UNIVERSITY OF BAHRAIN

College of Information Technology

جامعة البحرين
LIBRARY
University of Bahrain

Automated Design of Stream Cipher Using GADS and Simulated Annealing

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Master Degree in Information Technology

Submitted by

Amal Mohammed Al Hiddi

University number: 19991154

Supervised by

Wasan Shaker Awad

(Associate professor)

University of Bahrain

Kingdom of Bahrain

December, 2014

جامعة البحرين
LIBRARY
University of Bahrain

DECLARATION

I declare that this work is the result of my own investigation and that it has not already been accepted in substance for any degree, nor is it currently submitted for any degree.

Signed: ---------- Date: 22/12/2014

Amal Mohammed Al Hiddi (Candidate)

The Thesis defense committee considers the Thesis titled:

AUTOMATED DESIGN OF STREAM CIPHER USING GADS AND SIMULATED ANNEALING

Submitted by Amal Mohammed Al Hiddi to the College of Information Technology is satisfactory and acceptable for the Master of Science in Information Technology.

Thesis Defense Committee:

Dr. Wasan Shaker Awad
(Supervisor)

Department of Information Systems
University of Bahrain

Prof. Adnan Abdul-Aziz Gutub
(Member)

Umm Al-Qura University
Makkah - Saudi Arabia

Dr. Amjad Mahmood
(Member)

Department of Computer Science
University of Bahrain

Abstract

Stream cipher is important method in encryption for many reasons such as security level, easy to implement and efficient (in terms of space and time). The mechanism of how stream cipher work is that cryptographic key and algorithm are applied to each binary digit in a data stream, one bit at a time.

The main objective of this thesis is to propose a new effective algorithm to design stream cipher systems automatically using genetic programming with different representation method for representing the genetic programming population individuals compare with simulated annealing algorithm. Usually the individual programs of genetic program population represented using LISP expressions. In the proposed method, the programs are represented as strings of integers representing the individual program syntactic rule numbers. This representation method is called genetic algorithm for developing software (GADS). The performance of the proposed algorithm is studied by applying different genetic parameters and methods such as selection algorithm, crossover and mutation. Furthermore, it is compared with other representation methods such as LISP expression.

Acknowledgment

Praise be to Allah, who endowed me with courage, knowledge, guidance, health and energy towards the achievement of this study.

I would like to express my gratitude to my supervisor, Professor Dr. Wasan Shaker, for her valuable time, guidance, corporation, patience and useful comments that helped me to complete my thesis.

I am also grateful to all the lecturers and employees at UOB for bearing with us during our master study journey.

I dedicate this achievement to my beloved family: to my father and mother for their continuous support, to my husband for his encouragement that aided me during the most difficult times and helped me to realize my aim, and to my brothers and sister for their endless help.

I would also like to express my sincere thanks to the managers at the Central Informatics Organization for their assistance. My special thanks shall go to my direct manager, Rania Al Seesi, for her unremitting patience and relentless cooperation.

I would also like to express my earnest gratitude to Shaikh Salman AL Khalifa, General Director of Information Technology, and Mohammed Al Yassi, Director of Computer Operations who allowed me the opportunity to carry out this project.

My appreciation and thanks shall go to all my friends for their assistance and encouragement during my work on this project, and to those who I have not remembered at this time.

Table of Content

Subject	Page No.
Abstract	ii
Acknowledgment	iii
Table of Contents	iv
List of Tables	vii
List of Figures	ix
Abbreviations and Definitions	xii
Chapter One Introduction	
1.1 Overview	2
1.2 Statement of the Problem	3
1.3 Research objective	5
1.4 Research scope and limitations	5
1.5 Thesis Structure	5
Chapter Two Literature Review	
2.1 Chapter Overview	8
2.2 Genetic Algorithm Overview	8
2.3 Genetic Programming	10
2.3.1 Representation Scheme	13
2.3.1.1 LISP (LISt Processing)	13
2.3.1.2 GADS	14
2.3.2 Genetic Operations	16
2.3.2.1 Crossover	16

2.3.2.2 Mutation	19
2.3.3 Reproduction and Selection	19
2.3.3.1 Tournament Selection	20
2.3.3.2 Roulette Wheel Selection	21
2.3.3.3 Rank Selection	23
2.3.4 GP Complete Algorithm	23
2.4 Simulated Annealing	24
2.5 Stream Ciphers	26
2.5.1 LFSR	27
2.5.2 Randomness tests	28
2.6 Related Work	30
2.6.1 Hash function	31
2.6.2 Block cipher	31
2.6.3 Stream Ciphers	33
2.6.4 Related Work Discussion	34
Chapter Three Proposed Stream Cipher Design Method	
3.1 Problem Definition	37
3.2 Function and Terminal Sets	37
3.3 Representation Schema	39
3.4 The proposed System Architecture	45
3.5 Initial Population Generation	45
3.6 Convertor	46
3.7 Code Interpreter	50
3.8 Evaluator	50
3.9 New Population Generator	53
3.9.1Selection	54
3.9.2 Crossover Types	54

3.9.3 Mutation	54
3.10 Stream Cipher Design Algorithm	55
Chapter Four Results	
4.1 GADS Classes	59
4.2 Experiment Results	61
4.3 Comparison Between Different Chromosome Structures	67
4.4 Comparison Between Different Crossover Types	72
4.5 Comparison Between Different Selection Methods	77
4.6 SA Parameters	81
4.7 Comparison Between the Proposed GADS Method With SA and Without Using SA	83
4.8 Comparison between the proposed GADS method and previous SAP algorithm	84
4.9 Results Summary	85
Chapter Five Conclusion and future work	
5.1 Conclusion	91
5.2 Future Work	91
References	93
ملخص الرسالة باللغة العربية	

List of Tables

NO.	Subject	Page No.
1	Table (2.1) XOR table	28
2	Table (3.1) Functions library	39
3	Table (3.2) CFG rules set 1	40
4	Table (3.3) CFG rules set 2	41
5	Table (3.4) CFG rules set 3	41
6	Table (3.5) Terminal nodes and its equivalent Binary numbers	43
7	Table (3.6) Example of a program generation	44
8	Table (3.7) Example of parameters and output of Initial population generation algorithm	49
9	Table (4.1) Experiments of studying the performance of the proposed method	59
10	Table (4.2) GADS classes	60
11	Table (4.3) Results of experiments with different population sizes	62
12	Table (4.4) Results of experiments with different chromosome lengths	63
13	Table (4.5) Results of experiments with different number of generations	63
14	Table (4.6) Results of experiments with different mutation rates	64
15	Table (4.7) Results of experiments with different crossover rates	65
16	Table (4.8) Results of experiments with different FR	65

	block lengths	
17	Table (4.9) Results of experiment 3 with different lengths of FR and TR	66
18	Table (4.10) Results of experiment 6 with different lengths of FR block and TR block	67
19	Table (4.11) Experiment 9 results with different lengths of FR block and TR block	68
20	Table (4.12) The result of different SA initial temperature	82
21	Table (4.13) The result of different annealing schedule	82
22	Table (4.14) Comparison between GADS method with SA and GADS only	83
23	Table (4.15) A comparison between GADS and Awad's previous method	84
24	Table (4.16) Execution time comparison between GADS and Awad's previous method	85
25	Table (4.17) Summary of results	88

List of Figures

No.	Figure	Page No.
1	Figure (2.1) Genetic algorithm main steps	9
2	Figure (2.2) A flowchart representing genetic algorithm	11
3	Figure (2.3) The LISP tree	14
4	Figure (2.4) Ontogenetic mapping	15
5	Figure (2.5) Crossover in GP	17
6	Figure (2.6) Single point crossover	18
7	Figure (2.7) Multi point crossover	19
8	Figure (2.8) Mutation operation	20
9	Figure (2.9) Tournament selection algorithm	21
10	Figure (2.10) Roulette wheel selection algorithm	22
11	Figure (2.11) Genetic programming workflow	25
12	Figure (2.12) Stream cipher system	27
13	Figure (3.1) The problem definition	38
14	Figure (3.2) Non-linear combining functions	38
15	Figure (3.3) Examples of chromosome structures	47
16	Figure (3.4) The proposed system	47
17	Figure (3.5) The overall architecture of the proposed system	48
18	Figure (3.6) Initial population algorithm	48
19	Figure (3.7) Convertor algorithm	51
20	Figure (3.8) Code Interpreter algorithm	52
21	Figure (3.9) Flow diagram for fitness value calculation	53
22	Figure (3.10) The algorithm for new population generator	55
23	Figure (3.11) Stream cipher design algorithm	56
24	Figure (4.1) Example of GADS class 1	61

25	Figure (4.2) Example of GADS class 2	61
26	Figure (4.3) Example of GADS class 3	61
27	Figure (4.4) Result of comparing different chromosome structures with different population sizes	69
28	Figure (4.5) Result of comparing different chromosome structures with different chromosome lengths	70
29	Figure (4.6) Result of comparing different chromosome structures with different population sizes	70
30	Figure (4.7) Result of comparing different chromosome structures with different chromosome lengths	71
31	Figure (4.8) Result of comparing different chromosome structures with different population sizes	71
32	Figure (4.9) Result of comparing different chromosome structures with different chromosome lengths	72
33	Figure (4.10) Result of comparing different crossover types with different population size	73
34	Figure (4.11) Result of comparing different crossover types with different chromosome length	73
35	Figure (4.12) Result of comparing different crossover types with different population size	74
36	Figure (4.13) Result of comparing different crossover types with different chromosome length	75
37	Figure (4.14) Result of comparing different crossover types with different population size	76
38	Figure (4.15) Result of comparing different crossover types with different chromosome length	76
39	Figure (4.16) Result of comparing different selection methods with different population size	78

40	Figure (4.17) Result of comparing different selection methods with different chromosome length	78
41	Figure (4.18) Result of comparing different selection methods with different population size	79
42	Figure (4.19) Result of comparing different selection methods with different chromosome length	79
43	Figure (4.20) Result of comparing different selection methods with different population size	80
44	Figure (4.21) Result of comparing different selection methods with different chromosome length	81
45	Figure (4.22) Example number 1 of keystream generator	87
46	Figure (4.23) Example number 2 of keystream generator	87
47	Figure (4.24) Example number 3 of keystream generator	87
48	Figure (4.25) Example number 4 of keystream generator	87
49	Figure (4.26) Example number 5 of keystream generator	87
50	Figure (4.27) Example number 6 of keystream generator	88

Abbreviations

AES	Advanced Encryption Standards
AI	Artificial intelligence
ANN	Artificial Neural Network
BNF	Backus–Naur Form
CFG	Context-free grammar
CI	Computational intelligence
DES	Data Encryption Standard
EC	Evolutionary Computation
FR	Function Rules
GA	Genetic algorithm
GADS	Genetic Algorithm for Development Software
GEP	Gene Expression Programming
GP	Genetic programming
LFSR	Linear Feedback Shift Register
LISP	List Processing
LR	Leaves rules
SA	Simulated annealing
SAP	Simulated annealing programming
TS	Tabu search

TR	Terminals rules

Chapter 1 Introduction

1.1 Overview

1.2 Statement of the Problem

1.3 Research Objective

1.4 Research Scope and Limitations

1.5 Thesis Structure

Chapter 1 Introduction

1.1 Overview

Nowadays, in the organizations, there is a great dependence on information, computer systems, and computer communication and networking. Consequently, awareness of the need to security has been also increased. Information security is the process of protecting information from unauthorized access, use, disclosure, destruction, or modification. The organizations need to protect information from malicious actions that endanger the confidentiality of its information, such as snooping and traffic analysis. There are a number of information security goals, one of them is confidentiality. It is about protection of confidential information by encryption mechanism. The systems that are used to encrypt information are called cipher systems (cryptosystems) (Beker & Piper, 1982).

In the modern cipher systems, there are two kinds of cipher systems. One of them is block cipher and the other is stream cipher. The main idea of how the block cipher works is to divide the plaintext into blocks of the same size and encrypted by the same key. On the other hand, stream cipher works by encrypting each digit of plaintext by corresponding digit of the keystream (Beker & Piper, 1982). In this work, only stream cipher is considered.

It is not easy to design stream ciphers of good properties. Therefore, we need an intelligent technique for an automated design of stream ciphers. Genetic programming (GP) integrated with simulated annealing (SA) have been chosen to solve the problem of this study. GP is one of

the evolutionary computation techniques that can be used for automatic programming that invented by John R. Koza in 1990s (1992, 1994).. GP is an extension of genetic algorithm (GA) attributed to John H. Holland (Holland, 1976). It is about one of the important questions of computer science, that is, how the computer can do the tasks that it is required to do without being told exactly how to do it, and how it can learn from its own knowledge.

There are two main representation methods used to represent the individual programs of GP populations, LISP expressions and GADS. LISP is List Processing, and very old programming language proposed by John McCarthy in 1958 (1960) (1 year after Fortran). It is an artificial intelligence (AI) programming language and has symbol-oriented structure. GADS (Genetic Algorithm for Development Software) is a different representation in GP. The chromosomes are not represented as LISP expressions but as lists of integers representing the syntactic rules of programs.

SA is a random-search technique that is used to solve optimization problems. It is used when the structure of a space is not well understood or is not smooth (Kirkpatrick, Gelatt, & Vecchi, 1983). It is based on the technique that deal with heating and controlled cooling of a material to increase the size of its crystals and to reduce their defects.

1.2 Statement of the Problem

Encryption is an important mechanism used to protect information. Therefore, we can see many encryption techniques proposed with different levels of security and different structures. Stream cipher is an important class for many reasons, such as security degree, easy to

implement, efficient, and with no error propagation (Forouzan, 2008; Schneier, 1996).

Designing cipher systems of good characteristics, such as security level, efficiency, and ease of implementation, is a hard task. Therefore, this problem has attracted a number of researchers, and some of them proposed automated methods for designing cipher systems, as shown in section (2.6). This thesis is about to find an efficient and effective automated method for designing cipher systems to help the security researchers and industry to find strong cipher systems. The proposed method for designing cipher systems is based on GP, which is an application of GP. GP has been used to solve different kinds of problems and attracted many researchers. Thus, we can find so many studies about applying GP to solve complex problems, and most studies proved its effectiveness. Thus, this work is to study the effectiveness of GP in designing stream ciphers automatically. In the literature, we can find different studies about the automated design of stream cipher systems using computational intelligence techniques (CI). Awad (2011a, 2011b, 2013) proposed a different variations of GP-based stream cipher design algorithm, such as adaptive GP and simulated annealing programming (SAP). This thesis is to present a modified SAP algorithm using GADS.

Although GA (and GP) has gained many applications, simple GA suffers from many problems such as getting stuck in a local minimum and parameters dependence (Eiben, Hinterding, & Michalewic, 1999). There are many improvements proposed to enhance the performance of the GA. Therefore, in this work, to avoid the problem of getting stuck in a local minimum and to preserve good individuals into the next generation, SA is integrated with GP, the resulted algorithm is called SAP.

1.3 Research Objective

The main objectives of this research are:

1. To study the effectiveness of GADS combined with SA in designing stream cipher systems of high level of security.
2. To study the effectiveness of GADS in designing stream ciphers.
3. To compare between GADS and LISP-expression representation methods.
4. To study the effect of different GA methods and parameters on the performance of the proposed GADS-based design method.

1.4 Research Scope and Limitations

There are more than one kind of cipher systems (block, stream ..etc.) and each one has its own system structure and functions. In this research, we will use GP with different representation method and SA to design stream ciphers. Unfortunately we can't apply this design method to all kinds of ciphers. In this work, we will apply it to stream ciphers only. However, GP-automated design method presented here can be applied to design other ciphers by changing the function and terminal sets in addition to defining different fitness function.

1.5 Thesis Structure

This thesis consists of five chapters. In chapter 2, GA, GP, SA, and stream ciphers will be explained, including the two representation methods used in GP, which are LISP and GADS, number of genetic operations (crossover and mutation), and the selection methods. The remainder of the chapter gives some information about what is SA. At the

end of this chapter, we will display related works that discussed different problems of information security that used number of CI to solve it.

Chapter 3 is to describe the proposed method of designing stream ciphers using GADS compare with SA. Functions and terminals sets and representation schema is defined. After that the proposed architecture of the system is explain in details.

The result of implementing the proposed method is presented in chapter four. we will explore different results that is depending on various values of parameters, and several methods and structures that have been examined.

The last chapter of this thesis summarizes the findings of this project and makes various recommendations to improve the proposed method of designing stream cipher using the GP along with SA.

Chapter 2 Literature Review

2.1 Chapter Overview

2.2 Genetic Algorithm Overview

2.3 Genetic Programming

2.4 Simulated Annealing

2.5 Stream Ciphers

2.6 Related Work

Chapter 2 Literature Review

2.1 Chapter Overview

In order to understand the proposed method for designing stream ciphers automatically, it is necessary to comprehend some concepts and techniques. Thus, this chapter aims to present the concepts of GA and GP, the algorithms and their operations. Furthermore, it gives an overview of stream ciphers.

2.2 Genetic Algorithm Overview

GA is an intelligent search technique that can be applied to solve a set of optimization problems. The GA theoretical foundation was developed by John Holland (1975). The main idea of GA has been derived from the evolutional process in natural biological life: the natural population will develop according to the natural selection principle "survival of the fittest" (Beasley & Chu, 1996; Holland, 1975).

Before explaining GA, the following key terms need to be defined (Beasley & Chu, 1996; Holland, 1975):

1. Chromosome or population Individual: Any candidate solution in the search space of a problem, and it is an element in GA population.
2. Population: A set of chromosomes or individuals, and population size is the number of the chromosomes in the population.
3. Search Space: All possible solutions of the problem.

Usually GA works by creating an initial population with a set of chromosomes randomly. Then, evaluate the fitness of each chromosome

and give them score depending on how they perform in the fitness function. After that, two chromosomes will be selected according to their fitness score. The chromosome with higher fitness will have more chance to be selected according to selection method. The selected two chromosomes (parents) will be recombined to create new chromosomes (children) that will be added to the new population. These steps will be repeated until a proper solution is found or an termination criterion is met, as shown in figures (2.1) and (2.2).

```
Generate an initial population;
Evaluate fitness of individuals in the population;
repeat
    Select parents from the population;
    Recombine (mate) parents to produce children;
    Evaluate fitness of the children;
    Replace some or all of the population by the chil-
    dren;
until a satisfactory solution has been found;
```

Figure (2.1) Genetic algorithm main steps (Beasley & Chu, 1996)

The advantages of GA can be summarized as follows (Beasley & Chu, 1996; Holland, 1975):

1. The most powerful methods will create high quality solutions to a problem.
2. GA will work well in complex, poorly understood and large search spaces.
3. GA will be useful if there is no mathematical analysis available.
4. GA can work effectively when traditional search methods fail.

However, GA has also its own disadvantages: it can take a long time to run and provide the solution; therefore, it is not always feasible for real time use.

2.3 Genetic Programming

As defined by Koza (1992, 1994) who introduced the concept of GP, GP is an extension of GA. GP deals with computer programming by finding the solution to a particular problem as computer program rather than the solution itself. GP deals with one of the most important questions in computer science: how the computer can perform tasks without being told exactly how to do them. Thus, the search space of GP is the space of all possible computer programs composed of functions and terminals appropriate to the problem domain.

In order to design a GP algorithm to solve any problem, the following steps must be considered (Koza, 1994):

1. Determine the set of terminals.
2. Select the set of primitive functions.
3. Define the fitness function.
4. Decide on the parameters for controlling the run of GP.
5. Choose the method for designating a result of the run; usually, the single best-so-far individual of the last generation is designated as the result of the run.
6. Choose the representation method to represent the population individuals.

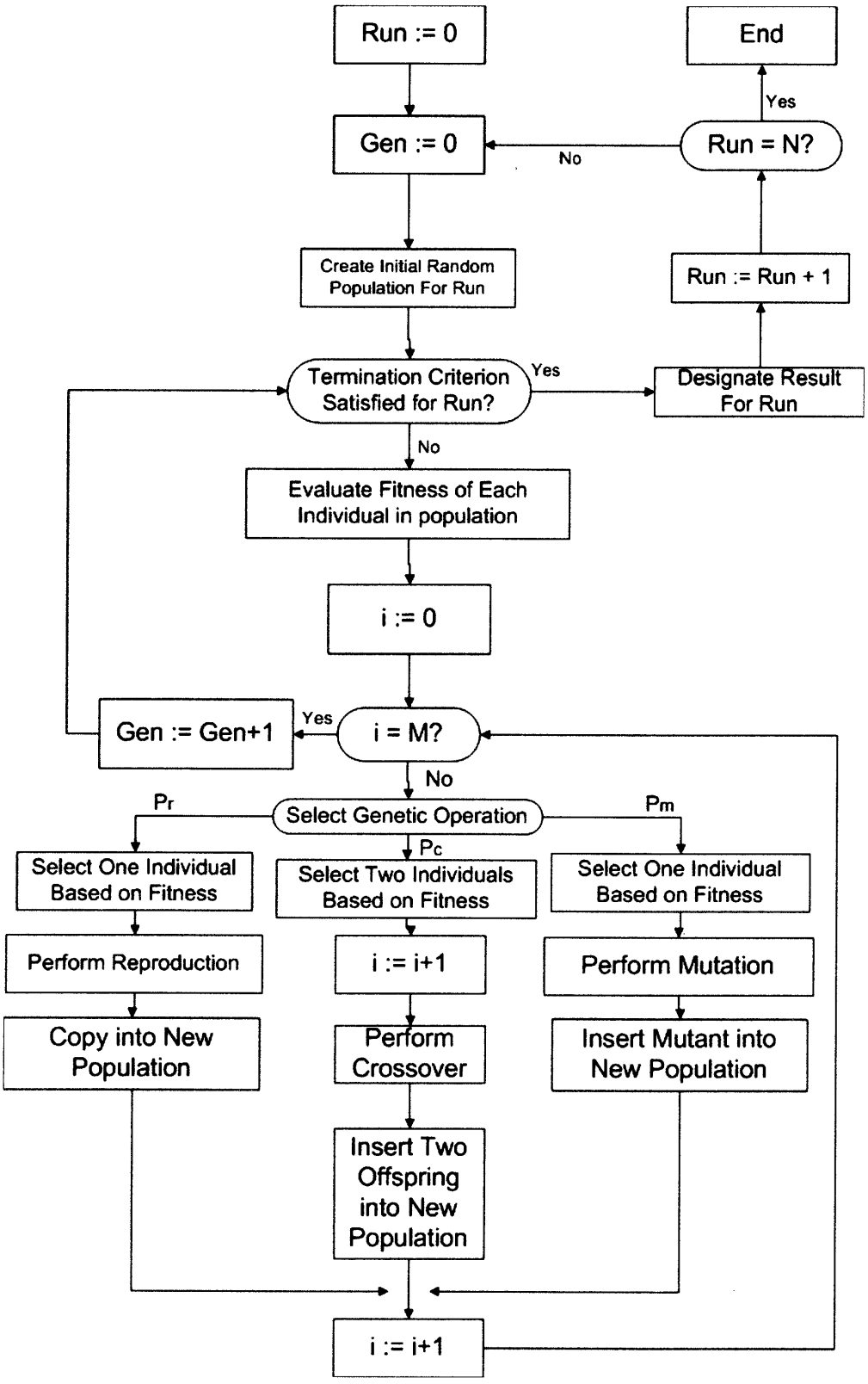


Figure (2.2) A flowchart representing genetic algorithm (Koza, 1994)

The solution of a problem evolved by GP is a computer program. Therefore, the population of GP is the set of candidate solutions which are programs. Each program is composed of a number of primitive functions and terminals.

To apply GP, it is necessary to define the terminal set. The terminals include the constants and variables of the population individuals (programs). The primitive functions are chosen based on the problem at hand. The function set may include arithmetic functions such as subtraction, addition, division and multiplication. It can be a set of logical functions (OR, AND, OXR ...etc.) and so on. Terminal and function sets together define the alphabetic of the program that will be evolved by GP (Koza, 1994).

The fitness function will evaluate each population program by measuring its performance in the problem environment. Accordingly, it is essential to define how fitness measure will work, based on the type and nature of the problem. This constitutes the most important and most difficult step in GP, because it determines how good the programs are in solving the problem. If the fitness measure is weak, then the program that will be evolved by GP at the end will also be weak (Koza, 1994).

The control parameters of GP need also to be identified. There are a number of controls parameters in GP (Koza, 1994):

1. Population size: the number of candidate solutions (programs/chromosomes) in each population of GP.
2. The maximum number of generations of GP run.
3. Maximum and minimum chromosome (program) length.
4. Genetic operation rates.

5. Other parameters specific to the problem.

2.3.1 Representation Scheme

This section will introduce two representation methods used to represent the population chromosomes of GP.

2.3.1.1 LISP (LISt Processing)

Koza et al. (Hernandez-Castro & Estevez-Tapiador, 2006; Koza, 1994; Holland, 1975) used LISP programming language (Lari & Young, 1990; Siebel & Peter, 2005) to represent the candidate programs in the GP populations. The LISP program consists of two types of entities: list and atoms. An atom can be constant such as 4, or variable like (Counter). The List in the LISP is represented by a collection of items placed between parentheses, such as (+ 6 8). The syntactic form in the LISP is called S-Expressions (symbolic expressions). In S-expression, the item in the outermost left parenthesis is a function that is applied to the other items of the list. For example, the LISP S-Expression:

(* 3 10 (- (+ 1 7 2) 5))

In this example, the first function in the list is "*", and it has three arguments: 3, 10 and the sub S-Expression (- (+ 1 7 2) 5)). In this sub S-Expression, the function is " –" and it has two arguments: the sub expression (+ 1 7 2) and constant 5. The sub expression (+ 1 7 2) has "+" as a function that will add the 3 arguments 1, 7 and 2. The result of the last sub expression will be 10, and the result of (- (+ 1 7 2) 5) will be 5. The final result will be 150. This S-expression can be represented by an expression tree as shown in figure (2.3).

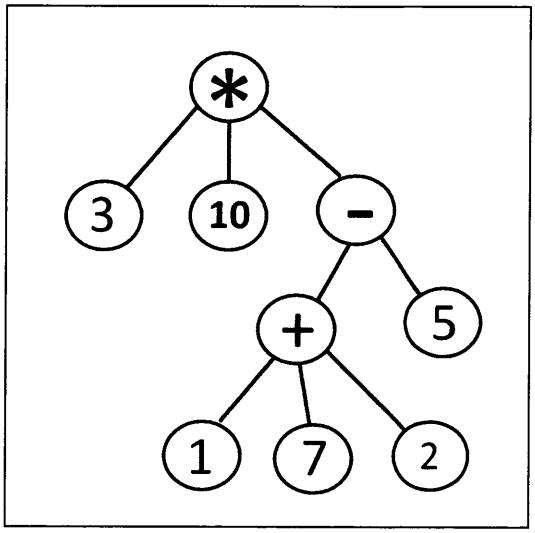


Figure (2.3) The LISP tree

2.3.1.2 GADS

GADS was first introduced by Paterson and Livesey in 1996 (Paterson & Livesey, 1996). The chromosomes of GADS are not represented using the tree structure but as a list of integers.

GADS makes use of BNF (Backus–Naur Form) (Naur, 1963). BNF is a method used for representing context-free grammar that is used to characterize the formal language syntax in computer science. In the 1950's, context-free grammar (CFG) was developed by Noam Chomsky (Chomsky, 1956). He developed BNF when he was working in IBM as a programming language designer to represent the syntax of the new programming language.

The rules of CFG for defining the syntax of a formal language take the form below:

S → X, where S is a non-terminal value. It must be single and on the left side. The X symbol can be terminal or non-terminal value, or a mix of both of them. Here the values can be string, single or even empty.

In GADS, a genotype (chromosome or genetic search space element) is represented as a list of integers (rather than a tree in normal GP). These integers will be the input for the appropriate generator which is used to produce the phenotype (program or solution space element). This process of mapping genotype-phenotype is called ontogenetic mapping as shown in figure (2.4). To modify a genotype, the same operations of GA such as crossover, mutation… etc. are used. To evaluate the fitness of a genotype, GADS tests the phenotype in the environment in which it operates. The result of the testing gives the fitness of the phenotype, and by implication, of its genotype (Paterson & Livesey, 1996 ; Bastaki & Awad, 2003).

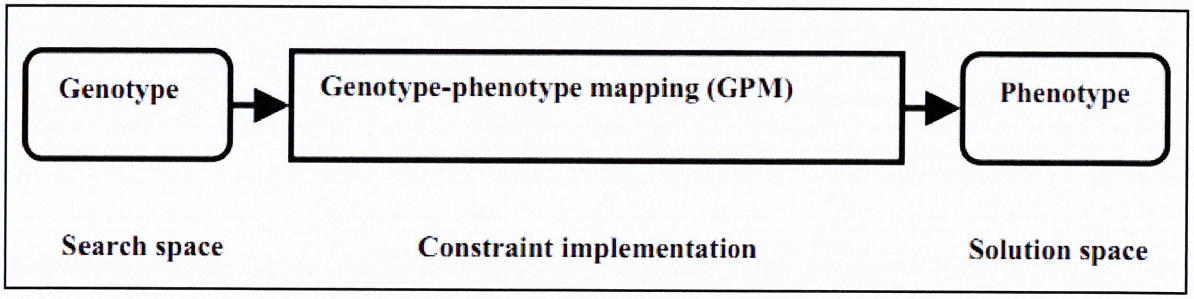


Figure (2.4) Ontogenetic mapping (Bastaki & Awad, 2003)

The advantages of using GADS (Kellegoz, Toklu, & Wilson, 2008; Bastaki & Shaker, 2003):

1. The programming language that will be used to produce the phenotype language depends on the problem domain which is addressed by GADS. Most of the GP uses LISP, but GADS is not limited to LISP.
2. The difference between genotype and phenotype is drawn clearly.
3. A traditional GA engine can be used to produce a solution of a problem as a program, and hence, using GADS will improve the efficiency of GP.

The problems of this approach are (Ryan, Collins, & Neill, 1998):

1. In case of a large grammar, the probability of choosing a specific rule in that grammar is decreased. Thus, to solve this problem and to improve the chance of selecting a particular rule, the rule in the grammar can be duplicated.
2. Sometimes, at the end of a chromosome, when it is created randomly, the non-terminals do not have any terminal chosen for them. The solution to this problem will be using a default value which is inserted into the chromosome. The default value must be tailored for each production rule.

2.3.2 Genetic Operations

This section will expose a number of genetic operations that can be used to update the current population and generate a new population. All genetic operations are applied in specific rates.

2.3.2.1 Crossover

In this genetic operation, more than one chromosome are selected from the current population to produce new chromosomes that will be added to the new population. This can be done by creating a crossover point in each chromosome, and all the genes behind that point will be swapped with the other chromosomes to produce the new chromosomes.

Before discussing the different types of crossovers, the crossover of GP using LISP will be discussed. The crossover is applied to the selected expression trees and the crossover point will be selected in two trees. Then the sub tree after this point will be swapped with the other parent

sub tree to create new two trees, as shown in figure (2.5) (Beasley & Chu, 1996; Koza, 1994).

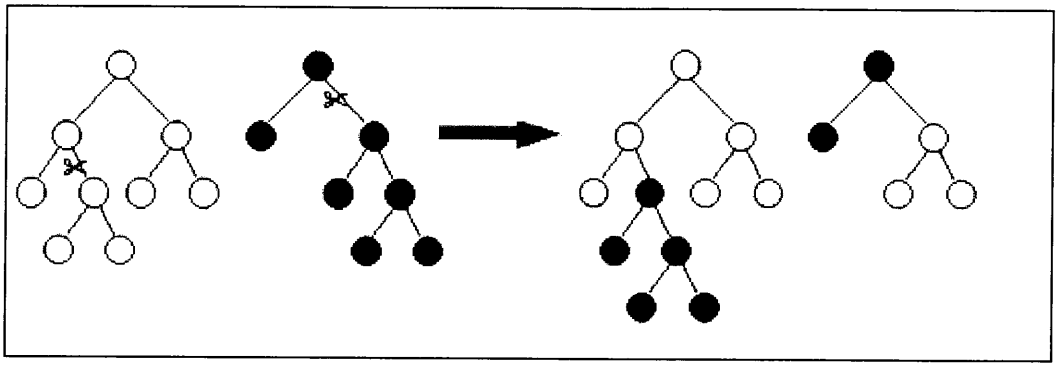


Figure (2.5) Crossover in GP

There are different types of crossover operation that can be applied in GA. These include:

1. Single point crossover

In this method, a single point position is chosen randomly in the parent chromosome. From this position until the end, the two chromosomes will exchange their values, as shown in figure (2.6) (Kellegoz, Toklu, & Wilson, 2008).

2. Multi point crossover

This method is a generalization from the previous one (Single point crossover). In a Multi point crossover several points are chosen randomly. Between each two points the chromosomes will exchange their values, as shown in figure (2.7) (Spears & Jong, 1990).

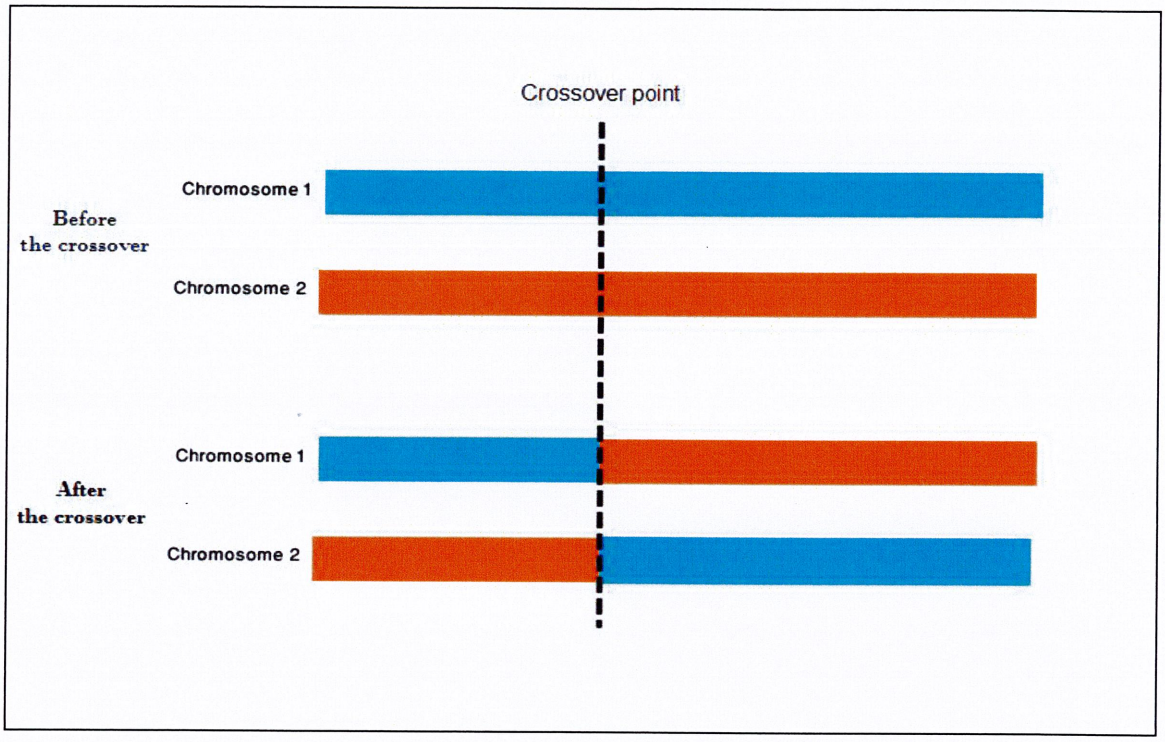


Figure (2.6) Single point crossover

3. Uniform Crossover

The Uniform Crossover method works when the chromosomes share their gene values to create the new chromosomes by letting the crossover operator determine that with some probability known as mixing ratio. For example, if the mixing ratio is set to 0.5, that means the new chromosomes have almost half of the genes from the first parent chromosome and the other half from the second parent chromosome. The crossover points will be chosen randomly.

The advantage of this method is that it allows the chromosomes to be mixed at the gene level rather than at the segment level. Yet, the disadvantage of using Uniform Crossover is that it can destroy building blocks (Syswerda, 1989).

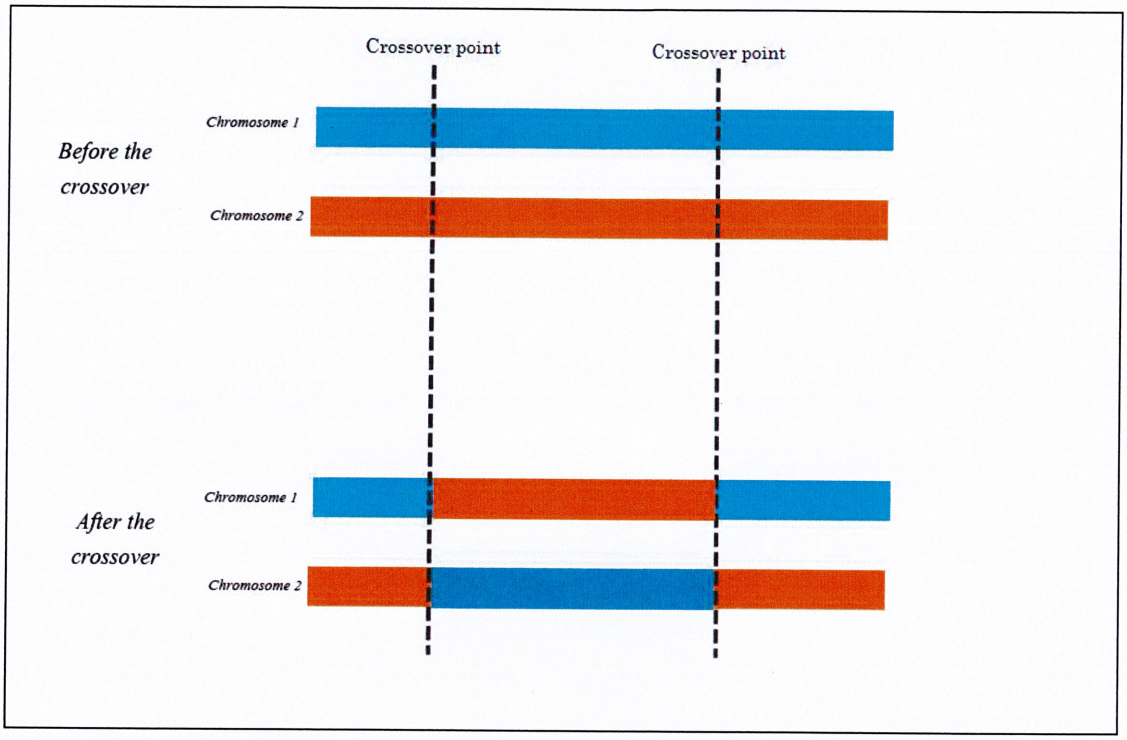


Figure (2.7) Multi point crossover

2.3.2.2 Mutation

Mutation operation will create a new chromosome from the selected parent of the old population by changing some parts of that chromosome randomly (Koza, 1994).

2.3.3 Reproduction and Selection

This process consists of selecting the chromosomes from the current population and copying them in the new one without any change. The selection depends on the fitness measurement. The purpose of using this operator is to reuse the individuals that have proven themselves capable of solving the problem so that they we will not be lost. But on the other hand, if the operator is overused, it can damage the goal of the GP; therefore, the operator has to be used wisely (Koza, 1994).

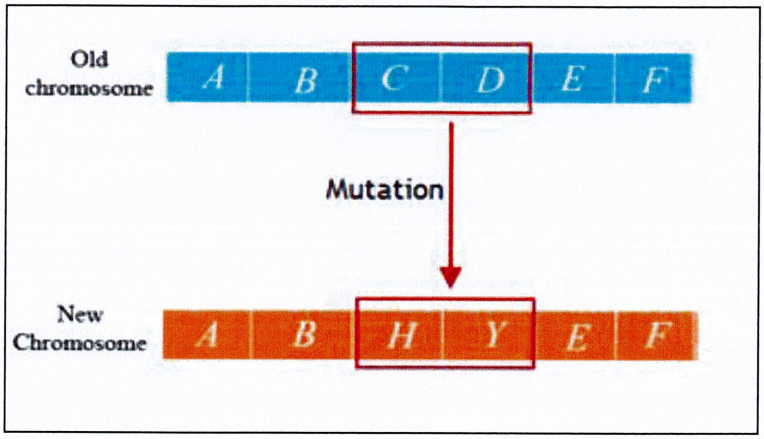


Figure (2.8) Mutation operation

The strategy of selection in GP (and GA) consists of choosing a chromosomes in the existing population and using it to generate a new population, so the fitness of the new population will be better. That can be done by giving the chromosomes with higher fitness high probability to move to the next population.

There are different methods of selection with different strategies, but all of them are built on the principle of survival of the fittest. In this section some of those methods will be discussed.

2.3.3.1 Tournament selection

Tournament selection is the most popular method for selection, for its simple implementation and efficiency. Tournament selection works by doing a tournament for the chromosomes in the population as competitors. The winner in this tournament will be the chromosome with the highest fitness among the other competitors. This chromosome will be inserted into the new population, as shown in figure (2.9).

The number of chromosomes competing in each tournament is referred to as tournament size. "Tournament selection gives a chance to all individuals to be selected and thus it preserves diversity, although

keeping diversity may degrade the convergence speed. In tournament selection, larger values of tournament size lead to higher expected loss of diversity. The larger tournament size means that a smaller portion of the population actually contributes to genetic diversity, making the search increasingly greedy in nature." (Miller, & Goldberg, 1995; Noraini, & John, 2011).

Algorithm: Tournament Selection.

Input: The population $P\ (\tau)$ the tournament size $t \in \{1,2,\ldots..,N\}$

Output: The population after selection $P\ (\tau)'$

Tournament $(t, J_1, \ldots..., J_N)$:

For $i \leftarrow 1$ **to** N **do**

$J'_i \leftarrow$ best fit individual out of t randomly picked individuals from $\{J_1, \ldots..., J_N\}$;

Return $\{J'_1, \ldots..., J'_N\}$

Figure (2.9) Tournament selection algorithm (Noraini & John, 2011)

2.3.3.2 Roulette Wheel Selection

As stated by Rakesh and Jyotishree (2012), Roulette wheel is the simplest selection method. In this approach, the chromosomes will be placed in the Roulette wheel according to their fitness values. The probability of the chromosome i, P (choice = i), can be calculated in Roulette Wheel as follows:

$$P\ (choice = i) = \text{def} \frac{fitness\ (i)}{\sum_{j=1}^{n} fitness\ (j)} \quad \text{----------(2.1)}$$

Then, the following algorithm can be applied to select the chromosomes, as shown in figure (2.10).

```
Algorithm: Roulette Wheel Selection.

r := random number, where 0≤ r < 1;
sum := 0;
for each individual i
{   sum := sum + P(choice = i);
    if r < sum
    { return i ;
    }
}
```

Figure (2.10) Roulette wheel selection algorithm (Rakesh & Jyotishree, 2012)

The disadvantage of Roulette Wheel selection method is that the chromosomes with higher fitness have more chance to be selected to move to the new population. That probability can lead to miss the best chromosomes in the old population; therefore, there is no guarantee for the good chromosomes to find a way to reach to the new population (Rakesh & Jyotishree, 2012).

On the other hand, the advantage of this method, as mentioned earlier, is that it is the simplest selection method, which makes it an easy medium to implement and simulate nature more truly. It also has the propensity of setting the threshold to the value of zero, and at the same time it is likely to converge to unlimited expansion. For all these reasons, it is much more appealing than the other methods.

2.3.3.3 Rank Selection

The Rank Selection method works by first sorting the chromosomes of the population according to their value of fitness and then ranking them. The chromosomes will be given a selection probability according to their ranks. The chromosome at the top of the list has more chances to be selected and transformed to the new population.

The advantage of the Rank selection method is that it can block very fit chromosomes from winning and can control the new population so early and ignore the chromosomes with less fitness value. This will result in reducing the population's genetic variety and may block the effort of finding reasonable solution. On the other hand, the disadvantage of this method lies in the fact that when implementing it, it requires sorting and ranking the whole population, which renders the procedure time-exhausting (Noraini, & John, 2011; Rakesh, & Jyotishree, 2012).

2.3.4 GP Complete Algorithm

The steps of GP algorithm are the following (Koza, 1993; Koza, 1994; Matthew, William, Guido, Jessen, & Martin, 2006):

1. Generate the initial population randomly.
2. Repeat:
    - Execute each program in the population and measure its fitness using the fitness function.
    - Apply selection and genetic operations to produce new population until the termination criterion is satisfied.
3. After the termination criterion is satisfied, the single best program in the population produced during the run (the best-so-far individual) is harvested and designated as the result of the run. If the run is

successful, the result may be a solution (or approximate solution) to the problem.

Figure (2.11) shows the flowchart of GP and the different genetic operations used in GP.

2.4. Simulated Annealing

Simulated Annealing (SA) was introduced by Kirkpatrik (Kirkpatrick, Gelatt, & Vecchi, 1983). It is an iterative improvement technique used to solve optimization problems, especially if we have a set of combinatorial optimization problems so we can find a good solution. SA can help to avoid the problem of getting stuck in a local minimum (Kirkpatrick, Gelatt, & Vecchi, 1983). The idea of how SA works comes from the annealing process in mineralogy: when the temperature is high, the molecules of liquid move freely and molecules respect each other when they are moving. When the molecules move slowly, the liquid cools down and thermal mobility is lost.

In SA algorithm, if the move to the next problem state improves the situation, it is always executed; otherwise, the algorithm makes the move with some probability less than 1. The probability decreases exponentially with the "badness" of the move. The algorithm parameter T (temperature) is used to moves are more likely to be allowed. As T tends to zero, they become more and more unlikely. The schedule determines the rate at which the temperature is lowered.

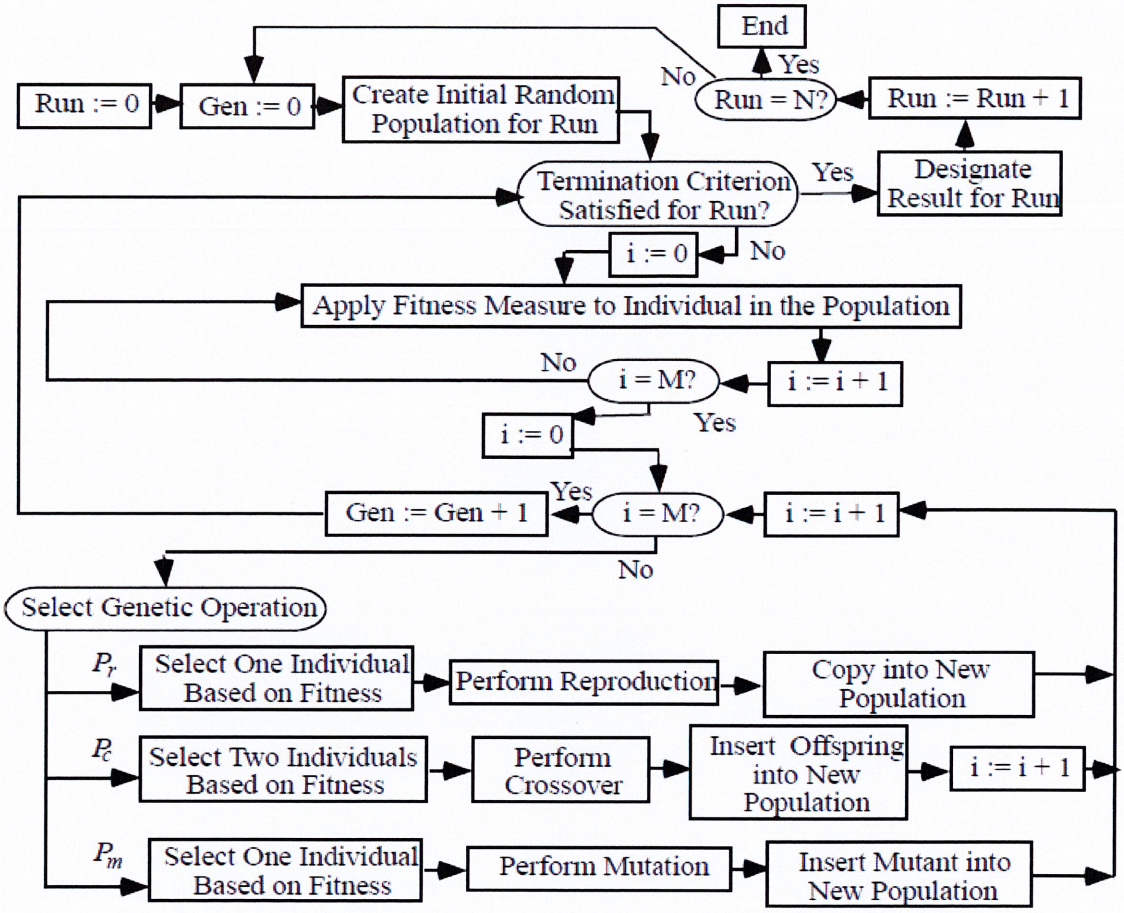


Figure (2.11) Genetic programming workflow (Koza, 1994)

Many problems of different types have been solved using SA. The good thing in SA is that it can be integrated with GA. That will help to maintain good individuals into the next generations. The complete algorithm of SA is the following:

1. Evaluate the initial state.
2. Initialize the temperature (T) according to the annealing schedule.
3. Loop until a solution is found, or until there are no new operators left to be applied in the current state.

4.1 Apply an operator to the current sate to produce a new state.

4.2 Evaluate the new state; if the new state is the goal then return it, otherwise:

If value of current state less than value of new state, then make the new state as current state, and set best-so-far to the new state.

If value of current state greater than value of new state, then E = value of current state - value of new state PR = $e^{-E/T,}$ make the new state as current state with probability PR.

4.3 Reduce T.

5. Return best-so-far as the answer.

2.5 Stream Ciphers

To protect information from unauthorized access encryption mechanism can be used. Nowadays, we can find many cipher systems (cryptosystems) used for encrypting the private information of different types, and a number of cipher systems have been proposed with different levels of security. These cryptosystems can be classified into block ciphers and stream ciphers. Block ciphers divide the plaintext into blocks and encipher each block independently, such as Data Encryption Standard (DES) and Advanced Encryption Standards (AES) systems. Stream ciphers, on the other hand, are extremely fast and easy to implement. In addition, they usually have very minimal hardware resource requirements. Therefore, stream ciphers are of great importance in applications where encryption speed is paramount and where area-constrained or memory-constrained devices make it impractical to use block ciphers. Stream ciphers can operate one data unit as small as a bit or a character (Forouzan, 2008; Schneier, 1996).

Every stream cryptosystem consists of two parts:

- Keystream (random sequence) generator, and

- Mixer (XOR for the binary sequence).

A keystream generator, which is the heart of stream ciphers, outputs a stream of bits (keystreams) XORed with a stream of plaintext bits to produce the stream of cipher text, as shown in figure (2.12).

There are different types of keystream generators; one of them is LFSR-based generators, in which the basic component is linear feedback shift register.

The challenge is how to design a good keystream generator because if the keystream is short and repeated frequently, that repetition will help the attacker to expose it (Menezes, Van Oorschot, & Vanstone, 2001; Qi, 2008). Thus, a good keystream must have the following features:

- Long and not repeated and does not have a specific pattern.
- Random.
- Be of large linear complexity.

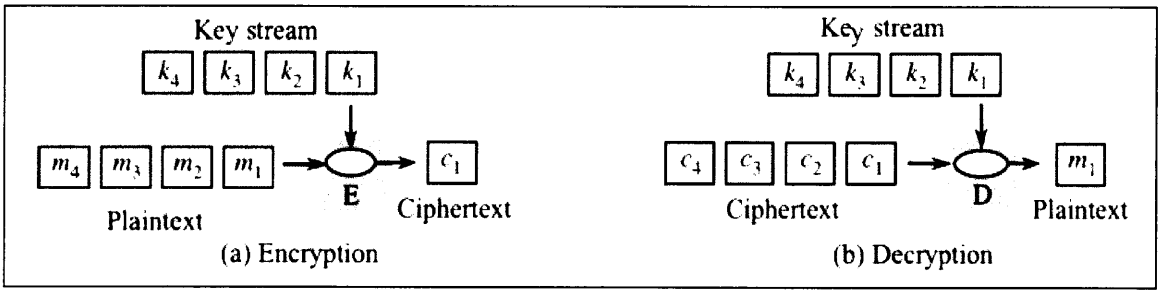


Figure (2.12) Stream cipher system (Forouzan, 2008)

2.5.1 LFSR

LFSR (Linear Feedback Shift Register) is a shift register with linear feedback function in which the only operation used is XOR. Table (2.1)

presents the behavior of XOR (Menezes, Van Oorschot, & Vanstone, 2001; Qi, 2008).

Table (2.1) XOR table

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

There is an initial input that LFSR uses at the beginning called seed. The tap is the position of the bits that will be used to tell the shift register where to find the bits that will XOR them to find the new bit that will be added to the keystream in the last significant bit (stage 0). The last bit will be removed.

To understand how LFSR works, let's assume that the seed is 1111 in {4-bits} shift register and that the tap sequence is {2, 3}. The LFSR will read the second and the third bits and XOR them. The result will be 0 and it will be inserted in the left side of the LFSR. The last bit on the right will be removed. Now the new state of LFSR is 0111. The procedure will be repeated.

The maximum length of Binary sequence generated by a LFSR of length n can be calculated using this formula below:

$$L = 2^n - 1 \quad \text{---------- (2.2)}$$

2.5.2 Randomness tests

The following are a number of randomness tests that can be applied to test randomness of keystreams (Knuth, 1998; L'ecuyer, & Simard, 2007; Yalçın, Suykens, & Vandewalle, 2004):

1. Frequency Test

A basic test that tests if there is no difference between number of 1's and 0's of the binary sequence, i.e., the number of zeros must be equal to the number of ones.

2. Serial Test

the same concept of previous test but it tests the sequences of two digits at a time (00, 01, 10 and 11). In general, a serial test calculates the number of times that $(X_{2i}, X_{2i}+1) = (p,r)$ exist with $0 \leq i < n$. For each pair of integers (p,r) with $0 \leq p, r < S$, this calculation has to be done and a chi-square test is applied to these $k=S^2$ categories with probability $1/S^2$ in each category.

3. Poker Test

"The poker test determines whether the sequences of length appear approximately the same number of times in the bitstream. The bitstream is divided into k ($k=[n/m]$) non-overlapping parts each of length m. Let n_i be the number of occurrences of the i^{th} type of sequence of length $m=1, 2, \ldots, 2^m$. The statistic is then: (Yalçın, Suykens, & Vandewalle, 2004)

$$X_3 = \frac{2^m}{k}\left(\sum_{i=0}^{2^m} n_i^2\right) - k \quad \text{----------} \quad (2.3)$$

Runs Test: it will divide the sequence into neighboring streams of 0's called gapes and neighboring stream of 1's called blocks. "If r_0^i is the number of gaps of length i, then half of the gaps will have length 1 bit, a quarter with length 2 bits, and an eighth with length 3 bits. If r_1^i is the number of blocks of length i, then the distribution of blocks is similar to

the number of gaps" (Kirkpatrick, Gelatt, & Vecchi, 1983). The statistic used in this test is shown in the equation below:

$$X_4 = \sum_{i=1}^{k} \frac{(B_i - e_i)^2}{e_i} + \sum_{i=1}^{k} \frac{(G_i - e_i)^2}{e_i} \quad \text{----------}$$

(2.4)

"With $e_i = (n - i + 3)/2^{i+2}$ (the expected number of gaps and blocks of length i), where *n* is the length of the random sequence andB_i, G_i is the number of blocks and gaps, respectively of length i in the bitstream for each $i = 1, 2 \ldots k$" (Yalçın, Suykens, & Vandewalle, 2004). Knuth (1998) views that this test method is very strong.

2.6 Related Work

Because of the ever increasing connections in the world; the important of cryptography has increased as cryptography is used to secure public communication media, such as the Internet, to make the users feel confident about the security of their communications. There are a number cryptography algorithms (or ciphers), with different security levels and performance. (Burke, McDonald, & Austin, 2000; Picek, & Golub, 2011).

In the literature you can find a number of researches that discuss different problems of information security. Furthermore, you can find many studies that make use of Evolutionary Computation (EC) algorithms to solve cryptography problems. There is an increased interest in this field in the recent years. There are a number of evolutionary computation techniques, such as GA, GP, tabu search (TS) and SA that have been used to solve a number of cryptology problems.

2.6.1 Hash function

"Cryptographic hash functions are fundamental primitives in modern cryptography and have many security applications (data integrity checking, cryptographic protocols, digital signatures, pseudo random number generators ... etc.) (Varrette, Muszyński, & Bouvry, 2012). A Hash function takes a message of any length, and then creates a fixed length hash value (Damg°ard, 1990).

Estébanez et al. (Estébanez, Hernández-Castro, Ribagorda, & Isasi, 2006) used GP to design hash functions to make them stronger and faster. At the end, they found that "GP system is able to produce competitive hash functions that are able to outperform other well-known, expert designed, commonly-used hash functions, and it is faster than other hash functions used in the experiments and which were generated by practiced humans with years of experience". As a result, designing hash functions using GP was a successful and that makes researchers more interested in applying GP to other methods in cryptography.

Varrette et al. (Varrette, Muszyński, & Bouvry, 2012) designed new hash functions using Gene Expression Programming (GEP). GEP was proposed by Ferreira (2002) as an extension of GP. The results of this design are quite promising.

2.6.2 Block cipher

Hernandez-Castro and Estevez-Tapiador (2006) discussed how we can create block ciphers using GP. They introduced a new block cipher design, called Wheedham, which operates on 512-bit blocks and keys of 256 bits, they offer its C code (directly translated from the GP Trees) and some preliminary security results. In the conclusion of this paper they

found that GP can be successfully applied to design competitive block ciphers. Still they suggested doing more improvements on it.

Xiangyang (2010) presents a new S-box optimization method based on the combination of genetic algorithm and TS. S-box is used in block cipher algorithm to provide the confounding effect that is necessary for cipher algorithm, and makes the dependency among cipher and plaintext as well as cipher text very complex, so that the cryptanalyst cannot use the dependency, and the S-box is the only non-linear component, its cipher strength will determine the security strength of the entire block cipher. After testing the new method, the results show that using the improved tabu-genetic algorithm to construct S-box is feasible and effective.

Hashem and others (Hashem, AL-Hamami, & AL-Hamami, 2011) generated a key for block ciphers using two methods of EC algorithms. One of them is Artificial Neural Network (ANN), and the second one by using GA. It was another successful experience in using EC algorithms in cryptography.

Julio et al. (Julio, Castro, Pedro, Viñuela, Cristóbal, & Arco-Calderón, 2003) presented how they designed a highly nonlinear functions using GP. The importance of finding a good design for highly nonlinear functions emanates from the significance of those functions. Nonlinear functions are useful in different key applications such as building hash functions for large databases and in the development of different cryptographic functions like message authentication codes, block ciphers... etc. The authors proved in their paper that the resulting functions that they design are extremely efficient both in hardware and in software.

2.6.3 Stream Ciphers

Ali (2014) proposed a design method for designing strong Boolean functions using Memetic algorithm. Boolean functions are used as nonlinear combining functions in certain stream ciphers. Good Boolean functions need to be found in cryptography so ciphers can resist cryptanalytic attack. Ali's paper introduces the Memetic algorithms as an extension of the traditional GA with a local hill climbing. It is like "a marriage between a population-based global technique and a local search made by each of the individuals". Ali shows that Memetic algorithm is used successfully to generate strong Boolean functions for cryptographic applications.

Millan et al. (Millan, Clark, & Dawson, 1998) introduced another design method for strong balanced Boolean functions using a modification of the original GA along with Hill climbing techniques that are adapted to locate balanced, highly nonlinear Boolean functions that also almost satisfy correlation immunity. The findings of this paper also highlight the effectiveness of the methods proposed.

Abdulsalam (2011) introduced a new design method for stream cipher using SA. Abdulsalam built a new key generator using the new technique that attracted a lot of attention because of its ability to solve very complex optimization problems. The findings of this paper show that "new algorithm is far more effective than blind search in finding good keystream randomness with large linear complexity". Abdulsalam proved that using new technique in designing stream cipher can be successful.

The main purpose of Awad's paper (2011a) is to design an effective algorithm for designing keystream generators, and also to study the

effectiveness of GP and SA to solve this problem. The proposed method was implemented using C++ programming language. The results showed that using GP yields good results. Furthermore, by using GP and SA together, the performance is highly improved rather than applying GP only. At the end of this paper the author stated: "the results obtained can be improved by considering more factors in the fitness evaluation, such as linear complexity, other randomness tests, and correlation immunity. Furthermore, other techniques and genetic operations can be examined and compared in order to find the best algorithm for solving the underlying problem". This will constitute the main subject to be addressed in this thesis.

2.6.4 Related Work Discussion

The main points that extract from previews studies are:

- They proved that using EC techniques can solve cryptography problems.
- Applying GP in cryptography gives promising results.
- Using improved method of EC in general and GP in specific give better results than the traditional method.
- Using new techniques in designing stream cipher lead to successful design.
- Applying GP combine with SA in designing stream cipher gives high performance rather than applying GP alone.

In this thesis, it is discuss how to design stream cipher using improve method of GP which is GADS because from the previews work it has been approved that using new technique in cryptography can gives better results. In this new design it will combine the improve GP with SA because it was already approve that design stream cipher with tradition

GP and SA give promising results, so it is trying to improve those results in this thesis

Chapter 3 Proposed Stream Cipher Design Method

3.1 Problem Definition

3.2 Function and Terminal Sets

3.3 Representation Schema

3.4 The Proposed System Architecture

3.5 Initial Population Generation

3.6 Convertor

3.7 Code Interpreter

3.8 Evaluator

3.9 New Population Generator

3.10 Stream Cipher Design Algorithm

Chapter 3 Proposed Stream Cipher Design Method

3.1 Problem Definition

Designing cipher system of good characteristics, such as security level, efficiency, and ease of implementation, is not an easy task. This explains why this area has been an appealing field of research and attracted the interest of many researchers, some of whom proposed automated methods for designing cipher systems. However, so far, while there are many studies about using GA and GP in block cipher design, there is just one research paper for stream cipher that discusses how to use GP with LISP as representation method in its design as discussed in chapter two. For this reason, we decided to consider the automated design of stream cipher based on GP combine with SA using different representation methods in order to obtain better results. The proposed method is used GADS because of its advantages that mention in chapter two. SA also used because in order to avoid the problem of getting stuck in a local minimum. Accordingly, the problem under study is formulated, as shown in figure (3.1).

3.2 Function and Terminal Sets

First, we have to define the set of functions. In GP, the structure under adaptation is a set of programs representing the nominee keystream generators. The main function in the proposed function set that will be used in this method is LFSR (Linear Feedback Shift Register). It is

obligatory to use LFSR because the keystream generators implemented in this work are based on LFSR.

The problem: Automated design of stream ciphers that fulfils desired properties, such as security level.

Input: The desired properties of stream cipher, such as keystream length.

Search space: Set of all possible programs (keystream generators).

States: Population of candidate solutions(programs).

Output: A program that represents LFSR-based keystream generator to generate pseudorandom Binary sequences.

Figure (3.1) The problem definition

The problem in LFSR is inherently linear makes the security of keystream generator is weak, and the amenability to be attacked is high. To avoid that, a number of LFSRs can be combined using a nonlinear combining functions f , as shown in figure (3.2) (Zhou, Qing, & Okamoto, 2001).

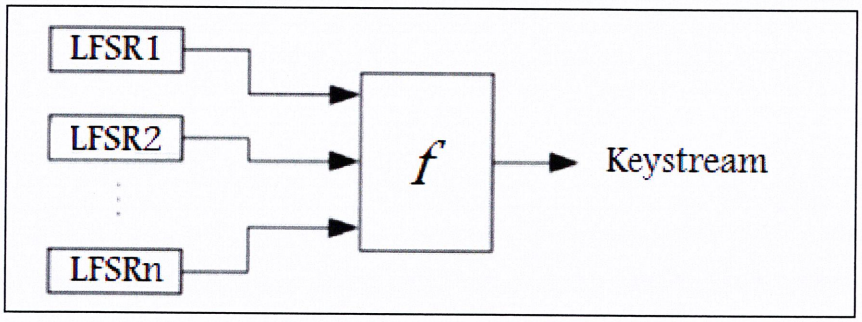


Figure (3.2) Non-linear combining functions (Zhou, Qing, & Okamoto, 2001)

For this reason, we need to include more functions in the function set. Thus, AND (bitwise and), OR (bitwise or) and XOR (addition mod 2) are chosen to be in the function set, as presented in table (3.1).

Table (3.1) Functions library

Symbol	Format	Description
#	# x	LFSR and x is a feedback function.
&	& x y	AND logic operation, x and y are sequences Binary numbers.
|	| x y	OR logic operation, x and y are sequences Binary numbers.
^	^ x y	XOR logic operation, x and y are sequences Binary numbers.

To apply the proposed method for generating a keystream, a representation of solutions to the problem must be appropriately chosen. The standard stream cipher uses the binary representation in which each solution (keystream) coded as a binary string. Therefore, the initial state of LFSR must contain a binary string.

3.3 Representation Schema

Usually in GP, LISP expressions are used to represent the population individual programs (chromosomes); in this work, the chromosomes are represented as sequences of integers. These integers represent the individual program syntactic (CFG) rule numbers. In order to find the best set of CFG rules, three sets have been examined, as shown in tables

(3.2, 3.3, 3.4, and 3.5). In GADS, the set of CFG rules used affects its performance as it be shown in chapter four.

Table (3.2) CFG rules set 1

Number	CFG Rule
0	S →& S S
1	S → | S S
2	S → ^ S S
3	S → # X
4	X →0 X
5	X → 1 X
6	X → 0
7	X → 1

According to these three sets, GADS can be classified into three classes, as follows:

1. GADS class1: It uses rules set 1, in which 0 and 1 are used to represent the terminals in CFG.
2. GADS class2: It uses rules set 2. It is similar to class1 except that rules 4 and 5 are duplicated to be of higher probability, and hence, we will have longer LFSR in the chromosomes.
3. GADS class3: It uses rules set 3. Instead of using 0 and 1 as terminals, the letters "a" to "h" are used in this class. Each of the letters will represent a sequence of three bits as shown in the table

(3.5). This representation will be more compact comparable with Binary representation.

For example, consider the chromosome generated randomly by the method algorithm is: 2, 3, 3, 5, 7, 14, 8, 12, then, the corresponding program will be generated as shown in table (3.6).

Table (3.3) CFG rules set 2

Number	CFG Rule
0	S →& S S
1	S → | S S
2	S → ^ S S
3	S → # X
4	X → 0 X
5	X →X 0
6	X → 1 X
7	X →X 1
8	X → 0
9	X → 1

Table (3.4) CFG rules set 3

Number	CFG Rule
0	S →& S S

1	S → | S S
2	S → ^ S S
3	S → # X
4	X →a X
5	X →b X
6	X →c X
7	X →d X
8	X → e X
9	X → f X
10	X → g X
11	X → h X
12	X → a
13	X → b
14	X → c
15	X → d
16	X → e
17	X → f
18	X → g
19	X → h

Table (3.5) Terminal nodes and its equivalent Binary numbers

Terminal latter	**Equivalent Binary number**
a	000
b	001
c	010
d	011
e	100
f	101
g	110
h	111

To find the best class suitable for the proposed designing method in terms of effectiveness and efficiency, these classes have been implemented to study their performance. Each class has been analyzed in order to choose a class that gives good results, as shown in chapter four.

Regarding the chromosomes structure and format, we proposed three chromosome structures, and we divided the CFG rules of table (3.4) into three categories, which are:

1. The first category includes rules 0 to 3, which represent the rules for generating functions which are AND, OR, XOR and LFSR. This category is called Function Rules (FR).

2. The second category includes the first set of terminals rules (TR) which includes the rules from 4 to 11. That's rules contain non-terminal and terminal values in CFG.

3. The third category is represented by the leaves rules (LR) that have all rules from 12. That's rules contain only terminal value in CFG.

The three chromosomes structures, as shown in figure (3.3), are:

1. Chromosome Structure1: A chromosome include any rule number without any restriction, as shown in figure (3.3.a).
2. Chromosome Structure2: A chromosome is composed of two parts. The first block is for FR numbers, and second is for TR numbers, as shown in figure (3.3.b).
3. Chromosome Structure3: A chromosome has three parts. The first is for FR, the second is for TR and the last block is for LR, figure (3.3.c).

Those three structures are the most suitable structures for my method, because the first one mix from all the categories randomly. The second and third structures, the first part in the chromosome will contain only the function rule which is the most important rules, since it has the LFSR function which the keystream generators implemented in this work based on it.

Table (3.6) Example of a program generation

Step	**Program (phenotype) generation**	**Rule applied**
1	^SS	2
2	^#XS	3

3	^#X#X	3
4	^#bX#X	5
5	^#bdX#X	7
6	^#bdc#X	14
7	^#bdc#eX	8
8	^#bdc#ea	12
9	^#001011010#100000	

3.4 The Proposed System Architecture

The proposed system is a GP-based system used to find best keystream generator that can generate a random Binary sequence of a given length, as shown in figure (3.4). That show the needed components to implement the proposed method.

3.5 Initial Population Generation

The GP population includes a number of chromosomes which consist of the numbers from 0 to 19 representing the CFG rule numbers, as shown in table (3.4). The algorithm that is used to generate the initial population for the proposed method is shown in figure (3.6). the time complexity of this algorithm is O(popsize * chromlength).

For example, consider table (3.7). It presents the algorithm parameters and its output.

3.6 Convertor

It is used to convert the genotype to phenotype. It is a procedure that works as follows: read the chromosome, then search for the FR (from 0 to 3). It converts any rule number found to what it represents in CFG rules. For example, if the number found is 3 in the chromosome, it will be converted to "# X". Then it will search for the TR (from 4 to 11) and replaces the nonterminal letter "X" with the number that represents it. Therefore, if it finds rule number 5 in the chromosome, it will search for first nonterminal letter "X" and replaces it with "b X". After that, it will replace the rest of nonterminal letter "X" with a number from 12 to 19. Finally, the phenotype (candidate keystream generator) will be generated to contain the functions and terminals only.

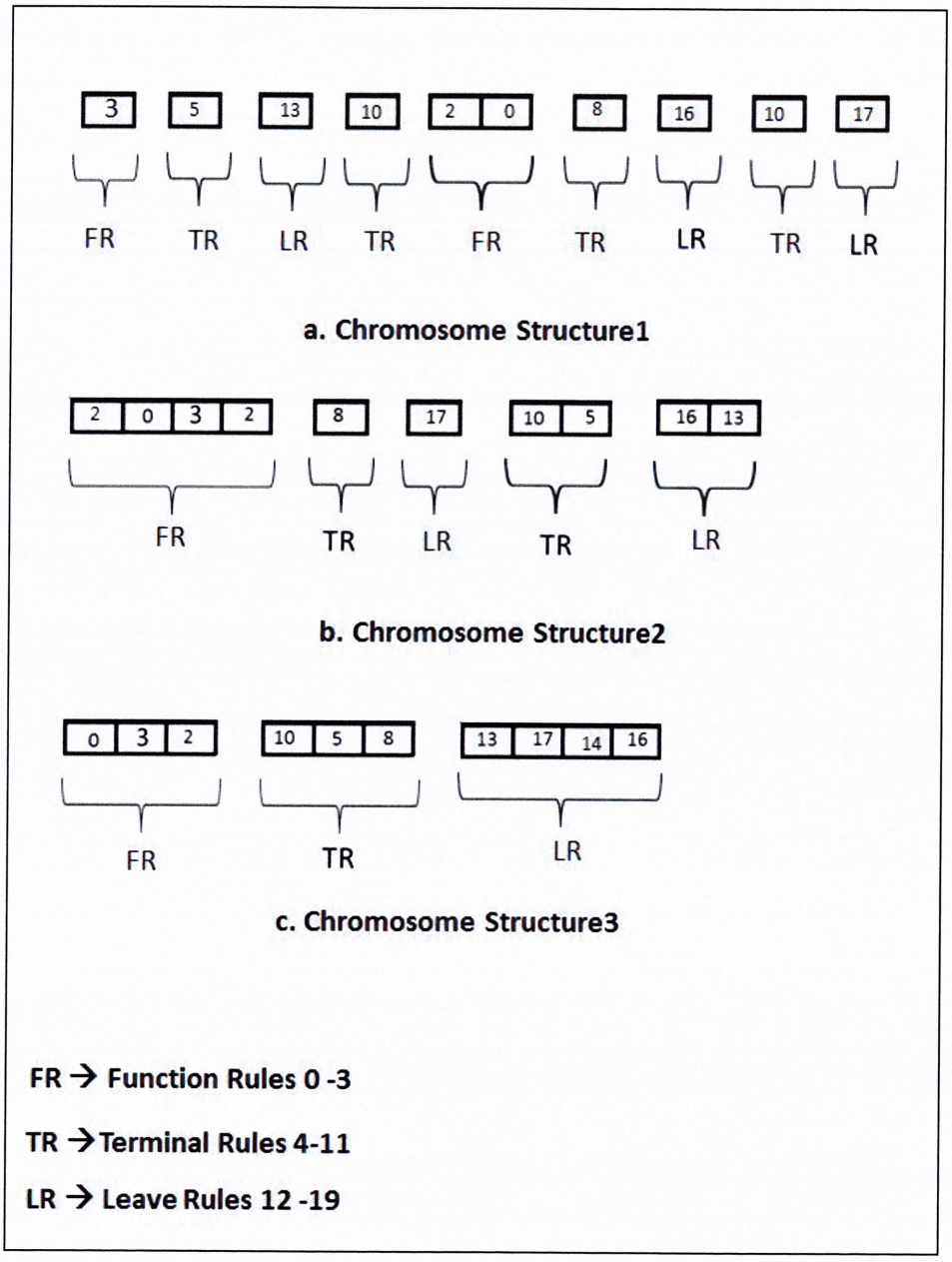


Figure (3.3) Examples of chromosome structures

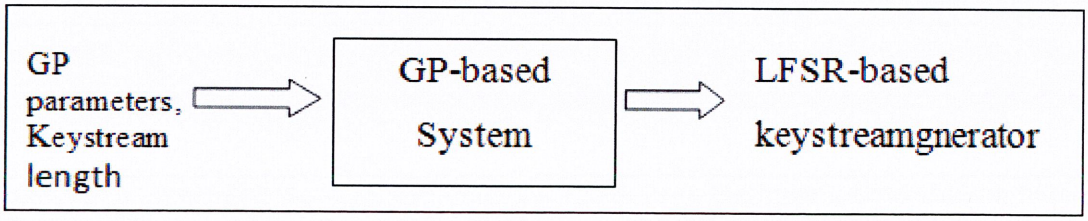


Figure (3.4) The proposed system

The overall architecture of the proposed system is shown in figure (3.5).

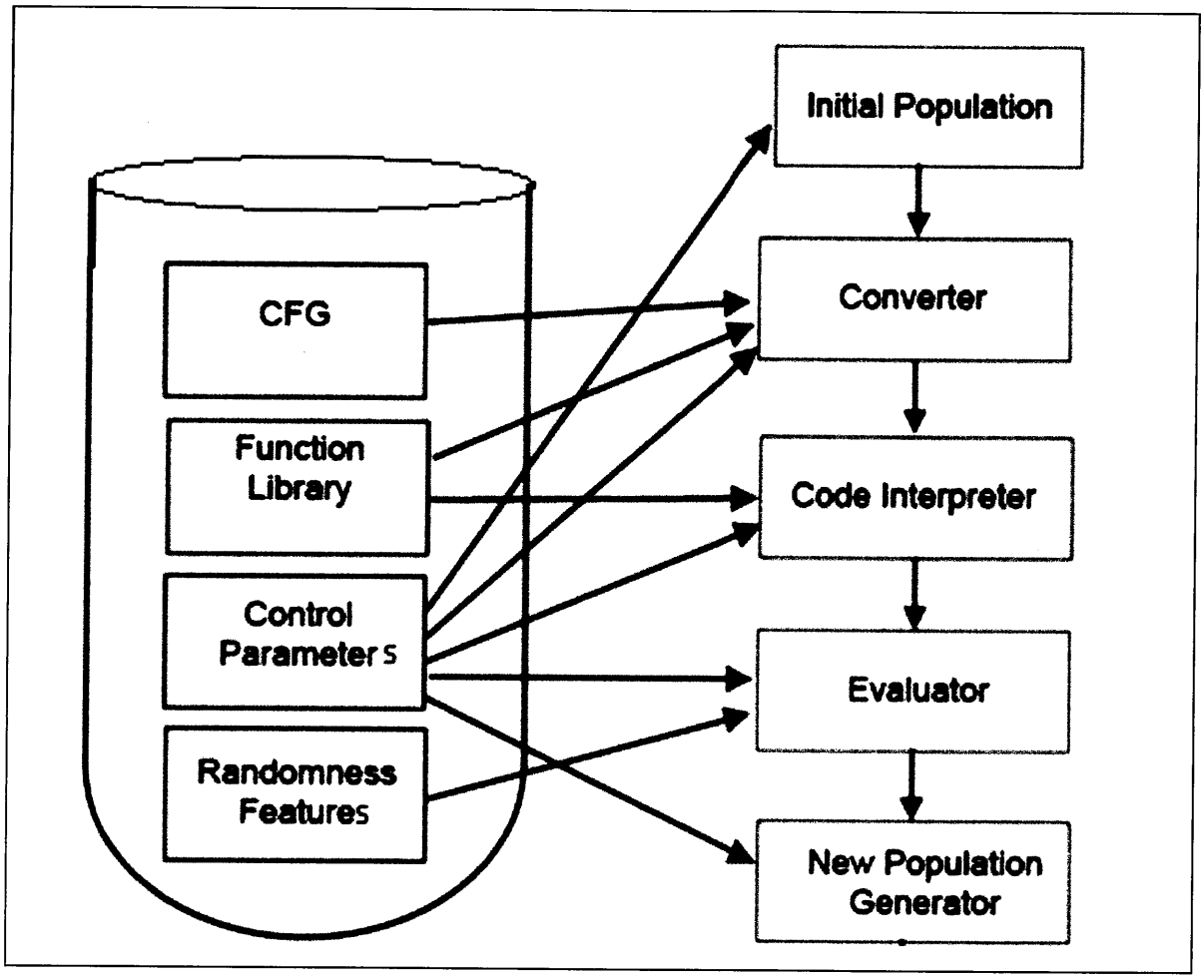


Figure (3.5) The overall architecture of the proposed system

Algorithm 1: (Initial Population Generation)
Input: population size (popsize), chromosome length (chromlength).
Output: initial population.

Begin
for i =1 to popsize do
 for j = 1 to chromlength do
Cromosome$_{ij}$ = random number from 0 to 19;
End;

Figure (3.6) Initial population algorithm

Table (3.7) Example of system parameters and output of Initial population generation algorithm

Chromosome length	8
Population size	5
Chromosome structure	1
Function operation	Generating 8 random numbers from 0 – 19 for each chromosome
Function Output (population)	4, 6, 7, 11, 3, 10, 6, 5 11, 2, 12, 15, 1, 19, 5, 5 7, 17, 15, 12, 1, 19, 5, 0 3, 2, 8, 11, 4, 19, 13, 9 5, 11, 12, 18, 6, 17, 7, 5

The problem in the above conversion method is: What if the chromosome does not contain rule number 3 that represents the LFSR function? This function is the core of the CFG we use, and without it, the keystreams cannot be generated. Thus, we need to modify the above conversion method. The converter will search for any nonterminal letter "S" and replaces it with LFSR function "# X". Also, in case it does not find any number from FR (0 to 3), it will add LFSR function.

Even with this method, still we have a problem, because the converter searches for the first nonterminal letter "X" that appeared in the chromosome and it is replaced by TR. Therefore, the first LFSR function will be very long and all other LFSR functions will be short since they contain only one or two letters as feedback functions. This will affect the performance of the proposed method.

For example:

The phenotype contain nonterminal "X": &^#X#X#X#X

The phenotype after replace the nonterminal "X": &^#dbddebc#f#a#h

To solve this problem, we opted for the modification of the program so that it contains a counter that counts the number of terminals for each LFSR function, and ensures that it will not extend four letters. Based on above, Algorithm 2 is the converter algorithm presented in figure (3.7) which is of time complexity $O(chromlength)^2$.

3.7 Code Interpreter

After converting genotype to phenotype, the generated phenotype (keystream generator) need to be executed in order to evaluate the generated keystream. The code interpreter is used to execute each key stream generator in GP populations. Code interpreter is the program that executes the instructions written in a programming language. It is used to analyze each statement in the program each time it is executed and then perform the desired action. Algorithm 3 is the algorithm of the code interpreter used in the proposed method. The algorithm for code interpreter is given in figure (3.8).

3.8 Evaluator

This component is for evaluating the chromosomes and computing the fitness value for each chromosome. The fitness value evaluates how good the keystream generator is and it important in GP to create the new population. After executing each candidate keystream generator, the resulted Binary keystream is examined. There are a number of metrics used to analyze keystream generators such as correlation immunity, linear complexity and keystream randomness (Awad, 2011a).

```
Algorithm 2: Convertor
Input: chromosome (genotype) of length chromlength
Output: phenotype, i.e., keystream generator (KS)

Begin
for i = 1 to chromlength do
Search in the chromosome for FR (from 0 to 3);
Add corresponding CFG rule to KS;
End for;
If KS is empty then
Add LFSR function;
Search for any nonterminal letter "S" left and replace it with
LFSR function;
for i = 1 to chromlength do
    Search in chromosome for TR and LR (from 4 to 19);
     Replace it with corresponding CFG rule;
       If the number of terminals of a LFSR function   =
MAX then
       Replace nonterminal letter "X" with LR (from 12 to
            19);
End for;
End;
```

Figure (3.7) Convertor algorithm

Randomness and keystream period length have been considered in this study because the results is compared with Awad (2011a) results so it should be the same evaluator method, as shown in figure (3.9). Thus, the fitness function emanates from the fact that in a random sequence:

1. $n_0 = n_1$
2. $n_{01} = n_{11} = n_{10} = n_{00}$, where, n_w is the frequency of w in the Binary sequence.

Algorithm 3: (Code Interpreter, **GACI** for short)
Input: individual program, keystream length in Bits, i=0;
Output: Binary keystream

Begin
If $program_i$= 'a'..'h' then convert it to Binary;
If $program_i$= '&' then return GACI and GACI;
If $program_i$= '|' then return GACI or GACI;
If $program_i$= '^' then return GACI xor GACI;
If $program_i$= '#' then generate keystream of a given length;
i= i+1;
End;

Figure (3.8) Code Interpreter algorithm

3. $1/2^i \times n_r$ runs in the Binary sequence are of length i, where n_r is the number of runs in the sequence, i.e.,

$$\left(\frac{1}{2^i} \times n_r\right) = n_i \text{ for all } i = 1 \text{ to M} \quad \text{---------- (3.1)}$$

In equation (3.2), n_i is the desired number of runs of length i, and M is maximum run length.

Three factors are considered in the fitness evaluation of the chromosomes which are:

1. Randomness of the generated keystream.
2. Keystream period length.
3. Program size.

Therefore, the fitness function *fit(x)*(Awad, 2011a, 2011b, 2013) used to evaluate each chromosome x can be defined as follows, where *size* is the period length, *wt* is a constant, and *length* is the program size:

$$f1 = |n_0 - n_1| \quad \text{---------- (3.2)}$$

$$f2 = \left|n_{00} - \frac{size}{4}\right| + \left|n_{01} - \frac{size}{4}\right| + \left|n_{10} - \frac{size}{4}\right| + \left|n_{11} - \frac{size}{4}\right| \quad \text{---------- (3.3)}$$

$$f3 = \sum_{i=1}^{M} \left| \left(\frac{1}{2^i} \times n_r \right) - n_i \right| \quad \text{---------- (3.4)}$$

$$fit(x) = \frac{size}{1 + f_1 + f_2 + f_3} + \frac{wt}{length(x)} \quad \text{---------- (3.5)}$$

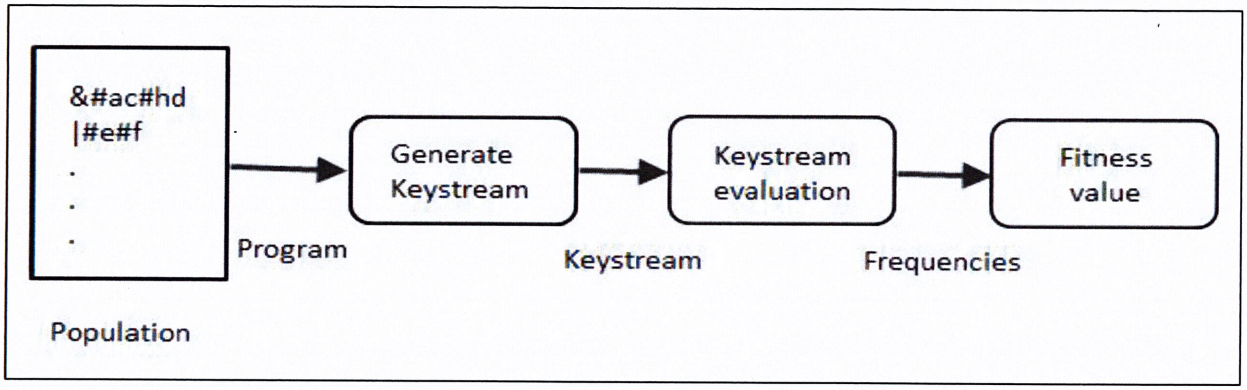


Figure (3.9) Flow diagram for fitness value calculation

3.9 New Population Generator

To generate a new population, a number of genetic operations are used, which are selection, crossover, mutation, and replacement. Two selection algorithms have been examined, tournament and roulette wheel. Also, single-point and multi-point crossovers have been studied with different rates. In the replacement, the whole population is replaced by newly generated population. Algorithm 4, as shown in figure (3.10), describes the process of generating a new population.

3.9.1 Selection

The selection mechanism determines which chromosomes are chosen for the production of the next population. The main principle of the selection strategy is "the better a chromosome, the higher its chance for being chosen."

In the proposed method for designing stream cipher, two methods of selection are used. To study the effect of using different methods on the whole design, the selection methods that will be used here are Tournament Selection and Roulette Wheel Selection.

3.9.2 Crossover Types

After the selection functions had been applied, it will shift two chromosomes to crossover function to generate the new population.

Two types of crossover operations will be applied in this proposed method. The first one is Single point and the second is Multi-point crossover. The reason of choosing those two types of crossover are they are simple and easy to implement so they often used in GA and GP (Sharapov, 2007).

3.9.3 Mutation

Mutation operation will create a new chromosome by turning one element in the old chromosome into a new element. Mutation will be applied depending on its chosen probability.

Mutation probability is basically a measure of the similarity that random elements of the chosen chromosome will be turned into something else. For example, if the chosen chromosome is of length 100 and we choose the mutation probability to be 1%, it means that 1 out of 100 bits (on average) picked at random will be changed.

Algorithm 4: (New population generator).

Input: population, population size (popsize), crossover rate (cr), and mutation rate (mr)

Output: New population

```
Begin
        For i = 1 to popsize/2 do
                Select two chromosomes by using selection
                procedure;
                Generate a random number r;
                If r ≤ cr then
                        Perform crossover on chromosome;
                       For each child chromosome do
                        Generate a random number r;
                        If r ≤ mr then
                            Perform mutation on chromosome;
                       End for;
              Else
               Add the selected two chromosomes to the new
               population;

           End for;
End;
```

Figure (3.10) The algorithm for new population generator

3. 10 Stream Cipher Design Algorithm

GP and SA are integrated to avoid the problem of getting stuck in a local minimum and to preserve good individuals into the next generation (Awad, 2011b). Algorithm 5 presents the main steps of designing stream cipher , as shown in figure (3.11). The time complexity for this algorithm is O(Max number of generation*popsize).

Algorithm 5: (Cipher Design)
Input: Keystream period length (size)
Output: LFSR-based keystream generator

```
Begin
        Generate the initial population (pop) randomly;
        Convert genotypes to phenotypes;
        Evaluate pop,
        T = 250:
        While not Max Number of generations do
                Generate a new population (pop1) by applying
                Crossover and mutation;
                Convert genotypes to phenotypes;
                  Evaluate the fitness of the new generated
                  chromosomes of pop1;
                Calculate the averages of fitness values for pop and
                pop1, av and av1 respectively;
                  If (av1 >av) then replace the old population by the
                  new one, i.e., pop = pop1
                  Else
                          Begin
                                  e = av - av1;
                                  Pr = -e/T;
                                  Generate a random number (rnd);
                                  If (exp(pr) >rnd) then pop = pop1;
                  End Else
                   EndIf


                 Reduce T;
        End While
                        Return the best chromosome of the last
generation;
End.
```

Figure (3.11) Stream cipher design algorithm

Chapter 4 Results

4.1 GADS Classes

4.2 Experiment Results

4.3 Comparison Between Different Chromosome Structures

4.4 Comparison Between Different Crossover Type

4.5 Comparison Between Different Selection Methods

4.6 SA Parameters

4.7 Comparison Between the Proposed GADS Method With SA and Without Using SA

4.8 Comparison between the proposed GADS method and previous SAP algorithm

4.9 Result Summary

Chapter 4 Results

The main objectives of this study are to devise an efficient and effective method for designing keystream generators of stream cipher automatically; in addition to assess the effectiveness of GADS in solving this problem. This chapter of the thesis presents the results and findings of applying the proposed method, and examines its effectiveness to solve the problem stated in the study. The proposed method has been implemented using C++ programming language.

To appraise the performance of using GADS in designing keystream generators of stream cipher, a number of experiments have been carried out to compare different genetic operations and parameters as shown in table (4.1). The results of each experiment have been obtained by running the program 20 times and finding the average of best fitness values of the 20 runs. The keystream length in all runs is 200 Bits, and the proposed designing algorithm control parameters are the following:

1. Keystream length in Bits.
2. Population size.
3. Chromosome length.
4. Maximum number of generations.
5. Crossover and mutation rates.
6. FR, LR, and TR block sizes.
7. SA initial temperature.
8. Annealing schedule.

The default value of the control parameters are: Population size is set to 350, chromosome length and maximum number of generation are

equal to 30, crossover rate is 1, mutation rate is equal to 0.05, SA initial temperature is 250 and annealing schedule is 0.95. This values had been chosen because they give the best results throw testing the proposed design of stream cipher, as it will be shown later in this chapter. The stopping parameter for the program is maximum number of generations.

Table (4.1) Experiments of studying the performance of the proposed method

	Exp. no.1	**Exp. no.2**	**Exp. no.3**	**Exp. no.4**	**Exp. no.5**	**Exp. no.6**	**Exp. no.7**	**Exp. no.8**	**Exp. no.9**
Chromosome structure	1	2	3	1	2	3	1	2	3
Crossover type	Single-point crossover			Multi-point crossover			Multi-point crossover		
Selection method	Tournament selection						Roulette Wheel Selection		

4.1 GADS Classes

In section 3.3, three different GADS classes have been introduced. Thus, in order to find the best class suitable for the proposed designing method in terms of effectiveness and efficiency, these classes have been implemented to study their performance. The problems of each class have been examined so as to choose a class that gives good results. As shown in table (4.2).

Table (4.2) GADS classes

GADS classes	**Problems**	**Improvement**	**Next enhancements class number**

Class 1	The problem we faced with this class is the probability of choosing equal rules of the CFG. Some rules need to be of higher probability to choose, such as rules 4 and 5 in (table 3.2), and hence, the LFSR will be very short and may contain only one bit. This will lead to generate weak keystream generator of low fitness values.	To avoid this problem, these rules are duplicated in the grammar to raise its use.	Class 2
Class 2	The length of LFSR may still be insufficient. It is still small and its fitness value will be unsatisfactory. That is because by using 0 and 1 as terminals that represent LFSR feedback function, we need to generate long Binary sequence which is inefficient.	Instead of using 0 and 1 as terminals, letters are used to get more compact representation of feedback functions. Each of the letters represents a Binary sequence.	Class 3

Figure (4.1) presents the actual run that shows the problem we face when using GADS class 1. The LFSR is very short and contains 1 or 2 bits most of the times.



Figure (4.1) Example of GADS class 1

Figure (4.2) shows an example of using GADS class 2. Most of LFSRs are very short.

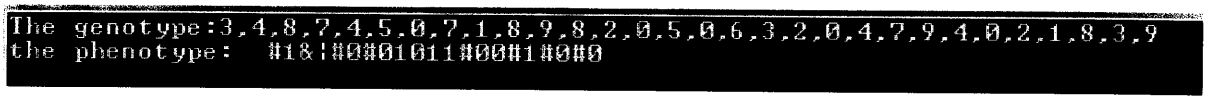


Figure (4.2) Example of GADS class 2

Figure (4.3) shows an example of using GADS class 3. We can see that the shortest LFSR contains 1 letter that represents 3 bits.

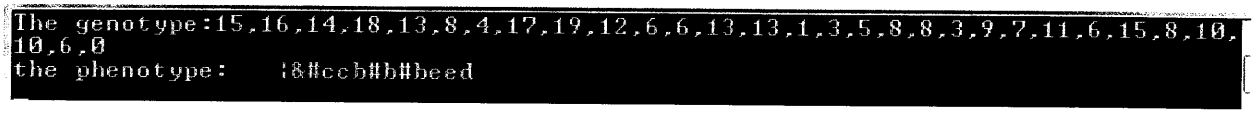


Figure (4.3) Example of GADS class 3

Thus, based on our experimental work, GADS class3 has been used in the proposed method.

4.2 Experiment Results

The results presented in table (4.3) are the results of using different population size values. From table (4.3), it is clear that by increasing the population size, better fitness values can be obtained for all chromosome structures, selection methods, and crossover methods. The best value is 35.06 when the population size equal to 350 . This result has been reached in many studies and this proves that GP works better with high

number of population size. Nevertheless, further increase of the population size does not always improve the solution. Moreover, the computational time is increased significantly (Alander, 1992).

The results shown in table (4.4) indicate that the optimal chromosome length is 30 because in that length the overall results were the best. Increasing chromosome length, above 50, does not improve the fitness values.

Table (4.5) presents the results as the average of best fitness values in different runs for different number of generations. Based on these results, the maximum number of generations has no great effect on the performance of the proposed method. However, the maximum number of generations should be more than 20 and not more than 30 to give better results.

Table (4.3) Results of experiments with different population sizes

	Fitness Average								
Population size	**Exp.1**	**Exp.2**	**Exp.3**	**Exp.4**	**Exp.5**	**Exp.6**	**Exp.7**	**Exp.8**	**Exp.9**
15	12.65	4.32	2.74	15.97	17.47	6.32	18.66	12.54	6.45
20	12.82	9.56	4.55	16.23	15.42	10.03	16.68	10.99	9.07
30	17.55	12.85	8.56	17.65	12.23	10.27	24.28	12.99	9.57
50	22.57	14.62	10.22	25.73	11.72	15.97	22.84	15.98	13.83
100	27.72	16.16	12.05	28.74	15.28	21.28	29.94	20.94	16.81
150	29.42	16.31	13.99	30.67	19.69	22.52	30.26	23.52	19.37
200	30.71	20.23	17.05	33.55	23.58	21.08	31.87	23.23	17.91
250	32.38	22.93	17.35	34.65	23.19	21.69	31.99	24.07	22.36
300	33.19	22.89	18.71	34.35	24.34	24.62	33.81	27.03	21.66
350	33.65	23.43	22.03	35.06	24.76	22.44	34.31	28.04	23.71

Table (4.4) Results of experiments with different chromosome lengths

	Fitness Average								
Chromosome Length	**Exp.1**	**Exp.2**	**Exp.3**	**Exp.4**	**Exp.5**	**Exp.6**	**Exp.7**	**Exp.8**	**Exp.9**
10	5.4	0.4	0.4	5.4	0.4	0.4	5.4	0.4	0.4
15	25.22	10.39	6.18	32.35	7.51	9.45	32.27	12.35	8.14
20	30.56	10.82	8.26	34.27	11.85	22.07	34.1	13.16	16.44
30	33.65	23.43	22.03	35.06	24.76	24.62	34.31	28.04	23.71
40	33.49	19.89	17.62	33.68	21.82	22.32	33.47	23.35	22.68
50	33.07	16.85	16.74	36.32	21.15	20.77	30.81	25.62	21.75
60	28.45	15.43	17.82	35.62	20.67	19.99	31.47	20.93	20.54

Table (4.5) Results of experiments with different number of generations

	Fitness Average								
Max Number of Generations	**Exp.1**	**Exp.2**	**Exp.3**	**Exp.4**	**Exp.5**	**Exp.6**	**Exp.7**	**Exp.8**	**Exp.9**
5	31.04	26.29	17.05	30.32	27.13	19.15	38.13	27.15	24.11
10	32.95	27.17	20.59	31.87	28.21	27.04	36.06	23.65	22.15
20	32.97	23.47	19.33	33.75	26.74	21.64	33.25	24.06	22.52
30	33.65	23.43	22.03	35.06	24.76	24.62	34.31	28.04	23.71
40	32.88	20.41	18.98	34.13	21.26	20.17	34.38	23.57	22.78
50	32.87	19.64	18.74	34.1	21.24	19.39	32.07	25.71	21.37
60	32.36	21.53	15.85	34.53	22.02	21.58	32.97	24.56	23.19
70	31.38	21.44	18.44	32.32	24.36	23.17	32.79	24.21	23.33
80	33.74	20.89	17.89	31.88	24.84	20.74	34.7	25.25	22.69

From the results of table (4.6), we can see that in general the best mutation rate is 0.05 which give the best results. The mutation rate (probability) should be set low because if it is high, the search will turn into random search, and the solution may be totally different from the previous solution. That will make us lose the benefits of using GP (Ochoa, Harvey, & Buxton, 1999).

Finally, table (4.7) shows that the top crossover rate is 1. The experimental results confirm that when we set the crossover rate (probability) to high values, we obtain better results. The higher the crossover rate, the quicker the exploitation proceeds and the more quickly new chromosomes are added to the population. On other hand, a low crossover rate may stagnate the search due to loss of exploration power (Spears & Jong, 1990).

Table (4.6) Results of experiments with different mutation rates

	Fitness Average								
Mutation Rate	**Exp.1**	**Exp.2**	**Exp.3**	**Exp.4**	**Exp.5**	**Exp.6**	**Exp.7**	**Exp.8**	**Exp.9**
0.02	32.96	21.55	16.09	35.61	23.82	22.39	32.74	25.73	23.19
0.03	33.45	20.95	17.09	34.07	23.19	21.28	33.36	23.66	23.16
0.04	34.21	22.34	19.83	34.9	23.77	21.67	31.65	24.93	21.73
0.05	33.65	23.43	22.03	35.06	24.76	24.62	34.31	28.04	23.71
0.06	31.39	21.04	14.89	33.2	23.52	20.42	33.52	26.69	22.58
0.07	32.12	20.09	15.29	34.37	19.79	21.38	32.18	25.82	22.03
0.08	31.03	20.23	19.14	32.62	23.68	19.17	33.26	24.93	22.40
0.09	30.79	21.19	18.62	32.96	21.49	20.98	30.69	24.79	22.95
0.1	30.92	22.38	16.78	32.44	21.91	19.61	32.70	23.88	18.55

Table (4.7) Results of experiments with different crossover rates

	Fitness Average								
Crossover Rate	**Exp.1**	**Exp.2**	**Exp.3**	**Exp.4**	**Exp.5**	**Exp.6**	**Exp.7**	**Exp.8**	**Exp.9**
0.1	0.99	0.57	0.82	0.99	1.96	1.72	0.99	1.38	1.81
0.3	1.04	4.76	2.83	2.96	4.84	1.79	19.58	7.53	12.39
0. 5	6.29	12.38	4.47	8.42	4.47	5.54	28.95	9.27	18.18
0. 7	17.13	11.82	18.96	25.72	9.36	7.48	30.96	15.41	20.16
0. 9	28.82	17.99	17.12	32.07	9.94	11.48	30.91	17.48	18.28
1	33.65	23.43	22.03	35.06	24.76	22.44	34.31	28.04	23.71

In addition to the results above, FR block size of chromosomes of structure 2 have been studied since, in the experiments with chromosome structure 2, we need to identify the length of FR block each time we run the program. The results of table (4.8) show that the best FR block size is 10.

Table (4.8) Results of experiments with different FR block lengths

FR Block Size	**Exp.2 Fitness value**	**Exp.5 Fitness value**	**Exp.8 Fitness value**
2	22.48	21.81	28.24
3	23.5	22.41	26.89
5	24.04	23.8	25.99
6	20.22	21.51	24.99
7	19.93	22.1	25.91
8	22.82	20.31	24.78
9	21.68	21.75	25.34

10	23.43	24.76	28.04
15	6.82	13.75	14.69

Furthermore, FR and TR block sizes have been studied in the experiments in which chromosome structure 3 is used. Table (4.9) presents the results of experiment 3 with different values of FR and TR block sizes. The worst fitness value average is 10.99 when FR and TR block lengths are 15 because there is no space for leave rules (LR) in the chromosomes since the chromosome length is equal to 30. The best fitness value is 25.12 when FR block length is 3 and TR block length is 6.

Table (4.9) Results of experiment 3 with different lengths of FR and TR

2	20.22	19.98	20.87	19.61	19.19	22.14	18.81	17.88	21.14
3	21.02	20.57	21.52	25.12	24.49	18.47	23.75	20.68	20.55
5	20.13	23.65	21.42	20.88	22.59	21.31	20.26	20.85	18.71
6	22.24	20.78	22.06	21.99	20.89	20.91	19.11	17.42	16.33
7	20.54	22.33	20.62	21.52	19.70	16.46	19.69	20.75	19.49
8	16.69	23.03	22.04	22.91	21.78	20.73	21.08	14.49	21.03
9	17.50	21.07	21.65	23.15	20.71	18.85	23.19	17.55	22.34
10	23.11	20.12	22.04	21.48	23.18	19.08	18.24	22.03	24.05
15	18.09	20.93	19.36	21.47	18.95	20.71	15.97	16.24	10.99
FR TR	**2**	**3**	**5**	**6**	**7**	**8**	**9**	**10**	**15**

Table (4.10) shows that the worst fitness average value is 11.96 when FR and TR block lengths are 15. The best fitness average value is 30.72 when FR block length is 15 and TR block length is 6.

Table (4.11) displays the results for different lengths of FR block and TR block. The worst fitness average value is 15.94 when FR and TR block lengths are 15. The best fitness values are 28.50 and 28.52 for FR block length 15 and TR block length 5, and when FR block length is equal to TR block length and is equal to 6, respectively.

Table (4.10) Results of experiment 6 with different lengths of FR block and TR block

2	20.45	17.89	19.75	21.2	19.41	18.97	19.57	18.21	19.54
3	20.03	25.38	23.83	25.52	20.03	22.85	23.01	24.04	22.15
5	20.14	25.79	17.89	24.67	21.63	23.47	21.41	24.62	19.24
6	21.56	23.92	21.51	19.38	23.41	19.84	25.07	20.93	21.61
7	21.72	23.81	27.14	25.13	22.58	22.72	18.17	21.66	20.45
8	21.64	23.24	23.78	22.04	22.91	19.56	24.35	19.38	21.81
9	21.32	21.82	24.41	24.45	27.47	21.99	23.08	19.68	25.54
10	24.42	25.24	26.34	26.84	22.99	21.11	23.42	21.49	23.97
15	19.04	25.79	24.55	30.72	24.81	15.63	22.21	20.21	11.96
FR TR	**2**	**3**	**5**	**6**	**7**	**8**	**9**	**10**	**15**

4.3 Comparison Between Different Chromosome Structures

In this section, different chromosome structures had been realized and tested. Then, they will be compared to each other to find the best chromosome structure that gives the best results in general.

First I will compare the results of experiments number 1, 2 and 3 when we change the population size and chromosome length, as shown in figures (4.4) and (4.5). I choose those experiments because I want to set the crossover type to " Single point crossover" and selection method to "Tournament Selection" and changing only the chromosome structure, to see the effect of changing it in the result.

Table (4.11) Experiment 9 results with different lengths of FR block and TR block

2	22.64	27.91	24.70	24.53	26.64	24.77	24.51	24.34	23.06
3	26.20	25.59	24.47	25.52	27.81	25.65	24.44	23.85	22.97
5	23.25	23.39	25.74	26.62	27.16	23.75	24.49	23.12	21.26
6	24.27	27.10	25.51	28.52	25.51	26.04	28.40	23.86	22.47
7	24.96	24.91	23.98	27.66	27.13	21.28	22.79	23.11	19.13
8	24.27	26.26	25.34	28.25	25.41	25.25	23.78	21.66	24.01
9	21.93	24.99	27.70	27.19	25.86	23.82	22.86	19.84	26.03
10	26.82	26.20	26.33	26.85	25.13	25.31	24.19	23.71	27.09
15	22.09	28.09	28.50	28.02	26.40	21.50	21.93	19.84	15.94
FR TR	**2**	**3**	**5**	**6**	**7**	**8**	**9**	**10**	**15**

Figure (4.4) shows the fitness value average when we change the population size at each generation. As the graphs shows in this figure, that the chromosome structure 1 give better result than the other structures. The best value for structure 1 is when population size equal to 350 and the fitness value equal to 33.65. The best values for other structures also at when the population size is 350 but the fitness values

are less than the first structure. The best value for chromosome structure 2 is 23.43 and for chromosome structure 3 is 22.03.

Also, the structure 1 take the lead in figure (4.5). the highest fitness value in structure1 is 33.65 when the chromosome length is 30. The highest value in the other structures are less than 25. The different between structure 2 and 3 is slightly different.

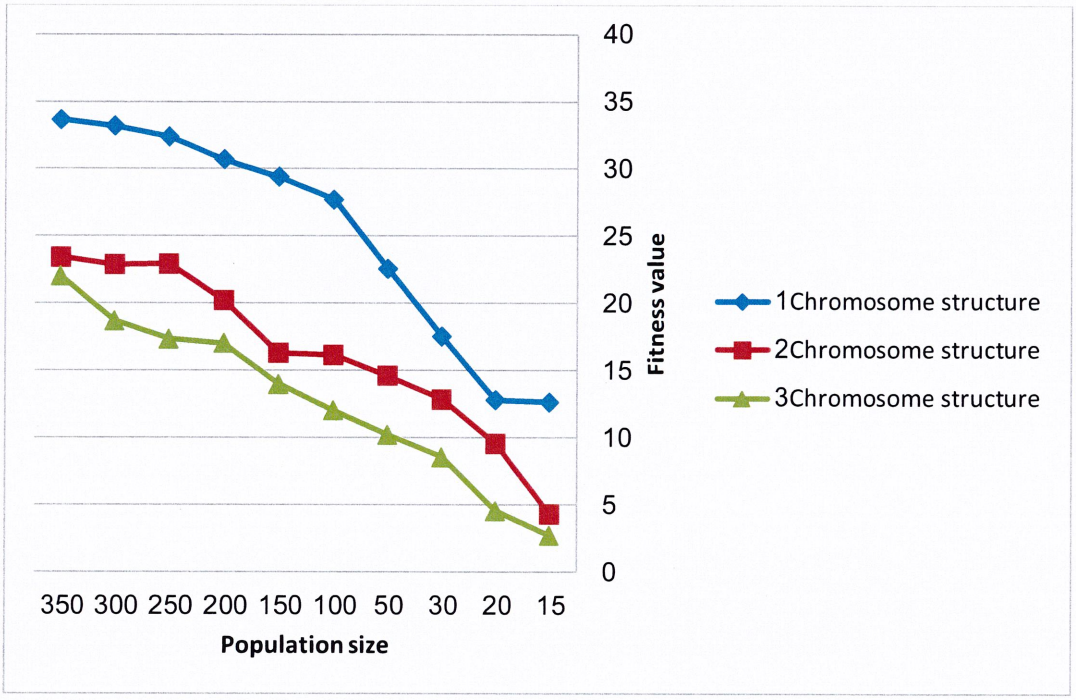


Figure (4.4) Results of comparing different chromosome structures with different population sizes

Figures (4.6) and (4.7) presents the results of the comparison with crossover type Multi point crossover and the selection method is Roulette Wheel Selection. The top value can be found in chromosome structure 1 with this fitness value 35.06. At the second place, Chromosome structure 2 and 3 are keeping compete with each other to occupy this place. Again, chromosome structure 1 is the best.

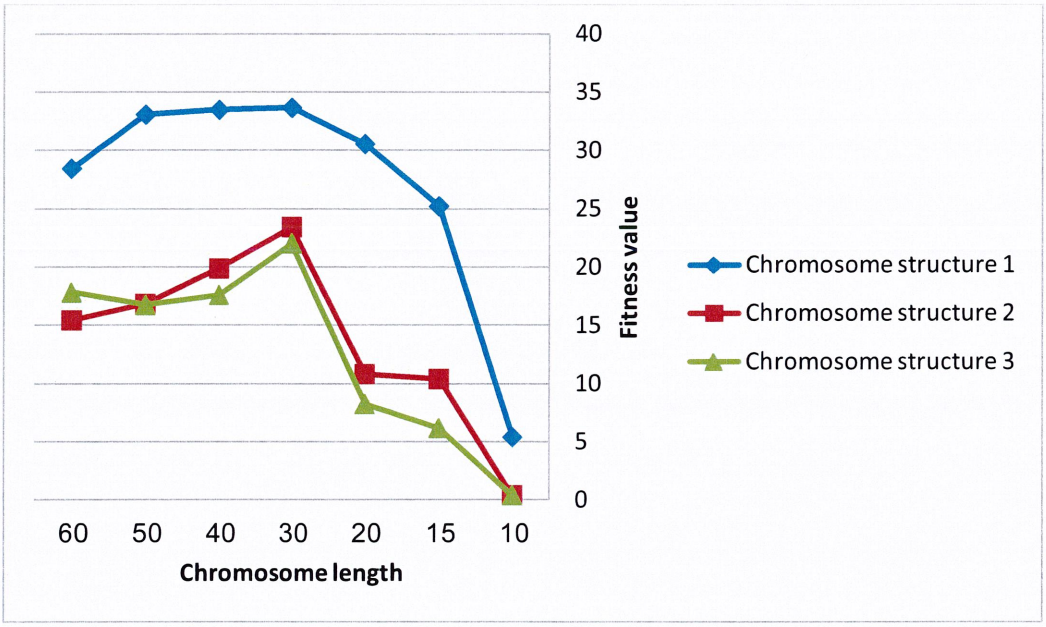


Figure (4.5) Results of comparing different chromosome structures with different chromosome lengths

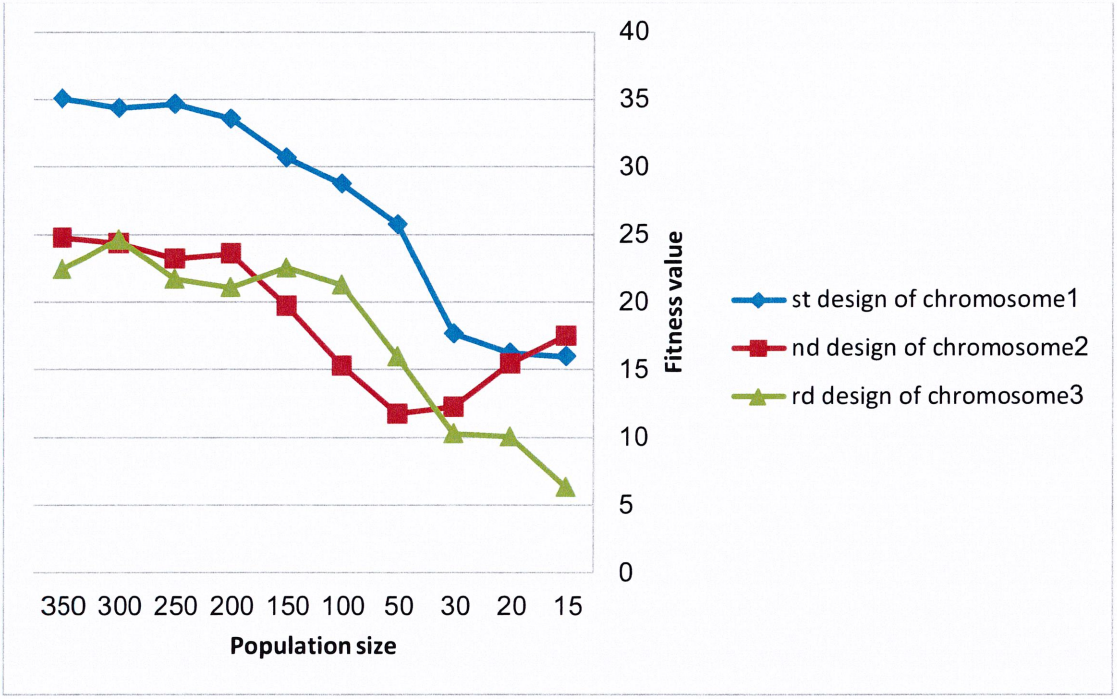


Figure (4.6) Results of comparing different chromosome structures with different population sizes

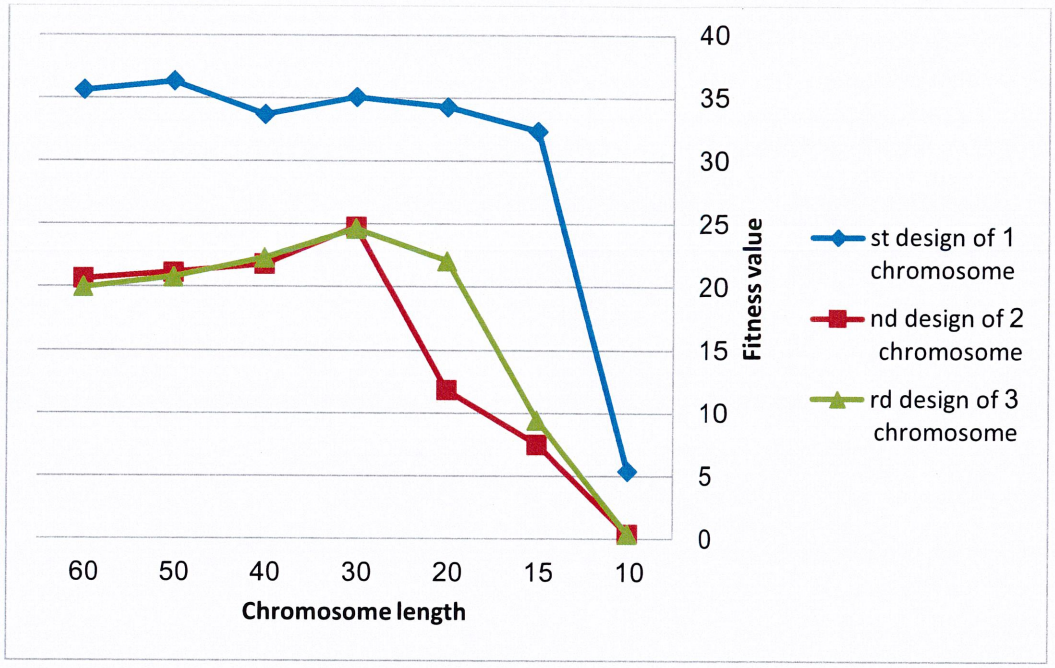


Figure (4.7) Results of comparing different chromosome structures with different chromosome lengths

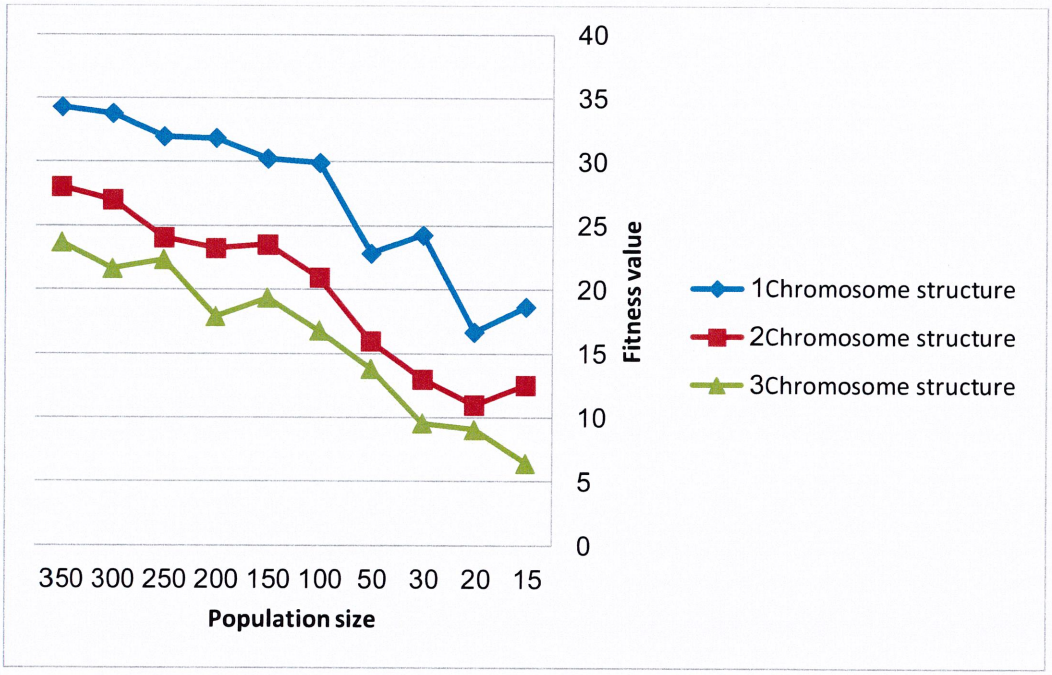


Figure (4.8) Results of comparing different chromosome structures with different population sizes

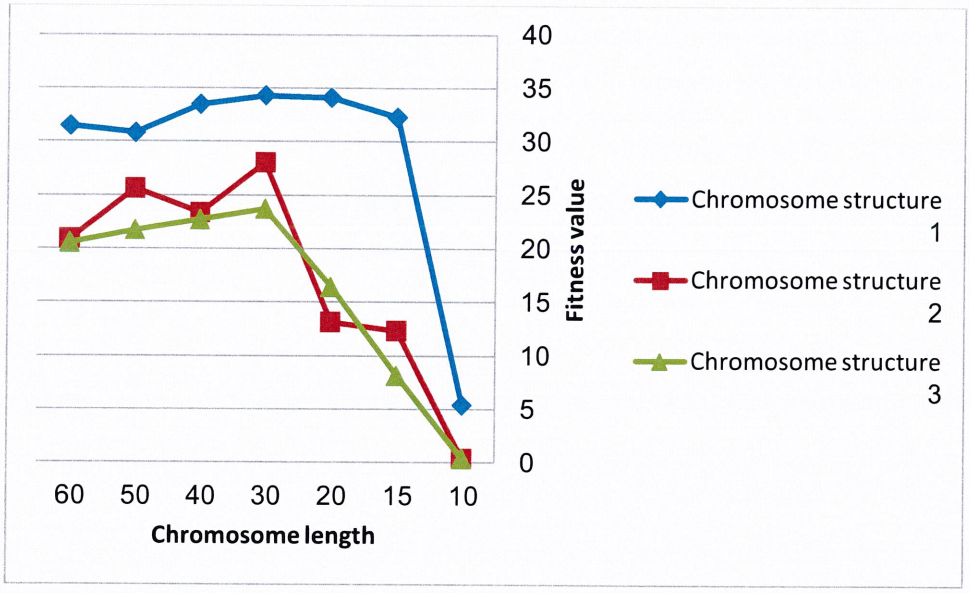


Figure (4.9) Results of comparing different chromosome structures with different chromosome lengths

The last comparison in this section will be between experiments number 7, 8 and 9. The default crossover type will be "Multi point crossover" and the selection method will be " Roulette Wheel Selection ", as shown in figures (4.8) and (4.9). Again, chromosome structure 1 proved that it is the best design. There is a huge difference between it and the other structures design. One reason that this structure gives the best result is that the other structures depend on the lengths of FR block and TR block. A more rigorous exploration of those parameters may produce better results when using chromosome structures 2 and 3.

4.4 Comparison Between Different Crossover Types

The other component that had been study its effect in the proposed method is the crossover type, and which type will be more suitable for designing the stream cipher.

In the following two figures (4.10) and (4.11), the comparison will be carried out when the chromosome structure is "1" and selection method is "Tournament Selection":

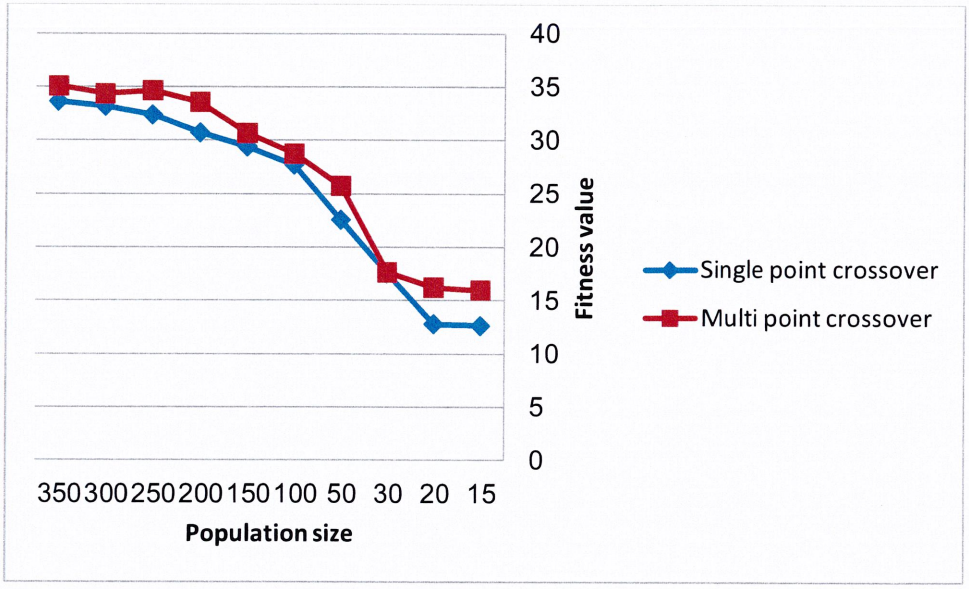


Figure (4.10) Results of comparing different crossover types with different population size

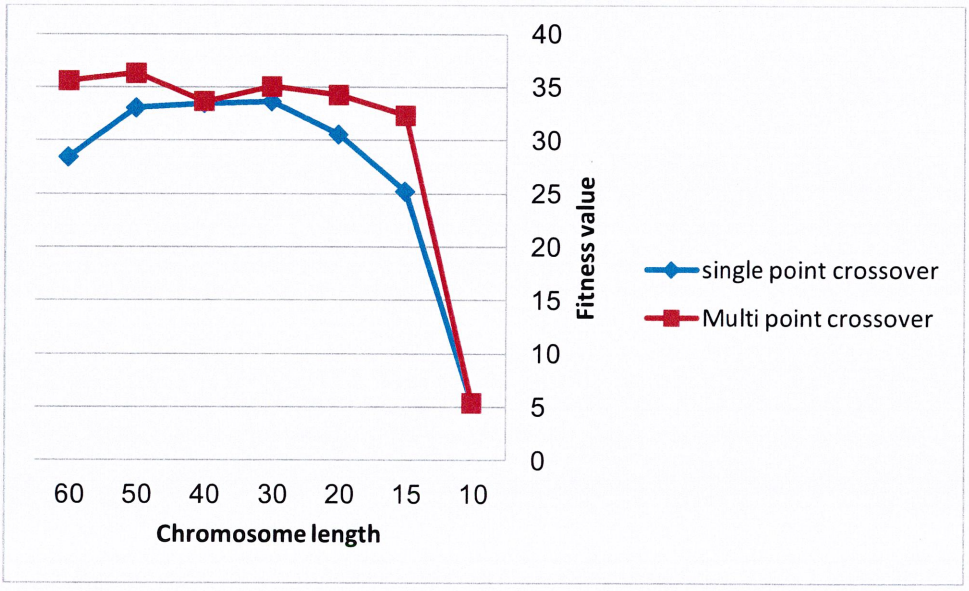


Figure (4.11) Results of comparing different crossover types with different chromosome length

The results in figures (4.10) and (4.11) show that using multi point crossover type gives better results than using the single point type; even if the difference between them is not huge. The best result in figure (4.10) is 35.06 when the population size equal to 350 and using multi point crossover. In figure (4.11), the best fitness value can be found is 36.32 when chromosome length is 50.

The chromosome structure in the second comparison will be "2" and the selection method is "Tournament Selection". Figures (4.12) and (4.13) show the results which indicate slightly that multi point crossover is better than single point crossover. The best fitness value in multi point crossover is 24.76 and in single point is 23.43.

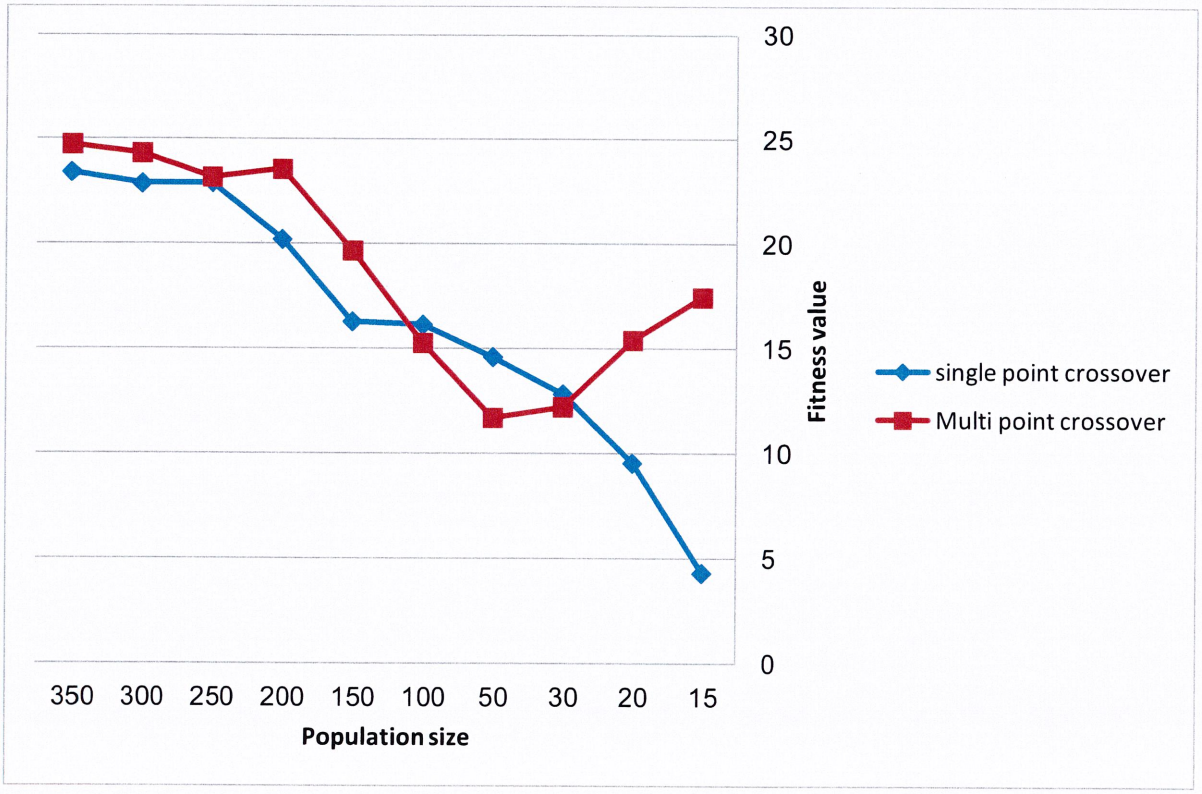


Figure (4.12) Result of comparing different crossover types with different population size

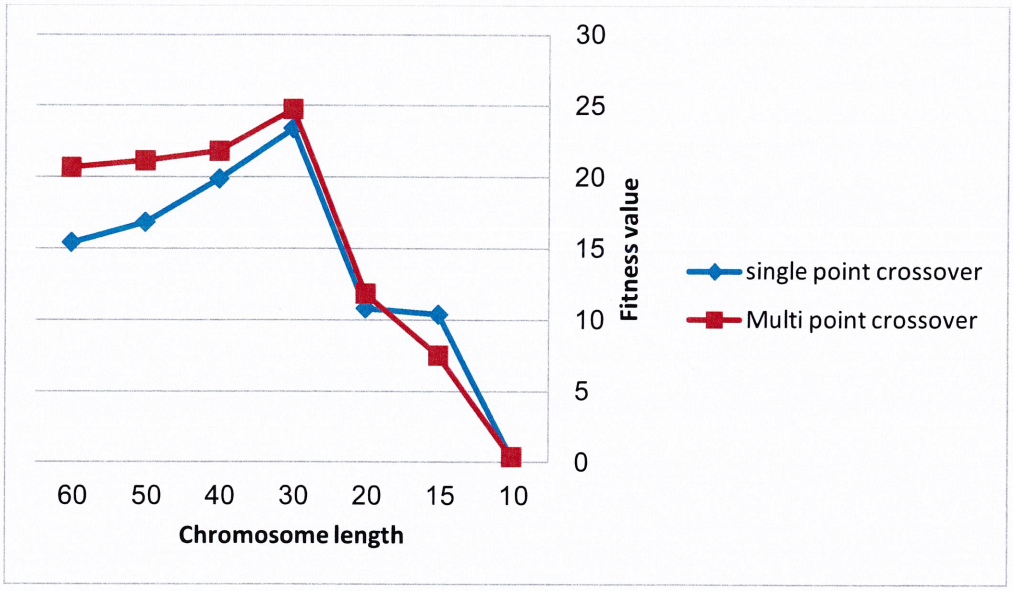


Figure (4.13) Result of comparing different crossover types with different chromosome length

The last comparison of changing the crossover type will be for chromosome structure 3 and Tournament Selection. The results of this comparison are illustrated below in figures (4.14) and (4.15). The difference between the crossover types goes up as the number of variables increases. The multi point crossover performance is higher than that of the single point.

The experimental results above confirm that the multi point crossover yields a better result than the single point crossover. This can be explained by the disruptive nature of multi-point crossover, which appears to encourage the exploration of the search space, rather than favors the convergence to highly fit individuals early in the search. In this way it makes the search more robust (Spears & Jong, 1990). For this reason, the multi point crossover type only implement with Roulette Wheel Selection.

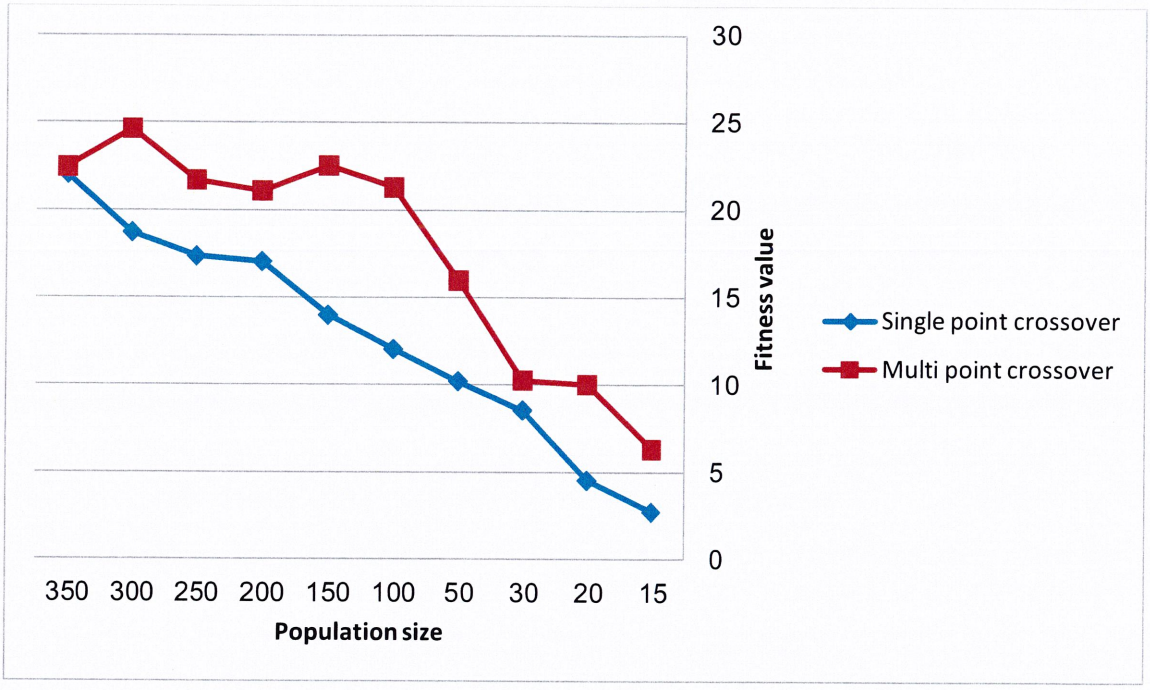


Figure (4.14) Result of comparing different crossover types with different population size

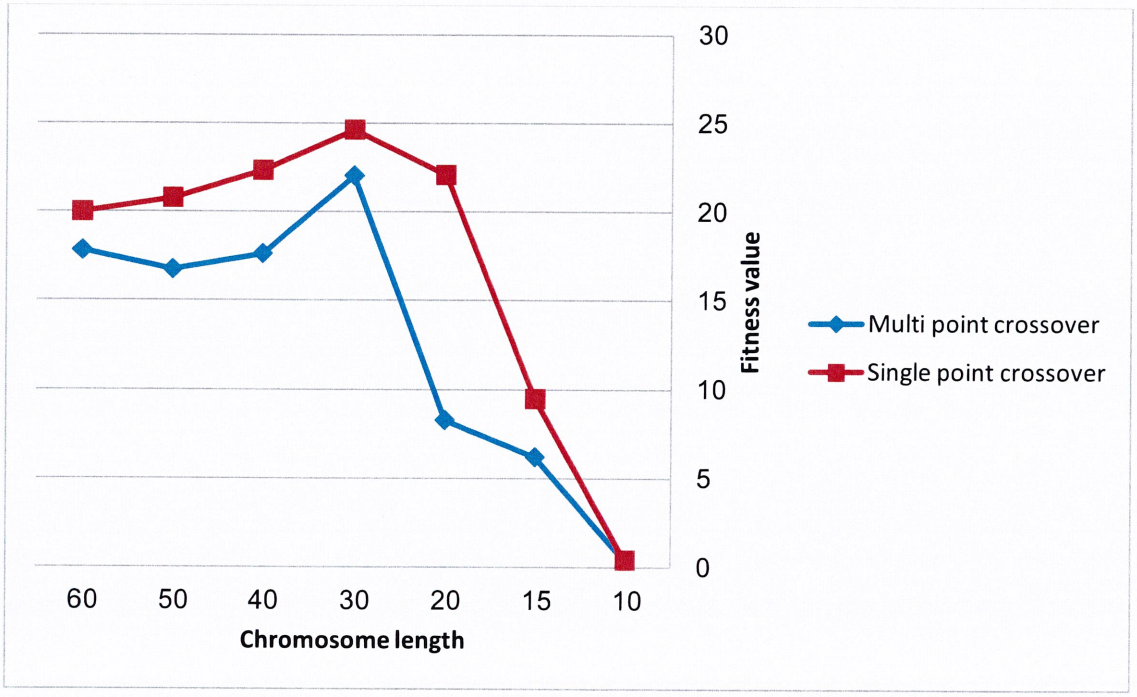


Figure (4.15) Result of comparing different crossover types with different chromosome length

4.5 Comparison Between Different Selection Methods

One of the important components used for creating keystream in stream cipher using GP is selection methods. There are many selection methods available to integrate with this method. Here in this thesis two methods have been chosen (as mentioned earlier in section 3.5.1). They are the Tournament Selection and Roulette Wheel Selection.

In this section, these methods will be tested so as to decide which of them gives a higher fitness value average in different designs.

First, figures (4.16) and (4.17) present the results obtained in applying chromosome structure 1 with multi point crossover. Figure (4.16) shows that the tournament selection gives a better result than the roulette wheel selection. On the other hand, figure (4.17) shows that until the chromosome length is equal to 40, there is near-equality in the results between the two selection methods. However; at the end of the tournament selection results improve better than the roulette wheel selection results when chromosome lengths are 50 and 60.

In figures (4.18) and (4.19) the results of different selection methods are presented with the following parameters: chromosome structure 2 and multi point crossover. In this comparison, the roulette wheel selection takes the lead. It gives higher results than tournament selection. The highest fitness value in roulette wheel is 28.04; whereas the highest result in tournament is 24.76.

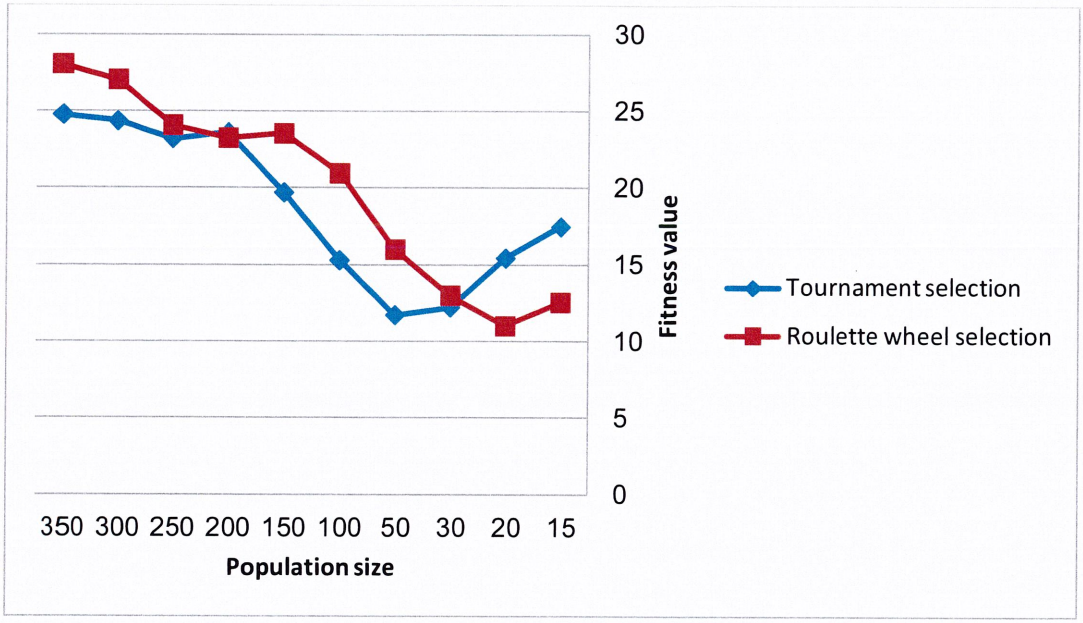


Figure (4.16) Result of comparing different selection methods with different population size

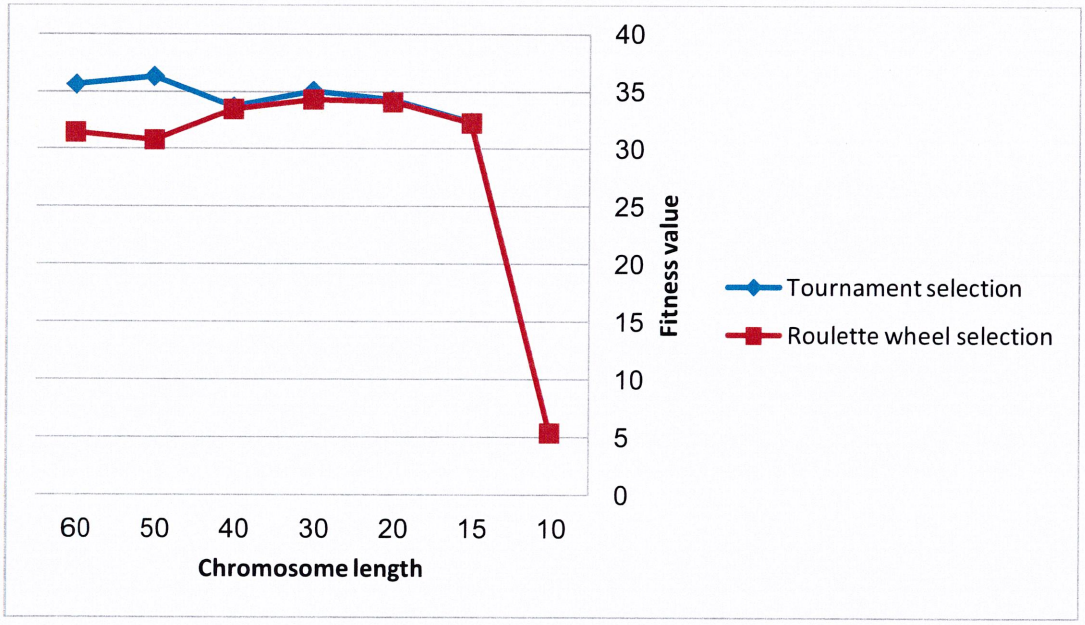


Figure (4.17) Result of comparing different selection methods with different chromosome length

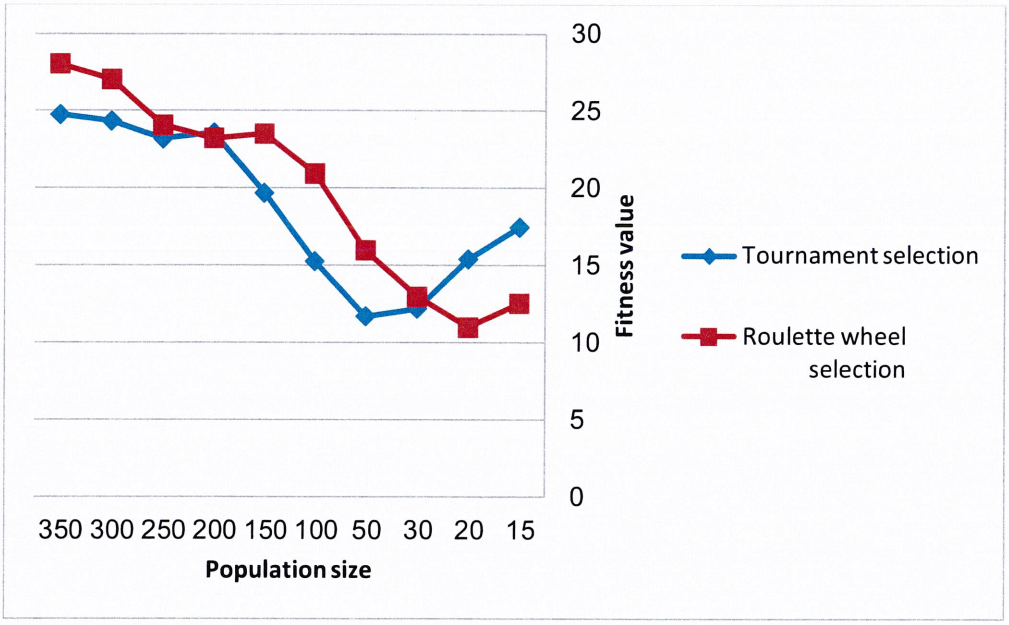


Figure (4.18) Result of comparing different selection methods with different population sizes

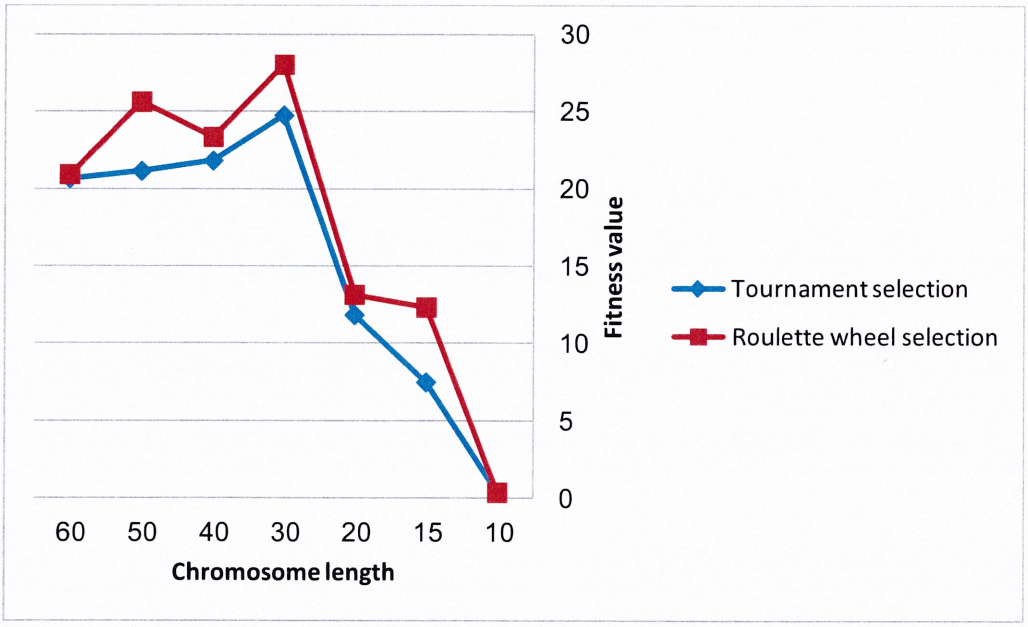


Figure (4.19) Result of comparing different selection methods with different chromosome length

In the last comparison in this section the fix components are: chromosome structure 3 and multi point crossover. Figures (4.20) and (4.21) show that there is a conflict between the two selection methods in

this comparison. At the beginning, tournament selection gives higher results. But when the population size or the chromosome length increases, the roulette wheel selection yields almost better results.

Qualitative analysis of the selection strategies must be done to adopt the best strategy since the selection performance is highly dependent on the different criteria at the time of selection of chromosomes. These criteria could be anything like type of population, type of representation method used, or population size (Noraini, & John, 2011).

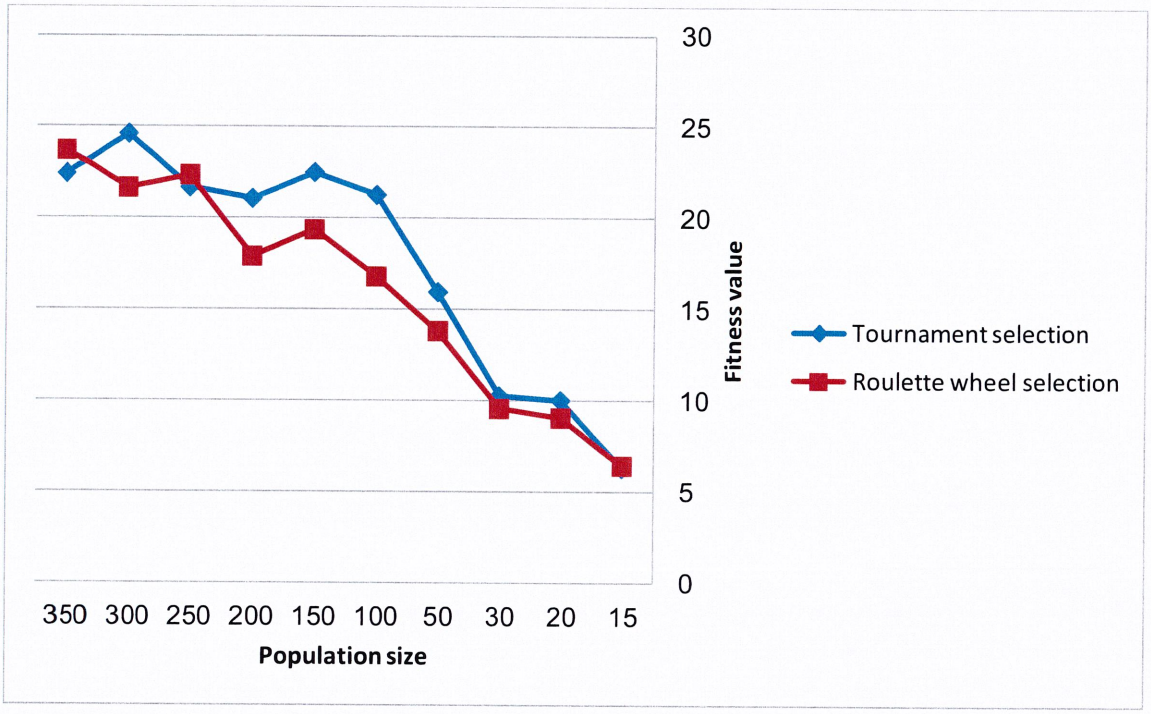


Figure (4.20) Result comparing different selection methods with different population sizes

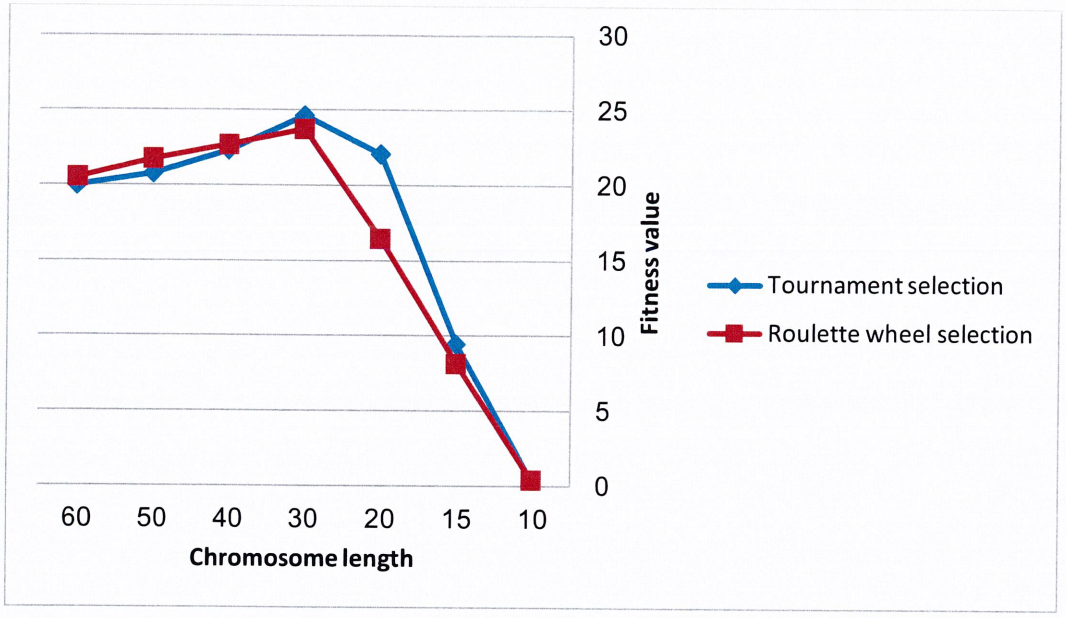


Figure (4.21) Result of comparing different selection methods with different chromosome lengths

4.6 SA Parameters

In this section we will study two important parameters in SA, those parameters are initial temperature and annealing schedule.

As Andresenz and Nouraniy (1998) said "Lowering the temperature by itself is not a sufficient recipe for finding the ground state. In addition to this the system should not be allowed to become significantly out of equilibrium with its environment during the annealing. Otherwise it does not reach the global minimum energy state but will freeze into one of the great many higher energy local minima". That was approved in this thesis, after run the experiment number 4 several times with different temperatures. This experiment had been chosen because it gives the best result as shown in section 4.2. Table (4.12) shows that increasing the initial temperature gives us better result because the initial temperature should be high enough to `melt' the system completely to give the best result.

Table (4.12) The result of different SA initial temperature

SA initial Temperature	Fitness value
50	30.03
100	32.04
150	32.82
200	34.31
250	35.06

The annealing schedule decides the degree of uphill movement permitted during the search in SA. For this reason annealing schedule had been chosen as the second SA parameter to be tested in this thesis. The results presented in table (4.13) shows the best schedule that tested and used in the proposed method in this thesis is 0.95.

Table (4.13) The result of different annealing schedule

Annealing schedule	Fitness value
0.75	33.58
0.80	33.27
0.85	33.45
0.90	33.77
0.95	35.06

Regarding the above result, we will use for all the other experiments the initial temperature equal to 250 and the annealing schedule set to 0.95.

4.7 Comparison Between the Proposed GADS Method With SA and Without Using SA

One objective of this thesis is to study the effectiveness of GP combined with SA in designing stream cipher systems of high level of security.

Table (4.14) Comparison between GADS method with SA and GADS only

Max Number of Generations	Fitness average	
	GADS with SA	GADS only
5	30.32	27.08
10	31.87	29.33
20	33.75	33.10
30	35.06	34.36
40	34.13	32.84
50	34.1	33.51
60	34.53	30.53
70	32.32	31.76
80	31.88	29.61

According to table (4.14), using SA with GP yields better results. That is because of the advantages of SA, it can help to avoid the problem of getting stuck in a local minimum.

4.8 Comparison Between the proposed GADS Method and the Previous SAP Algorithm

Finally, we need to compare between the proposed GADS combine with SA method of this chapter and the previous SAP algorithm, proposed by Awad (2011a, 2011b), in which LISP expression is used as a representation method for designing stream ciphers. Table (4.15) presents the best fitness values obtained in the two methods, for different number of generations. As we can see, the proposed method did not improve the results of Awad's previous method used for the automated design of stream ciphers.

Table (4.15) A comparison between GADS and Awad's previous method

MAX Number of Generations	Fitness value	
	GADS	SAP
30	35.06	34.6199
50	34.1	35.7214
70	32.32	36.3741
90	35.65	35.3944

One important feature that show if one system is better than the other is the execution time. Therefore, the execution time of GADS and Awad's previous method are compared. The results in table (4.16) proved that the proposed method in this thesis that uses GADS need less time to run than previous SAP algorithm. That is because of the advantages of GADS (as mention in section 2.3.1.2), it uses the velocity of GA engine. However, it is clear that the GADS give good results, which proves that GADS can be used to design an efficient and effective stream cipher.

Table (4.16) Execution time comparison between GADS and Awad's previous method

MAX Number of Generations	Execution Time in Seconds	
	GADS	SAP
30	35	41
50	44	67
70	69	89
90	86	111

4.9 Result Summary

In this chapter, several experiments have been contacted for designing stream cipher using GADS and SA. The method was tested on three different components and several values of different parameters. By observing the results of the nine experiments we can conclude the following:

1. Increasing the population size yields better results. The best population size that found in this thesis is 350.
2. The best mutation rate found is 0.05.
3. The best crossover rate found is 1, which is the highest rate.
4. The maximum number of generations has no effect on the proposed method performance. In general, it should not be less than 20 and not greater than 30.
5. The best chromosome length is 30 integers.

6. The findings of all the experiment proved that the best chromosome structure is chromosome structure 1, in which the rule numbers are mixed in the chromosomes.

7. The results also showed that using multi-point crossover gives better results compared to single-point crossover.

8. Regarding the selection method, the results above showed that there is conflict between the two selection methods in this comparison which are tournament selection method and roulette wheel selection method.

9. In term of fitness values, the proposed method of this chapter did not improve the results of Awad's previous method used for the automated design of stream ciphers but execution time for it is faster than the Awad's method.

10. Increasing the initial temperature and annealing schedule in SA gives us better results. Since the initial temperature should be high enough to `melt' the system completely and should be reduced towards its `freezing point' as the search progresses (Andresenz, & Nouraniy, 1998).

11. Using GADS combine with SA give better result in term of fitness values than using GADS alone in designing stream cipher.

Summary of the results is shown in table (4.17).

The following are a number of keystream generators evolved by the proposed automated design method. In the following figures, we can see the genotypes, phenotypes, their fitness values, and the Binary keystream generated by each generator (phenotype). As shown in figures (4.22), (4.23), (4.24), (4.25), (4.26), (4.27), the keystreams have long period length (since that there is no pattern in the whole 200 Bits which is the keystream length) and are random (pass randomness tests). Accordingly,

it is hard to recover the cipher's key from the keystream because it is unpredictable.

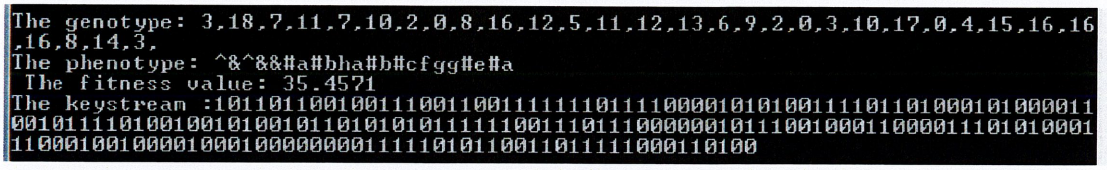


Figure (4.22) Example number 1 of keystream generator

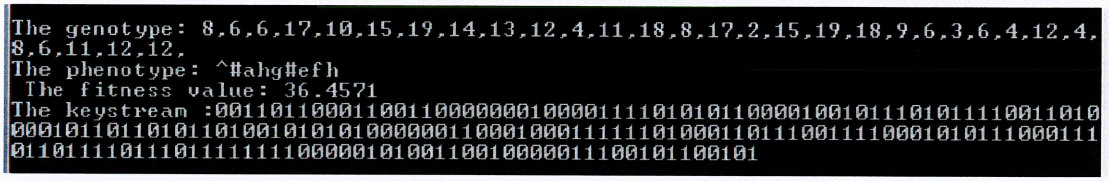


Figure (4.23) Example number 2 of keystream generator

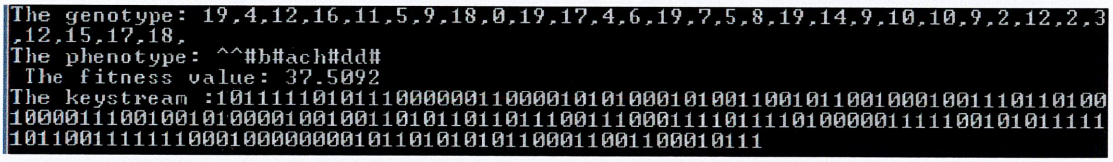


Figure (4.24) Example number 3 of keystream generator

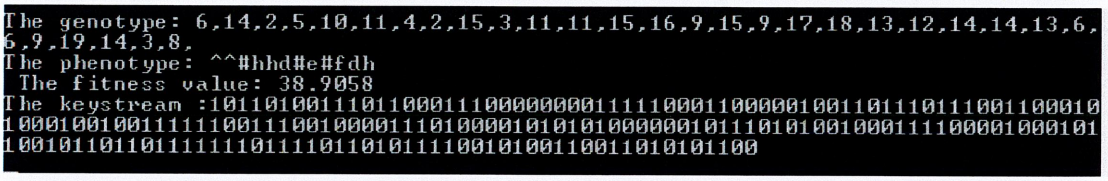


Figure (4.25) Example number 4 of keystream generator

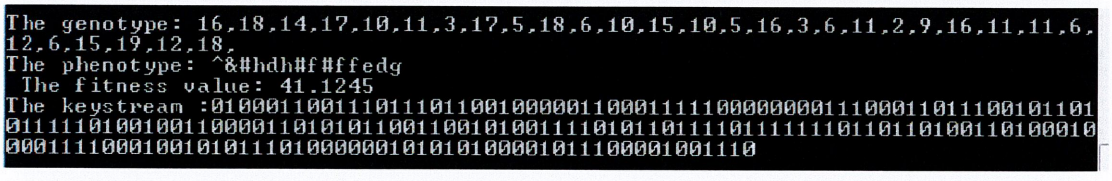


Figure (4.26) Example number 5 of keystream generator

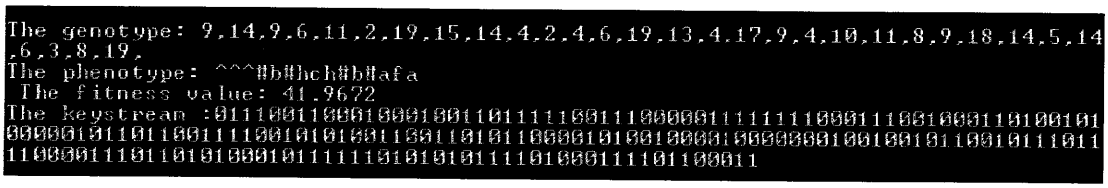


Figure (4.27) Example number 6 of keystream generator

Table (4.17) Summary of results

The parameters or components	Best value or type
Population size	350
Chromosome length	30
Number of generations	30
Mutation rate	0.05
Crossover rate	1
Chromosome structure	Structure 1
Crossover type	Multi point crossover
Selection method	Both (tournament & roulette wheel selection)
SA initial temperature	250
Annealing schedule	0.95

Chapter 5 Conclusion and Future Work

5.1 Conclusion

5.2 Future Work

Chapter 5 Conclusion and Future Work

5.1 Conclusion

This thesis presented an effective automated stream ciphers design using GP. The proposed method is a modified version of Awad (2011a, 2011b) SAP algorithm, which has been modified in this work by using GADS. In GADS, the population chromosomes (genotypes) are represented as sequences of integers rather than LISP expressions. Thus, GA operations, such as crossover and mutation, can be applied easily and efficiently. Three chromosomes structure have been proposed and analyzed. Different GA operations and parameters have been examined, which are, chromosome structure, chromosome length, selection strategy, population size, crossover type, maximum number of generations, crossover rate and mutation rate. It has been found that the best results obtained when using chromosome structure 1, tournament and roulette wheel selection, 30 integers as chromosome length, 0.05 as mutation rate, 30 as maximum number of generations, above 50 individuals as population size, and multi-point crossover with crossover rate equal to 1.

Furthermore, it has been shown that GADS is as effective as GP using LISP, with the advantage of using GA efficient engine.

5.2 Future Work

1. More investigation is required about the chromosome structures and FR, TR, and LR block sizes. How can we modify it to give us better result.

2. Examining more genetic operations, such as, replacement, and selection.
3. Examining the dynamic setting of genetic operation rates.
4. Examining other CI tools.

Those and other questions and extensions to this work are still unanswered. The work performed, in within this thesis, dose however leave a number of questions open, that open for us a number of possible directions for future work to potentially further improve the design of the stream cipher or any other encryption methods.

References

Abdulsalam, A. A. (2011). Keystream Generator Based On Simulated Annealing. *Journal of Applied Computer Science & Mathematics, 10,* 48-53.

Alander, J. T. (1992). On optimal population size of genetic algorithms. *Proceedings of the IEEE Computer Systems and Software Engineering,* 65-69.

Ali, I. K. (2014). Design Strong Boolean Functions using Memetic Algorithm. Retrieved July, 12, 2014, from http://www.iasj.net/iasj?func=fulltext&aId=53139

Andresenz, B., & Nouraniy, Y. (1998). A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General*, 31, 8373-8385

Awad, W. S. (2011a). On the application of evolutionary computation techniques in designing stream cipher systems. *International Journal of Computational Intelligence Systems*, 4(5), 921-928.

Awad, W. S. (2011b). Designing stream cipher systems using genetic programming. *Learning and optimization (LION5),* LNCS, 6683, 308-320.

Awad, W. S., (2013). The effect of mutation operation on GP- based stream ciphers design algorithm. *5th International Conference on Agents and Artificial Intelligence,* 15-18 Feb, Spain, 455-450.

Al Jadaan, O., Lakishmi, R., & Rao, C. R. (2005). Improved selection operator for genetic algorithm. *Journal of Theoretical and Applied Information Technology*, 4(4), 269-277.

Alander, J. T. (1992). On optimal population size of genetic algorithms. *Proceedings of the IEEE Computer Systems and Software Engineering,* 4-8 May, The Hague, Netherlands, 65-69.

Bastaki, Y., & Shaker, W. (2003). New Developments in Genetic Algorithms for Developing Software. *Studies in Informatics and Control,* 12 (4), 277-283.

Beasley, J. E., & Chu, P. C. (1996). A genetic Algorithm for the set covering problem. *European Journal of Operational Research*, 94(2), 392-404.

Beker, H., & Piper, F. (1982). *Cipher systems: The protection of communications*. United Kingdom: Northwood Publications.

Bertsimas, D., & Tsitsiklis, J. (1993). Simulated Annealing. *Statistical Science, 8(1),* 10-15.

Burke, J., McDonald, J., & Austin, T. (2000). Architectural support for fast symmetric-key cryptography. *ACM New York, 28 (5),* 178-189.

Caballero-Gil, P. (2000). New upper bounds on the linear complexity. Computers and Mathematics with Applications, *39(3), 31-38.*

Chomsky, N. (1956). Three models for the description of language. *IEEE Transactions on Information Theory*, *2*(3), 113-124.

Damg°ard, I. B. (1990). A Design Principle for Hash Functions, Advances in Cryptology-CRYPTO'89, (LNCS 435, pp. 416-427) : Springer

Eiben, A. E., Hinterding, R., & Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, *3*(2), 124-141.

Estébanez, C., Castro, J. C., Ribagorda, A., & Isasi, P. (2006). Evolving hash functions by means of genetic programming. *Proceedings of the 8th*

annual conference on Genetic and evolutionary computation, *2*, 1861-1862.

Ferreira, C. (2002). *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence*. Portugal: Angra do Heroismo.

Forouzan, B. A. (2008). *Cryptography and network security*. NY: McGRAW-HILL.

Garcia L. J., & Fuster-Sabater, A. (2000). On the linear complexity of the sequences generated by nonlinear filtering. Information Processing Letters, *76, 67-73.*

Goldberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning. NY: Addison-Wesley.

Goresky, M., & Klapper, A. (2006). Pseudonoise sequence based on algebraic feedback shift registers. *IEEE Transactions on Information Theory*, 52(4), 1649-1662.

Gustafson, H., et al. (1994). A computer package for measuring the strength of encryption algorithm. Computer and Security, *13(9), 687-697.*

Hashem, S. H., AL-Hamami, M. A., & AL-Hamami, A. H. (2011). Developing a Block-Cipher-Key Generator Using Philosophy of Data Fusion Technique. *Journal of Emerging Trends in Computing and Information Sciences, 2(5)*, 222-227

Hernandez-Castro,J. C., & Estevez-Tapiador, J. M. (2006). Wheedham: An Automatically Designed Block Cipher by means of Genetic Programming. *IEEE Congress on Evolutionary Computation,* 192-199.

Holland, J. H. (1975). Adaptive in natural and artificial systems. Michigan: University of Michigan.

Al Jadaan, O., Lakishmi, R., & Rao, C. R. (2005). Improved Selection Operator for GA. *Journal of Theoretical and Applied Information Technology*, 269-277.

Julio, C. H., Castro, Pedro, I., Viñuela, Cristóbal, L., & Arco-Calderón, d. (2003). Finding Efficient Nonlinear Functions by Means of Genetic Programming. *Knowledge-Based Intelligent Information and Engineering Systems, Lecture Notes in Computer Science, 2773*, 1192-1198.

Kellegoz, T., Toklu, B., & Wilson, J. (2008). Comparing efficiencies of genetic crossover operators for one machine total weighted tardiness problem. *Applied Mathematics and Computation*, 199(2), 590-598.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science New Series, 220 (4598)*, 671-680.

Koza, J. R. (1992). *Genetic programming*. Cambridge: MIT press.

Koza, J.R. (1994). *GP II: Automatic discovery of reusable programs*. Cambridge: MIT press.

Knuth, D. E. (1998). *The art of computer programming* (2nd ed.). Reading, MA: Addison-Wesley Pub. Co.

Lari, K., & Young, S. J. (1990). The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer speech and language,* 4(1), 34-56.

L'ecuyer, P., & Simard, R. J. (2007). TestU01: A C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software, 33*(4), 22-40.

Massey, J. L. (1976). Shift register sequences and BCH decoding. *IEEE Transaction on Information Theory*, 15(1), 122-127.

Matthew J. S., William M., Guido L., Jessen Y., & Martin A. (2006). *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*: Springer.

McCarthy, J. (1979). *The implementation of Lisp, History of Lisp*. Stanford University.

Menezes, A. J., Van Oorschot, P.C., & Vanstone, S. A. (2001). *Handbook of Applied Cryptography*: CRC Press.

Mitchell, M. (1996). *An Introduction to Genetic Algorithm*. Cambridge: MIT Press.

Millan, W., Clark, A., & Dawson, E. (1998). Heuristic Design of cryptographically strong balanced Boolean Functions. *Advances in Cryptology — EUROCRYPT'98, Lecture Notes in Computer Science, 1403*, 489-499.

Miller, B. L., & Goldberg, D. E. (1995). Genetic Algorithms. Tournament Selection and the Effects of Noise, *Complex Systems, 9*(3), 193- 212.

Naur, P. (1963). Revised report on the algorithmic language ALGOL 60. *Commun. ACM, 6 (1)*, 1-17.

Noraini, M. R., & John. G.(2011). Genetic Algorithm Performance with Different Selection Strategies in Solving TSP. *Proceedings of the World Congress on Engineering, 2,* 1134-1139.

Norman P. , Mike L., & Ky, S. (1997). Evolving caching algorithms in C by genetic programming: MIT Press, 262-267.

Ochoa, G., Harvey, I., & Buxton, H. (1999). On recombination and optimal mutation rates. *In the Proceedings of the Genetic and Evolutionary Computation Conference,* 488-495.

O'Neill, M., & Ryan, C. (2003). *Grammatical evolution: Evolutionary automatic programming in an arbitrary language* (4th ed.). Boston: Kluwer Academic Publishers.

Paar, C., & Pelzl, J. (2009). *Understanding cryptography: A textbook for students and practitioners*. Berlin: Springer.

Paterson, N. R., & Livesey, M. (1996). Distinguishing genotype and phenotype in genetic programming. *Genetic Programming 1996 Conference*, 141-150.

Picek, S., & Golub, M. (2011). On Evolutionary Computation Methods in Cryptography. MIPRO 2011 Proceedings of the 34th International Convention, 34, 1496-1501.

Qi, H. (2008). *Stream Ciphers and Linear Complexity, Thesis of Master degree in Science*, Available from National University of Singapore NUS Dissertations and Theses database.

Rakesh, K., & Jyotishree, (2012). Blending Roulette Wheel Selection & Rank Selection in Genetic Algorithms. International Journal of Machine Learning and Computing, 2 (4).

Rueppel, R.A. (1986). *Aanalysis and design of stream ciphers*. Berlin: Springer-Verlag.

Ryan, C., Collins, J., & Neill, M. O. (1998). *Grammatical evolution: Evolving programs for an arbitrary language*. Lecture Notes in Computer Science, 1391, 83-96.

Schneier, B. (1996). Applied cryptography: Protocols, algorithms, and source code in C (2nd ed.). New York: Wiley.

Shannon, C. E. (1949). Communication theory of secrecy systems. *The Bell System Technical Journal, 28*, 656-715

Seibel, P. (2005). Practical COMMON LISP. Berkeley, CA: Apress.

Stallings, W. (2006). Cryptography and network security: Principles and practices (4th ed.). Upper Saddle River, N.J.: Pearson/Prentice Hall.

Spears, W., & Jong, K. (1990). A formal analysis of the role of multi-point crossover in genetic algorithms. Annals of Mathematics and Artificial Intelligence, 1-26.

Syswerda, G. (1989). Uniform Crossover in Genetic Algorithms. *International Conference on Genetic Algorithms*, *3*, 2-9.

Szaban, M., Seredynski F., & Bouvry P. (2006). Collective behavior of rules for cellular automata-based stream ciphers. *IEEE Congress on Evolutionary Computation*, July, Vancouver, BC, 179-183.

Tobias, B., & Lothar, T. (1995). *A comparison of selection schemes used in Genetics Algorithms. TIK Report.*

Varrette, S., Muszyński, J., & Bouvry, P. (2012). Hash function generation by means of Gene Expression Programming. *Annales UMCS Informatica AI XII,* 37-53.

Xiangyang, X. (2010). A new genetic algorithm and tabu search for s-box optimization. *International Conference On Computer Design And Appliations 2010*, *4*, 492-495.

Yalcin, M., Suykens, J., & Vandewalle, J. (2004). True Random Bit Generation From a Double-Scroll Attractor. IEEE Transactions on Circuits and Systems I: Regular Papers, 51(7), 1395-1404.

Zalzala, A., & Fleming, P. (1997). Genetic algorithms in engineering systems. London: Institution of Electrical Engineers.

Zeng, K., Yang, C., & Rao, T.R.N. (1991). Pseudorandom bit generator in stream cipher cryptography. *Computer*, 24(2), 8-17.

Zhou, J., Qing, S., & Okamoto, T. (2001). Information and Communications Security. Paper presented at Third International Conference, ICICS 2001, Xian, China, 1334: Springer Science & Business Media.

ملخص الرسالة باللغة العربية

في الوقت الراهن تعتمد المؤسسات على الأنظمة المعلوماتية التي تحفظ وتنتقل من خلال الحواسيب وشبكات التواصل المعلوماتية. من أساسيات استخدام هذه الأنظمة المحافظة على أمن المعلومات وسريتها بحيث لا يطلع عليها إلا من كان مخولا بذلك.

نظام التشفير من أهم الآليات المستخدمة للمحافظة على أمن المعلومات. هناك الكثير من التقنيات المستخدمة بالتشفير تختلف فيما بينها بدرجة الأمان التي توفرها وكيفية عملها. من أهم نظم التشفير المعتمدة هذه الأيام نظام التشفير الأنسيابي وذلك لدرجة الأمان العالية التي يوفرها وسهولة تنفيذه وكفاءته.

تصميم هذا النوع من نظم التشفير ليست عملية سهلة. العديد من الأبحاث والدراسات أجريت في هذا المجال. تقدم هذه الأطروحة تصميم جديد لنظام التشفير الأنسيابي يعتمد على البرمجة الجينية لكن بطريقة تمثيل جديدة تستخدم لتمثيل مكونات البرمجة الجينية. هذه الطريقة الجديدة تسمى الخوارزمية الجينية لتطوير البرمجيات مع خوارزمية محاكاة التلدين.

سنقوم بتجربة التصميم الجديد مع استخدام عوامل متغيرة كثيرة للتأكد من كيفية أداء التصميم المقترح بأكثر من بيئة. وستتم مقارنة النتائج للتوصل لأفضل بيئة تناسب التصميم. كما سيتم مقارنة أداء هذا التصميم مع التصميم الأساسي لنظام التشفير الأنسيابي الذي يستخدم الطريقة التقليدية بالبرمجة الجينية لاكتشاف أيهما افضل.

جامعة البحرين

قسم تقنية المعلومات

جامعة البحرين
المكتبة
LIBRARY
University of Bahrain

التصميم المؤتمت لأنظمة التشفير الانسيابي باستخدام البرمجة الجينية وخوارزمية محاكاة التلدن

مخطط بحث لاستكمال متطلبات الحصول على درجة الماجستير

في تقنية المعلومات

إعـــــداد

أمل محمد الحدي

الرقم الجامعي: 19991154

إشـــــراف

وسن شاكر عواد

(استاذ مشارك)

جامعة البحرين

مملكة البحرين

ديسمبر 2014