CONSTRUCTION OF BOOLEAN FUNCTION BY HEURISTIC SEARCH FOR CRYPTO-SYSTEMS

A DISSERTATION

submitted in partial fulfillment of the requirements for the award of the degree
of
MASTER OF TECHNOLOGY
in
ELECTRONICS AND COMMUNICATION ENGINEERING
(With Specialization in Communication Systems)

By
DHEERAJ KUMAR SHARMA

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE -247 667 (INDIA)
JUNE, 2009

CANDIDATE'S DECLARATION

I hereby declare that the work being presented in this dissertation entitled "**Construction of Boolean function by Heuristic search for Crypto-Systems**" in partial fulfillment of the requirements for the award or the degree of MASTER OF TECHNOLOGY with specialization in COMMUNICATION SYSTEMS, submitted in the Department of Electronics and Computer Engineering, Indian Institute of Technology, Roorkee is an authentic record of my own work carried out from July 2007 to June 2008, under the guidance and supervision of **Mr. S. CHAKRAVORTHY,** Assistant Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology, Roorkee and **Dr. SUGATA GANGOPADHYAY,** Assistant Professor, Department of Mathematics, Indian Institute of Technology, Roorkee.

I have not submitted the matter embodied in this dissertation for the award of any other degree or diploma.

Date: 29/06/09

Place: Roorkee

[illegible]

DHEERAJ KUMAR SHARMA

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of our knowledge and belief.

[illegible]

S. Chakravorty

Date:

Place:

Dr. SUGATA GANGOPADHYAY	**Mr. S. CHAKRAVORTY**
Assistant Professor,	Assistant Professor,
Department of Mathematics,	E&C Department,
IIT Roorkee,	IIT Roorkee,
Roorkee - 247667.	Roorkee - 247667.

ACKNOWLEDGEMENTS

It is my privilege and pleasure to express my profound sense of respect, gratitude and indebtedness to my guides, **Mr. S. Chakravorty**, Asst. Professor, Department of Electronics and Computer Engineering and **Dr. Sugata Gangopadhyay**, Asst. Professor, Dept. of Mathematics, Indian Institute of Technology, Roorkee, for their inspiration, guidance, constructive criticisms and encouragement throughout this dissertation work.

Thanks are due to the Lab staff Communication Systems Lab, Department of Electronics and Computer Engineering, IIT Roorkee for providing necessary facilities.

I gratefully acknowledge my sincere thanks to my family members for their inspirational impetus and moral support during course of this work.

The pleasure of nearing completion of the course requirements is immense, but with it carries the pain of leaving behind these wonderful two years of life in the sprawling green campus of this great historical institute.

I am greatly indebted to all my friends, who have graciously applied themselves to the task of helping me with ample morale support and valuable suggestions. Finally, I would like to extend my gratitude to all those persons who directly or indirectly helped me in the process and contributed towards this work.

[illegible]

(DHEERAJ KUMAR SHARMA)

ABSTRACT

In conventional cryptography, Boolean functions play a major role in the construction of symmetric key primitives such as block ciphers and stream ciphers. Various criteria, including balancedness, nonlinearity, autocorrelation, algebraic degree and algebraic immunity are used for measuring the cryptographic strength of Boolean function. Block and stream ciphers are made from Boolean functions that usually require a compromise between several conflicting cryptographic criteria. This dissertation work focuses on study of various properties of Boolean functions and construction of Boolean function by heuristic approach with a compromise between several conflicting cryptographic criteria (nonlinearity, autocorrelation, algebraic immunity).

A Boolean function, when used in cryptosystems, should be designed properly to resist algebraic attacks. Algebraic Immunity is a measure of the capability of a Boolean function to withstand algebraic attacks. So far, the Boolean functions were designed keeping in mind the other cryptographic criteria, and then it has been checked whether it can provide good algebraic immunity too. In this dissertation, algebraic immunity has chosen as one of the criterion along with nonlinearity and autocorrelation for search and Boolean functions with highest possible algebraic immunity of $\left\lceil \frac{n}{2} \right\rceil$ have been constructed.

Results with optimum tradeoff among properties like nonlinearity, algebraic degree, algebraic immunity and autocorrelation have been obtained, that remained as an open problem. For the first time, Boolean function of 5 variables with nonlinearity of 12, autocorrelation value 8, algebraic immunity 3 and algebraic degree 3 has been constructed.

TABLE OF CONTENTS

Chapter 1
INTRODUCTION

Digital communication is increasingly replacing face to face contact and direct physical exchange in transactions. Internet shopping is now high profile, many major payments in shops and supermarkets are made by credit or debit card, electronic cash (e.g. Mondex) is now emerging on the horizon, auctions are being held over the World Wide Web and a great deal of day-to-day communication is effected by email. The non-digital world has developed mechanisms to ensure interactions take place in an appropriate manner. We send confidential messages by special courier, we have passports against which our faces may be checked, we sign documents in the presence of esteemed members of society who may subsequently confirm any agreements made if there is a dispute. Stockbrokers routinely have their telephone calls taped so that disputes about what was agreed at some point may be resolved. The notes in one's wallet may be held to the light to reveal watermarks and other indicators of authenticity. Moving to the digital world does not relieve us of providing similar guarantees. We cannot see the people with whom we are interacting and consequently issues of trust must arise. Since communications media are generally shared between many parties, many of whom we may have little reason to trust, we must cater for the possible subversion of our communications in transit. We need to develop means of transacting that ensure legitimate expectations are met despite a potentially very hostile environment that is the medium. There will also be limits to how far legitimate parties in transactions will be trusted [1].

1.1 Cryptology

Cryptology is at the heart of providing such guarantees. Cryptology is the uniting name for a broad scientific field in which one studies the mathematical techniques of designing, analysis and attacking information security services. Cryptology consists of two subfields; *cryptography* and *cryptanalysis*. Cryptography is the field in which one study techniques for providing security services and cryptanalysis is the field in which one studies the techniques for attacking the security services [2]. Providing security services is not an easy task.

The very nature of security makes more difficult the task at hand. *Network security* measures are needed to protect data during transmission. A model for network security is shown in figure 1.1.

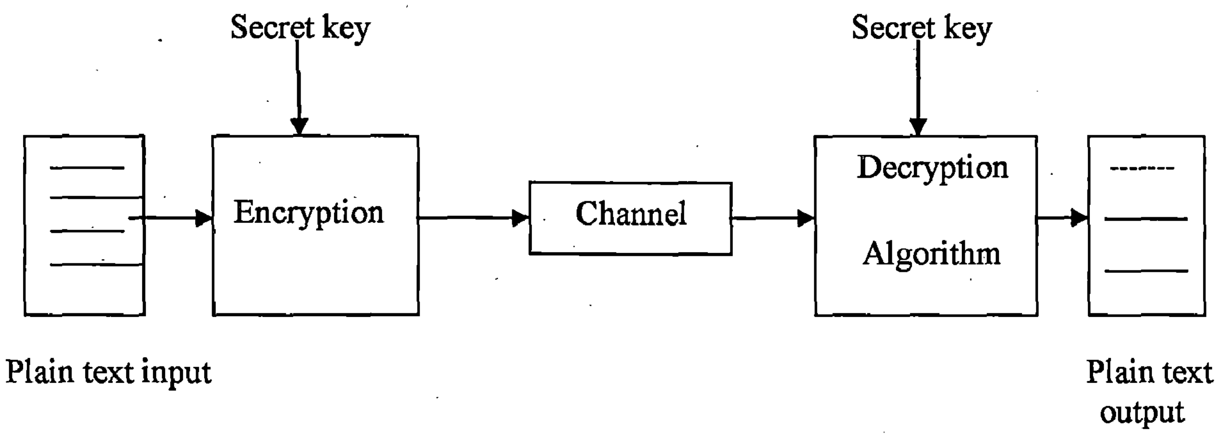

Fig.1.1 Model for Network Security.

Whereas most disciplines solve tasks unimpeded by external agents, the cryptographer must develop techniques that are resilient to perverse, malicious and potentially well funded attempts to subvert his or her efforts (i.e. break the system). In contrast, although genetic algorithms researcher might well compete with colleagues for computation time, it is unlikely he will face malicious attempts to subvert his techniques in action.

Cryptographers are, in a sense, concerned with creating problems that are artificially hard, so hard that an enemy will not be able to solve them. Suppose an Embassy encrypts diplomatic communications using a particular cryptosystem and a particular secret key K. Without knowledge of the secret key information it should be impractical for an enemy to determine the contents of any message sent within its useful life. Having intercepted the encrypted text (cipher text) in transit, an enemy could decrypt with each possible secret key in turn (generally referred to as a 'brute force' attack) to determine the one actually used for encryption (the correct key will produce the original and presumably intelligible text). If the secret key space is of sufficient size, this attack is infeasible. The problem is just too hard to solve in this way. Brute force, however, is the least sophisticated of attacks. There is an armory of devices available to the professional cryptanalyst and a successful cryptosystem must resist each. A large keyspace may protect against brute force attack, but is no guarantee that a system cannot be broken by more sophisticated means. In practice, cryptosystem designers aim to make breaking systems using known types of attack infeasible (and in some cases provably so), aim to reduce features that might form the basis of an attack, or else rely on past experience to justify unproven assumptions (e.g. the difficulty of factoring)[1].

Cryptographic techniques are typically divided into two generic types: *symmetric-key* and *public-key*. *Symmetric key* technique is the technique in which decryption key can be derived from encryption key and vice-versa. *Public key* technique is the technique in which encryption key is public but decryption key is private to receiver and decryption key cannot be derived from encryption key [2].

Symmetric key systems [3] are broadly divided into two classes:-

1. Stream Ciphers.
2. Block Ciphers.

In *Stream Cipher Cryptography*, a pseudo random sequence of bits of length equal to message length is generated. This sequence is then bitwise XOR-ed with message sequence and then message is transmitted. At the receiving end deciphering is done by generating same pseudo random sequence and again XOR-ing the cipher bits with random bits. The seed (initial start) of pseudo random generator is obtained from secret key.

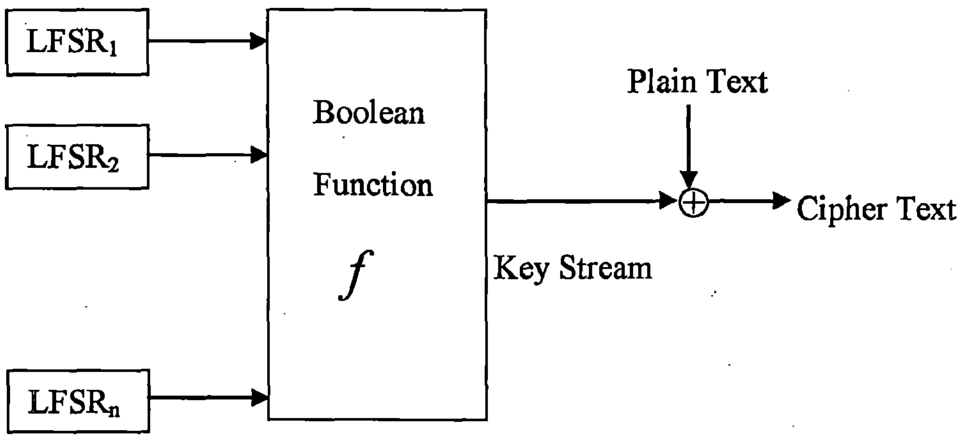

Fig. 1.2 Model for LFSR Based Encryption Scheme

Linear Feedback Shift registers (LFSR's) are important building blocks in stream ciphers. A non- linear combiner model of stream ciphers, where the outputs of several linear feedback shift registers (LFSR's) are combined by using a nonlinear Boolean function to produce a key stream, is shown in Fig. 1.2. To resist the cipher text from attacks, different design criteria have been proposed for both the LFSR's and the combining Boolean functions. Balancedness, high algebraic degree, a high nonlinearity, correlation immunity, good autocorrelation and high algebraic immunity are main criteria for design of Boolean function. A Boolean function used in stream cipher should be balanced, which is required for pseudo randomness of key stream. To resist from divide and conquer attack, the Boolean function should be correlation immune of high order [4].

In *Block Cipher Cryptography,* message is divided in blocks and each block is separately enciphered with same key stream and transmitted. Most of the block ciphers use S-boxes for introducing non-linearity.

Different construction methods of Boolean functions are

- *Primary* constructions, which produce functions directly.
- *Secondary* constructions which give new functions from previously designed ones. Some examples are: Direct sums of functions, Siegenthaler's construction [4], Tarannikov's elementary construction [5], construction by heuristic search [6] and Multiobjective Random bit climber [7].

Multiobjective Random bit climber [7] used to find Boolean functions satisfying multiple criteria.

Heuristic search is concerned with the development and application of general purpose optimization techniques and has been successfully used across many scientific, engineering and commercial domains.

1.2 Motivation for Heuristic Search

Mathematics remains the most powerful tool in science and engineering. A vast number of techniques have been developed to solve problems posed. These techniques often provide exact answers. There is, for example, a *formula* for the roots of a quadratic equation. Still it is difficult to get solution of many practical problems. The well-known Traveling Salesman Problem (TSP) is a good example:

> Consider a set of N cities, indexed 1......N. Each pair (i, j) of cities is connected by a road of length d_{ij} . A salesman lives in town 1. Starting from town 1, the salesman must carry out a tour, visiting each town in turn, and then return home. In what order should he visit the cities to give the shortest round tour [1]?

There is no known efficient method for finding a minimal length tour of a large number of cities. Enumeration over all tours would reveal the answer eventually but since there are $\frac{(N-1)!}{2}$ possible tours this approach rapidly becomes infeasible. In practice, an optimal solution to such problems is not expected. Rather, the solution space is navigated in a practically effective way to reach excellent, but not *necessarily* optimal, solutions. This is sensible since such problems are often concerned with efficient use of resources. In practice, a planner is not asked 'What is the shortest length tour?' He or she is asked 'What is the best tour you can suggest within a reasonable time?'

To construct Boolean functions achieving good properties (nonlinearity, algebraic degree, autocorrelation, algebraic immunity) simultaneously from big sample space (e.g. for 8 variable Boolean function, there is $2^{2^8} = 2^{64}$ sample points) is a difficult problem. It is becoming exhaustive to construct a Boolean function with good tradeoff between properties with construction methods like direct construction of Boolean functions, Siegenthaler's construction [4], Tarannikov's elementary construction [5], etc. So, heuristic search technique is used also for construction of Boolean function. But it does not guarantee for the optimal solution.

Exchanging guarantees of optimality for computational tractability in this way is at the heart of heuristic search. Often drawing loose inspiration from natural processes, researchers have created combinatorial search techniques that can produce effective answers where other techniques fail. Techniques such as simulated annealing [8] (based loosely on the cooling process of molten metals) and Hill climbing method [9] have seen effective application across a huge range of disciplines.

1.3 Problem statement

Construction of Boolean function with desired properties is not an easy task. The objective of this dissertation is to construct Boolean functions by heuristic approach while achieving following good properties:

- nonlinearity
- algebraic immunity
- autocorrelation

Specifically, the following tasks have been undertaken:

i) Representation of Boolean function (ANF, WHT)
ii) Review of heuristic approach
iii) Construction of Boolean function by hill climbing method, simulated annealing and combination of both methods.

1.4 Organization of Report

Including this introductory chapter, the report is organized in six chapters. *Chapter 2* provides the essential definitions and different representations of Boolean functions. The essential properties of Boolean functions and their significance are included in the same chapter.

Chapter 3 provides an overview of heuristic search. The uses of hill climbing and simulated annealing algorithms for construction of Boolean function have been discussed in detail in this *chapter 3*. *Chapter 4* introduces the concept of cost functions and outlines their importance. The details of cost functions used in the present work are elaborated and optimization techniques used to minimize these cost functions have been explained in detail in this chapter.

Chapter 5 contains results of the search for "good" Boolean functions having good properties(algebraic degree, nonlinearity, autocorrelation and algebraic immunity) with satisfactory tradeoff and compares the results with some of the earlier results of other authors. *Chapter 6* concludes the report and includes scope of future work in the field.

Chapter 2
DEFINITIONS AND PRELIMINARIES

To resist cipher form attacks, it is necessary for a Boolean function to satisfy some bounds on its cryptographic properties. It is difficult to obtain a Boolean function satisfying all the cryptographic properties. There is a tradeoff between some of the properties like Nonlinearity and Algebraic Immunity [10, 11]. Some of these properties can be described in terms of Walsh Hadamard Transform, Auto-Correlation transform and Hamming weight. *Hamming Weight* of a Boolean function, which is a measure of its difference from null space, is defined as number of ones in the function. It is denoted by *wt (f)*. *The Hamming distance* between two n-variable Boolean functions f and g is defined as size of the set$\left\{x \in F_2^n / f(x) \neq g(x)\right\}$. It is denoted by *d (f, g)*[12].

In this chapter several well known related representations and cryptographically desirable properties of Boolean functions are described. First, the truth table is defined and some simple operations are discussed. Next, Algebraic Normal Form ANF is considered in detail. The Walsh Hadamard transform (WHT) provides a unique representation of Boolean functions that is vital to cryptographic work since it expresses a Boolean function in terms of its correlation to all linear functions. It is presented along with some important theorems that describe an essential tradeoff between cryptographic properties.

The Galois field of order 2 will be denoted by F_2 and Galois field of order 2^n will be denoted by F_{2^n} and the corresponding vector space by F_2^n. Addition operator over F_2 is denoted by +. A Boolean function is essentially a function which maps one or more binary input variables to one binary output variable. We write this as a mapping from a vector $x = (x_1, x_2 ... x_n)$ to a single output f, where $x_i \in F_2$, $1 \leq i \leq n$..

$$f : F_2^n \rightarrow F_2$$

For n input variables there exists 2^n possible combination of inputs and since each input represents a particular monomial, so there will be 2^n monomials. Therefore there exist 2^{2^n} distinct Boolean functions and we denote set of all Boolean functions by β_n

2.1 Truth Table Representation

The binary truth table (TT) of a single output Boolean function f(x): $F_2^n \rightarrow F_2$ is a list of the outputs for every possible combination of input variables. Explicit storage of a binary truth table requires 2^n bits with the input ordering taken to be the natural lexicographical ordering of binary n-tuples. A Boolean function may be implemented in RAM as a look up table (LUT) if enough memory is available. Alternatively a Boolean function can be implemented using combinational logic: a network of logic gates that generates the output of the function corresponding to the input values. The complexity of a Boolean function can be defined as the minimum number of gates required to construct the circuit. Any circuit representation can be expressed as a formula using the basic operations of AND, OR, XOR and inversion (complementation) [3].

From the truth table of representation, we can write Boolean function in sum of products form. In the following example (Table 2.1), we have three inputs x_1, x_2 and x_3 and output f. For m variable Boolean function, the table will consist of m columns for input, one column for output and 2^m rows for enumeration of input variables.

x_1	x_2	x_3	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 2.1 The Truth table of the Boolean function $f(x_1, x_2, x_3) = x_1 x_2 + x_2 x_3 + x_3$.

It is often useful to consider Boolean functions over the set {1,-1} rather than {0, 1} and we introduce a "dash" notation to distinguish these forms. The state S of a Boolean function can be defined by storing the truth table outputs of Boolean function in an array. For example, consider a 3 variable Boolean function of Table 2.1, and then the state S will be of Boolean function will be (01000111). Then, this Boolean function will have 2^8 states.

x_1	x_2	x_3	f	f'
0	0	0	0	1
0	0	1	1	-1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	1	-1
1	1	0	1	-1
1	1	1	1	-1

Table 2.2 Polarity Truth table Representation of Boolean Function f

The polarity truth table (Table 2.2), $f'(x)$ of Boolean function $f(x)$ is given by

$$f'(x) = (-1)^{f(x)} = 1 - 2f(x) \qquad (2.1)$$

The binary truth table can be obtained from the polarity truth table by

$$f(x) = \frac{1-f'(x)}{2} \qquad (2.2)$$

The state S of a Boolean function can be defined by storing the truth table outputs of Boolean function in an array. For example, consider a 5 variable Boolean function, the state S can be stored as (01100101010001100010100011000111) in the form of an array.

2.2 Algebraic Normal Form

The algebraic normal transform was introduced by Zhegalkin in 1927[13]. An n-variable Boolean function $f(x_1 \ldots, x_n)$ can be considered to be a multivariate polynomial over F_2. This polynomial can be expressed as a sum of products representation of all distinct r-th order products ($0 \leq r \leq n$) of the variables. More precisely, $f(x_1, \ldots, x_n)$ can be written as

$$f(x_1, \ldots, x_n) = \sum_{u \in F_2^n} \lambda_u \left(\prod_{i=1}^{n} x_i^{u_i} \right) \quad , \quad \lambda_u \in F_2 \; , \quad u = (u_1, \ldots\ldots\ldots, u_n). \tag{2.3}$$

where, u corresponds to an n-tuple.

This representation of f is called the *algebraic normal form* (ANF) of f. The *algebraic degree* of f, denoted by $deg(f)$, is the maximal value of the Hamming weight of u such that $\lambda_u \neq 0$. There is a one-to-one correspondence between the truth table and the ANF via so called inversion formulae.

Example: $f(x_1, x_2, x_3, x_4, x_5) = x_1 + x_2 + x_2x_4 + x_3x_4 + (x_2 + x_3 + x_1x_4 + x_2x_4 + x_3x_4)\, x_5$

This function can be written as $f(x_1, x_2, x_3, x_4, x_5) = \sum_{u \in F_2^5} \lambda_u \left(\prod_{i=1}^{5} x_i^{u_i}\right)$

The set of x values for which $f(x) = 1$ ($f(x) = 0$) is called the on-set (off-set), and is denoted by $S_1(f)$ ($S_0(f)$). The ANF of f is fully specified by its on-set using the following expansion,

$$f(x_1, \ldots\ldots, x_n) . = \sum_{\tau \in S_1(f)} \left(\prod_{i=1}^{n} (x_i + \tau_i + 1)\right) \quad , \qquad \tau = (\tau_1 \ldots \tau_n) \tag{2.4}$$

The Algebraic Normal Form (ANF) of a Boolean function is an XOR sum of AND products. There are 2^n possible combinations of n input variables for f, so there are 2^n distinct product terms and every XOR sum is a formula for the corresponding Boolean function. The number of different ANFs is equal to the number of different truth tables and they are in one-to- one correspondence. It is therefore a unique representation: $ANF_f = ANF_g$ if and only if $f(x) = g(x)$ for all $x \in F_2^n$. The ANF can be stored as 2^n binary coefficients λ_u of the terms in the XOR sum.

The functions of degree at most one are called as *affine* functions. The affine functions with constant term equal to zero are called as *linear* functions. The set of all *affine* functions of n variables is denoted by A_n

Consider two n-1 variable Boolean functions f_1 and f_2 over F_2 then $f = f_1 \| f_2$ is an n- variable Boolean function over F_2 is *concatenation* of f_1 and f_2 [6]. It means that the upper half part of the truth table of f correspond to f_1 and the lower half to f_2. The ANF of f is then given by

$$f(x_1, \ldots, x_n) = (1+x_n)f_1(x_1, \ldots, x_{n-1}) + x_n f_2(x_1, \ldots, x_{n-1}). \tag{2.5}$$

2.3 Walsh-Hadamard Transform

The Walsh-Hadamard transform is an orthogonal transform like the discrete Fourier transform. S. Golomb was apparently the first to consider the Walsh-Hadamard transform of Boolean functions [14, 15]. The Walsh Hadamard transform (WHT) is one of the important tools required for the analysis of Boolean functions. The WHT of an n-variable function $f(x_1, \ldots, x_n)$ is the real-valued function over F_2^n whose value at every $a \in F_2^n$ is defined as

$$\hat{f}(a) = \sum_{x \in F_2^n} (-1)^{f(x)+a.x} \quad , \tag{2.6}$$

where $a.x = a_1x_1+a_2x_2+\ldots\ldots\ldots+a_nx_n$ is a normal inner product in F_2^n.

The WHT expresses a Boolean function uniquely in terms of its correlation with all linear functions. This is also helpful in calculation of nonlinearity of Boolean functions.

Example: Given a truth table we can calculate Walsh Transform. Consider a function $f(x_1, x_2, x_3)$ shown in Table 2.3

x_3	x_2	x_1	f
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Table 2.3 Truth table of $f(x_1, x_2, x_3)$

Consider $a = (a_3,a_2,a_1)$ and $x = (x_3,x_2,x_1)$

(i) For $a = (0,0,0)$, $a.x = 0$.

x_3	x_2	x_1	f	$f'(x) = (-1)^{f(x)}$	$a.x = 0$	$(-1)^{a.x}$	$(-1)^{f(x)+a.x}$
0	0	0	1	-1	0	1	-1
0	0	1	1	-1	0	1	-1
0	1	0	0	1	0	1	1
0	1	1	1	-1	0	1	-1
1	0	0	1	-1	0	1	-1
1	0	1	0	1	0	1	1
1	1	0	0	1	0	1	1
1	1	1	0	1	0	1	1

Therefore, $\hat{f}(000)$ = (-1)+(-1)+1+(-1)+(-1)+1+1+1 = 0

(ii) For $a = (0,0,1)$, $a.x = x_1$

x_3	x_2	x_1	f	$f'(x) = (-1)^{f(x)}$	$a.x = x_1$	$(-1)^{a.x}$	$(-1)^{f(x)+a.x}$
0	0	0	1	-1	0	1	-1
0	0	1	1	-1	1	-1	1
0	1	0	0	1	0	1	1
0	1	1	1	-1	1	-1	1
1	0	0	1	-1	0	1	-1
1	0	1	0	1	1	-1	-1
1	1	0	0	1	0	1	1
1	1	1	0	1	1	-1	-1

Therefore, $\hat{f}(001)$ = (-1)+1+1+1+(-1)+(-1)+1+(-1) = 0.

(iii) For $a = (0,1,0)$, $a.x = x_2$

x_3	x_2	x_1	f	$f'(x) = (-1)^{f(x)}$	$a.x = x_2$	$(-1)^{a.x}$	$(-1)^{f(x)+a.x}$
0	0	0	1	-1	0	1	-1
0	0	1	1	-1	0	1	-1
0	1	0	0	1	1	-1	-1
0	1	1	1	-1	1	-1	1
1	0	0	1	-1	0	1	-1
1	0	1	0	1	0	1	1
1	1	0	0	1	1	-1	-1
1	1	1	0	1	1	-1	-1

Therefore, $\hat{f}(010)$ = (-1)+(-1)+(-1)+1+(-1)+1+(-1)+(-1) = -4

(iv) For $a = (0,1,1)$, $a.x = x_1 + x_2$

x_3	x_2	x_1	f	$f'(x) = (-1)^{f(x)}$	$a.x = x_1 + x_2$	$(-1)^{a.x}$	$(-1)^{f(x)+a.x}$
0	0	0	1	-1	0	1	-1
0	0	1	1	-1	1	-1	-1
0	1	0	0	1	1	-1	1
0	1	1	1	-1	0	1	-1
1	0	0	1	-1	0	1	-1
1	0	1	0	1	1	-1	-1
1	1	0	0	1	1	-1	-1
1	1	1	0	1	0	1	1

Therefore, $\hat{f}(011) = (-1)+(-1)+(1)+(-1)+(-1)+(-1)+(-1)+(1) = -4$

(v) For $a = (1,0,0)$, $a.x = x_3$

x_3	x_2	x_1	f	$f'(x) = (-1)^{f(x)}$	$a.x = x_3$	$(-1)^{a.x}$	$(-1)^{f(x)+a.x}$
0	0	0	1	-1	0	1	-1
0	0	1	1	-1	0	1	-1
0	1	0	0	1	0	1	1
0	1	1	1	-1	0	1	-1
1	0	0	1	-1	1	-1	1
1	0	1	0	1	1	-1	-1
1	1	0	0	1	1	-1	-1
1	1	1	0	1	1	-1	-1

Therefore, $\hat{f}(100) = (-1)+(-1)+(1)+(-1)+(1)+(-1)+(-1)+(-1) = -4$

(vi) For $a = (1,0,1)$, $a.x = x_3 + x_1$

x_3	x_2	x_1	f	$f'(x) = (-1)^{f(x)}$	$a.x = x_3 + x_1$	$(-1)^{a.x}$	$(-1)^{f(x)+a.x}$
0	0	0	1	-1	0	1	-1
0	0	1	1	-1	1	-1	1
0	1	0	0	1	0	1	1
0	1	1	1	-1	1	-1	1
1	0	0	1	-1	1	-1	1
1	0	1	0	1	0	1	1
1	1	0	0	1	1	-1	-1
1	1	1	0	1	0	1	1

Therefore, $\hat{f}(101) = (-1)+1+1+1+1+1+(-1)+1 = 4$

(vii) For $a = (1,1,0)$, $a.x = x_3 + x_2$

x_3	x_2	x_1	f	$f'(x) = (-1)^{f(x)}$	$a.x = x_3 + x_2$	$(-1)^{a.x}$	$(-1)^{f(x)+a.x}$
0	0	0	1	-1	0	1	-1
0	0	1	1	-1	0	1	-1
0	1	0	0	1	1	-1	-1
0	1	1	1	-1	1	-1	1
1	0	0	1	-1	1	-1	1
1	0	1	0	1	1	-1	-1
1	1	0	0	1	0	1	1
1	1	1	0	1	0	1	1

Therefore, $\hat{f}(110) = (-1)+(-1)+(-1)+1+1+(-1)+1+1 = 0$

(viii) For $a = (1,1,1)$, $a.x = x_3 + x_2 + x_1$

x_3	x_2	x_1	f	$f'(x) = (-1)^{f(x)}$	$a.x = x_3 + x_2 + x_1$	$(-1)^{a.x}$	$(-1)^{f(x)+a.x}$
0	0	0	1	-1	0	1	-1
0	0	1	1	-1	1	-1	-1
0	1	0	0	1	1	-1	1
0	1	1	1	-1	0	1	-1
1	0	0	1	-1	1	-1	1
1	0	1	0	1	0	1	1
1	1	0	0	1	0	1	1
1	1	1	0	1	1	-1	-1

Therefore, $\hat{f}(111) = (-1)+(-1)+1+(-1)+1+1+1+(-1) = 0$

In this way, we can calculate the WHT of *f(x)* as shown below in table

a_3	a_2	a_1	$\hat{f}$
0	0	0	0
0	0	1	0
0	1	0	-4
0	1	1	-4
1	0	0	-4
1	0	1	4
1	1	0	0
1	1	1	0

Table 2.4 Walsh Hadamard Transform of *f*

Walsh Hadamard Transform is calculated by Fast WT algorithm [16].

Consider a function $f(x_1, x_2, x_3)$ shown in Table 2.3. Table 2.3 has been reproduced for convenience.

x_3	x_2	x_1	f
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

First, we generate the polarity truth table (Fig 2.1) of $f(x_1, x_2, x_3)$. Next, each pair of elements is modified by an "in-phase butterfly"; that is, the values in each pair produce two results which replace the original pair, wherever they were originally located. The left result will be the two values added; and the right will be the first less second. That is, $(a', b') = (a + b, a - b)$ where (a, b) is original pair. So for the values (-1, 1) we get (-1+1, -1-1) or (0, -2). We start pairing out adjacent elements, then every other element, then every 4th element, then every eighth element and so on until the correct pairing is impossible, as shown in Figure 2.1.

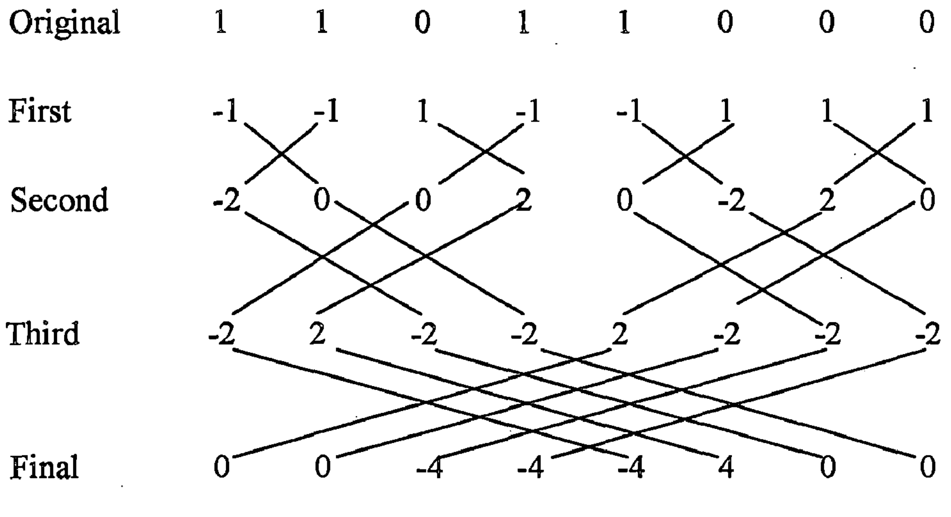

Fig.2.1 An 8-element Fast Walsh Transform.

The WHT of f is shown in Table 2.4. Table 2.4 has been reproduced here for convenience.

x_1	x_2	x_3	$\hat{f}$
0	0	0	0
0	0	1	0
0	1	0	-4
0	1	1	-4
1	0	0	-4
1	0	1	4
1	1	0	0
1	1	1	0

Theorem 2.1(Parseval) Let $f(x_1,\ldots,x_n)$ be a real valued function with domain the vector space F_2^n with Walsh Hadamard Transform $\hat{f}(a)$ where $a \in F_2^n$, then[15]

$$\sum_{a \in F_2^n} \hat{f}^2(a) = 2^{2n}. \tag{2.7}$$

2.4 Balancedness

A Boolean function is said to be balanced if its truth table has equal number of 1's and 0's. In other words, if f is an n-variable Boolean function then it will be balanced if

$$\mathrm{wt}\,(f) = 2^{n-1} \tag{2.11}$$

From the definition of Walsh Hadamard Transform, the sufficient and necessary condition for a function to be balanced is [3]

$$\hat{f}(\underline{0}) = 0 \tag{2.12}$$

where $\underline{0}$= (0,0,0.......0) in F_2^n.

2.5 Auto-Correlation Function

Consider a n-variable Boolean function $f(x_1,\ldots x_n)$. Then auto-correlation function will be defined as

$$\hat{r}_f(s) = \sum_x f'(x)\, f'(x+s) \tag{2.8}$$

where $f'(x) = (-1)^{f(x)}$ and x and s range over F_2^n.

Therefore $\hat{r}_f(s)$ will also in F_2^n .The maximum value of auto correlation is denoted as AC_f .i.e. $AC_f = max_{s\neq 0}|\sum_x f'(x)f'(x+s)|$[3,9]. For every Boolean function we have $\hat{r}_f(0)$ $=2^n$ since $(\hat{f}(x))^2 = 1$.

For balanced Boolean function, Maitra conjecture [17] for even n provides the bound on autocorrelation ACB(n) by relation

$$ACB(n) = 2^{\frac{n}{2}} + ACB(\tfrac{n}{2})$$

where ACB(n) represents autocorrelation bound for n variable Boolean function.

Example: Given a truth table we can calculate Auto-Correlation Transform. Consider a function $f(x_1, x_2, x_3)$ shown in Table 2.5.

x_3	x_2	x_1	f	f'
0	0	0	1	-1
0	0	1	1	-1
0	1	0	0	1
0	1	1	1	-1
1	0	0	1	-1
1	0	1	0	1
1	1	0	0	1
1	1	1	0	1

Table 2.5 Truth table of $f(x_1, x_2, x_3)$

Consider $s = (s_3,s_2,s_1)$ and $x=(x_3,x_2,x_1)$

(i) For s=(0,0,0)

x_3	x_2	x_1	$f(x)$	$f'(x)$	$f'(x+s)$	$f'(x)f'(x+s)$
0	0	0	1	-1	-1	1
0	0	1	1	-1	-1	1
0	1	0	0	1	1	1
0	1	1	1	-1	-1	1
1	0	0	1	-1	-1	1
1	0	1	0	1	1	1
1	1	0	0	1	1	1
1	1	1	0	1	1	1

Therefore, $\hat{r}_f(s)$ = 1+1+1+1+1+1+1+1 = 8.

(ii) For s=(0,0,1)

x_3	x_2	x_1	$f(x)$	$f'(x)$	$f'(x+s)$	$f'(x)f'(x+s)$
0	0	0	1	-1	-1	1
0	0	1	1	-1	-1	1
0	1	0	0	1	-1	-1
0	1	1	1	-1	1	-1
1	0	0	1	-1	1	-1
1	0	1	0	1	-1	-1
1	1	0	0	1	-1	-1
1	1	1	0	1	1	1

Therefore, $\hat{r}_f(s)$ = 1+1+(-1)+(-1)+(-1)+(-1)+(-1)+1 = -2.

(iii) For s=(0,1,0)

x_3	x_2	x_1	$f(x)$	$f'(x)$	$f'(x+s)$	$f'(x)f'(x+s)$
0	0	0	1	-1	1	-1
0	0	1	1	-1	-1	1
0	1	0	0	1	-1	-1
0	1	1	1	-1	-1	1
1	0	0	1	-1	1	-1
1	0	1	0	1	-1	-1
1	1	0	0	1	-1	-1
1	1	1	0	1	1	1

Therefore, $\hat{r}_f(s)$ = (-1)+1+(-1)+1+(-1)+(-1)+(-1)+1 = -2.

(iv) For s=(0,1,1)

x_3	x_2	x_1	$f(x)$	$f'(x)$	$f'(x+s)$	$f'(x)f'(x+s)$
0	0	0	1	-1	-1	1
0	0	1	1	-1	1	-1
0	1	0	0	1	-1	-1
0	1	1	1	-1	-1	1
1	0	0	1	-1	1	-1
1	0	1	0	1	-1	-1
1	1	0	0	1	1	1
1	1	1	0	1	-1	-1

Therefore, $\hat{r}_f(s)$ =1+(-1)+(-1)+1+(-1)+(-1)+1+(-1) = -2.

(v) For s=(1,0,0)

x_3	x_2	x_1	$f(x)$	$f'(x)$	$f'(x+s)$	$f'(x)f'(x+s)$
0	0	0	1	-1	-1	1
0	0	1	1	-1	1	-1
0	1	0	0	1	1	1
0	1	1	1	-1	1	-1
1	0	0	1	-1	-1	1
1	0	1	0	1	-1	-1
1	1	0	0	1	1	1
1	1	1	0	1	-1	-1

Therefore, $\hat{r}_f(s)$ =1+(-1)+1+(-1)+1+(-1)+1+(-1) = 0.

(vi) For s=(1,0,1)

x_3	x_2	x_1	$f(x)$	$f'(x)$	$f'(x+s)$	$f'(x)f'(x+s)$
0	0	0	1	-1	1	-1
0	0	1	1	-1	-1	1
0	1	0	0	1	1	1
0	1	1	1	-1	-1	1
1	0	0	1	-1	-1	1
1	0	1	0	1	-1	-1
1	1	0	0	1	-1	-1
1	1	1	0	1	-1	1

Therefore, $\hat{r}_f(s)$ =(-1)+1+1+1+1+(-1)+(-1)+1 = 2.

(vii) For s=(1,1,0)

x_3	x_2	x_1	$f(x)$	$f'(x)$	$f'(x+s)$	$f'(x)f'(x+s)$
0	0	0	1	-1	-1	1
0	0	1	1	-1	1	-1
0	1	0	0	1	-1	-1
0	1	1	1	-1	1	-1
1	0	0	1	-1	1	-1
1	0	1	0	1	-1	-1
1	1	0	0	1	-1	-1
1	1	1	0	1	-1	-1

Therefore, $\hat{r}_f(s)$ =1+(-1)+(-1)+(-1)+(-1)+(-1)+(-1)+(-1) = 6.

(viii) For s=(1,1,1)

x_3	x_2	x_1	$f(x)$	$f'(x)$	$f'(x+s)$	$f'(x)f'(x+s)$
0	0	0	1	-1	1	-1
0	0	1	1	-1	1	-1
0	1	0	0	1	1	1
0	1	1	1	-1	-1	1
1	0	0	1	-1	-1	1
1	0	1	0	1	1	1
1	1	0	0	1	-1	-1
1	1	1	0	1	-1	-1

Therefore, $\hat{r}_f(s)$ =(-1)+(-1)+1+1+1+1+(-1)+(-1) = 0.

So, for s $\neq$ 0, we have

$AC_f = \max_s|\hat{r}f\ (s)| = 6$.

2.6 Non Linearity

The nonlinearity of a Boolean function is defined as the minimum Hamming distance to any affine function [3,10]. Consider A_n be a set of all n-variable affine functions. Then Nonlinearity nl (f) of Boolean function f is defined as

$$N_f = \text{nl}\ (f) = \min_{g \epsilon A_n} d(f, g) \tag{2.9}$$

It is sometimes written in terms of WHT forms as

$$N_f = \text{nl}\ (f) = 2^{n-1} - \frac{1}{2}\max_{a \epsilon \mathbb{F}_2^n} |\hat{f}(a)| \tag{2.10}$$

To maximize the nonlinearity, $\max_{a \epsilon \mathbb{F}_2^n} |\hat{f}(a)|$ should be minimum and the minimum of $\max_{a \epsilon \mathbb{F}_2^n} |\hat{f}(a)|$ is $2^{n/2}$ i.e. for maximum nonlinearity $\hat{f}(a) = \pm 2^{n/2}$. So, the maximum achievable value for nonlinearity for n-variable Boolean function is 2^{n-1}-$2^{(n-2)/2}$. It is achievable only for even n. The functions achieving this value are called as bent functions [10].

For balanced Boolean functions, Dobertin's conjecture [18] states that on an even number n of inputs, the highest achievable nonlinearity satisfies Nl(n) = $2^{n-1} - 2^{\frac{n}{2}} +$ Nl$(\frac{n}{2})$, where Nl(n) represents nonlinearity of n-variable Boolean function.

High nonlinearity is required to resist affine approximations of Boolean function. If we are able to fix some inputs of n-variable Boolean function then function can be approximated by affine function [3,10,11].

Linear cryptanalysis is a very powerful cryptanalytic method for stream ciphers. A function with low nonlinearity is prone to linear approximation attack. Linear approximation means approximating the combining function by a linear function. Thus for symmetric cipher applications we need functions with high nonlinearity [19].

2.7 Algebraic Immunity

Very recently, a new attack that uses cleverly over defined systems of multivariate nonlinear equations to recover the secret key has gained a lot of attention (the idea of using such systems comes from Shannon , but the improvement in the efficiency of the method is recent)[20]. It is known as algebraic attack. Given a Boolean function on variables, different kinds of scenarios related to low-degree multiples of have been studied in [21,22]. Consider *f(x)* and *g(x)* be two n-variable Boolean functions such that *f*g=0* or *(1+f)*g=0* where '*' is multiplication of GF(2) elements.. Then *g(x)* is called as annihilator of *f(x).* The core of the analysis is to find minimum (or low) degree annihilators of *f(x)* or (1+*f(x)*). To mount the algebraic attack, one needs only low-degree annihilators [21,22]. The immunity of Boolean functions against algebraic attacks is called as algebraic immunity. The highest possible algebraic immunity is $\left\lceil \frac{n}{2} \right\rceil$[10,21,22,23].

It has been observed that a Boolean function used as a cryptographic primitive, and interpreted as a multivariate polynomial over F_2, should not have low degree multiples obtained by multiplication with low degree nonzero functions. The functions with low nonlinearity are more prone to attack. Functions having low-degree subfunctions are weak in terms of algebraic immunity. Some functions are symmetric, so they are at risk of attacks. Carlet et. al. present a construction method to generate Boolean functions on n variables with highest possible algebraic immunity $\left\lceil \frac{n}{2} \right\rceil$[23].

2.8 Algebraic degree

From Equation no. 2.3, the *algebraic normal form* (ANF) of $f(x_1, \ldots, x_n)$ can written as

$$f(x_1, \ldots, x_n) = \sum_{u \in F_2^n} \lambda_u \left(\prod_{i=1}^{n} x_i^{u_i} \right) \quad , \ \lambda_u \in F_2 \ , \quad u = (u_1, \ldots\ldots\ldots, u_n).$$

where, u corresponds to an n-tuple.

The *algebraic degree* of f, denoted by $deg(f)$, is the maximal value of the Hamming weight of u such that $\lambda_u \neq 0$.

High algebraic degree resists certain attacks and is therefore desirable in both stream and block ciphers. In the stream cipher model, the combining function f is so chosen that it increases the linear complexity of the resulting key stream. High algebraic degree provides high linear complexity [22].

Chapter 3
HILL CLIMBING TECHNIQUE FOR HEURISTIC SEARCH

The basic idea of heuristic search is that, rather than trying all possible search paths, we try and focus on paths that seem to be getting us nearer our goal state. Of course, we generally can't be sure that we are really near our goal state - it could be that we will have to take some amazingly complicated and circuitous sequence of steps to get there. But we might be able to have a good guess. Heuristics are used to help us make that guess.

There are very few applications of heuristic search techniques to modern-day cryptological design or analysis problems. This is a little surprising since the heuristic search and cryptology research communities seem, at a fundamental level, to share one major interest — solving computationally 'hard' problems. This chapter provides a brief introduction to guided search techniques and use of hill climbing technique for search of Boolean function of cryptological use.

3.1 Guided Search

To resist ciphers from Cryptanalytic attack, we need to construct Boolean function with desirable properties. The main desirable properties are balancedness, high nonlinearity, low autocorrelation, high algebraic immunity. The tradeoffs between these have received a lot of attention in Boolean function literature [10,11,25]. The more criteria that have to be taken into account, the more difficult it is to construct Boolean function. In the past the main options for construction of Boolean functions were random generation and direct construction. Direct constructions can produce functions that are optimum with regard to the designed property, but they may be weak for other cryptographic properties. So, there exists a tradeoff between main criteria and determining the optimum compromise attainable is an open problem. Recent work has moved to construction of Boolean function with the aspects of computer search. So, *guided search* techniques are one of the solutions to find Boolean function with optimum compromise in desirable properties. Some authors reported have good results with guided search [16,24]. Guided search have been defined below in brief.

For some problems there may be no alternative to enumerative or sampling-based approaches. This is generally due to lack of (approximate) continuity in the function $f(x)$ i.e. the value of $f(x)$ at a specific point x_1 gives little exploitable information. Cryptology revels in lack of continuity. Indeed, certain cryptographic goodness criteria can be thought of as discontinuity measures (e.g. for a 64-bit block cipher it might be required that keys which differ by a single bit should produce ciphertexts that differ on average by 32 bits: small input changes can have radical output effects). Solutions (inputs to the cost functions) that are 'near-by' or 'close' will not give outputs that are radically different. Information gleaned from function evaluation will be used to influence the progress of the search. This is *guided search.* The notion of closeness can be formalized as a function. For a specific value the set of all points that are in the immediate neighborhood can be defined by some function $N(x)$:

$$N: X \rightarrow 2^x.$$

Here the search moves through a series of points $x_1, x_2, x_3, \ldots\ldots. x_{final}$ with each point being in the neighborhood of the point which precedes it. At each point x_n the value of $f(x)$ is evaluated for one or more points in $N(x_n)$ and the information used to determine whether the search should 'move' to a particular point in that neighborhood. There are several strategies for selecting points in the neighborhood and deciding which move, if any, should be taken. Examples of this kind of strategies are Hill climb method [9], Simulated Annealing method [8], Genetic Algorithms (GA) [25], Tabu Search [26] and some local search methods.

3.2 Gradient Search-Hill Climbing

Gradient search methods sample or enumerate the values of $f(x)$ in the neighborhood of the current solution x_{curr}. If the search moves only to a neighbor if it improves the value of *f(x)*then the search is a form of 'hill-climbing' or gradient ascent. If the neighborhood is huge then sampling may be carried out to find an improving move. Accepting a move that makes the greatest improvement gives rise to what is known as steepest ascent. If the search takes the first improving move it encounters, it is said to be a 'greedy' gradient ascent. The terms gradient ascent and gradient descent are used depending on whether the problem at hand is couched as a maximization or a minimization problem. The problem with such techniques is obvious. If the search starts in the wrong place the result may be a local optimum [1].

The **hill-climbing** search algorithm is shown in Fig. 3.1[27]. It is simply a loop that continually moves in the direction of increasing value-that is, uphill. It terminates when it reaches a "peak" where no neighbor has a higher value. The algorithm does not maintain a search tree, so the current node data structure need only record the state. Hill-climbing does not look ahead beyond the immediate neighbors of the current state [27].

```
Function HILL-CLIMBING(problem) returns a state that is a local maximum
        inputs: problem, a problem
        local variables: current, a node
                         neighbor, a node
        current ← MAKE-NODE(INITIAL-STATE[problem])
        loop do
            neighbor← a highest-valued successor of current.
            if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
            current←neighbor
```

Fig. 3.1 The hill-climbing search algorithm

Hill climbing remains an important technique nevertheless sometimes one simply has a hill to climb. Furthermore, robust non-linear optimization techniques may get close to optimal solutions but use hill-climbing to carry out the very final stages of optimization efficiently [1]. Hill climbing is used in Artificial Intelligence, Business problems, Planted Bisection problems.

The hill climbing approach to Boolean function design was introduced by W. Millan et. al. in 1997 as a means of improving the nonlinearity of a given Boolean function by making well chosen alterations of one or two places of the truth table [9,28]. If $\Delta_{WHT}(w)$ represents the change in WHT value, then it is easy to show that any single truth table change causes $\Delta_{WHT}(w) \in \{-2,2\}$ for all w. Similarly, any two truth table changes cause $\Delta_{WHT}(w) \in \{-4,0,4\}$. By starting with a balanced function, we can hill climb to a more nonlinear balanced Boolean function. The approach did not make an alteration to the truth table unless the nonlinearity is improved by a change in WHT value [9,28]. This approach can also be used for improvement of autocorrelation and for both autocorrelation and nonlinearity.

3.2.1 Non-linearity Targeted

In this approach, the nonlinearity of Boolean function is targeted [9,28]. The truth table outputs of Boolean function are changed if the nonlinearity increases. Derivation of the rules for the change of two output values of Boolean function is given below.

Consider a given Boolean function $f(x)$ in polarity truth table form $f'(x)$. Now let the truth table output be complemented for two distinct inputs x_1 and x_2. We have $g'(x_i) = -f'(x_i)$ for $i \in \{1,2\}$ and $g'(x) = f'(x)$ for other x. Now consider the WHT of $g(x)$.

$$\begin{aligned}
\hat{g}(w) &= \textstyle\sum_{x \in \mathbb{F}_2^n} (-1)^{g(x)+w.x} \\
&= (-1)^{g(x_1)+w.x_1} + (-1)^{g(x_2)+w.x_2} + \textstyle\sum_{x \# \{x_1,x_2\}} (-1)^{g(x)+w.x} \\
&= g'(x_1)(-1)^{w.x_1} + g'(x_2)(-1)^{w.x_2} + \textstyle\sum_{x \# \{x_1,x_2\}} (-1)^{g(x)+w.x} \\
&= -f'(x_1)(-1)^{w.x_1} - f'(x_2)(-1)^{w.x_2} + \textstyle\sum_{x \# \{x_1,x_2\}} (-1)^{g(x)+w.x} \\
&= -(f'(x_1)(-1)^{w.x_1} + f'(x_2)(-1)^{w.x_2}) + \textstyle\sum_{x \# \{x_1,x_2\}} (-1)^{g(x)+w.x} \qquad (3.1)
\end{aligned}$$

The change in the WHT value for all w is

$$\Delta_{WHT}(w) = \hat{g}(w) - \hat{f}(w) \qquad (3.2)$$

It follows directly that

$$\Delta_{WHT}(w) = -2(f'(x_1)(-1)^{w.x_1} + f'(x_2)(-1)^{w.x_2}) \qquad (3.3)$$

This result is used to directly update the WHT in each iteration of a 2-step hill climbing program. It is now a straightforward matter to determine the conditions required for the choice of two distinct inputs x_1 and x_2 to complement so that the WHT values change as required. It is clear that the two truth table changes ensure $\Delta_{WHT}(w) \in \{-4,0,4\}$. This method can also be used for change of four distinct inputs [9,28].

3.2.2 Auto-Correlation Targeted

Here, the autocorrelation of Boolean function is targeted [9,28]. If with the change of truth table outputs, autocorrelation decreases, then change is kept as such. Derivation of the rules for the change of two output values of Boolean function is given below.

Consider changing a Boolean function $f(x)$ by complementing the output for two distinct inputs x_1 and x_2, creating a function $g(x)$ with autocorrelation given by:

$$\hat{r}_g(s) = \sum_x g'(x)\, g'(x+s)$$

$$= 2g'(x_1)g'(x_1+s) + 2g'(x_2)g'(x_2+s) + \sum_{x \neq \{x_1,x_2,x_1+s,x_2+s\}} g'(x)g'(x+s)$$

$$= -2f'(x_1)f'(x_1+s) - 2f'(x_2)f'(x_2+s) + \textstyle\sum_{x\neq\{x_1,x_2,x_1+s,x_2+s\}} f'(x)f'(x+s) \quad (3.4)$$

For each $s \neq 0$, the change in the value of autocorrelation is

$$\Delta_{AC}(s) = \hat{r}_g(s) - \hat{r}_f(s)$$

$$= -2f'(x_1)g'(x_1+s) - 2f'(x_2)g'(x_2+s) - 2f'(x_1)f'(x_1+s) - 2f'(x_2)f'(x_2+s) \quad (3.5)$$

For $x_1 + x_2 = s$, we have,

$$g'(x_1+s) = g'(x_2) = -f'(x_2) \text{ and } g'(x_2+s) = g'(x_1) = -f'(x_1).$$

In this case the formula for autocorrelation changes collapses to

$$\Delta_{AC}(s = x_1 + x_2) = 0.$$

In the remaining general case, we have

$$\Delta_{AC}(s \neq x_1 + x_2) = -4f'(x_1)f'(x_1+s) - 4f'(x_2)f'(x_2+s)$$

Noting that the pair (x_1, x_2) was chosen so that $f(x_1) \neq f(x_2)$, we can determine that

$$\Delta_{AC}(s) = -8 \Leftrightarrow f(x_i) = f(x_i+s) \text{ for } i = \{1,2\},$$

$$\Delta_{AC}(s) = +8 \Leftrightarrow f(x_i) \neq f(x_i+s) \text{ for } i = \{1,2\},$$

$$\Delta_{AC}(s) \neq -8 \Leftrightarrow \text{not both } f(x_i) = f(x_i+s) \text{ for } i = \{1,2\} \text{ and}$$

$$\Delta_{AC}(s) \neq +8 \Leftrightarrow \text{not both } f(x_i) \neq f(x_i+s) \text{ for } i = \{1,2\}.$$

If there is requirement for improvement of autocorrelation only and wish to maintain Hamming weight, then the truth table outputs for any pair (x_1; x_2) are complemented that satisfies all of the following conditions [28]:

(i) $f(x_1) \neq f(x_2)$

(ii) $x_1 + x_2 \neq s$ and both $f(x_i) = f(x_i + s)$ for $i = \{1,2\}$, for all $\{s : \hat{r}(s) = AC_{max}\}$

(iii) $x_1 + x_2 \neq s$ and both $f(x_i) \neq f(x_i + s)$ for $i = \{1,2\}$, for all $\{s : \hat{r}(s) = -AC_{max}\}$

(iv) if $x_1 + x_2 \neq s$ then not both $f(x_i) \neq f(x_i + s)$ for $i = \{1,2\}$, for all $\{s : \hat{r}(s) = AC_{max} - 8\}$.

(v) if $x_1 + x_2 \neq s$ then not both $f(x_i) = f(x_i + s)$ for $i = \{1,2\}$, for all $\{s : \hat{r}(s) = -AC_{max} + 8\}$.

Chapter 4
SIMULATED ANNEALING ALGORITHM FOR HEURISTIC SERACH

In 1983 Kirkpatrick et al.[8] proposed *simulated annealing*, a new search technique inspired by the cooling processes of molten metal. There is a deep and useful connection between statistical mechanics (the behavior of systems with many degrees of freedom in thermal equilibrium at a finite temperature) and multivariate or combinatorial optimization (finding the minimum of a given function depending on many parameters). The analogy with annealing in solids provides a method for optimization of the properties of very large and complex systems. This technique is a generic probabilistic heuristic technique, namely locates a good approximation to the global minimum of a given function in a large search space.

4.1 Cost Function

To use heuristic search we need an *evaluation function/cost function* that scores a node in the search tree according to how close to the target/goal state it seems to be. This will just be a guess, but it should still be useful. For example, for finding a route between two towns a possible evaluation function might be a "as the crow flies" distance between the town being considered and the target town. It may turn out that this does not accurately reflect the actual (by road) distance - maybe there aren't any good roads from this town to target town. However, it provides a quick way of guessing that helps in the search [1,24].

The general aim is to find optimal solutions to problems that are structured as a function of some decision variables, perhaps in the presence of some constraints [1]. These can be formulated as:

Minimize $f(x)$ with respect to $x \in X$, subject to constraint elements of C.

The set X of all possible vectors $x = (x_1 \ldots x_n)$ of decision variables will generally be referred to as the *solution space* for the search problem at hand. The set *C* represents the imposition of constraints. Searches may be restricted to consider only elements of *C*. Alternatively, the problem may be recast as 'Minimize *g(x)* subject to $x \in X$' where *g(x)* contains a component that punishes *x* outside *C*. Such values of *x* are said to be 'priced out'. The function *f* (or *g*) is generally referred to as a *cost function*. When problems are similarly couched as maximization problems the term *fitness function* is used. There is complete freedom over which functions are used for the problem at hand. Experience shows that the choice of function is an important success factor in applying many search techniques. The best functions are those that give the best results when used. Unfortunately, it is difficult to predict in advance which functions will work best. Experimentation is the only solution to predict best cost function [1].

Solution vectors *x* may be designs (e.g. the truth table of a Boolean function used as a component in a cryptosystem) or analysis artifacts (e.g. a vector of 64 key bits sought by a cryptanalyst). To find them the designer or analyst is free to employ whatever techniques seem most suitable from the vast array available. Solution techniques span a range of sophistication.

An example can be considered as construction of a Boolean function with desirable properties (balancedness, high nonlinearity, low autocorrelation). So, there is a need to make a *cost function (fitness function)* considering desirable properties of Boolean function. Some of the well known cost functions used recently for construction of Boolean functions is discussed in subsections 4.1.1 and 4.1.2.

4.1.1 Cost function for Nonlinearity

In chapter 3, hill climbing method has been explained for construction of Boolean function with good cryptographic properties. In hill climbing method, we are targeting nonlinearity (NLT) to construct Boolean function. The objective function is taken as *fitness function*, i.e. the fitness of a function *f* on *n* input variables is given by

$$fitness(f) = N_f = \mathrm{nl}\,(f) = 2^{n-1} - \frac{1}{2}\max_{a \in \mathbb{F}_2^n} |\hat{f}(a)|$$

From above equation, to maximize nonlinearity, $\max_{a\in F_2^n}|\hat{f}(a)|$ has to be minimized. So, maximization of nonlinearity can be viewed as minimization of $\max_{a\in F_2^n}|\hat{f}(a)|$ (WHT). Therefore, the cost function is given by

$$cost(f) = WH_{max}(f) = \max_{a\in F_2^n}|\hat{f}(a)|$$

From Theorem 2.1 (Parseval's equation),

$$\sum_{a\in F_2^n} \hat{f}^2(a) = 2^{2n}.$$

This relation constrains $WH_{max}(f) = \max_{a\in F_2^n}|\hat{f}(a)|$ to be at least $2^{\frac{n}{2}}$. This bound can only be achieved when, for each a, $|\hat{f}(a)| = 2^{\frac{n}{2}}$. When some $|\hat{f}(a)|$ are less than $2^{\frac{n}{2}}$, then Parseval's Theorem ensures that some $|\hat{f}(a)|$ is greater than $2^{\frac{n}{2}}$. Thus minimizing the spread of WHT seems to be a possible means for achieving good nonlinearity. For each a, $|\hat{f}(a)| = 2^{\frac{n}{2}}$ is achieved by bent function. But bent functions are not balanced, exist for only even number of input variables and also have zero autocorrelation [29]. Therefore, considering the minimization of spread of Walsh Hadamard Transform, a cost function can be

$$cost(f) = \sum_{a\in F_2^n} \left||\hat{f}(a)| - 2^{\frac{n}{2}}\right| \tag{4.1}$$

This cost function is a simple candidate for targeting nonlinearity and autocorrelation. Functions having, $|\hat{f}(a)| = 2^{\frac{n}{2}}$, for all a, must also have $\hat{f}(\underline{0}) = 2^{\frac{n}{2}}$. But, for balanced functions, $\hat{f}(\underline{0}) = 0$ from Equation 2.12. So functions that achieve $|\hat{f}(a)| = 2^{\frac{n}{2}}$, for each a, cannot be balanced. Thus, a new cost function for balanced Boolean function can be

$$cost(f) = \sum_{a\in F_2^n} \left||\hat{f}(a)| - X\right|^R \tag{4.2}$$

where X and R are variable parameters. It is difficult to predict the best values of X and R for Boolean functions that are balanced and those with odd number of variables. Some parametric flexibility is justified. With the help of parametric variations of X and R, one can make variations on cost functions and it has been shown by Clark et. al. [1, 6] that it is possible to get Boolean functions with desirable properties by making parametric variations. Clark et. al. have taken X ranging from -16 to 30 and mostly R =3 in their search[1,6].

Nonlinearity and autocorrelation criteria can be handled by the above cost function. Taking into consideration of balancedness, starting function can be taken as balanced Boolean function, and algebraic degree and algebraic immunity are ignored. Simulated annealing algorithm is applied as an optimization technique on this cost function. Algebraic degree and algebraic immunity criteria can be considered in "Stopping Criteria" of Simulated Annealing algorithm.

4.1.2 Cost function for Autocorrelation

In hill climbing method, we are targeting autocorrelation (ACT) to construct Boolean function. In this case, the objective function is taken as *cost function*, i.e. the cost of a function *f* on *n* input variables is given by

$$cost(f) = AC_f = max_{s\neq 0}\left|\sum_x f'(x)f'(x+s)\right| \qquad (4.3)$$

Some modification in *cost function* related to autocorrelation has been done by Zhang and Zheng in 1995[30]. A new *cost function* known as the sum of squares measure σ_f (considering all values of autocorrelation function $\hat{r}_f(s)$) has been introduced by Zhang and Zheng [30]

$$\sigma_f = \sum_{s=0}^{2^{n-1}} \hat{r}_f(s)^2$$

Constructions for both even and odd *n* are offered by Zhang and Zheng.

A typical optimization approach to multi-criteria problems is to take a weighted sum of the individual cost functions. For the target criteria, this would lead to consideration of cost functions like

$$Cost(f) = \alpha\, WH_{max}(f) + \beta\, AC_f + \gamma\, (n\text{-}Degree(f)) + \delta\, Imbalance(f)$$

where α, β, γ and δ are weighted values, *Degree(f)* shows the value of degree of Boolean function and *Imbalance(f)* shows the difference of number of zero's of Boolean function from balanced Boolean function.

4.2 Simulated Annealing Algorithm for Boolean function

An example of application of simulated annealing can be shown in cryptography for construction of Boolean function with desirable properties. To obtain Boolean function with optimum tradeoff in properties, optimization of a cost function has to be done. It becomes exhaustive to search Boolean function with optimum value of cost function because of large no. of states of Boolean functions. Simulated annealing is one of the guided search method used as a solution of this problem. This method used to construct Boolean function is described below.

The simulated annealing algorithm is shown in Figure 4.1[1,6]. Consider a function *f(S)* is varying with state S. Let search starts at some initial state $S = S_0$. We define state *S* by storing outputs of Boolean function in an array. There is a control parameter *T* known as "synthetic temperature". There is another parameter *J* known as State Selecting Parameter. For each value of *J*, we choose the new state *N*(*S*) from the neighborhood of previous state *S* and we move a number *MIL* (Moves in Inner Loop) of moves to new states. The change in value, $\delta(f(Y)-f(S))$, of *f* is calculated. If it improves the value of *f*(*S*) (i.e., if $\delta < 0$ for a minimization problem) then a move to that state is taken ($S = Y$); if not, then it is taken with some probability. The probability acceptance has been done by generating a random value *U(0,1)* and performing the indicated comparison in Figure 4.1. The algorithm terminates when the stopping criterion is met. The common stopping criteria is met when some maximum number *MaxIL* (Maximum number of Inner Loop) of consecutive unproductive inner loops have been executed. In between measurement of desirable properties has also been done and stored at each state *S*. The basic simulated annealing algorithm has proven remarkably effective over a range of problems [6].

```
$S = S_0$
$J = J_0$
Repeat
{
for ( int $i = 0$; $i < MIL$; $i$++)
    {
        Select $Y \in N(S)$
        $\delta = f(Y) - f(S)$
        if ($\delta < 0$) then
```

```
                    $S = Y$
            else
                    Generate $U = U(0, 1)$
                    if ($U < \exp(-\delta/T)$) then $S = Y$
        }
    $J=J+MIL$
    }
    Until stopping criterion is met
```

Fig. 4.1 Basic Simulated Annealing for Minimization Problems

4.3 Simulated Annealing and Hill climbing

Search has been conducted for both balanced and nonbalanced Boolean function. For balanced Boolean function, a valid move simply swaps two dissimilar vector elements and so preserves the balancedness of Boolean function. In formal terms, we can define the neighborhood of a function $f(x)$ as follows. The function $g(x)$ is in the neighborhood of $f(x)$ if for two inputs x_1 and x_2, $g(x_1) = f(x_1) + 1$ and $g(x_2) = f(x_2) + 1$. The approach is as follows:

1. Use an annealing-based search to minimize the value of cost function. Let the best solution produced during the search be $f_{sa}(x)$.
2. Hill-climb from $f_{sa}(x)$ with respect to nonlinearity or autocorrelation or algebraic immunity (we shall term these the Non-Linearity Targeted (NLT) and Auto-Correlation Targeted (ACT) respectively) to produce the final solution $f_{sahc}(x)$.
3. Measure the nonlinearity, autocorrelation, algebraic degree and algebraic immunity of $f_{sahc}(x)$.

"Stopping Criteria" of Simulated annealing method can be used for targeting cryptographic properties [1,6].

Chapter 5
RESULTS AND DISCUSSION

The following properties of Boolean function have been targeted in our search:

- nonlinearity
- autocorrelation
- algebraic immunity

Following heuristic methods have been used for search:

(i) Hill climbing method.

(ii) Simulated Annealing method.

(iii) Simulated Annealing and hill climbing

Search has been conducted for both balanced and nonbalanced Boolean functions. The search programme is written in C language. In this program, we have defined the state S of an n-variable Boolean function by storing the truth table outputs of the Boolean function in an 2^n element array. For example, for a 5 variable Boolean function, the state can be stored as (01100101010001100010100011000111) in the form of an array.

The starting Boolean function has been taken with initial state S_0. This state has been chosen randomly by taking random runs of 0's and 1's. Consider an n-variable Boolean function with state S_0 = (0011....1010). We calculate the properties of Boolean function at state S_0. To choose the neighbor state, we randomly select two positions x_1 and x_2, $1 \leq (x_1, x_2) \leq 2^n$, in the array and change the value of Boolean function at those positions (i.e. if '0' then change to '1' and if '1' then change to '0'). Let the state obtained after the change be S_1. Next, we calculate properties at state S_1. Following the same procedure as above, we obtain states S_2, S_3, ... and so on and test these for the Boolean properties.

To ensure that all successive states correspond to balanced Boolean functions, we chose states by simply testing all pairs (x_1; x_2) such that the values at position x_1 and x_2 are not equal i.e. $f(x_1) \neq f(x_2)$.

5.1 Search Results for Hill Climbing Method

Hill climbing method has been described in section 3.2. Let us assume that present state is S_i. We calculate properties of Boolean function at state S_i. Next, we select S_j in the neighbor of State S_i and calculate the properties at state S_j. If the targeted property improves with the change of state then, state S_j is stored and used as the present state. If the targeted property does not improve with the change of state then, state S_j is not stored and previous state S_i is used to search new state. This is continued till we reach state S_f which achieves the target value of the property. The final state obtained by search is stored in a file and corresponds to the desired Boolean function.

Higher values of the targeted property are obtained with a compromise in other properties. For example, for an 8-variable Boolean function, we get a maximum nonlinearity of 114, with autocorrelation of 56 and algebraic degree of 1, whereas much lower autocorrelation (24) and higher algebraic degree (8) values may be obtained by sacrificing nonlinearity.

5.1.1 Results for Nonlinearity Targeted (NLT) Search

In this method, we search for Boolean functions with higher nonlinearity. A target value of nonlinearity is set and the search proceeds till this value is achieved. Table 5.1 shows the best values of nonlinearity obtained using NLT without consideration of balancedness. The best values of nonlinearity obtained for balanced Boolean function are shown in Table 5.2. Table 5.2 also compares the results for balanced Boolean function reported by other authors. We observe that other authors have obtained significantly better results than us for n > 8. The possible reason can be the use of starting Boolean function with proper nonlinearity or use of other techniques or exhaustive search.

n	5	6	7	8	9	10	11	12
nl	12	26	54	114	234	481	972	1976

nl stands for nonlinearity

Table 5.1 Best values of Nonlinearity obtained using NLT

n	5	6	7	8	9	10	11	12
Dobertin's Conjecture[18]		26		116		492		2010
Genetic Algorithms[31]	12	26	56	116	236	484	980	1976
Clark[6]	12	26	56	116	238	486	984	1992
Our Results	12	24	54	114	234	481	972	1974

Table 5.2 Comparing the Nonlinearity of balanced Boolean functions

5.1.2 Results for Autocorrelation Targeted (ACT) Search

In this method, we search for Boolean functions with lower autocorrelation. A target value for autocorrelation is set and the search proceeds till this value is achieved. Table 5.3 shows the best values of autocorrelation obtained using ACT without considering balancedness. Table 5.4 shows best values of autocorrelation obtained in our search for balanced Boolean functions and compares the results with those of [30] and [17].

n	5	6	7	8	9	10	11	12
ac	8	16	18	24	40	56	88	132

ac stands for autocorrelation

Table 5.3 Best values of autocorrelation obtained using ACT

n	5	6	7	8	9	10	11	12
Zhang and Zheng[30]	8	16	16	24	32	48	64	96
Maitra Construction [17]	8	16	16	24	32	40	64	80
Maitra Conjecture [17]		16		24		40		80
Clark[6]	8	16	16	16	40	56	88	128
Our Results	8	16	18	24	42	62	94	140

Table 5.4 Comparing the Autocorrelation of balanced Boolean functions

We observe that our results are much worse than those of others for n > 8. The possible reason can be use of starting Boolean function with proper autocorrelation or use of other techniques or exhaustive search.

5.2 Search Results for Simulated Annealing Method

In this method, we perform the search by varying X and R parameters of cost function of Equation 4.1. Table 5.5 shows X and R values used together with the parameters of the annealing algorithm (section 4.2). MIL is the number of Moves in Inner Loop. MaxIL represents the Maximum number of Inner Loops used for the search. The search proceeds as follows. At each value of *J*, the state selecting parameter, we select the new state S_j from the neighbor of previous state S_i. The properties of Boolean function are measured at the end of each inner loop and stored in a file. Some results among these stored results are selected. The best values of autocorrelation, nonlinearity and algebraic immunity obtained for Boolean functions with the corresponding values of X and R is shown in Tables 5.6, 5.7 and 5.8.

n	X Range (min→max)	R values	MIL	MaxIL
5	(-10→10)	3.0	50-100	400
6	(-10→10)	2.5, 3.0	50-100	300-500
7	(-10→15)	2.5, 3.0	100-200	200-300
8	(-16→16)	2.0, 2.5, 3.0	100-200	200-300
9	(-16→20)	2.0, 2.5, 3.0	100-200	200-300
10	(-20→32)	2.0, 2.5, 3.0	50-100	200-300
11	(-20→40)	2.0, 2.5, 3.0	50-100	100-200
12	(-30→64)	2.0, 3.0	50-100	100-200

Table 5.5 Search Parameters used

n	5	6	7	8	9	10	11	12
nl	12	26	56	116	234	480	970	1990
(X,R)	(10,3)	(10,3)	(15,2.5)	(16,2.5)	(20,3)	(25,2.5)	(30,2.5)	(35, 3)

nl stands for nonlinearity

Table 5.6 Best values of nonlinearity obtained for Boolean functions

n	5	6	7	8	9	10	11	12
ac	8	16	18	22	42	58	94	134
(X,R)	(10,3)	(10,3)	(12,3)	(14,3)	(20,2.5)	(30,3)	(25,2.5)	(30,2.5)

ac stands for autocorrelation

Table 5.7 Best values of autocorrelation obtained for Boolean function

n	5	6	7	8	9	10	11	12
ai	3	3	4	4	5	5	6	6
(X,R)	(10,3)	(10,3)	(12,3)	(16,2.5)	(20,2.5)	(30,3)	(25,2.5)	(30,2.5)
	(5,2.5)	(8,2)	(15,2)	(14,3)	(15,2)	(25,2)	(30,3)	(36,3)

ai stands for algebraic immunity

Table 5.8 Best values of algebraic immunity obtained for Boolean function

It may be observed that it is possible to get the same values of algebraic immunity for different sets of values of X and R. But it is not true for nonlinearity or autocorrelation. The possible reason might be that algebraic immunity is independent of X and R parameters.

Table 5.9 and 5.10 show the best values of nonlinearity and autocorrelation obtained in our search for balanced Boolean functions alongwith a comparison with the results reported by other authors. It is apparent from Table 5.9 that for $n \leq 8$, our search results has the same nonlinearity as those of others results while the autocorrelation results are also nearby same. But for $n > 8$, other authors have obtained significantly better results than us. The possible reason can be use of starting Boolean function with proper value of property or use of other techniques or exhaustive search.

n	5	6	7	8	9	10	11	12
Dobertin's Conjecture[18]		26		116		492		2010
Genetic Algorithms[31]	12	26	56	116	236	484	980	1976
Clark[6]	12	26	56	116	238	486	984	1992
Our Results	12	26	56	116	234	474	970	1974

Table 5.9 Comparing the Nonlinearity of balanced Boolean functions

n	5	6	7	8	9	10	11	12
Zhang and Zheng[30]	8	16	16	24	32	48	64	96
Maitra Construction[17]	8	16	16	24	32	40	64	80
Maitra Conjecture[17]		16		24		40		80
Clark[6]	8	16	16	16	40	56	88	128
Our Results	8	16	18	24	42	58	94	134

Table 5.10 Comparing the Autocorrelation of balanced Boolean functions

5.3 Search Results for Simulated Annealing and Hill Climbing Method

This method has been described in section 4.3. Consider Fig. 4.1 of Simulated Annealing algorithm. In this algorithm, properties of Boolean function are calculated at the end of each inner loop and "Stopping Criteria" is used for targeting values of properties.

Clark [6] has not considered algebraic immunity as a constraint for search of Boolean function. Clark has considered only nonlinearity and autocorrelation as constraints for search of Boolean function. We have searched for Boolean functions with highest achievable algebraic immunity and for Boolean functions with tradeoff among nonlinearity, autocorrelation and algebraic immunity. The best results of Boolean function obtained during search with tradeoff among properties are shown in Table 5.11.

(5,3,12,8,3) balanced Boolean function is the best achievable 5-variable Boolean function, where quadruplet *(n, d, nl, ac, ai)* represents *n* variable Boolean function with algebraic degree *d*, nonlinearity *nl*, autocorrelation *ac* and algebraic immunity *ai*. This Boolean function has highest achievable nonlinearity (12), lowest achievable autocorrelation (8), and highest achievable algebraic immunity (3) and is balanced also. However, it has a lower algebraic degree of 3, compared to the maximum possible value of 4 for a 5-variable balanced Boolean function. For $n \leq 8$ variables, we are getting better Boolean function than for $n > 8$ variables. Some Boolean functions of Table 5.12 are given in Appendix.

(5,3,12,8,3)	(6,5,24,16,3)	(7,6,54,18,3)	(8,7,110,40,4)
(9,8,232,59,4)	(10,10,479,80,5)	(11,9,970,110,6)	(12,11,1974,176,6)

where quadruplet *(n, d, nl, ac, ai)* represents *n* variable Boolean function with al degree *d*, nonlinearity *nl*, autocorrelation *ac* and algebraic immunity *ai*.

Table 5.11 Best values (*n*, *d*, *nl*, *ac*, *ai*) with a tradeoff among nonlinearity, autocorrelation and algebraic immunity

5.3.1 Results for Nonlinearity Targeted (NLT) Search

In this method, we search for Boolean functions with higher nonlinearity. Table 5.12 shows the results obtained using NLT and a comparison with results of [6] and [7] are shown in Table 5.13. These results are obtained by considering the tradeoff among nonlinearity, autocorrelation and algebraic degree. We observe that some of our results are same as those of other authors and one of them is better than other author's results. This might be because of collective use of simulated annealing and hill climbing method. Boolean function (5,4,12,8) is better than what both [6] and [7] report.

(5,3,12,8)	(6,4,26,16)	(7,6,56,28)	(8,1,118,62)
(5,4,12,8)	(6,5,26,16)		(8,7,112,37)
(9,8,234,64)	(10,9,481,80)	(11,9,970,110)	(12,11,1974,176)
(9,8,232,59)			

where quadruplet *(n, d, nl, ac)* represents *n* variable Boolean function with degree *d*, nonlinearity *n* and autocorrelation *ac*.

Table 5.12 Results using NLT (Nonlinearity Targeted)

J.A. Clark[6]	(5,3,12,8) (5,4,12,16)	(6,5,26,16)	(7, 6, 56, 16)	(8,7,116,24) (8,5,112,16)
	(9,8,238,40)	(10,9,486,72) (10,9,484,64)	(11,9,984,96) (11,10,982,96)	(12,11,1974,176) (12,10,1990,144)
Multiobjective Approach [7]	(5, 3, 12, 8) (5, 4, 12, 16) (5, 4, 10 8)	(6, 5, 26, 16)	(7,5,56,16) (7,6,54,16)	(8,7,116,24) (8,5,112,16)
Our Results	(5,3,12,8) (5,4,12,8)	(6,5,26,16) (6,5,24,16)	(7,6,56,28) (7,6,54,18)	(8,1,118,62) (8,7,112,37)
	(9,8,234,64) (9,8,232,59)	(10,9,481,80)	(11,9,970,110)	(12,11,1974,176) (12,11,1972,176)

where quadruplet *(n, d, nl, ac)* represents *n* variable Boolean function with degree *d*, nonlinearity *n* and autocorrelation *ac*.

Table 5.13 Comparison of results using NLT (Nonlinearity Targeted) with a tradeoff among nonlinearity, autocorrelation and algebraic degree.

5.3.2 Results for Autocorrelation Targeted (ACT) Search

In this method, we search for Boolean functions with lower autocorrelation. Table 5.14 shows the results obtained by targeting autocorrelation. For n ≤ 7, we are getting best results obtained in known literature with a tradeoff among algebraic degree, autocorrelation and nonlinearity. Table 5.15 shows the comparison of our results those of [6] and [7] using ACT. These results are obtained by considering the tradeoff among nonlinearity, autocorrelation and algebraic degree. We can observe from Table 5.15 that for n ≤ 7, our results are better than other results.

(5,3,12,8) (5,4,12,8)	(6,5,24,16)	(7,6,54,18)	(8,7,112,32) (8,7,112,37)
(9,8,232,59)	(10,9,479,71)	(11,9,970,110)	(12,11,1974,176)

where quadruplet *(n, d, nl, ac)* represents *n* variable Boolean function with degree *d*, nonlinearity *n* and autocorrelation *ac*.

Table 5.14 Results using ACT (Autocorrelation Targeted)

J.A. Clark[6]	(5,3,12,8) (5,4,12,16)	(6,5,26,16)	(7, 6, 56, 16)	(8,7,116,24) (8,5,112,16)
	(9,8,238,40)	(10,9,484,56)	(11,10,982,88)	(12,11,1986,128)
Multiobje--ctive Approach [7]	(5,3,12,8) (5,4,10,8)	(6,5,26,16)	(7,5,56,16) (7,6,54,16)	(8,7,116,24) (8,5,112,16)
Our Results	(5,3,12,8) (5,4,12,8)	(6,5,26,16) (6,5,24,16)	(7,6,54,18)	(8,7,112,32) (8,7,112,37)
	(9,8,232,59)	(10,9,479,71)	(11,9,970,110)	(12,11,1960,142)

where quadruplet *(n, d, nl, ac)* represents *n* variable Boolean function with degree *d*, nonlinearity *n* and autocorrelation *ac.*

Table 5.15 Comparison of results using ACT (Autocorrelation Targeted) with a tradeoff among nonlinearity, autocorrelation and algebraic degree.

5.3.3 Results for Algebraic Immunity Targeted (AIT) Search

In this method, algebraic immunity of Boolean function has been targeted. Table 5.16 shows the results obtained by targeting algebraic immunity. It is apparent from Table 5.16 that we have obtained maximum achievable algebraic immunity of $\lceil \frac{n}{2} \rceil$ for n-variable Boolean function. Some Boolean functions obtained during search using NLT and ACT also have good algebraic immunity.

n	5	6	7	8	9	10	11	12
ai	3	3	4	4	5	5	6	6

ai stands for algebraic immunity

Table 5.16 Best values obtained by targeting algebraic immunity

Chapter 6
CONCLUSION

Cryptography needs ways to find good Boolean functions so that ciphers can resist cryptanalytic attack. The main properties required to resist cryptanalytic attack are high nonlinearity, low autocorrelation and high algebraic immunity. The dissertation work has focused on study of properties of Boolean function and construction of Boolean function by heuristic approach for crypto-systems.

Heuristic Search is a simple search technique. Heuristic Search can be used to solve exhaustive search problems. In this dissertation work, heuristic approach has been used for search of Boolean functions with tradeoff among nonlinearity, autocorrelation and algebraic immunity. We have attained some results better than other techniques like evolutionary multiobjective approach [7], direct construction [4,5] and exhaustive search. The Boolean functions (5,3,12,8,3) and (6,5,24,16,3) obtained by heuristic search are unachievable to the best of my knowledge in known literature. We have got some results better than Clark[6] results in lower variables.

The range of properties addressed shows that heuristic search is a flexible framework for Boolean function investigation.

6.1 Future Work

Here, we come up with some proposals to continue the investigation performed in this dissertation study.

- We get good results for $n \leq 8$ variable Boolean functions. But the limitations of the techniques become apparent when one attempts to generate functions with nine variables and above. So, there is a need to find better cost function for $n \geq 9$.

- ➢ The cost function considered in this dissertation does not asssumed algebraic immunity and algebraic degree. So, one has to determine general relationship between nonlinearity and algebraic immunity and evolve better cost function.

Other criteria which have not dealt within this work include:

- Correlation immunity (An n-variable Boolean function is m-th order correlation immune if there is no change in probability distribution of its output when any of its m inputs is kept constant)

REFERENCES

[1] J.A. Clark, "Metaheuristic Search as a Cryptological Tool", Ph.D. Thesis, University of York, UK, Dec. 2001.

[2] W. Stallings, "Cryptography and network security: principles and practice", 4th Edition , Prentice Hall, 2006.

[3] K.C. Gupta, “Cryptographic and combinational properties of Boolean functions and S-boxes”, Ph. D. thesis, Indian Statistical Institute, India, 2004.

[4] T. Siegenthaler, “Correlation-immunity of nonlinear combining functions for cryptographic applications”, *IEEE Transactions on Information Theory*, Vol. IT-30, pp.776-780, September, 1984.

[5] C. Carlet, “On secondary construction of resilient function and bent functions”, In *Progress in Computer Science and Applied Logic*, Vol.23, pp. 3-28, Birkhauser-Verlag, 2004.

[6] J.A. Clark, J.L. Jacob, S. Stepney, S. Maitra, and W. Millan, “Evolving Boolean Functions Satisfying Multiple Criteria”; Lecture Notes in Computer Science, Vol.2551, pp. 246-259, 2002.

[7] Hernan Aguirre, Hiroyuki Okazaki and Yasushi Fuwa, “An Evolutionary Multiobjective Approach to Design Highly Non-linear Boolean Functions”, *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pp. 749-756, July, 2007, London, England.

[8] S. Kirkpatrick, Jr.C.D. Gelatt, and M.P. Vecchi, "Optimization by Simulated Annealing". *Science*, Vol. 220, pp. 671-680, May 1983.

[9] W. Millan, A. Clark and E. Dawson, “Boolean Function Design using Hill climbing Methods”, Lecture Notes in Computer Science, Vol.1587, pp. 1-11, Springer-Verlag, 1999.

[10] M. Lobanov, “Tight Bound Between Nonlinearity and Algebraic Immunity”, [Online]. Available: http://eprint.iacr.org/

[11] E. Elsheh, A. BenHamza and A. Youssef, “On the nonlinearity profile of cryptographic Boolean functions”, *Journal of Universal Computer Science*, Vol. 4, Issue No. 4, pp. 705–717, 1998.

[12] S.Lin, D.J. Costello, “Error Control Coding”, Englewood Cliffs, New Jersey, 1983.

[13] P.E. Dunne, "The Complexity of Boolean Networks", A.P.I.C. Studies in Data Processing No. 29, Academic Press, 1988.

[14] S.W. Golomb, "Shift Register Sequences". San Francisco: Holden-Day, 1967.

[15] G. Xiao and J.L. Massey, “A spectral characterization of correlation-immune combining functions”, *IEEE Trans. On Information Theory*, Vol. IT-34, pp. 569-571. 1988.

[16] F.J.M. Williams and N.J.A. Sloane, "The Theory of Error Correcting Codes", North Holland Publishing Company, Amsterdam, 1977.

[17] S. Maitra, "Highly nonlinear balanced Boolean functions with very good autocorrelation property", In *Workshop on Coding and Cryptography - WCC*, Electronic Notes in Discrete Mathematics, Vol. 6, Elsevier Science, Paris, January, 2001.

[18] H. Dobbertin, "Construction of bent functions and balanced functions with high nonlinearity", Lecture Notes in Computer Science, Vol. 1008, pp. 61–74, Springer-Verlag, 1994.

[19] N.T. Courtois, J. Pieprzyk, "Cryptanalysis of Block Ciphers with Overdefined Systems of Equations", Lecture Notes in Computer Science, Vol.2501, pp. 267-287, Springer-Verlag, 2002.

[20] C. E. Shannon, “Communication theory of secrecy systems,” *Bell Syst. Tech. J.*, Vol. 28, pp. 656–715, 1949.

[21] W. Meier, E. Pasalic, and C. Carlet, "Algebraic Attacks and Decomposition of Boolean Functions", Lecture Notes in Computer Science, Vol. 3027, pp. 474–491. Springer Verlag, 2004.

[22] N.T. Courtois and W. Meier, "Algebraic Attacks on Stream Ciphers with Linear Feedback", Lecture Notes in Computer Science, Vol. 2656, pp. 345-359, Springer Verlag, 2003.

[23] C. Carlet, D.K. Dalai, K.C. Gupta, and S. Maitra, "Algebraic Immunity for Cryptographically Significant Boolean Functions: Analysis and Construction", *IEEE Transactions on Information Theory*, Vol. 52, pp. 3105-3121, July, 2006.

[24] J.A. Clark and J.L. Jacob, "Two-Stage Optimisation in the Design of Boolean Functions", Lecture Notes in Computer Science, Vol. 1841, pp. 242–254. Springer Verlag. 2000.

[25] P. Sarkar, S. Maitra, "Construction of Nonlinear Boolean Functions with Important Cryptographic Properties", Lecture Notes in Computer Science, Vol. 1807, pp. 485–506. Springer Verlag. 2003.

[26] F. Glover and M. Laguna, "Tabu Search", Kluwer, Norwell, MA, 1997.

[27] S. Russell, P. Norvig, "Artificial Intelligence: A Modern Approach", 2nd Edition, Prentice Hall Series in Artificial Intelligence, 2003.

[28] W. Millan, A. Clark and E. Dawson, "Smart Hill Climbing Finds Better Boolean Functions", In *Workshop on Selected Areas in Cryptology*, *Workshop Record*, pp. 50–63, 1997.

[29] O.S. Rothaus, "On Bent Functions", *Journal of Combinatorial Theory: Series A*, Vol. 20, pp. 300–305, 1976.

[30] X-M. Zhang and Y. Zheng, "GAC – the criterion for global avalanche characteristics of cryptographic functions", *Journal of Universal Computer Science*, Vol.1, Issue No.5, pp. 316–333, 1995.

[31] W. Millan, A. Clark and E. Dawson, "Heuristic Design of Cryptographically Strong Balanced Boolean Functions", Lecture Notes in Computer Science, Vol. 1403, pp. 489–499. Springer Verlag. 1998.

APPENDIX

1. (5,3,12,8,3) Boolean function

The best result obtained in 5 variable Boolean function with optimum compromise in cryptographic criteria is given below:

00101110001011011001001101011100

This Boolean function is balanced also.
Algebraic degree: 3
Nonlinearity: 12
Autocorrelation: 8
Algebraic Immunity: 3

2. (6,5,24,16,3) Boolean function

The best result obtained in 6 variable Boolean function with optimum compromise in cryptographic criteria is given below:

0001000110010111011010001010000010100101001010100010101010011100

Algebraic degree: 5
Nonlinearity: 24
Autocorrelation: 16
Algebraic Immunity: 3

3. (7,6,54,18,3) Boolean function

The best result obtained in 7 variable Boolean function with optimum compromise in cryptographic criteria is given below:

0100001001001011111100111001010010001010010011101011101011101101101001110000011110010100100000100100101010100000001101110100011 1

Algebraic degree: 6
Nonlinearity: 54
Autocorrelation: 18
Algebraic Immunity: 3

4. (8,7,110,40,4) Boolean function

The best result obtained in 8 variable Boolean function with optimum compromise in cryptographic criteria is given below:

0100110111000101001010110001110100100111110100010110101001110111100000001101010101011110110001011001111111000000111100011101100100010000000110101010010111010100001100011110011110001001101000100100011111110010001101101011001110011001001101100101001000110101 1

This Boolean function is balanced also.
Algebraic degree: 7
Nonlinearity: 110
Autocorrelation: 40
Algebraic Immunity: 4

5. (9,8,232,59,4) Boolean function

The best result obtained in 9 variable Boolean function with optimum compromise in cryptographic criteria is given below:

01110111101011001010001011111001101010010011000000111101100011110111110011011101110101000010101010111101000101011111011110100000000101110101110000111110100001111111100110000000100011000000110110100001000100110011100001010101011010100100000010100010010000000100100000101001010111000001011011

1001010001010001000100101101110110001100001110011110101111001001000010000
011001111101010110101001000101000010000001101001011010011100110001111100
110000010111001011100101010001100010110010010111001111000101001011001010
11100001

Algebraic degree: 8

Nonlinearity: 232

Autocorrelation: 59

Algebraic Immunity: 4

6. (10,10,479,80,5) Boolean function

The best result obtained in 10 variable Boolean function with optimum compromise in cryptographic criteria is given below:

0110001000110000010101101000010001011011110101101110011100001110111111010
110001001010111101111011001000100101111110011001110101110110100100010111
110000000101011100010100100110011110010101101111000100001100000011000000
1110000001011000000010000011100010110101010110100010001110111010001111111
110110110000011001000100011110001111100010010010100001000010010101111000
110001011100111110011011100011011111001100111101111100111010010001000111
010010010111100011111111010111111010001001100111100100110110110010011001
101000011111111110101011100110100101101101011101110101100101001011010110
100011010110010101100000001111111100101011001110011101001011010011011100
000101011101010110100110011100111111001000111100101110000011001111100001
010010110010001100100011000110110010011111111010110010101011011010011010
001111111110111100111110101001100000010001100011100111010111000100001100
100111000011111101001001011101011001101010101111011100100100001111011011
100111010011110011011000100010001010101001011110101111010000101111010011
1001001110011101010110000

Algebraic degree: 10

Nonlinearity: 479

Autocorrelation: 80

Algebraic Immunity: 5