2	66.5	45.9	98.3
0.05	0.0000001	0.00005	57.0	6.0	54.0	87.1
0.05	0.000001	0.00001	96.1	69.4	55.3	86.4
0.05	0.000001	0.00005	89.8	84.6	49.0	93.7

Table 57 Test Results for ID-A, where NR=416, NS=2400, NE=4, BR=0.0003.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	45.9	15.7	39.5	171.9
0.0003	0.00000001	0.00005	59.9	19.2	41.5	168.1
0.0003	0.0000001	0.00001	55.6	12.7	41.5	118.4
0.0003	0.0000001	0.00005	83.2	3.8	34.2	126.6
0.0003	0.000001	0.00001	72.7	28.1	46.3	133.1
0.0003	0.000001	0.00005	78.2	27.5	48.5	171.9
0.001	0.00000001	0.00001	79.0	42.0	45.4	113.4
0.001	0.00000001	0.00005	68.5	25.2	46.0	178.5
0.001	0.0000001	0.00001	71.2	21.2	43.5	137.5
0.001	0.0000001	0.00005	84.8	31.2	51.2	116.4
0.001	0.000001	0.00001	70.9	27.6	36.5	115.1
0.001	0.000001	0.00005	54.1	23.5	38.2	112.6
0.05	0.00000001	0.00001	13.1	0.9	43.6	150.7
0.05	0.00000001	0.00005	47.5	5.8	50.3	121.9
0.05	0.0000001	0.00001	82.6	59.6	44.6	140.7
0.05	0.0000001	0.00005	90.2	19.9	49.3	83.4
0.05	0.000001	0.00001	62.9	33.7	42.9	116.1
0.05	0.000001	0.00005	78.9	71.9	37.3	52.3

Table 58 Test Results for ID-A, where NR= 416, NS= 2400, NE= 4, BR= 0.001.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	55.1	24.7	35.9	135.3
0.0003	0.00000001	0.00005	78.5	41.1	39.3	137.1
0.0003	0.0000001	0.00001	67.8	24.5	39.6	152.1
0.0003	0.0000001	0.00005	82.1	76.3	34.3	54.9
0.0003	0.000001	0.00001	79.4	41.5	44.8	164.3
0.0003	0.000001	0.00005	94.9	24.6	42.6	142.8
0.001	0.00000001	0.00001	90.4	18.7	44.9	129.8
0.001	0.00000001	0.00005	55.4	13.2	45.0	138.7
0.001	0.0000001	0.00001	77.4	34.2	50.9	173.7
0.001	0.0000001	0.00005	95.5	10.2	49.3	141.1
0.001	0.000001	0.00001	21.2	3.0	41.8	102.8
0.001	0.000001	0.00005	80.0	22.7	45.4	190.9
0.05	0.00000001	0.00001	23.9	1.6	41.8	134.3
0.05	0.00000001	0.00005	69.9	15.3	41.8	176.2
0.05	0.0000001	0.00001	68.1	13.2	43.4	115.1
0.05	0.0000001	0.00005	37.6	7.6	39.8	143.7
0.05	0.000001	0.00001	64.1	6.4	43.6	116.5
0.05	0.000001	0.00005	35.3	9.7	38.9	118.4

Table 59 Test Results for ID-A, where NR= 416, NS= 2400, NE= 4, BR= 0.008.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	39.5	6.7	42.7	93.2
0.0003	0.00000001	0.00005	69.6	2.4	49.6	108.0
0.0003	0.0000001	0.00001	56.6	7.5	42.8	174.5
0.0003	0.0000001	0.00005	76.5	3.7	53.0	153.8
0.0003	0.000001	0.00001	62.4	0.6	39.6	132.4
0.0003	0.000001	0.00005	70.5	8.6	43.2	213.3
0.001	0.00000001	0.00001	97.2	67.5	48.1	92.4
0.001	0.00000001	0.00005	90.7	75.2	48.7	70.4
0.001	0.0000001	0.00001	57.4	10.4	48.6	224.9
0.001	0.0000001	0.00005	50.4	10.0	40.1	95.6
0.001	0.000001	0.00001	72.2	12.2	40.4	177.3
0.001	0.000001	0.00005	74.0	45.8	40.2	199.5
0.05	0.00000001	0.00001	98.3	14.4	46.4	162.5
0.05	0.00000001	0.00005	48.1	4.3	44.1	107.6
0.05	0.0000001	0.00001	25.1	0.3	46.6	111.8
0.05	0.0000001	0.00005	93.6	71.3	45.3	69.9
0.05	0.000001	0.00001	88.3	58.4	44.8	134.0
0.05	0.000001	0.00005	86.7	66.5	46.9	67.1

Table 60 Test Results for ID-A, where NR= 480, NS= 400, NE= 1, BR= 0.0003.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	46.9	10.4	56.1	187.0
0.0003	0.00000001	0.00005	75.2	52.3	51.6	163.1
0.0003	0.0000001	0.00001	70.1	47.9	51.6	216.5
0.0003	0.0000001	0.00005	72.9	40.3	50.4	182.5
0.0003	0.000001	0.00001	78.9	24.4	58.6	211.8
0.0003	0.000001	0.00005	89.9	52.4	60.1	188.8
0.001	0.00000001	0.00001	72.7	47.1	69.4	174.7
0.001	0.00000001	0.00005	69.7	48.4	60.7	186.3
0.001	0.0000001	0.00001	71.2	25.1	66.9	196.8
0.001	0.0000001	0.00005	71.2	25.1	66.9	196.9
0.001	0.000001	0.00001	49.6	10.6	50.2	189.3
0.001	0.000001	0.00005	67.9	38.3	54.9	121.2
0.05	0.00000001	0.00001	37.6	7.3	57.5	175.9
0.05	0.00000001	0.00005	36.1	11.4	56.4	171.7
0.05	0.0000001	0.00001	56.1	8.8	57.5	130.2
0.05	0.0000001	0.00005	74.5	23.4	68.7	174.2
0.05	0.000001	0.00001	70.4	38.1	64.2	222.7
0.05	0.000001	0.00005	79.3	23.5	59.3	162.5

Table 61 Test Results for ID-A, where NR= 480, NS= 400, NE= 1, BR= 0.001.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	94.8	17.6	52.8	122.3
0.0003	0.00000001	0.00005	92.6	81.4	53.6	100.7
0.0003	0.0000001	0.00001	49.4	13.5	53.2	192.0
0.0003	0.0000001	0.00005	84.8	23.8	55.5	154.1
0.0003	0.000001	0.00001	67.7	10.8	59.8	149.4
0.0003	0.000001	0.00005	75.0	18.4	57.7	206.0
0.001	0.00000001	0.00001	64.6	25.6	52.0	234.8
0.001	0.00000001	0.00005	91.1	68.6	57.4	116.2
0.001	0.0000001	0.00001	93.0	18.6	51.5	189.1
0.001	0.0000001	0.00005	88.0	19.2	59.6	148.8
0.001	0.000001	0.00001	50.3	17.9	47.3	213.5
0.001	0.000001	0.00005	82.2	72.2	51.2	87.1
0.05	0.00000001	0.00001	51.0	16.8	55.8	158.4
0.05	0.00000001	0.00005	60.2	11.0	50.5	174.3
0.05	0.0000001	0.00001	14.7	2.8	49.7	135.7
0.05	0.0000001	0.00005	41.2	7.7	49.0	205.5
0.05	0.000001	0.00001	46.4	10.4	48.6	181.0
0.05	0.000001	0.00005	62.0	33.0	55.8	224.2

Table 62 Test Results for ID-A, where NR= 480, NS= 400, NE= 1, BR= 0.008.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	93.4	74.6	43.0	108.1
0.0003	0.00000001	0.00005	74.3	20.6	52.4	157.3
0.0003	0.0000001	0.00001	64.6	20.8	45.2	186.0
0.0003	0.0000001	0.00005	72.1	5.6	59.9	172.5
0.0003	0.000001	0.00001	47.3	4.5	62.1	259.9
0.0003	0.000001	0.00005	85.6	77.9	62.1	96.4
0.001	0.00000001	0.00001	54.9	6.5	70.3	153.1
0.001	0.00000001	0.00005	50.3	2.9	66.8	201.3
0.001	0.0000001	0.00001	29.2	6.1	55.3	179.0
0.001	0.0000001	0.00005	90.9	66.3	58.5	109.5
0.001	0.000001	0.00001	50.9	12.9	61.0	192.5
0.001	0.000001	0.00005	86.9	69.1	65.3	110.4
0.05	0.00000001	0.00001	20.7	3.3	50.2	228.5
0.05	0.00000001	0.00005	99.4	66.1	49.2	90.5
0.05	0.0000001	0.00001	86.4	65.8	50.4	109.0
0.05	0.0000001	0.00005	45.3	1.1	48.1	145.4
0.05	0.000001	0.00001	22.0	0.6	55.2	119.2
0.05	0.000001	0.00005	81.0	72.3	54.2	96.2

Table 63 Test Results for ID-A, where NR= 480, NS= 400, NE= 4, BR= 0.0003.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	55.8	8.3	56.5	181.2
0.0003	0.00000001	0.00005	87.4	21.7	52.6	140.8
0.0003	0.0000001	0.00001	81.1	90.8	56.0	59.2
0.0003	0.0000001	0.00005	70.4	17.6	53.9	128.4
0.0003	0.000001	0.00001	89.1	79.5	59.2	99.8
0.0003	0.000001	0.00005	86.5	54.7	58.0	197.4
0.001	0.00000001	0.00001	79.4	28.4	65.4	159.8
0.001	0.00000001	0.00005	70.5	12.9	53.0	180.8
0.001	0.0000001	0.00001	58.8	16.8	48.9	167.7
0.001	0.0000001	0.00005	79.5	6.9	65.5	222.9
0.001	0.000001	0.00001	92.3	38.8	55.3	121.3
0.001	0.000001	0.00005	90.0	85.3	49.1	80.2
0.05	0.00000001	0.00001	91.0	86.5	55.1	68.3
0.05	0.00000001	0.00005	38.6	4.6	46.3	126.7
0.05	0.0000001	0.00001	44.1	5.8	62.2	166.7
0.05	0.0000001	0.00005	45.1	9.8	60.7	153.7
0.05	0.000001	0.00001	61.3	6.9	59.3	142.7
0.05	0.000001	0.00005	61.5	9.3	55.0	141.5

Table 64 Test Results for ID-A, where NR= 480, NS= 400, NE= 4, BR= 0.001.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	77.7	22.3	52.1	75.7
0.0003	0.00000001	0.00005	75.2	5.0	67.4	222.3
0.0003	0.0000001	0.00001	87.8	16.2	55.2	175.1
0.0003	0.0000001	0.00005	64.1	37.2	48.6	147.0
0.0003	0.000001	0.00001	75.6	54.9	59.2	172.1
0.0003	0.000001	0.00005	89.5	19.8	59.2	208.3
0.001	0.00000001	0.00001	77.2	37.1	50.8	149.1
0.001	0.00000001	0.00005	86.8	27.7	56.9	214.8
0.001	0.0000001	0.00001	80.4	19.3	60.1	288.9
0.001	0.0000001	0.00005	80.8	19.6	56.9	241.1
0.001	0.000001	0.00001	94.4	69.2	57.6	89.1
0.001	0.000001	0.00005	76.3	22.5	58.1	127.5
0.05	0.00000001	0.00001	23.3	4.2	50.9	253.5
0.05	0.00000001	0.00005	36.8	7.7	53.0	115.8
0.05	0.0000001	0.00001	13.4	2.3	52.0	161.9
0.05	0.0000001	0.00005	86.4	70.2	53.4	72.0
0.05	0.000001	0.00001	67.9	33.5	44.2	176.9
0.05	0.000001	0.00005	60.4	19.4	43.8	188.2

Table 65 Test Results for ID-A, where NR= 480, NS= 400, NE= 4, BR= 0.008.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	69.4	4.8	57.0	195.2
0.0003	0.00000001	0.00005	89.3	20.1	55.8	152.3
0.0003	0.0000001	0.00001	51.3	3.9	50.4	128.1
0.0003	0.0000001	0.00005	94.2	68.1	49.5	75.9
0.0003	0.000001	0.00001	31.5	3.6	51.2	104.0
0.0003	0.000001	0.00005	56.1	8.9	51.8	132.7
0.001	0.00000001	0.00001	24.3	4.0	44.1	156.5
0.001	0.00000001	0.00005	45.3	10.5	49.9	110.3
0.001	0.0000001	0.00001	59.5	8.4	56.3	114.2
0.001	0.0000001	0.00005	82.5	67.2	50.3	91.0
0.001	0.000001	0.00001	84.7	31.4	52.0	171.7
0.001	0.000001	0.00005	80.4	2.5	61.1	124.0
0.05	0.00000001	0.00001	64.8	23.0	46.8	244.7
0.05	0.00000001	0.00005	48.5	3.1	50.9	150.6
0.05	0.0000001	0.00001	94.6	68.7	59.6	92.9
0.05	0.0000001	0.00005	72.4	9.5	54.5	180.5
0.05	0.000001	0.00001	31.1	4.7	45.5	101.7
0.05	0.000001	0.00005	51.2	6.5	47.8	187.6

Table 66 Test Results for ID-A, where NR=480, NS=1000, NE=1, BR=0.0003.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	68.9	38.3	54.2	182.4
0.0003	0.00000001	0.00005	87.4	66.9	50.9	163.0
0.0003	0.0000001	0.00001	68.1	39.5	53.1	156.9
0.0003	0.0000001	0.00005	68.3	39.7	51.7	204.7
0.0003	0.000001	0.00001	68.9	46.1	50.0	144.1
0.0003	0.000001	0.00005	76.0	30.8	56.6	170.5
0.001	0.00000001	0.00001	74.3	37.4	64.1	194.9
0.001	0.00000001	0.00005	74.8	36.2	65.9	176.7
0.001	0.0000001	0.00001	78.0	47.3	57.6	230.8
0.001	0.0000001	0.00005	83.2	78.8	59.0	135.0
0.001	0.000001	0.00001	86.0	28.1	53.5	193.7
0.001	0.000001	0.00005	58.3	25.6	54.1	130.2
0.05	0.00000001	0.00001	25.0	3.7	56.8	139.7
0.05	0.00000001	0.00005	66.9	36.5	50.3	210.1
0.05	0.0000001	0.00001	82.9	21.7	65.7	204.6
0.05	0.0000001	0.00005	90.4	15.5	58.5	194.9
0.05	0.000001	0.00001	51.8	12.8	55.9	164.6
0.05	0.000001	0.00005	45.7	11.9	54.6	181.8

Table 67 Test Results for ID-A, where NR= 480, NS= 1000, NE= 1, BR= 0.001.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	60.7	17.9	54.3	149.1
0.0003	0.00000001	0.00005	97.5	70.0	66.2	103.8
0.0003	0.0000001	0.00001	33.9	6.3	67.4	126.9
0.0003	0.0000001	0.00005	49.2	18.1	53.0	218.2
0.0003	0.000001	0.00001	55.2	13.3	58.8	185.8
0.0003	0.000001	0.00005	55.6	11.1	59.9	187.6
0.001	0.00000001	0.00001	24.4	5.6	49.3	105.2
0.001	0.00000001	0.00005	89.7	73.7	55.0	90.2
0.001	0.0000001	0.00001	59.5	29.3	48.7	159.1
0.001	0.0000001	0.00005	86.7	19.3	58.6	212.3
0.001	0.000001	0.00001	53.1	24.3	62.2	209.6
0.001	0.000001	0.00005	87.3	46.6	59.8	199.2
0.05	0.00000001	0.00001	79.4	41.4	51.9	160.1
0.05	0.00000001	0.00005	49.3	4.8	52.7	187.4
0.05	0.0000001	0.00001	79.5	71.6	47.4	67.0
0.05	0.0000001	0.00005	27.6	6.5	48.7	130.3
0.05	0.000001	0.00001	41.9	11.7	54.0	181.4
0.05	0.000001	0.00005	38.0	7.0	57.0	179.8

Table 68 Test Results for ID-A, where NR= 480, NS= 1000, NE= 1, BR= 0.008.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	35.2	6.3	70.3	169.9
0.0003	0.00000001	0.00005	41.0	6.5	61.2	118.3
0.0003	0.0000001	0.00001	79.4	1.7	47.6	173.0
0.0003	0.0000001	0.00005	62.4	5.3	47.5	115.2
0.0003	0.000001	0.00001	83.3	0.5	61.9	113.1
0.0003	0.000001	0.00005	58.7	1.3	62.0	148.8
0.001	0.00000001	0.00001	45.1	3.0	67.5	161.5
0.001	0.00000001	0.00005	94.2	12.4	72.6	272.1
0.001	0.0000001	0.00001	61.8	5.2	63.7	215.6
0.001	0.0000001	0.00005	81.9	4.9	57.4	213.8
0.001	0.000001	0.00001	90.5	74.8	61.9	110.7
0.001	0.000001	0.00005	55.5	1.9	61.3	181.2
0.05	0.00000001	0.00001	55.9	4.0	50.1	223.2
0.05	0.00000001	0.00005	34.6	0.5	55.5	149.2
0.05	0.0000001	0.00001	74.1	2.8	50.0	144.9
0.05	0.0000001	0.00005	93.3	71.2	47.1	88.8
0.05	0.000001	0.00001	21.9	3.1	54.6	191.0
0.05	0.000001	0.00005	39.3	4.8	54.7	200.6

Table 69 Test Results for ID-A, where NR=480, NS=1000, NE=4, BR=0.0003.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	69.6	10.2	56.5	173.6
0.0003	0.00000001	0.00005	40.7	8.1	52.2	181.5
0.0003	0.0000001	0.00001	52.2	17.6	49.1	174.5
0.0003	0.0000001	0.00005	74.3	10.2	49.6	192.6
0.0003	0.000001	0.00001	81.9	44.4	60.0	136.6
0.0003	0.000001	0.00005	83.6	78.3	61.4	123.6
0.001	0.00000001	0.00001	70.6	42.0	53.4	175.8
0.001	0.00000001	0.00005	82.1	34.1	50.5	156.5
0.001	0.0000001	0.00001	73.8	12.8	65.4	173.8
0.001	0.0000001	0.00005	66.1	34.1	60.1	204.8
0.001	0.000001	0.00001	68.8	25.9	56.8	146.6
0.001	0.000001	0.00005	41.1	10.6	52.4	163.0
0.05	0.00000001	0.00001	67.7	21.7	52.7	153.8
0.05	0.00000001	0.00005	73.6	3.8	54.6	164.9
0.05	0.0000001	0.00001	55.9	9.6	66.8	225.9
0.05	0.0000001	0.00005	88.6	14.4	56.7	218.4
0.05	0.000001	0.00001	67.6	4.1	53.0	154.0
0.05	0.000001	0.00005	76.4	7.7	56.7	178.7

Table 70 Test Results for ID-A, where NR= 480, NS= 1000, NE= 4, BR= 0.001.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	59.4	21.6	56.5	200.7
0.0003	0.00000001	0.00005	83.7	74.6	58.2	73.0
0.0003	0.0000001	0.00001	86.3	42.9	49.0	203.7
0.0003	0.0000001	0.00005	63.0	14.1	56.9	207.3
0.0003	0.000001	0.00001	62.8	22.6	53.1	91.5
0.0003	0.000001	0.00005	60.9	17.7	47.7	176.3
0.001	0.00000001	0.00001	79.8	20.0	53.1	139.9
0.001	0.00000001	0.00005	48.2	2.4	63.8	138.4
0.001	0.0000001	0.00001	64.9	25.0	54.0	178.3
0.001	0.0000001	0.00005	84.3	12.1	51.3	128.2
0.001	0.000001	0.00001	85.0	27.8	49.0	182.6
0.001	0.000001	0.00005	73.2	14.7	48.8	180.2
0.05	0.00000001	0.00001	78.5	70.6	45.8	76.1
0.05	0.00000001	0.00005	81.4	71.6	46.8	78.1
0.05	0.0000001	0.00001	87.2	66.7	50.1	67.5
0.05	0.0000001	0.00005	44.4	17.2	47.6	151.3
0.05	0.000001	0.00001	51.4	10.8	51.8	157.5
0.05	0.000001	0.00005	50.5	10.1	55.8	163.4

Table 71 Test Results for ID-A, where NR= 480, NS= 1000, NE= 4, BR= 0.008.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	82.6	73.7	58.5	74.5
0.0003	0.00000001	0.00005	35.2	8.8	44.6	175.7
0.0003	0.0000001	0.00001	76.2	21.3	45.9	310.4
0.0003	0.0000001	0.00005	52.1	7.0	43.7	151.7
0.0003	0.000001	0.00001	71.2	5.9	61.8	144.8
0.0003	0.000001	0.00005	60.0	4.1	64.7	214.1
0.001	0.00000001	0.00001	88.5	16.5	64.3	124.9
0.001	0.00000001	0.00005	55.3	13.3	52.5	87.4
0.001	0.0000001	0.00001	87.9	75.2	48.1	99.3
0.001	0.0000001	0.00005	67.2	14.6	48.0	140.7
0.001	0.000001	0.00001	86.3	67.9	59.2	87.9
0.001	0.000001	0.00005	84.1	90.9	58.7	50.0
0.05	0.00000001	0.00001	79.9	1.1	55.4	115.7
0.05	0.00000001	0.00005	77.1	6.3	55.7	173.4
0.05	0.0000001	0.00001	84.1	30.9	49.7	155.1
0.05	0.0000001	0.00005	67.3	12.8	48.5	105.8
0.05	0.000001	0.00001	73.7	24.1	59.4	221.6
0.05	0.000001	0.00005	55.6	8.4	69.8	273.4

Table 72 Test Results for ID-A, where NR=480, NS=2400, NE=1, BR=0.0003.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	63.3	26.0	49.5	174.1
0.0003	0.00000001	0.00005	70.9	31.6	52.4	175.4
0.0003	0.0000001	0.00001	75.9	37.8	58.1	199.3
0.0003	0.0000001	0.00005	58.9	22.4	56.0	183.1
0.0003	0.000001	0.00001	66.7	26.2	58.5	169.3
0.0003	0.000001	0.00005	80.7	26.8	58.8	177.4
0.001	0.00000001	0.00001	81.5	86.4	55.7	162.0
0.001	0.00000001	0.00005	74.1	62.5	51.4	147.2
0.001	0.0000001	0.00001	87.5	36.9	61.3	171.6
0.001	0.0000001	0.00005	67.3	44.1	49.0	127.5
0.001	0.000001	0.00001	84.0	33.1	64.9	181.2
0.001	0.000001	0.00005	64.1	37.3	61.6	217.9
0.05	0.00000001	0.00001	36.3	10.1	46.2	173.5
0.05	0.00000001	0.00005	48.5	14.9	47.3	179.1
0.05	0.0000001	0.00001	56.2	22.9	64.7	189.8
0.05	0.0000001	0.00005	55.2	16.7	60.5	163.8
0.05	0.000001	0.00001	64.7	13.1	64.6	171.5
0.05	0.000001	0.00005	51.6	18.0	56.1	143.5

Table 73 Test Results for ID-A, where NR= 480, NS= 2400, NE= 1, BR= 0.001.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	92.1	8.2	55.5	179.9
0.0003	0.00000001	0.00005	95.9	73.9	53.6	82.2
0.0003	0.0000001	0.00001	79.8	16.8	58.4	193.5
0.0003	0.0000001	0.00005	80.0	78.7	62.7	144.2
0.0003	0.000001	0.00001	78.6	36.5	67.4	200.7
0.0003	0.000001	0.00005	75.6	22.6	70.4	195.5
0.001	0.00000001	0.00001	65.2	22.7	50.9	197.6
0.001	0.00000001	0.00005	64.3	23.5	53.9	182.3
0.001	0.0000001	0.00001	91.3	15.9	51.4	131.2
0.001	0.0000001	0.00005	93.7	7.2	54.8	158.0
0.001	0.000001	0.00001	51.1	19.2	52.6	222.0
0.001	0.000001	0.00005	76.2	21.6	58.6	185.4
0.05	0.00000001	0.00001	46.3	12.0	54.8	152.1
0.05	0.00000001	0.00005	82.6	10.0	63.8	200.6
0.05	0.0000001	0.00001	49.8	11.7	58.8	159.4
0.05	0.0000001	0.00005	67.2	15.9	61.2	165.1
0.05	0.000001	0.00001	79.7	4.4	61.4	165.5
0.05	0.000001	0.00005	96.1	15.8	64.1	177.9

Table 74 Test Results for ID-A, where NR= 480, NS= 2400, NE= 1, BR= 0.008.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	53.9	0.4	50.7	124.5
0.0003	0.00000001	0.00005	62.1	9.4	48.9	142.7
0.0003	0.0000001	0.00001	90.1	0.6	57.3	167.0
0.0003	0.0000001	0.00005	15.8	0.1	56.2	112.9
0.0003	0.000001	0.00001	82.3	0.7	66.5	149.2
0.0003	0.000001	0.00005	51.9	0.3	59.9	98.8
0.001	0.00000001	0.00001	84.3	5.1	78.5	293.5
0.001	0.00000001	0.00005	58.7	0.1	71.5	184.9
0.001	0.0000001	0.00001	60.0	1.7	49.3	165.1
0.001	0.0000001	0.00005	80.5	2.7	52.4	214.5
0.001	0.000001	0.00001	65.8	4.6	67.4	132.3
0.001	0.000001	0.00005	53.2	0.7	68.9	155.9
0.05	0.00000001	0.00001	91.1	2.8	56.3	153.9
0.05	0.00000001	0.00005	68.8	0.4	56.3	203.0
0.05	0.0000001	0.00001	11.9	1.3	51.5	182.0
0.05	0.0000001	0.00005	90.1	0.7	53.5	190.3
0.05	0.000001	0.00001	94.3	10.6	52.6	188.0
0.05	0.000001	0.00005	77.9	2.0	57.3	193.4

Table 75 Test Results for ID-A, where NR=480, NS=2400, NE=4, BR=0.0003.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	87.5	39.8	54.7	224.9
0.0003	0.00000001	0.00005	70.0	41.2	48.2	178.9
0.0003	0.0000001	0.00001	90.2	32.0	61.2	186.6
0.0003	0.0000001	0.00005	73.2	38.6	52.4	204.9
0.0003	0.000001	0.00001	89.0	35.2	58.8	190.0
0.0003	0.000001	0.00005	73.6	11.1	57.8	146.7
0.001	0.00000001	0.00001	93.7	50.0	57.0	161.9
0.001	0.00000001	0.00005	45.7	12.1	53.2	139.0
0.001	0.0000001	0.00001	55.6	18.7	55.1	152.4
0.001	0.0000001	0.00005	96.8	40.0	54.6	174.0
0.001	0.000001	0.00001	85.4	22.2	58.3	163.7
0.001	0.000001	0.00005	73.0	45.4	53.4	172.1
0.05	0.00000001	0.00001	27.7	2.4	53.1	176.1
0.05	0.00000001	0.00005	48.5	1.7	55.9	106.3
0.05	0.0000001	0.00001	67.7	28.1	63.8	167.9
0.05	0.0000001	0.00005	43.2	11.7	57.3	238.5
0.05	0.000001	0.00001	64.5	28.7	50.5	164.4
0.05	0.000001	0.00005	54.9	16.7	48.5	163.2

Table 76 Test Results for ID-A, where NR= 480, NS= 2400, NE= 4, BR= 0.001.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	91.7	70.4	69.0	97.0
0.0003	0.00000001	0.00005	83.7	69.0	49.8	90.3
0.0003	0.0000001	0.00001	51.4	7.2	56.2	166.2
0.0003	0.0000001	0.00005	85.4	22.5	63.3	169.2
0.0003	0.000001	0.00001	50.3	19.2	57.3	207.4
0.0003	0.000001	0.00005	57.2	22.6	54.5	207.0
0.001	0.00000001	0.00001	72.5	44.1	49.9	217.6
0.001	0.00000001	0.00005	73.4	26.0	50.5	221.7
0.001	0.0000001	0.00001	79.0	8.0	52.4	104.5
0.001	0.0000001	0.00005	67.4	13.0	51.6	163.8
0.001	0.000001	0.00001	81.3	51.2	69.0	208.2
0.001	0.000001	0.00005	92.1	23.3	51.2	198.4
0.05	0.00000001	0.00001	59.6	12.8	56.8	189.9
0.05	0.00000001	0.00005	54.4	11.2	61.5	234.2
0.05	0.0000001	0.00001	24.6	5.5	46.7	153.5
0.05	0.0000001	0.00005	31.3	10.0	44.4	141.7
0.05	0.000001	0.00001	13.7	1.5	51.2	146.6
0.05	0.000001	0.00005	11.3	1.2	51.2	89.1

Table 77 Test Results for ID-A, where NR= 480, NS= 2400, NE= 4, BR= 0.008.

RPR	TR	MR	% Acc	% Comp	TrT	TeT
0.0003	0.00000001	0.00001	99.2	66.0	64.2	105.5
0.0003	0.00000001	0.00005	92.6	78.9	61.9	82.4
0.0003	0.0000001	0.00001	74.5	4.3	51.6	166.2
0.0003	0.0000001	0.00005	83.4	70.0	59.1	89.5
0.0003	0.000001	0.00001	83.9	71.7	58.7	113.8
0.0003	0.000001	0.00005	40.1	8.8	58.3	241.9
0.001	0.00000001	0.00001	52.8	7.4	63.8	275.7
0.001	0.00000001	0.00005	65.3	2.1	70.0	203.0
0.001	0.0000001	0.00001	38.7	6.9	56.1	200.2
0.001	0.0000001	0.00005	33.6	3.9	58.2	150.9
0.001	0.000001	0.00001	93.5	71.0	73.2	136.2
0.001	0.000001	0.00005	41.3	1.7	57.4	133.0
0.05	0.00000001	0.00001	88.0	4.2	56.6	224.9
0.05	0.00000001	0.00005	83.1	9.0	56.1	262.3
0.05	0.0000001	0.00001	92.4	67.0	57.2	105.2
0.05	0.0000001	0.00005	95.8	4.1	55.1	159.8
0.05	0.000001	0.00001	44.1	12.6	54.1	213.5
0.05	0.000001	0.00005	28.0	5.9	62.7	248.9

APPENDIX E

Code for Intrusion Detector A

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define DEFAULT_STRENGTH 1.0
#define ZERO_CHAR '0'
#define ONE_CHAR '1'
#define MATCH_ANY_CHAR 'x'

#define MODLUS 2147483647
#define MULT1       24112
#define MULT2       26143
#define RAND_RATIO_20 0.5
#define RAND_RATIO_21 0.5
#define RAND_RATIO_30 0.1
#define RAND_RATIO_31 0.1
#define RAND_RATIO_32 0.8

/* Set the default seeds for all 100 streams. */
static long zrng[] =
{         1,
 1973272912,  281629770,   20006270, 1280689831, 2096730329,
 1933576050,  913566091,  246780520, 1363774876,   04901985,
 1511192140, 1259851944,  824064364,  150493284,  242708531,
   75253171, 1964472944, 1202299975,  233217322, 1911216000,
  726370533,  403498145,  993232223, 1103205531,  762430696,
 1922803170, 1385516923,   76271663,  413682397,  726466604,
  336157058, 1432650381, 1120463904,  595778810,  877722890,
 1046574445,   68911991, 2088367019,  748545416,  622401386,
 2122378830,  640690903, 1774806513, 2132545692, 2079249579,
   78130110,  852776735, 1187867272, 1351423507, 1645973084,
 1997049139,  922510944, 2045512870,  898585771,  243649545,
 1004818771,  773686062,  403188473,  372279877, 1901633463,
  498067494, 2087759558,  493157915,  597104727, 1530940798,
 1814496276,  536444882, 1663153658,  855503735,   67784357,
 1432404475,  619691088,  119025595,  880802310,  176192644,
 1116780070,  277854671, 1366580350, 1142483975, 2026948561,
 1053920743,  786262391, 1792203830, 1494667770, 1923011392,
 1433700034, 1244184613, 1147297105,  539712780, 1545929719,
  190641742, 1645390429,  264907697,  620389253, 1502074852,
  927711160,  364849192, 2049576050,  638580085,  547070247
};
#define list_head_ptr(list) ((list)->head_ptr)
#define list_next_ptr(list) ((list)->next_ptr)
#define list_condition_ptr(list) ((list)->condition_ptr)
#define list_message_ptr(list) ((list)->message_ptr)
#define list_action_ptr(list) ((list)->action_ptr)
#define list_strength_ptr(list) ((list)->strength_ptr)
struct list_node {
```

```
     char *condition_ptr;
     char *message_ptr;
     char *action_ptr;
     double *strength_ptr;
     struct list_node *next_ptr;
};

struct list {
     struct list_node *head_ptr;
     int size;
     double *strength_of_environment_ptr;
};

struct secondary_list_node {
     struct list_node *hosted_node_ptr;
     struct secondary_list_node *next_ptr;
};

struct secondary_list {
     struct secondary_list_node *head_ptr;
};

void generate_rule_set(struct list *rule_set_ptr);
struct list_node *create_one_rule_ptr();
void generate_condition(char *new_condition_ptr);
void generate_message(char *new_message_ptr);
void generate_action(char *new_action_ptr);
void train(struct list *rule_set_ptr);
void process_one_training_step (struct list *rule_set_ptr, char
     *input_message_ptr);
struct list_node *choose_node_for_action_ptr (struct list
     *rule_set_ptr, char *input_message_ptr);
struct list_node *hold_an_auction_ptr (struct list *rule_set_ptr,
     double *activator_strength_ptr, char *input_message_ptr);
char check_for_loop (struct secondary_list *loop_list_ptr, struct
     list_node *auction_winner_ptr);
double collect_life_tax(struct list *rule_set_ptr);
void pay_subsidies(struct list *rule_set_ptr, double
     total_subsidy);
void process_genetic_operations (struct list *rule_set_ptr);
void eliminate_weak_rules (struct list *rule_set_ptr);
struct list_node *choose_a_rule_to_be_eliminated_ptr(struct list
     *rule_set_ptr);
void reproduce_strong_rules(struct list *rule_set_ptr);
struct list_node *reproduce_one_rule (struct list *rule_set_ptr);
void choose_two_rules(int *first_parent_index_ptr, int
     *second_parent_index_ptr, int number_of_rules);
void cross_over(char *child_ptr, char *first_parent_ptr, char
     *second_parent_ptr);
void test(struct list *rule_set_ptr);
void process_one_test_step (struct list *rule_set_ptr, char
     *input_message_ptr, char *correct_action_ptr);
struct list_node *test_choose_node_for_action_ptr (struct list
     *rule_set_ptr, char *input_message_ptr);
struct list_node *test_hold_an_auction_ptr (struct list
     *rule_set_ptr, double *activator_strength_ptr, char
     *input_message_ptr);
```

```
struct list *list_init_ptr ();
struct list_node *list_alloc_node_memory_ptr();
struct list *list_duplicate_ptr(struct list
     *list_to_be_duplicated_ptr);
void list_copy_node (struct list_node *destination_node_ptr,
     struct list_node *source_node_ptr);
void list_print (struct list *list_to_be_printed_ptr);
void list_print_node (struct list_node *node_to_be_printed_ptr,
     char as_a_single_node);
void list_insert(struct list *list_to_be_inserted_to_ptr, struct
     list_node *node_to_be_inserted_ptr);
void list_insert_as_head(struct list *list_to_be_inserted_to_ptr,
     struct list_node *node_to_be_inserted_ptr);
void list_insert_next(struct list *list_to_be_inserted_to_ptr,
     struct list_node *node_to_be_inserted_after_ptr, struct
     list_node *node_to_be_inserted_ptr);
struct list_node *list_locate_ptr (struct list
     *list_to_be_located_in_ptr, struct list_node
     *node_to_be_located_ptr, char *same_exists_ptr);
int list_compare_two_nodes (struct list_node
     *first_list_node_to_compare_ptr, struct list_node
     *second_list_node_to_compare_ptr);
void list_destroy_list (struct list *list_to_be_destroyed_ptr);
void list_remove_head (struct list *list_to_be_removed_from_ptr);
void list_remove_next (struct list *list_to_be_removed_from_ptr,
     struct list_node *previous_node_ptr);
struct list_node *list_get_node_by_index_ptr (struct list
     *list_to_get_from_ptr, int node_index);
void list_destroy_node (struct list_node
     *node_to_be_destroyed_ptr);
struct secondary_list_node *create_secondary_list_node_ptr(struct
     list_node *node_to_be_hosted_ptr);
struct secondary_list *secondary_list_init_ptr ();
void secondary_list_insert_as_head(struct secondary_list
     *secondary_list_to_be_inserted_to_ptr, struct
     secondary_list_node *secondary_list_node_to_be_inserted_ptr);
void secondary_list_destroy_list (struct secondary_list
     *secondary_list_to_be_destroyed_ptr);
void secondary_list_remove_head (struct secondary_list
     *secondary_list_to_be_removed_from_ptr);
void secondary_list_destroy_node (struct secondary_list_node
     *secondary_list_node_to_be_destroyed_ptr);
char message_match_condition (char *message, char *condition);
char chars_match (char state_char, char condition_char);
void mutate_rule(struct list_node *rule_to_be_mutated_ptr);
void mutate_bit2(char *bit_to_be_mutated_ptr);
void mutate_bit3(char *bit_to_be_mutated_ptr);
char rand_char2();
char rand_char3();
float lcgrand(int stream);
void lcgrandst (long zset, int stream);
long lcgrandgt (int stream);

/* Global Variables */
FILE *out_file_ptr;
int num_bits;
int num_rules;
```

```
int num_steps;
int num_eli;
double bid_rate;
double rp_rate;
double tax_rate;
double mut_rate;

int number_of_true_positive;
int number_of_true_negative;
int number_of_false_positive;
int number_of_false_negative;

int main (){
     struct list *rule_set_ptr;

     FILE *param_file_ptr;
     extern FILE *out_file_ptr;
     extern int num_bits;
     extern int num_rules;
     extern int num_steps;
     extern int num_eli;
     extern double bid_rate;
     extern double rp_rate;
     extern double tax_rate;
     extern double mut_rate;

     /* Open the Parameters File */
     param_file_ptr = fopen("ida.prm","r");
     /* Make sure that it is opened */
     if(param_file_ptr == NULL){
          printf("\n\nERROR: main : NO PARAMETERSS FILE PRESENT
               !!!\n\n");
          printf("\n\nSO QUITTING!!!\n\n");
          system("PAUSE");
          exit(1);
     }
     /* Open (create if necessary) The Output File */
     out_file_ptr = fopen("ida.out","a+");
     /* Make sure that it is opened */
     if(out_file_ptr == NULL){
          printf("\n\nERROR: main : OUTPUT FILE CAN NOT BE OPENED
               CREATED!!!\n\n");
          printf("\n\nSO QUITTING!!!\n\n");
          system("PAUSE");
          exit(1);
     }
     /* Start Processing the Parameter sets */
     /* Read One Set of Parameters */
     fscanf(param_file_ptr, "%d %d %d %d %lf %lf %lf %lf",
          &num_bits, &num_rules, &num_steps, &num_eli, &bid_rate,
          &rp_rate, &tax_rate, &mut_rate);
     printf("%d %d %d %d %f %f %f %f\n", num_bits, num_rules,
          num_steps, num_eli, bid_rate, rp_rate, tax_rate,
          mut_rate);
     fprintf(out_file_ptr, "%10d, %10d, %10d, %10d, %15.10f,
          %15.10f, %15.10f, %15.10f, ", num_bits, num_rules,
          num_steps, num_eli, bid_rate, rp_rate, tax_rate,
```

```
          mut_rate);

     /* Reset the Random Number Stream */
     lcgrandst (1, 0);

     /* Initialize the list that will accomodate the Rule Set */
     rule_set_ptr = list_init_ptr();
     /* Generate the Rule Set */
     generate_rule_set(rule_set_ptr);
     /* Train the Rule Set */
     train(rule_set_ptr);
     /* Test the Rule Set */
     test(rule_set_ptr);
     /* Destroy the Rule Set */
     list_destroy_list (rule_set_ptr);

     return(0);
}

void generate_rule_set(struct list *rule_set_ptr){

     struct list_node *new_rule_ptr;

     while(rule_set_ptr->size < num_rules) {
          new_rule_ptr = create_one_rule_ptr();
          list_insert(rule_set_ptr, new_rule_ptr);
     }
}

struct list_node *create_one_rule_ptr() {
     struct list_node *new_rule_ptr;

     new_rule_ptr = list_alloc_node_memory_ptr();
     generate_condition(new_rule_ptr->condition_ptr);
     generate_message(new_rule_ptr->message_ptr);
     generate_action(new_rule_ptr->action_ptr);
     *(new_rule_ptr->strength_ptr) = DEFAULT_STRENGTH;

     return(new_rule_ptr);
}

void generate_condition(char *new_condition_ptr) {
     int bit_counter;

     for (bit_counter = 0; bit_counter < num_bits; bit_counter++){
          *new_condition_ptr = rand_char3();
          new_condition_ptr++;
     }
     *new_condition_ptr = NULL;
}

void generate_message(char *new_message_ptr) {
     int bit_counter;

     for (bit_counter = 0; bit_counter < num_bits; bit_counter++){
          *new_message_ptr = rand_char2();
          new_message_ptr++;
```

```
     }
     *new_message_ptr = NULL;
}

void generate_action(char *new_action_ptr) {

     *new_action_ptr = rand_char2();
     new_action_ptr++;
     *new_action_ptr = NULL;
}


void train(struct list *rule_set_ptr) {
     FILE *training_file_ptr;
     extern FILE *out_file_ptr;
     char *input_message_ptr;
     int training_step_counter;
     double training_start_time;
     double training_end_time;
     double training_duration;

     /* Open The Training File */
     training_file_ptr = fopen("id.trn","r");
     /* Make sure that it is opened */
     if(training_file_ptr == NULL){
          printf("\n\nERROR: main : NO TRAINING FILE PRESENT
               !!!\n\n");
          printf("\n\nSO QUITTING!!!\n\n");
          system("PAUSE");
          exit(1);
     }
     /* Reset Input Message Counter */
     training_step_counter = 0;
     /* Allocate Memory for an Input Message */
     input_message_ptr = (char *)malloc((num_bits + 1) *
          sizeof(char));

     /* Get Training Start Time */
     training_start_time = clock();

     /* Start Training (Train until End of Training File) */
     while(!feof(training_file_ptr)){
          training_step_counter++;
          /* Scan an Input Message */
          fscanf(training_file_ptr, "%s", input_message_ptr);
          /* Process One Training Step */
          /*list_print(rule_set_ptr);
          system("pause");*/
          process_one_training_step(rule_set_ptr,
               input_message_ptr);
          /* Process Genetic Operations once in num_steps Steps */
          if((training_step_counter % num_steps) == 0){
               process_genetic_operations (rule_set_ptr);
          }
     }
     /* Get Training End Time */
     training_end_time = clock();
```

```
     /* Close the Training File */
     /* Compute Training Time */
     training_duration = (training_end_time -
          training_start_time)/CLOCKS_PER_SEC;
     /* Output the Training Time */
     fprintf(out_file_ptr,"%10.3f,", training_duration);
}

void process_one_training_step (struct list *rule_set_ptr, char
*input_message_ptr) {
     double total_tax;
     struct list_node *active_rule_ptr;
     double mut_prob;
     int mut_rule_index;
     struct list_node *mut_rule_ptr;
     active_rule_ptr = NULL;
     total_tax = 0.0;

     active_rule_ptr = choose_node_for_action_ptr(rule_set_ptr,
          input_message_ptr);

     if (*(active_rule_ptr->action_ptr) == '0') {
          *(active_rule_ptr->strength_ptr) += rp_rate *
               (*(rule_set_ptr->strength_of_environment_ptr));
          *(rule_set_ptr->strength_of_environment_ptr) *= (1 -
               rp_rate);
     }
     else if (*(active_rule_ptr->action_ptr) == '1'){
          *(rule_set_ptr->strength_of_environment_ptr) += rp_rate *
               (*(active_rule_ptr->strength_ptr));
          *(active_rule_ptr->strength_ptr) *= (1 - rp_rate);
     }

     total_tax = collect_life_tax(rule_set_ptr);

     pay_subsidies(rule_set_ptr, total_tax);

     mut_prob = lcgrand(0);
     if (mut_prob < mut_rate){
          mut_rule_index = ((double)num_rules)*lcgrand(0);
          mut_rule_ptr = list_get_node_by_index_ptr(rule_set_ptr,
               mut_rule_index);
          mutate_rule(mut_rule_ptr);
     }
}

struct list_node *choose_node_for_action_ptr (struct list
     *rule_set_ptr, char *input_message_ptr) {
     struct secondary_list *loop_list_ptr;
     struct list_node *auction_winner_ptr;
     struct list_node *previous_auction_winner_ptr;
     struct secondary_list_node
          *secondary_list_node_for_auction_winner_ptr;
     char loop_warning;

     loop_list_ptr = secondary_list_init_ptr();
```

```
     auction_winner_ptr = NULL;
     previous_auction_winner_ptr = NULL;


     auction_winner_ptr =
hold_an_auction_ptr(rule_set_ptr,rule_set_ptr-
>strength_of_environment_ptr,input_message_ptr);
     while(auction_winner_ptr != NULL) {
          secondary_list_node_for_auction_winner_ptr =
               create_secondary_list_node_ptr(auction_winner_ptr);
          secondary_list_insert_as_head (loop_list_ptr,
               secondary_list_node_for_auction_winner_ptr);
          previous_auction_winner_ptr = auction_winner_ptr;
          auction_winner_ptr = hold_an_auction_ptr(rule_set_ptr,
               previous_auction_winner_ptr->strength_ptr,
               previous_auction_winner_ptr->message_ptr);
          loop_warning = check_for_loop (loop_list_ptr,
               auction_winner_ptr);
          if (loop_warning == 1) {
               mutate_rule(auction_winner_ptr);
               secondary_list_destroy_list(loop_list_ptr);
               loop_list_ptr = secondary_list_init_ptr();
          }
     }

     secondary_list_destroy_list(loop_list_ptr);

     return(previous_auction_winner_ptr);
}

struct list_node *hold_an_auction_ptr (struct list *rule_set_ptr,
     double *activator_strength_ptr, char *input_message_ptr) {
     struct list_node *current_rule_ptr;
     struct list_node *auction_winner_ptr;
     struct secondary_list *bidder_list_ptr;
     struct secondary_list_node *current_bidder_host_ptr;
     double bid_of_current_rule;
     double total_strength;
     double rand_num;

     current_rule_ptr = list_head_ptr(rule_set_ptr);
     bidder_list_ptr = secondary_list_init_ptr();
     auction_winner_ptr = NULL;
     bid_of_current_rule = 0.0;
     total_strength = 0.0;

     while (current_rule_ptr != NULL) {

          if(message_match_condition(input_message_ptr,
          list_condition_ptr(current_rule_ptr))){
               bid_of_current_rule = bid_rate *
                    (*list_strength_ptr(current_rule_ptr));
               *(list_strength_ptr(current_rule_ptr)) *= (1 -
                    bid_rate);
               *activator_strength_ptr = *activator_strength_ptr +
                    bid_of_current_rule;
```

```
                current_bidder_host_ptr =
                     create_secondary_list_node_ptr(current_rule_ptr
                     );
                secondary_list_insert_as_head(bidder_list_ptr,
                     current_bidder_host_ptr);
           }
           current_rule_ptr = list_next_ptr(current_rule_ptr);
      }
      current_bidder_host_ptr = list_head_ptr(bidder_list_ptr);

      while (current_bidder_host_ptr != NULL) {
           total_strength += (*(current_bidder_host_ptr->
                hosted_node_ptr->strength_ptr));
           current_bidder_host_ptr =
                list_next_ptr(current_bidder_host_ptr);
      }
      rand_num = (double)(total_strength*lcgrand(0));
      current_bidder_host_ptr = list_head_ptr(bidder_list_ptr);
      if(rand_num > (*(current_bidder_host_ptr->hosted_node_ptr->
      strength_ptr))) {
           rand_num -= (*(current_bidder_host_ptr->hosted_node_ptr->
                strength_ptr));
           current_bidder_host_ptr =
                list_next_ptr(current_bidder_host_ptr);
      }

      while (rand_num > (*(current_bidder_host_ptr->
      hosted_node_ptr->strength_ptr))) {
           rand_num -= (*(current_bidder_host_ptr-> hosted_node_ptr
                ->strength_ptr));
           current_bidder_host_ptr =
                list_next_ptr(current_bidder_host_ptr);
      }
      auction_winner_ptr = current_bidder_host_ptr->
           hosted_node_ptr;

      secondary_list_destroy_list(bidder_list_ptr);

      return(auction_winner_ptr);
}

char check_for_loop (struct secondary_list *loop_list_ptr, struct
list_node *auction_winner_ptr) {
      char loop_warning;
      struct secondary_list_node *current_loop_list_node_ptr;
      struct list_node *current_node_ptr;

      loop_warning = 0;
      current_loop_list_node_ptr = list_head_ptr(loop_list_ptr);
      while (current_loop_list_node_ptr != NULL) {
           current_node_ptr = current_loop_list_node_ptr->
                hosted_node_ptr;
           if (list_compare_two_nodes
           (current_node_ptr,auction_winner_ptr) == 0) {
                loop_warning = 1;
           }
           current_loop_list_node_ptr =
```

```
                list_next_ptr(current_loop_list_node_ptr);
     }
     return(loop_warning);
}

double collect_life_tax(struct list *rule_set_ptr) {
     struct list_node *current_rule_ptr;
     double total_tax;

     current_rule_ptr = NULL;
     total_tax = 0.0;

     current_rule_ptr = list_head_ptr(rule_set_ptr);
     while (current_rule_ptr != NULL) {
          if (*(current_rule_ptr->action_ptr) == '0') {
               total_tax += tax_rate * *(current_rule_ptr->
                    strength_ptr);
               *(current_rule_ptr->strength_ptr) *= (1 - tax_rate);
          }
          current_rule_ptr = list_next_ptr(current_rule_ptr);
     }
     return(total_tax);
}

void pay_subsidies(struct list *rule_set_ptr, double
total_subsidy) {
     struct list_node *current_rule_ptr;
     double subsidy_per_rule;
     int number_of_rules_for_subsidy;

     current_rule_ptr = NULL;
     subsidy_per_rule = 0.0;
     number_of_rules_for_subsidy = 0;

     current_rule_ptr = list_head_ptr(rule_set_ptr);
     while (current_rule_ptr != NULL) {
          if (*(current_rule_ptr->action_ptr) == '1') {
               number_of_rules_for_subsidy++;
          }
          current_rule_ptr = list_next_ptr(current_rule_ptr);
     }
     subsidy_per_rule = total_subsidy /
          number_of_rules_for_subsidy;
     current_rule_ptr = list_head_ptr(rule_set_ptr);
     while (current_rule_ptr != NULL) {
          if (*(current_rule_ptr->action_ptr) == '1') {
               *(current_rule_ptr->strength_ptr) +=
                    subsidy_per_rule;
          }
          current_rule_ptr = list_next_ptr(current_rule_ptr);
     }
}

void process_genetic_operations (struct list *rule_set_ptr) {
     eliminate_weak_rules (rule_set_ptr);
     reproduce_strong_rules (rule_set_ptr);
}
```

```
void eliminate_weak_rules (struct list *rule_set_ptr) {
     struct list_node *prior_rule_ptr;
     int rules_eliminated;
     prior_rule_ptr = NULL;

     for (rules_eliminated = 0; rules_eliminated < num_eli;
rules_eliminated++) {
          prior_rule_ptr =
               choose_a_rule_to_be_eliminated_ptr(rule_set_ptr);

          if (prior_rule_ptr == NULL) {
               list_remove_head(rule_set_ptr);
          }
          else {
               list_remove_next(rule_set_ptr, prior_rule_ptr);
          }
     }
}

struct list_node *choose_a_rule_to_be_eliminated_ptr(struct list
*rule_set_ptr) {
     struct list_node *current_rule_ptr;
     struct list_node *rule_at_back_ptr;
     struct list_node *prior_to_weakest_rule_ptr;

     double lowest_strength;

     current_rule_ptr = NULL;
     rule_at_back_ptr = NULL;
     prior_to_weakest_rule_ptr = NULL;
     lowest_strength = 0.0;

     current_rule_ptr = list_head_ptr(rule_set_ptr);
     lowest_strength = *(current_rule_ptr->strength_ptr);

     while(current_rule_ptr != NULL) {
          if ((*(current_rule_ptr->strength_ptr)) <
          lowest_strength) {
               prior_to_weakest_rule_ptr = rule_at_back_ptr;
          }
          rule_at_back_ptr = current_rule_ptr;
          current_rule_ptr = list_next_ptr(current_rule_ptr);
     }
     return(prior_to_weakest_rule_ptr);
}

void reproduce_strong_rules(struct list *rule_set_ptr) {
     struct list_node *new_rule_ptr;
     struct list *parent_list_ptr;

     new_rule_ptr = NULL;
     parent_list_ptr = list_duplicate_ptr(rule_set_ptr);
     while((rule_set_ptr->size) < num_rules) {
          new_rule_ptr = reproduce_one_rule(parent_list_ptr);
          list_insert(rule_set_ptr, new_rule_ptr);
     }
```

```
     list_destroy_list(parent_list_ptr);
}

struct list_node *reproduce_one_rule (struct list *rule_set_ptr) {
     int first_parent_index;
     int second_parent_index;
     struct list_node *first_parent_ptr;
     struct list_node *second_parent_ptr;
     struct list_node *new_rule_ptr;

     first_parent_index = 0;
     second_parent_index = 0;
     first_parent_ptr = NULL;
     second_parent_ptr = NULL;
     new_rule_ptr = NULL;

    choose_two_rules(&first_parent_index, &second_parent_index,
          rule_set_ptr->size);
     first_parent_ptr = list_get_node_by_index_ptr(rule_set_ptr,
          first_parent_index);
     second_parent_ptr =list_get_node_by_index_ptr(rule_set_ptr,
          second_parent_index);
     new_rule_ptr = list_alloc_node_memory_ptr();
          cross_over(new_rule_ptr->condition_ptr, first_parent_ptr
          -> condition_ptr, second_parent_ptr->condition_ptr);
     cross_over(new_rule_ptr->message_ptr, first_parent_ptr->
          message_ptr, second_parent_ptr->message_ptr);
     cross_over(new_rule_ptr->action_ptr, first_parent_ptr->
          action_ptr, second_parent_ptr->action_ptr);
     *(new_rule_ptr->strength_ptr) = DEFAULT_STRENGTH;

     return (new_rule_ptr);
}

void choose_two_rules(int *first_parent_index_ptr, int
*second_parent_index_ptr, int number_of_rules) {
     *first_parent_index_ptr = (int)(number_of_rules *
          lcgrand(0));
     *second_parent_index_ptr = (int)(number_of_rules *
          lcgrand(0));
     while(*first_parent_index_ptr == *second_parent_index_ptr) {
          *second_parent_index_ptr = (int)(number_of_rules *
               lcgrand(0));
     }
}

void cross_over(char *child_ptr, char *first_parent_ptr, char
*second_parent_ptr) {
     int parent_len;
     int crossing_point;

     parent_len = 0;
     crossing_point = 0;

     if(strlen(first_parent_ptr) == strlen(second_parent_ptr)){
          parent_len = strlen(first_parent_ptr);
     }
```

```
        else {
             printf("There is something wrong here cross_over");
        }
        crossing_point = (int)((parent_len - 1) * lcgrand(0));

        strncpy(child_ptr, first_parent_ptr, crossing_point);

        child_ptr = child_ptr + crossing_point;
        second_parent_ptr = second_parent_ptr + crossing_point;

        strncpy(child_ptr, second_parent_ptr, (parent_len -
             crossing_point));

        child_ptr = child_ptr + (parent_len - crossing_point);

        *child_ptr = NULL;
}

void test(struct list *rule_set_ptr) {
        FILE *test_file_ptr;
        char *input_message_ptr;
        char *correct_action_ptr;

        double test_start_time;
        double test_end_time;
        double test_duration;

        extern int number_of_true_positive;
        extern int number_of_true_negative;
        extern int number_of_false_positive;
        extern int number_of_false_negative;


        /* Open The Test File */
        test_file_ptr = fopen("id.tst","r");
        /* Make sure that it is opened */
             if(test_file_ptr == NULL){
             printf("\n\nERROR: main : NO TEST FILE PRESENT !!!\n\n");
             printf("\n\nSO QUITTING!!!\n\n");
             system("PAUSE");
             exit(1);
        }

        /* Allocate Memory for an Input Message and Correct Action*/
        input_message_ptr = (char *)malloc((num_bits + 1) *
             sizeof(char));
        correct_action_ptr = (char *)malloc(2*sizeof(char));
        /* Reset Oupput Counters */
        number_of_true_positive = 0;
        number_of_true_negative = 0;
        number_of_false_positive = 0;
        number_of_false_negative = 0;

        /* Get Test Start Time */
        test_start_time = clock();
        /* Start Test (Process Test until End of Test File */
        while(!feof(test_file_ptr)){
```

```
          /* Scan an Input Message and Correct Action */
     fscanf(test_file_ptr, "%s %s", input_message_ptr,
          correct_action_ptr);
          /* Process One Test Step */
     process_one_test_step(rule_set_ptr, input_message_ptr,
          correct_action_ptr);
     }
     /* Get Test End Time */
     test_end_time = clock();
     /* Close The Test File */
     fclose(test_file_ptr);
     /* Compute Test Time */
     test_duration = (test_end_time -
          test_start_time)/CLOCKS_PER_SEC;
     /* Output the Test Time */
     fprintf(out_file_ptr,"%10.3f,", test_duration);
     /* Output the Test Results */
     fprintf(out_file_ptr,"%10d,%10d,%10d,%10d\n",
          number_of_true_positive, number_of_false_positive,
          number_of_true_negative, number_of_false_negative);
}

void process_one_test_step (struct list *rule_set_ptr, char
*input_message_ptr, char *correct_action_ptr) {
     struct list_node *active_rule_ptr;

     extern int number_of_true_positive;
     extern int number_of_true_negative;
     extern int number_of_false_positive;
     extern int number_of_false_negative;

     active_rule_ptr = NULL;

     active_rule_ptr = test_choose_node_for_action_ptr
          (rule_set_ptr, input_message_ptr);
     if (((active_rule_ptr->action_ptr) == NULL) &&
          ((*correct_action_ptr) == '0')) {
          number_of_false_negative++;
     }
     else if (((active_rule_ptr->action_ptr) == NULL) &&
          ((*correct_action_ptr) == '1')){
          number_of_true_negative++;
     }
     else if (((*(active_rule_ptr->action_ptr)) == '0') &&
     ((*correct_action_ptr) == '0')) {
          number_of_true_positive++;
          }
     else if (((*(active_rule_ptr->action_ptr)) == '1') &&
     ((*correct_action_ptr) == '1')){
          number_of_true_negative++;
     }
     else if (((*(active_rule_ptr->action_ptr)) == '0') &&
     ((*correct_action_ptr) == '1')){
          number_of_false_positive++;
     }
     else if (((*(active_rule_ptr->action_ptr)) == '1') &&
     ((*correct_action_ptr) == '0')){
```

```
          number_of_false_negative++;
     }
}

struct list_node *test_choose_node_for_action_ptr (struct list
*rule_set_ptr, char *input_message_ptr) {
     struct secondary_list *loop_list_ptr;
     struct list_node *auction_winner_ptr;
     struct list_node *previous_auction_winner_ptr;
     struct secondary_list_node
          *secondary_list_node_for_auction_winner_ptr;
     char loop_warning;

     loop_warning = 0;
     loop_list_ptr = secondary_list_init_ptr();

     auction_winner_ptr = NULL;
     previous_auction_winner_ptr = NULL;
     auction_winner_ptr =
          test_hold_an_auction_ptr(rule_set_ptr,rule_set_ptr->
          strength_of_environment_ptr,input_message_ptr);
     while(auction_winner_ptr != NULL) {
          secondary_list_node_for_auction_winner_ptr =
               create_secondary_list_node_ptr(auction_winner_ptr);
          secondary_list_insert_as_head (loop_list_ptr,
               secondary_list_node_for_auction_winner_ptr);
          previous_auction_winner_ptr = auction_winner_ptr;
          auction_winner_ptr =
               test_hold_an_auction_ptr(rule_set_ptr,
               previous_auction_winner_ptr->
               strength_ptr,previous_auction_winner_ptr->
               message_ptr);
          loop_warning = check_for_loop (loop_list_ptr,
               auction_winner_ptr);
          if (loop_warning == 1) {
               auction_winner_ptr = NULL;
               previous_auction_winner_ptr = NULL;
          }
     }
     secondary_list_destroy_list(loop_list_ptr);

     return(previous_auction_winner_ptr);
}

struct list_node *test_hold_an_auction_ptr (struct list
*rule_set_ptr, double *activator_strength_ptr, char
*input_message_ptr) {
     struct list_node *current_rule_ptr;
     struct list_node *auction_winner_ptr;
     struct secondary_list *bidder_list_ptr;
     struct secondary_list_node *current_bidder_host_ptr;
     double bid_of_current_rule;
     double total_strength;
     double rand_num;

     current_rule_ptr = list_head_ptr(rule_set_ptr);
     bidder_list_ptr = secondary_list_init_ptr();
```

```
     auction_winner_ptr = NULL;
     bid_of_current_rule = 0.0;
     total_strength = 0.0;

     while (current_rule_ptr != NULL) {
          if(message_match_condition(input_message_ptr,
          list_condition_ptr(current_rule_ptr))){
               current_bidder_host_ptr =
               create_secondary_list_node_ptr(current_rule_ptr);
               secondary_list_insert_as_head(bidder_list_ptr,
               current_bidder_host_ptr);
          }
          current_rule_ptr = list_next_ptr(current_rule_ptr);
     }
     current_bidder_host_ptr = list_head_ptr(bidder_list_ptr);
     while (current_bidder_host_ptr != NULL) {
          total_strength += (*(current_bidder_host_ptr->
          hosted_node_ptr->strength_ptr));
          current_bidder_host_ptr =
          list_next_ptr(current_bidder_host_ptr);
     }
     rand_num = (double)(total_strength*lcgrand(0));
     current_bidder_host_ptr = list_head_ptr(bidder_list_ptr);
      if(rand_num > (*(current_bidder_host_ptr->hosted_node_ptr->
          strength_ptr))) {
          rand_num -= (*(current_bidder_host_ptr->hosted_node_ptr->
               strength_ptr));
          current_bidder_host_ptr =
               list_next_ptr(current_bidder_host_ptr);
     }

     while (rand_num > (*(current_bidder_host_ptr->
     hosted_node_ptr->strength_ptr))) {
          rand_num -= (*(current_bidder_host_ptr->hosted_node_ptr->
               strength_ptr));
          current_bidder_host_ptr =
               list_next_ptr(current_bidder_host_ptr);
     }

          auction_winner_ptr = current_bidder_host_ptr->
               hosted_node_ptr;

     secondary_list_destroy_list(bidder_list_ptr);

     return(auction_winner_ptr);
}

struct list *list_init_ptr (){

     struct list *list_to_be_init_ptr;

     /* Allocate memory for the List */
     list_to_be_init_ptr=malloc(sizeof(struct list));
     /* Make sure that it has been allocated */
     if (list_to_be_init_ptr==NULL) {
          printf("\n\nlist_init_ptr: ERROR : NOT ENOUGH MEMORY :
               list_to_be_init_ptr\n");
```

```
            fprintf(stderr,"\n\nlist_init_ptr: ERROR : NOT ENOUGH
                MEMORY : list_to_be_init_ptr\n");
            system("PAUSE");
            exit(1);
        }
        /* Allocate memory for the strength of environment */
        list_to_be_init_ptr->strength_of_environment_ptr =
            malloc(sizeof(double));
        /* Make sure that it has been allocated */
        if ((list_to_be_init_ptr-> strength_of_environment_ptr) ==
        NULL) {
            printf("\n\nlist_init_ptr : ERROR : NOT ENOUGH MEMORY :
                strength_of_environment_ptr\n");
            fprintf(stderr,"\n\nlist_init_ptr : ERROR : NOT ENOUGH
                MEMORY : strength_of_environment_ptr\n");
            system("PAUSE");
            exit(1);
        }

        /* Set initial Values */
        list_to_be_init_ptr->head_ptr = NULL;
        list_to_be_init_ptr->size=0;
        *(list_to_be_init_ptr->strength_of_environment_ptr) =
            DEFAULT_STRENGTH;
        return(list_to_be_init_ptr);
    }

    struct list_node *list_alloc_node_memory_ptr(){
        struct list_node *new_node_ptr;
        extern int num_bits;

        new_node_ptr = NULL;

        new_node_ptr = malloc(sizeof(struct list_node));
        if (new_node_ptr == NULL) {
            printf("\n\ncreate_one_rule_ptr: ERROR : NOT ENOUGH
                MEMORY : new_node_ptr\n");
            fprintf(stderr,"\n\ncreate_one_rule_ptr: ERROR : NOT
                ENOUGH MEMORY : new_node_ptr\n");
            system("PAUSE");
            exit(1);
        }
        new_node_ptr->condition_ptr = (char *)malloc((num_bits + 1)
            *sizeof(char));
        if (new_node_ptr == NULL) {
            printf("\n\ncreate_one_rule_ptr: ERROR : NOT ENOUGH
                MEMORY : new_node_ptr->condition_ptr\n");
            fprintf(stderr,"\n\ncreate_one_rule_ptr: ERROR : NOT
                ENOUGH MEMORY : new_node_ptr->condition_ptr\n");
            system("PAUSE");
            exit(1);
        }
            new_node_ptr->message_ptr = (char *)malloc((num_bits + 1)
            *sizeof(char));
        if (new_node_ptr == NULL) {
            printf("\n\ncreate_one_rule_ptr: ERROR : NOT ENOUGH
                MEMORY : new_node_ptr->message_ptr\n");
```

```
            fprintf(stderr,"\n\ncreate_one_rule_ptr: ERROR : NOT
                ENOUGH MEMORY : new_node_ptr->message_ptr\n");
            system("PAUSE");
            exit(1);
        }

        new_node_ptr->action_ptr = (char *)malloc(2*sizeof(char));
        if (new_node_ptr == NULL) {
            printf("\n\ncreate_one_rule_ptr: ERROR : NOT ENOUGH
                MEMORY : new_node_ptr->action_ptr\n");
            fprintf(stderr,"\n\ncreate_one_rule_ptr: ERROR : NOT
                ENOUGH MEMORY : new_node_ptr->action_ptr\n");
            system("PAUSE");
            exit(1);
        }
        new_node_ptr->strength_ptr = (double *)
            malloc(2*sizeof(double));
        if (new_node_ptr == NULL) {
            printf("\n\ncreate_one_rule_ptr: ERROR : NOT ENOUGH
                MEMORY : new_node_ptr->strength_ptr\n");
            fprintf(stderr,"\n\ncreate_one_rule_ptr: ERROR : NOT
                ENOUGH MEMORY : new_node_ptr->strength_ptr\n");
            system("PAUSE");
            exit(1);
        }
        return (new_node_ptr);
    }
    struct list *list_duplicate_ptr(struct list
    *list_to_be_duplicated_ptr) {
        struct list *duplicate_list_ptr;
        struct list_node *duplicate_node_ptr;
        struct list_node *current_node_ptr;
        struct list_node *tail_node_ptr;


        duplicate_list_ptr = list_init_ptr();
        tail_node_ptr = NULL;

        current_node_ptr = list_head_ptr(list_to_be_duplicated_ptr);
        duplicate_node_ptr = list_alloc_node_memory_ptr();
        list_copy_node (duplicate_node_ptr, current_node_ptr);
        list_insert_as_head(duplicate_list_ptr, duplicate_node_ptr);
        tail_node_ptr = duplicate_node_ptr;
        current_node_ptr = list_next_ptr(current_node_ptr);
        while(current_node_ptr != NULL) {
            duplicate_node_ptr = list_alloc_node_memory_ptr();
            list_copy_node (duplicate_node_ptr, current_node_ptr);
            list_insert_next(duplicate_list_ptr, tail_node_ptr,
                duplicate_node_ptr);
            tail_node_ptr = duplicate_node_ptr;
            current_node_ptr = list_next_ptr(current_node_ptr);
        }
        return(duplicate_list_ptr);
    }

    void list_copy_node (struct list_node *destination_node_ptr,
    struct list_node *source_node_ptr) {
```

```
     strcpy((destination_node_ptr->condition_ptr),
          (source_node_ptr->condition_ptr));
     strcpy((destination_node_ptr->message_ptr), (source_node_ptr
          ->message_ptr));
     strcpy((destination_node_ptr->action_ptr), (source_node_ptr->
          action_ptr));
     (*(destination_node_ptr->strength_ptr)) = (*(source_node_ptr
          ->strength_ptr));
     destination_node_ptr->next_ptr = NULL;
}

void list_print (struct list *list_to_be_printed_ptr) {
     struct list_node *current_node_ptr;
     printf("\n      Size = %d\n", list_to_be_printed_ptr->size);
     printf("        Strength of Environment = %f\n",
          *(list_to_be_printed_ptr->strength_of_environment_ptr));
     printf("      +------------+------------+--------+----------
          +\n");
     printf("      |  Condition |   Message  | Action | Strength
          |\n");
     printf("      +------------+------------+--------+----------
          +\n");
     current_node_ptr = list_head_ptr(list_to_be_printed_ptr);

     while (!(current_node_ptr==NULL)){
          list_print_node (current_node_ptr , 0);
          current_node_ptr=list_next_ptr(current_node_ptr);
     }
}

void list_print_node (struct list_node *node_to_be_printed_ptr,
char as_a_single_node) {
     if (as_a_single_node == 1){
          printf("      +------------+------------+--------+-------
               ---+\n");
          printf("      |  Condition |   Message  | Action |
               Strength |\n");
          printf("      +------------+------------+--------+-------
               ---+\n");
     }
     printf("      | %s | %s |    %s   | %f |\n",
          node_to_be_printed_ptr->condition_ptr,
          node_to_be_printed_ptr->message_ptr,
          node_to_be_printed_ptr->action_ptr,
          *(node_to_be_printed_ptr->strength_ptr));
     printf("      +------------+------------+--------+----------
          +\n");
}

void list_insert(struct list *list_to_be_inserted_to_ptr, struct
list_node *node_to_be_inserted_ptr) {
     struct list_node *node_to_be_inserted_after_ptr;
     char *same_exists_ptr;

     same_exists_ptr = (char *)malloc(sizeof(char));
     *same_exists_ptr = 0;
```

```
     node_to_be_inserted_after_ptr =
          list_locate_ptr(list_to_be_inserted_to_ptr,
          node_to_be_inserted_ptr, same_exists_ptr);
     if (*same_exists_ptr != 1) {
          if (node_to_be_inserted_after_ptr == NULL) {
               list_insert_as_head(list_to_be_inserted_to_ptr,
                    node_to_be_inserted_ptr);
          }
          else {
               list_insert_next(list_to_be_inserted_to_ptr,
                    node_to_be_inserted_after_ptr,
                    node_to_be_inserted_ptr);
          }
     }
}
void list_insert_as_head(struct list *list_to_be_inserted_to_ptr,
struct list_node *node_to_be_inserted_ptr) {
     node_to_be_inserted_ptr->next_ptr=list_to_be_inserted_to_ptr-
>head_ptr;
     list_to_be_inserted_to_ptr->head_ptr=node_to_be_inserted_ptr;
     (list_to_be_inserted_to_ptr->size)++;
}

void list_insert_next(struct list *list_to_be_inserted_to_ptr,
     struct list_node *node_to_be_inserted_after_ptr, struct
     list_node *node_to_be_inserted_ptr){
     node_to_be_inserted_ptr-> next_ptr =
          list_next_ptr(node_to_be_inserted_after_ptr);
     node_to_be_inserted_after_ptr->
          next_ptr=node_to_be_inserted_ptr;
     (list_to_be_inserted_to_ptr->size)++;
}

struct list_node *list_locate_ptr (struct list
*list_to_be_located_in_ptr, struct list_node
*node_to_be_located_ptr, char *same_exists_ptr) {
     struct list_node *current_node_ptr;
     struct list_node *node_at_back_ptr;

     *same_exists_ptr = 0;

     node_at_back_ptr = NULL;

     if(!((list_to_be_located_in_ptr->head_ptr)==NULL)){
          current_node_ptr = list_to_be_located_in_ptr->head_ptr;

          while((list_compare_two_nodes(node_to_be_located_ptr,
          current_node_ptr) > 0)) {
               node_at_back_ptr=current_node_ptr;
               current_node_ptr=list_next_ptr(current_node_ptr);
               if (current_node_ptr==NULL) {
                    break;
               }
          }
               if((current_node_ptr != NULL) &&
                    (list_compare_two_nodes(node_to_be_located_ptr,
                    current_node_ptr) == 0)) {
```

```
                *same_exists_ptr = 1;
            }
        }
        return(node_at_back_ptr);
}

int list_compare_two_nodes (struct list_node
*first_list_node_to_compare_ptr, struct list_node
*second_list_node_to_compare_ptr) {
        int overall_comparison_result, condition_comparison_result,
        message_comparison_result, action_comparison_result;

        condition_comparison_result =
            strcmp(first_list_node_to_compare_ptr->
            condition_ptr,second_list_node_to_compare_ptr-
            >condition_ptr);
        if (condition_comparison_result == 0){
            message_comparison_result =
                strcmp(first_list_node_to_compare_ptr->
                message_ptr,second_list_node_to_compare_ptr->
                message_ptr);
            if(message_comparison_result == 0 ){
                action_comparison_result =
                    strcmp(first_list_node_to_compare_ptr->
                    message_ptr, second_list_node_to_compare_ptr->
                    message_ptr);
                if (action_comparison_result == 0){
                    overall_comparison_result = 0;
                }
                else
                    overall_comparison_result =
                        action_comparison_result;
            }
            else {
                overall_comparison_result =
                    message_comparison_result;
            }
        }
        else{
            overall_comparison_result = condition_comparison_result;
        }
return(overall_comparison_result);
}


void list_destroy_list (struct list *list_to_be_destroyed_ptr) {
        while ((list_to_be_destroyed_ptr->head_ptr) != NULL) {
            list_remove_head (list_to_be_destroyed_ptr);
        }

        free(list_to_be_destroyed_ptr->strength_of_environment_ptr);
        free(list_to_be_destroyed_ptr);
}

void list_remove_head (struct list *list_to_be_removed_from_ptr) {
        struct list_node *node_to_be_removed_ptr;
```

```
     node_to_be_removed_ptr =
          list_head_ptr(list_to_be_removed_from_ptr);
     if(node_to_be_removed_ptr == NULL) {
          printf("There is something wrong here
               list_remove_head\n");
     }
     list_to_be_removed_from_ptr->head_ptr =
          list_next_ptr(node_to_be_removed_ptr);
     list_destroy_node(node_to_be_removed_ptr);
     (list_to_be_removed_from_ptr->size)--;
}

void list_remove_next (struct list *list_to_be_removed_from_ptr,
struct list_node *previous_node_ptr) {
     struct list_node *node_to_be_removed_ptr;

     node_to_be_removed_ptr = list_next_ptr(previous_node_ptr);
     if(node_to_be_removed_ptr == NULL) {
         printf("There is something wrong here!
               list_remove_next\n");
     }
     previous_node_ptr->next_ptr =
          list_next_ptr(node_to_be_removed_ptr);

     list_destroy_node(node_to_be_removed_ptr);
     (list_to_be_removed_from_ptr->size)--;
}

struct list_node *list_get_node_by_index_ptr (struct list
*list_to_get_from_ptr, int node_index) {
     struct list_node *current_node_ptr;
     int index;

     current_node_ptr = NULL;
     index = 0;

     current_node_ptr = list_head_ptr(list_to_get_from_ptr);

     for(index = 0; index < node_index; index++) {
          current_node_ptr = list_next_ptr(current_node_ptr);
     }

     return(current_node_ptr);
}

void list_destroy_node (struct list_node
*node_to_be_destroyed_ptr) {

     free(node_to_be_destroyed_ptr->condition_ptr);
     free(node_to_be_destroyed_ptr->message_ptr);
     free(node_to_be_destroyed_ptr->action_ptr);
     free(node_to_be_destroyed_ptr->strength_ptr);

     free(node_to_be_destroyed_ptr);
}

struct secondary_list_node *create_secondary_list_node_ptr(struct
```

```
list_node *node_to_be_hosted_ptr) {
     struct secondary_list_node *new_secondary_list_node_ptr;

     new_secondary_list_node_ptr = malloc(sizeof(struct
          secondary_list_node));

     new_secondary_list_node_ptr->hosted_node_ptr =
          node_to_be_hosted_ptr;
     new_secondary_list_node_ptr->next_ptr = NULL;

     return(new_secondary_list_node_ptr);
}

struct secondary_list *secondary_list_init_ptr (){

     struct secondary_list *secondary_list_to_be_init_ptr;

     secondary_list_to_be_init_ptr = malloc(sizeof(struct
          secondary_list));
     if (secondary_list_to_be_init_ptr==NULL) {
          printf("\n\nlist_init_ptr: ERROR : NOT ENOUGH MEMORY :
               list_to_be_init_ptr\n");
          fprintf(stderr,"\n\nlist_init_ptr: ERROR : NOT ENOUGH
               MEMORY : list_to_be_init_ptr\n");
          system("PAUSE");
          exit(1);
     }

     secondary_list_to_be_init_ptr->head_ptr=NULL;

     return(secondary_list_to_be_init_ptr);
}

void secondary_list_insert_as_head(struct secondary_list
*secondary_list_to_be_inserted_to_ptr, struct secondary_list_node
*secondary_list_node_to_be_inserted_ptr) {
     secondary_list_node_to_be_inserted_ptr->
          next_ptr=secondary_list_to_be_inserted_to_ptr->head_ptr;
     secondary_list_to_be_inserted_to_ptr->
          head_ptr=secondary_list_node_to_be_inserted_ptr;
}

void secondary_list_destroy_list (struct secondary_list
*secondary_list_to_be_destroyed_ptr) {
     while ((secondary_list_to_be_destroyed_ptr->head_ptr) !=
     NULL) {
          secondary_list_remove_head
               (secondary_list_to_be_destroyed_ptr);
     }
     free(secondary_list_to_be_destroyed_ptr);
}

void secondary_list_remove_head (struct secondary_list
*secondary_list_to_be_removed_from_ptr) {
     struct secondary_list_node
          *secondary_list_node_to_be_removed_ptr;
```

```
     secondary_list_node_to_be_removed_ptr =
          list_head_ptr(secondary_list_to_be_removed_from_ptr);
     if(secondary_list_node_to_be_removed_ptr == NULL) {
          printf("there is something worng here! 65381741\n");
     }
     secondary_list_to_be_removed_from_ptr->head_ptr =
          list_next_ptr(secondary_list_node_to_be_removed_ptr);
     secondary_list_destroy_node
          (secondary_list_node_to_be_removed_ptr);
}

void secondary_list_destroy_node (struct secondary_list_node
*secondary_list_node_to_be_destroyed_ptr) {
     free(secondary_list_node_to_be_destroyed_ptr);
}

char message_match_condition (char *message, char *condition) {
     char match;
     int bit_counter;

     match = 1;

     if ((strlen(message) == (num_bits)) && (strlen(condition) ==
     (num_bits))){
          for(bit_counter = 0; bit_counter < num_bits;
          bit_counter++) {
               if((chars_match(*(message + bit_counter),*(condition
               + bit_counter))) == 0 ){
                    match = 0;
               }
          }
     }
     else {
          printf("\nLTM  : state_match_condition : state = %s\n",
          message);
          printf("\nLTM  : state_match_condition : condition =
          %s\n", condition);
          printf("There is something wrong here 5486721");
          system("pause");
     }
     return(match);
}

char chars_match (char state_char, char condition_char){
     char match;

     match = 0;

     if ((state_char == condition_char) || (condition_char ==
     MATCH_ANY_CHAR)) {
          match=1;
     }
     return(match);
}
void mutate_rule(struct list_node *rule_to_be_mutated_ptr) {
     char random_trunc;
```

```
     random_trunc = ((double)(2 * num_bits)) * lcgrand(0);

     if (random_trunc < num_bits) {
          mutate_bit3(((rule_to_be_mutated_ptr->condition_ptr) +
          random_trunc));
     }
     else if (random_trunc < (2 * num_bits)) {
          mutate_bit2(((rule_to_be_mutated_ptr->message_ptr) +
          random_trunc - num_bits));
     }
     else {
          mutate_bit2(rule_to_be_mutated_ptr->action_ptr);
     }
}

void mutate_bit2(char *bit_to_be_mutated_ptr) {
     if (*bit_to_be_mutated_ptr == ZERO_CHAR){
          *bit_to_be_mutated_ptr = ONE_CHAR;
     }
     else {
          *bit_to_be_mutated_ptr = ZERO_CHAR;
     }
}

void mutate_bit3(char *bit_to_be_mutated_ptr) {
     float random_number;

     random_number = lcgrand(0);
     if (*bit_to_be_mutated_ptr == ZERO_CHAR){
          switch ((random_number <= 0.5) + (random_number <= 1)) {
               case 2 : {
                    *bit_to_be_mutated_ptr = ONE_CHAR;
                    break;
               }
               case 1  : {
                    *bit_to_be_mutated_ptr = MATCH_ANY_CHAR;
                    break;
               }
               default : {
                    printf("There is something Wrong Here
                    mutate_bit3 1");
               }
          }
     }
     else if (*bit_to_be_mutated_ptr == ONE_CHAR){
          switch ((random_number <= 0.5) + (random_number <= 1)) {
               case 2 : {
                    *bit_to_be_mutated_ptr = ZERO_CHAR;
                    break;
               }
               case 1  : {
                    *bit_to_be_mutated_ptr = MATCH_ANY_CHAR;
                    break;
               }
               default : {
                    printf("There is something wrong here!
                    mutate_bit3 2");
```

```
                }
            }
        }
        else if (*bit_to_be_mutated_ptr == MATCH_ANY_CHAR){
            switch ((random_number <= 0.5) + (random_number <= 1)) {
                case 2 : {
                    *bit_to_be_mutated_ptr = ZERO_CHAR;
                    break;
                }
                case 1  : {
                    *bit_to_be_mutated_ptr = ONE_CHAR;
                    break;
                }
                default : {
                    printf("There is something wrong here!
                    mutate_bit3 3");
                }
            }
        }
}

char rand_char2() {
        float random_number;
        char new_rand_char;

        new_rand_char=0;
        random_number = lcgrand(0);

        switch ((random_number<=RAND_RATIO_20)+(random_number <=
        RAND_RATIO_20 + RAND_RATIO_21)) {
            case 2 : {
                new_rand_char= ZERO_CHAR;
                break;
            }
            case 1  : {
                new_rand_char= ONE_CHAR;
                break;
            }
            default : {
                printf("there is something worng here!");
            }
        }
        return(new_rand_char);
}

char rand_char3() {
        float random_number;
        char new_rand_char;

        new_rand_char=0;
        random_number = lcgrand(0);

        switch ((random_number <= RAND_RATIO_30) +
        (random_number<=RAND_RATIO_30 + RAND_RATIO_31)+(random_number
        <= RAND_RATIO_30 + RAND_RATIO_31 + RAND_RATIO_32)) {
            case 3 : {
                new_rand_char = ZERO_CHAR;
```

```
                break;
            }
            case 2 : {
                new_rand_char = ONE_CHAR;
                break;
            }
            case 1  : {
                new_rand_char= MATCH_ANY_CHAR;
                break;
            }
            default : {
                printf("there is something worng here!");
            }
        }
        return(new_rand_char);
}

float lcgrand(int stream) {
    long zi, lowprd, hi31;

    zi     = zrng[stream];
    lowprd = (zi & 65535) * MULT1;
    hi31   = (zi >> 16) * MULT1 + (lowprd >> 16);
    zi     = ((lowprd & 65535) - MODLUS) +
             ((hi31 & 32767) << 16) + (hi31 >> 15);
    if (zi < 0) zi += MODLUS;
    lowprd = (zi & 65535) * MULT2;
    hi31   = (zi >> 16) * MULT2 + (lowprd >> 16);
    zi     = ((lowprd & 65535) - MODLUS) +
             ((hi31 & 32767) << 16) + (hi31 >> 15);
    if (zi < 0) zi += MODLUS;
    zrng[stream] = zi;
    return (zi >> 7 | 1) / 16777216.0;
}

void lcgrandst (long zset, int stream) { /* Set the current zrng
for stream "stream" to zset. */
    zrng[stream] = zset;
}

long lcgrandgt (int stream) {/* Return the current zrng for stream
"stream". */
    return zrng[stream];
}
```

APPENDIX F

Code for Intrusion Detector B

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define MATCH_ANY_CHAR 'x'

#define list_head_ptr(list) ((list)->head_ptr)
#define list_next_ptr(list) ((list)->next_ptr)

struct list_node {
     char *condition_ptr;
     struct list_node *next_ptr;
};

struct list {
     struct list_node *head_ptr;
     int size;
};

void train(struct list *rule_set_ptr);
void process_one_training_step(struct list *rule_set_ptr, char
*input_message_ptr);
void compact(struct list *rule_set_ptr);
char search_for_match(struct list *rule_set_ptr, char
*input_message_ptr);
char message_match_condition (char *message, char *condition);
char combinable(char *condition_1_ptr, char *condition_2_ptr);
struct list_node *combine_two_rules(struct list_node *rule_1_ptr,
struct list_node *rule_2_ptr);
char chars_match (char message_char, char condition_char);
void reduce (struct list *rule_set_ptr);
char count_match_any (struct list_node *rule_ptr);
void test(struct list *rule_set_ptr);
char list_contained (struct list *rule_set_ptr, struct list_node
*rule_ptr);
char list_rule_containes (struct list_node *current_rule_ptr,
struct list_node *rule_ptr);
struct list *list_init_ptr ();
struct list_node *list_alloc_node_memory_ptr();
struct list_node *list_create_node_ptr(char *condition_ptr);
struct list_node *list_locate_ptr (struct list
*list_to_be_located_in_ptr, struct list_node
*node_to_be_located_ptr);
void list_insert(struct list *list_to_be_inserted_to_ptr, struct
list_node *node_to_be_inserted_ptr);
void list_insert_as_head(struct list *list_to_be_inserted_to_ptr,
struct list_node *node_to_be_inserted_ptr);
void list_insert_next(struct list *list_to_be_inserted_to_ptr,
struct list_node *node_to_be_inserted_after_ptr, struct list_node
```

```
*node_to_be_inserted_ptr);
void list_print (struct list *list_to_be_printed_ptr);
void list_print_node (struct list_node *node_to_be_printed_ptr,
char as_a_single_node);
void list_remove_head (struct list *list_to_be_removed_from_ptr);
void list_remove_next (struct list *list_to_be_removed_from_ptr,
struct list_node *previous_node_ptr);
void list_nd_remove_head (struct list
*list_to_be_removed_from_ptr);
void list_nd_remove_next (struct list
*list_to_be_removed_from_ptr, struct list_node
*previous_node_ptr);
void list_destroy_node (struct list_node
*node_to_be_destroyed_ptr);


FILE *out_file_ptr;
int num_bits;

int main() {
     FILE *param_file_ptr;
     struct list *rule_set_ptr;

     /* Open the Parameterfile File */
     param_file_ptr = fopen("idb.prm", "r");
          if(param_file_ptr == NULL){
          printf("\n\nERROR: main : NO Parameter FILE PRESENT
          !!!\n\n");
          printf("\n\nSO QUITTING!!!\n\n");
          system("PAUSE");
          exit(1);
     }
     /* Read the Parameter */
     fscanf(param_file_ptr, "%d", &num_bits);
     /* Close the Parameter File */
     fclose(param_file_ptr);
     /* Open (create if necessary) The Output File */
     out_file_ptr = fopen("idb.out","a+");
     /* Make sure that it is opened */
     if(out_file_ptr == NULL){
          printf("\n\nERROR: main : OUTPUT FILE CAN NOT BE OPENED
          CREATED!!!\n\n");
          printf("\n\nSO QUITTING!!!\n\n");
          system("PAUSE");
          exit(1);
     }

     rule_set_ptr = list_init_ptr();

     train(rule_set_ptr);

     reduce(rule_set_ptr);

     test(rule_set_ptr);


     return(0);
```

```
}
void train(struct list *rule_set_ptr){
     FILE *training_file_ptr;
     char *input_message_ptr;
     double training_start_time;
     double training_end_time;
     double training_duration;

     /* Open The Training File */
     training_file_ptr = fopen("id.trn","r");
     /* Make sure that it is opened */
     if(training_file_ptr == NULL){
          printf("\n\nERROR: main : NO TRAINING FILE PRESENT
          !!!\n\n");
          printf("\n\nSO QUITTING!!!\n\n");
          system("PAUSE");
          exit(1);
     }
     input_message_ptr = (char *)malloc((num_bits + 1 )*
     sizeof(char));

     training_start_time = clock();
     while(!feof(training_file_ptr)) {
          fscanf(training_file_ptr, "%s", input_message_ptr);
          process_one_training_step(rule_set_ptr,
          input_message_ptr);
     }
     training_end_time = clock();
          training_duration = (training_end_time -
          training_start_time)/CLOCKS_PER_SEC;
     fprintf(out_file_ptr,"%7.3f,", training_duration);
     list_print(rule_set_ptr);
}

void process_one_training_step(struct list *rule_set_ptr, char
*input_message_ptr){
     struct list_node *new_rule_ptr;
     char match_exists;

     new_rule_ptr = NULL;
     match_exists = 0;

     new_rule_ptr = list_create_node_ptr(input_message_ptr);
     list_insert(rule_set_ptr, new_rule_ptr);
     compact(rule_set_ptr);

}
void compact(struct list *rule_set_ptr) {
     struct list_node *current_rule_1_ptr;
     struct list_node *current_rule_2_ptr;
     struct list_node *rule_at_back_1_ptr;
     struct list_node *rule_at_back_2_ptr;
     struct list_node *combined_rule_ptr;
     char combine;
     char combined;

     do {
```

```
          current_rule_1_ptr = NULL;
          current_rule_2_ptr = NULL;
          rule_at_back_1_ptr = NULL;
          rule_at_back_2_ptr = NULL;
          combine = 0;
          combined = 0;
          current_rule_1_ptr = list_head_ptr(rule_set_ptr);
          while (current_rule_1_ptr != NULL) {
               combine = 0;
               rule_at_back_2_ptr = current_rule_1_ptr;
               current_rule_2_ptr =
               list_next_ptr(current_rule_1_ptr);
               while(current_rule_2_ptr != NULL){
                    combine = 0;
                    combine = combinable(current_rule_1_ptr->
                    condition_ptr, current_rule_2_ptr->
                    condition_ptr);
                    if (combine == 1){
                         combined = 1;
                         combined_rule_ptr =
                         combine_two_rules(current_rule_1_ptr,
                         current_rule_2_ptr);
                         if(rule_at_back_2_ptr == NULL) {
                              list_remove_head(rule_set_ptr);
                         }
                         else {
                              list_remove_next(rule_set_ptr,
                              rule_at_back_2_ptr);
                         }
                         if (rule_at_back_1_ptr == NULL) {
                              list_remove_head(rule_set_ptr);
                         }
                         else {
                              list_remove_next(rule_set_ptr,
                              rule_at_back_1_ptr);
                         }
                         list_insert(rule_set_ptr,
                         combined_rule_ptr);
                         break;
                    } else {
                         rule_at_back_2_ptr = current_rule_2_ptr;
                         current_rule_2_ptr =
                         list_next_ptr(current_rule_2_ptr);
                    }
               }
               if (combine == 1) {
                    break;
               }
               else {
                    rule_at_back_1_ptr = current_rule_1_ptr;
                    current_rule_1_ptr =
                    list_next_ptr(current_rule_1_ptr);
               }
          }
     }while (combined == 1);
}
```

```
char search_for_match(struct list *rule_set_ptr, char
*input_message_ptr) {
     struct list_node *current_rule_ptr;
     char match_exists;

     current_rule_ptr = list_head_ptr(rule_set_ptr);
     match_exists = 0;  /* does not exist */

     while(current_rule_ptr != NULL){
          if (message_match_condition(input_message_ptr,
          current_rule_ptr->condition_ptr) == 1){
               match_exists = 1;
               break;
          }
          current_rule_ptr = list_next_ptr(current_rule_ptr);
     }
     return(match_exists);
}


char message_match_condition (char *message, char *condition) {
     char match;
     int bit_counter;

     match = 1; /* it matches */

     if ((strlen(message) == (num_bits)) && (strlen(condition) ==
     (num_bits))){
          for(bit_counter = 0; bit_counter < num_bits;
          bit_counter++) {
               if((chars_match(*(message + bit_counter),*(condition
               + bit_counter))) == 0 ){
                    match = 0;
               }
          }
     }
     else {
          printf("\nLTM  : state_match_condition : state = %s\n",
          message);
          printf("\nLTM  : state_match_condition : condition =
          %s\n", condition);
          printf("There is something wrong here 5486721");
          system("pause");
     }
     return(match);
}
char combinable(char *condition_1_ptr, char *condition_2_ptr) {
     char diff_chars;
     char char_index;

     diff_chars = 0;
     char_index = 0;

     for (char_index = 0; char_index < num_bits; char_index++) {
          if((*(condition_1_ptr + char_index)) !=
          (*(condition_2_ptr + char_index))) {
```

```
                diff_chars++;
            }
            if(diff_chars == 2) {
                break;
            }
        }
        return(diff_chars);
}

struct list_node *combine_two_rules(struct list_node *rule_1_ptr,
struct list_node *rule_2_ptr) {
        struct list_node *new_rule_ptr;
        int index;

        new_rule_ptr = list_alloc_node_memory_ptr();

        index = 0;
        while((*((rule_1_ptr->condition_ptr)+index)) ==
        (*((rule_2_ptr->condition_ptr)+index))) {
            (*((new_rule_ptr->condition_ptr)+index)) =
            (*((rule_1_ptr->condition_ptr)+index));
            index++;
        }
        (*((new_rule_ptr->condition_ptr)+index)) = 'x';
        index++;
        while((*((rule_1_ptr->condition_ptr)+index)) != NULL) {
            (*((new_rule_ptr->condition_ptr)+index)) =
(*((rule_1_ptr->condition_ptr)+index));
            index++;
        }
        (*((new_rule_ptr->condition_ptr)+index)) = NULL;
        return(new_rule_ptr);
}
char chars_match (char message_char, char condition_char){
        char match;

        match = 0;

        if ((message_char == condition_char) || (condition_char ==
        MATCH_ANY_CHAR)) {
            match=1;
        }
        return(match);
}

void reduce (struct list *rule_set_ptr) {
        char num_match_any;
        char match_any_count;
        char contained;
        struct list_node *current_rule_ptr;
        struct list_node *previous_rule_ptr;
        struct list_node *next_rule_ptr;

        num_match_any = 0;
        match_any_count = 0;
        contained = 0;
        current_rule_ptr = NULL;
```

```
     previous_rule_ptr = NULL;
     next_rule_ptr = NULL;

     for(num_match_any = 0; num_match_any < num_bits;
     num_match_any++) {
          previous_rule_ptr = NULL;
          current_rule_ptr = list_head_ptr(rule_set_ptr);
          next_rule_ptr = list_next_ptr(current_rule_ptr);
          while (next_rule_ptr != NULL) {
               match_any_count = count_match_any(current_rule_ptr);
               if (match_any_count == num_match_any){
                    if (previous_rule_ptr == NULL) {
                         list_nd_remove_head(rule_set_ptr);
                    }
                    else {
                         list_nd_remove_next(rule_set_ptr,
                         previous_rule_ptr);
                    }
                    contained = list_contained (rule_set_ptr,
                    current_rule_ptr);
                    if (contained == 1) {
                         list_destroy_node(current_rule_ptr);
                    }
                    else {
                         list_insert(rule_set_ptr,
                         current_rule_ptr);
                         previous_rule_ptr = current_rule_ptr;
                    }
                    current_rule_ptr = next_rule_ptr;
                    next_rule_ptr = list_next_ptr(next_rule_ptr);
               }
               else {
                    previous_rule_ptr = current_rule_ptr;
                    current_rule_ptr = next_rule_ptr;
                    next_rule_ptr = list_next_ptr(next_rule_ptr);
               }
          }
     }
}

char count_match_any (struct list_node *rule_ptr) {
     char match_any_count;
     char index;

     match_any_count = 0;

     for (index = 0; index < num_bits; index++) {
          if ((*((rule_ptr->condition_ptr)+index)) ==
          MATCH_ANY_CHAR) {
               match_any_count++;
          }
     }

     return(match_any_count);
}

void test(struct list *rule_set_ptr){
```

```
FILE *test_file_ptr;
char *input_message_ptr;
char *correct_action_ptr;
char normal_action;

int number_of_true_positive;
int number_of_true_negative;
int number_of_false_positive;
int number_of_false_negative;

double test_start_time;
double test_end_time;
double test_duration;

number_of_true_positive = 0;
number_of_true_negative = 0;
number_of_false_positive = 0;
number_of_false_negative = 0;

test_file_ptr = fopen("id.tst","r");

input_message_ptr = (char *)malloc((num_bits + 1) *
sizeof(char));
correct_action_ptr = (char *)malloc(2*sizeof(char));

test_start_time = clock();

while(!feof(test_file_ptr)){
     fscanf(test_file_ptr, "%s %s", input_message_ptr,
     correct_action_ptr);
     normal_action = search_for_match(rule_set_ptr,
     input_message_ptr);

     if((normal_action == 1) && (*correct_action_ptr == '0')){
          number_of_true_positive++;
     }
     else if ((normal_action == 1 ) && (*correct_action_ptr ==
     '1')){
          number_of_false_positive++;
     }
     else if ((normal_action == 0 ) && (*correct_action_ptr ==
     '1')){
          number_of_true_negative++;
     }
     else if ((normal_action == 0 ) && (*correct_action_ptr ==
     '0')){
          number_of_false_negative++;
     }
     else {
          printf("there is something worng here !");
          system("PAUSE");
     }
}
test_end_time = clock();
test_duration = (test_end_time -
test_start_time)/CLOCKS_PER_SEC;
fprintf(out_file_ptr,"%7.3f,", test_duration);
```

```
          fprintf(out_file_ptr,"%7d,%7d,%7d,%7d\n",
          number_of_true_positive, number_of_false_negative,
          number_of_true_negative, number_of_false_positive);
}

char list_contained (struct list *rule_set_ptr, struct list_node
*rule_ptr) {
     struct list_node *current_rule_ptr;
     char contained;

     current_rule_ptr = NULL;
     contained = 0;

     current_rule_ptr = list_head_ptr(rule_set_ptr);
     while(current_rule_ptr != NULL) {
          contained = 0;
          if (list_rule_containes(current_rule_ptr, rule_ptr) == 1)
          {
               contained = 1;
               break;
          }
          current_rule_ptr = list_next_ptr(current_rule_ptr);
     }
     return(contained);
}

char list_rule_containes (struct list_node *current_rule_ptr,
struct list_node *rule_ptr) {
     char containes;
     char index;

     index = 0;
     containes = 1;

     for (index = 0; index < num_bits; index++) {
          if (((*((current_rule_ptr->condition_ptr) + index)) !=
          MATCH_ANY_CHAR) && ((*((current_rule_ptr->condition_ptr)
          + index)) != (*((rule_ptr->condition_ptr) + index)))) {
               containes = 0;
          }
     }
     return(containes);
}

struct list *list_init_ptr (){

     struct list *list_to_be_init_ptr;


     list_to_be_init_ptr = malloc(sizeof(struct list));
     if (list_to_be_init_ptr==NULL) {
          printf("\n\nlist_init_ptr: ERROR : NOT ENOUGH MEMORY :
          list_to_be_init_ptr\n");
          fprintf(stderr,"\n\nlist_init_ptr: ERROR : NOT ENOUGH
          MEMORY : list_to_be_init_ptr\n");
          system("PAUSE");
          exit(1);
```

```
     }

     list_to_be_init_ptr->head_ptr = NULL;
     list_to_be_init_ptr->size = 0;

     return(list_to_be_init_ptr);
}
struct list_node *list_alloc_node_memory_ptr(){
     struct list_node *new_node_ptr;

     new_node_ptr = NULL;

     new_node_ptr = malloc(sizeof(struct list_node));
     if (new_node_ptr == NULL) {
          printf("\n\ncreate_one_rule_ptr: ERROR : NOT ENOUGH
          MEMORY : new_node_ptr\n");
          fprintf(stderr,"\n\ncreate_one_rule_ptr: ERROR : NOT
          ENOUGH MEMORY : new_node_ptr\n");
          system("PAUSE");
          exit(1);
     }

     new_node_ptr->condition_ptr = (char
     *)malloc((num_bits+1)*sizeof(char));
     if (new_node_ptr == NULL) {
          printf("\n\ncreate_one_rule_ptr: ERROR : NOT ENOUGH
          MEMORY : new_node_ptr->condition_ptr\n");
          fprintf(stderr,"\n\ncreate_one_rule_ptr: ERROR : NOT
          ENOUGH MEMORY : new_node_ptr->condition_ptr\n");
          system("PAUSE");
          exit(1);
     }

     new_node_ptr->next_ptr = NULL;

     return (new_node_ptr);
}

struct list_node *list_create_node_ptr(char *condition_ptr) {
     struct list_node *new_node_ptr;

     new_node_ptr = list_alloc_node_memory_ptr();
     strcpy(new_node_ptr->condition_ptr, condition_ptr);
     new_node_ptr->next_ptr = NULL;

     return(new_node_ptr);
}
struct list_node *list_locate_ptr (struct list
*list_to_be_located_in_ptr, struct list_node
*node_to_be_located_ptr) {
     struct list_node *current_node_ptr;
     struct list_node *node_at_back_ptr;

     current_node_ptr = NULL;
     node_at_back_ptr = NULL;

     if(!((list_to_be_located_in_ptr->head_ptr)==NULL)){
```

```
          current_node_ptr = list_to_be_located_in_ptr->head_ptr;

          while(strcmp(node_to_be_located_ptr->condition_ptr,
          current_node_ptr->condition_ptr) > 0) {
               node_at_back_ptr=current_node_ptr;
               current_node_ptr=list_next_ptr(current_node_ptr);
               if (current_node_ptr==NULL) {
                    break;
               }
          }
     }

     return(node_at_back_ptr);
}

void list_insert(struct list *list_to_be_inserted_to_ptr, struct
list_node *node_to_be_inserted_ptr) {
     struct list_node *node_to_be_inserted_after_ptr;


     node_to_be_inserted_after_ptr =
     list_locate_ptr(list_to_be_inserted_to_ptr,
     node_to_be_inserted_ptr);
     if (node_to_be_inserted_after_ptr == NULL) {
          list_insert_as_head(list_to_be_inserted_to_ptr,
     node_to_be_inserted_ptr);
     }
     else {
          list_insert_next(list_to_be_inserted_to_ptr,
          node_to_be_inserted_after_ptr, node_to_be_inserted_ptr);
     }
}

void list_insert_as_head(struct list *list_to_be_inserted_to_ptr,
struct list_node *node_to_be_inserted_ptr) {
     node_to_be_inserted_ptr->next_ptr=list_to_be_inserted_to_ptr
     ->head_ptr;
     list_to_be_inserted_to_ptr->head_ptr=node_to_be_inserted_ptr;
     (list_to_be_inserted_to_ptr->size)++;
}

void list_insert_next(struct list *list_to_be_inserted_to_ptr,
struct list_node *node_to_be_inserted_after_ptr, struct list_node
*node_to_be_inserted_ptr){
     node_to_be_inserted_ptr->
     next_ptr=list_next_ptr(node_to_be_inserted_after_ptr);
     node_to_be_inserted_after_ptr->
     next_ptr=node_to_be_inserted_ptr;
     (list_to_be_inserted_to_ptr->size)++;
}
void list_remove_head (struct list *list_to_be_removed_from_ptr) {
     struct list_node *node_to_be_removed_ptr;

     node_to_be_removed_ptr =
     list_head_ptr(list_to_be_removed_from_ptr);
     if(node_to_be_removed_ptr == NULL) {
          printf("there is something worng here! 65281741\n");
```

```
     }
     list_to_be_removed_from_ptr->head_ptr =
     list_next_ptr(node_to_be_removed_ptr);
     list_destroy_node(node_to_be_removed_ptr);
     (list_to_be_removed_from_ptr->size)--;
}
void list_print (struct list *list_to_be_printed_ptr) {
     struct list_node *current_node_ptr;

     #ifdef DEBUG2
          printf("\n\n                                list_print :
          Printing\n");
     #endif /* DEBUG2 */

     printf("\n      Size = %d\n", list_to_be_printed_ptr->size);
     system("pause");
     current_node_ptr = list_head_ptr(list_to_be_printed_ptr);

     while (!(current_node_ptr==NULL)){
          list_print_node (current_node_ptr , 0);
          current_node_ptr=list_next_ptr(current_node_ptr);
     }
}

void list_print_node (struct list_node *node_to_be_printed_ptr,
char as_a_single_node) {
     if (as_a_single_node == 1){
     printf("      +------------+\n");
     }

     if (node_to_be_printed_ptr == NULL) {
          printf("      |    NULL    | \n");
          printf("      +------------+\n");
     }
     else {
          printf("      | %s | \n", node_to_be_printed_ptr->
          condition_ptr);
          printf("      +------------+\n");
     }
}


void list_remove_next (struct list *list_to_be_removed_from_ptr,
struct list_node *previous_node_ptr) {
     struct list_node *node_to_be_removed_ptr;
     node_to_be_removed_ptr = list_next_ptr(previous_node_ptr);
     if(node_to_be_removed_ptr == NULL) {
          printf("there is something worng here! 65421741\n");
          system("pause");
     }
     previous_node_ptr->next_ptr =
     list_next_ptr(node_to_be_removed_ptr);

     list_destroy_node(node_to_be_removed_ptr);
     (list_to_be_removed_from_ptr->size)--;
}
```

```
void list_nd_remove_head (struct list
*list_to_be_removed_from_ptr) {
     struct list_node *node_to_be_removed_ptr;

     node_to_be_removed_ptr =
     list_head_ptr(list_to_be_removed_from_ptr);
     if(node_to_be_removed_ptr == NULL) {
          printf("there is something worng here! 65381741\n");
     }
     list_to_be_removed_from_ptr->head_ptr =
     list_next_ptr(node_to_be_removed_ptr);
     node_to_be_removed_ptr->next_ptr = NULL;
     (list_to_be_removed_from_ptr->size)--;
}

void list_nd_remove_next (struct list
*list_to_be_removed_from_ptr, struct list_node *previous_node_ptr)
{
     struct list_node *node_to_be_removed_ptr;
     node_to_be_removed_ptr = list_next_ptr(previous_node_ptr);
     if(node_to_be_removed_ptr == NULL) {
          printf("there is something worng here! 65521741\n");
          system("pause");
     }
     previous_node_ptr->next_ptr =
     list_next_ptr(node_to_be_removed_ptr);
     node_to_be_removed_ptr->next_ptr = NULL;

     (list_to_be_removed_from_ptr->size)--;
}
void list_destroy_node (struct list_node
*node_to_be_destroyed_ptr) {

     free(node_to_be_destroyed_ptr->condition_ptr);

     free(node_to_be_destroyed_ptr);
}
```