Design Strong Boolean Functions Using Memetic Algorithm

Ismail Khalil. Ali
Department of Software Engineering
Baghdad College of Economic Science University
Boolean functions are used as nonlinear combining functions in certain stream ciphers. The design of Boolean functions with good combinations cryptographic properties remains an important research challenge. Advances heuristic optimization design is one option for achieving this goal. This paper concentrates to investigate the possibilities of memetic algorithms for creating cryptographic properties required by strong Boolean functions. The main properties required are (balance, high nonlinearity, and low autocorrelation. It will be noticed that, genetic algorithm with hill climbing algorithm can compete with theoretical construction for functions with number of variables and can be applied to good effect.

**Keywords:** Boolean functions, memetic algorithm, genetic algorithm, hill climbing algorithm.

1. **Introduction**

Cryptography needs ways to find good Boolean functions so that ciphers can resist cryptanalytic attack. The main properties required are high nonlinearity and low autocorrelation, so that linear cryptanalysis [7] and differential cryptanalysis [5] do not succeed faster than exhaustive key search.

Boolean functions play a central role in the design of most symmetric cryptosystems and in their security. In stream ciphers, Boolean functions may be used to combine the outputs of several bit streams. Figure 1 illustrates a classic stream cipher model. The plaintext stream Pn of bits is XOR-ed with pseudo-random bit stream Kn to give a cipher text stream Cn. The plaintext is recovered by the receiver by XOR-ing the cipher stream with the same pseudo-random stream. The pseudo-random stream is formed from several bit streams generated by Linear Feedback Shift Registers (LFSRs) using a suitable combining function f. The initial state of the registers forms the secret key. The function f must be sufficiently complex that cryptanalysts will not be able to determine the initial state of the registers, even when they know what the plaintext is (and so can recover the pseudorandom keystream Kn).
An $n$-variable Boolean function $f$ is a function from the set $Z_2^n$ of all binary vectors $x = (x_1, x_2, \ldots, x_n)$ of length $n$ to the field $Z_2 = \{0, 1\}$. The number $n$ of variables is rarely large in practice. However, determining and studying those Boolean functions, which satisfy some conditions needed for cryptographic uses, is not feasible through an exhaustive computer investigation, since the number of $n$-variable Boolean functions is too large when $n \geq 6$. However, clever computer investigations are useful for imagining or testing conjectures, and sometimes for generating interesting functions. Every $n$-variable Boolean function can be represented with its truth table. But the representation of Boolean functions, which is most usually used in cryptography, is the $n$-variable polynomial representation over $Z_2$. This polynomial representation is called the Algebraic Normal Form, in brief, ANF [4]. Table 1 shows the truth table of the Boolean function of three variables.

Table 1: The truth table of the Boolean function $f(x_1, x_2, x_3) = x_1 x_2 \oplus x_2 x_3 \oplus x_3$

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$X$</th>
<th>$f(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Many authors have attempted to use guided search techniques to evolve Boolean functions [1] [2] [3] [9] [10] [11]. In this paper, using memetic algorithms, we demonstrate how to evolve various functions with profiles of desirable properties.

2. Preliminaries

This section summarizes the basic cryptographic definitions needed. We shall denote the binary truth table of Boolean function by \( f: \mathbb{Z}_2^2 \rightarrow \mathbb{Z}_2 \) mapping each combination of \( n \) binary variables to some binary value.

**Balanced**: An \( n \) variable Boolean function \( f(x) \) is balanced if the output column in the truth table contains an equal number of 0's and 1's.

**Linear Boolean Function**: A linear Boolean function, selected by \( w \in \mathbb{Z}_2^n \), is denoted by:

\[
L_w(x) = w_1x_1 \oplus w_2x_2 \oplus \ldots \oplus w_nx_n
\]

**Affine Function**: The set of affine function is the set of linear functions and their complements, \( A_{w,c}(x) = L_w(x) \oplus c \)

**Walsh Hadmad Transform**: For a Boolean function \( f \) the Walsh-Hadamard Transform \( F \) is defined by:

\[
F(w) = \sum_{x \in \mathbb{Z}_2^n} f(x) L_w(x)
\]

We denote the maximum absolute value taken by transform by: \( WH_{\text{max}}(f) = \max_{w \in \mathbb{Z}_2^n} |F(w)| \) it is related to the non-linearity of \( f \).

**Non-linearity of \( f \)**: The non-linearity \( N_f \) of a Boolean function \( f \) is the minimum distance to any affine function. It is given by:

\[
N_f = \frac{1}{2} (2^n - WH_{\text{max}}(f))
\]

**Autocorrelation**: The autocorrelation \( AC_f \) of a Boolean function \( f \) is given by:

\[
AC_f = \max |\sum_x f(x)f(x \oplus s)|
\]

where \( x \) and \( s \) range over \( \mathbb{Z}_2^n \) and \( x \oplus s \) denotes bitwise XOR (and so produces a result in \( \mathbb{Z}_2^n \)).

3. Memetic Algorithm
The memetic algorithms [8] can be viewed as a marriage between a population-based global technique and a local search made by each of the individuals. They are a special kind of genetic algorithm with a local hill climbing. Like genetic algorithm, memetic Algorithms are a population based approach. They have shown that they are orders of magnitude faster than traditional genetic algorithm for some problem domains. In a memetic algorithm the population is initialized at random or using a heuristic. Then, each individual makes local search to improve its fitness. To form a new population for the next generation, higher quality individuals are selected. The selection phase is identical inform to that used in the classical search selection phase. Once two parents have been selected, their chromosomes are combined and the classical operators of crossover are applied to generate new individuals. The latter are enhanced using a local search technique. The role of local search in memetic algorithms is to locate the local optimum more efficiently then the traditional search. Figure 2 explains the generic implementation of memetic algorithm [6].

![Figure 2: The memetic algorithm](image)

### 3.1 Genetic Algorithm

A Genetic Algorithm (GA) (Holland, 1975) is an optimization method, which is inspired by nature, precisely speaking, by the process of biological evolution. Its three main mechanisms, *selection*, *recombination* (*crossover*) and *mutation* are applied iteratively to a set of solutions, called the *population*. During each iteration step, also called generation, a new population of solutions is generated by applying a *crossover operator*. This operator takes two solutions and produces one or more new solutions by recombining their elements in a certain predefined way. With a given probability, the resulting solutions may be subject to a subsequent mutation step, which slightly modifies solution components in a random manner. The probability of a solution to be chosen for crossover depends on its *fitness value*, which is strongly related to the solution's quality with respect to the objective function. A higher fitness
value yields a higher selection probability. The Genetic Algorithm we use for our experiments can be described as a Standard Genetic Algorithm, which has been modified in order to fit better into the problem environment. Figure 3 shows the structure of a GA [12]:

$$t = 0$$
$$\text{initialize } P(t)$$
$$\text{evaluate } P(t)$$
$$\text{while (not termination-condition) do}$$
$$\quad t = t + 1$$
$$\quad \text{select } P(t) \text{ from } P(t - 1)$$
$$\quad \text{crossover}$$
$$\quad \text{mutation}$$
$$\quad \text{evaluate } P(t)$$
$$\text{end while}$$

3.2 Hill climbing local search algorithm

The hill climbing search algorithm [11] is a local search, often also referred to as a (descent algorithm), is the simplest form of neighborhood local search and can be considered as a greedy algorithm. It accepts a new solution only if it improves the objective function value. The algorithm terminates if no improving solution can be found in some state. For our tests, we use a modified hill climber, which also allows neutral transitions with respect to the solution quality. The termination criterion is satisfied if neither an improving nor an equivalent, solution can be found or if a maximum number of neutral transitions have been reached. Figure 4 shown the hill climbing local search algorithm [6].

While (termination condition is not satisfied)
    do
        $\leftarrow$ New sol neighbors (best sol);
        If new sol is better the actual sol then
            $\leftarrow$ Bes sol actual sol
        End if
    End while

Figure 4: The Hill climbing local search algorithm

4. Designing Boolean Functions using Memetic Algorithms

The research reported concentrates on three criteria and investigates whether memetic algorithm can be applied to good effect. These criteria are balance, high nonlinearity, and low autocorrelation. These criteria, in
various combinations, have proven of interest to cryptological researchers (from both theoretical and optimization perspectives). The implementation of the memetic algorithm has to be taken. They will be discussed first.

### 4.1 Solution encoding

The classical bit string representation is well suited for this problem. A Boolean function is simply represented by its truth table, where the inputs are lexicographically ordered, thus $(0, \ldots, 0), (0, \ldots, 0, 1), (0, \ldots, 0, 1, 0), (1, \ldots, 1)$. An array of integers is used to represent the table. An integer can typically hold 32 bits. The truth table of a Boolean function on $n$ variables contains $2n$ bits. The number of doubles needed to represent the truth table thus equals $2^n / 32$.

### 4.2 Initial population

The initial population is created randomly, hereby ensuring that the created functions are balanced. Therefore $2^{n-1}$ different bit positions are chosen to be 1, the others being 0.

### 4.3 Selection procedure

Tournament selection is used in this work. This method randomly selects a number $k$ of individuals and selects the best one from this set into the next generation.

### 4.4 Genetic operators

The choice of the genetic operators is important for the convergence of the algorithm. In [9], it is stated that classical crossover, as well as XOR-ing the individuals, do not facilitate convergence to good solutions. They propose a merge operator, which shows to be effective. It is therefore adopted in our algorithm. The merge operation is defined as [10].

### 4.5 Fitness function

Optimization-based work aimed at producing highly nonlinear functions has generally used nonlinearity itself as the fitness function, i.e. the fitness of a function $f$ on $n$ input variables is given by:

$$fit(f) = N_f = \frac{1}{2} (2^n - WH_{max}(f))$$
Similarly, with low autocorrelation as the target, the autocorrelation itself has been used as the fitness function, i.e. the fitness function is given by:

$$fit(f) = AC_f = \max_s \left| \sum_x f(x) f(x \oplus s) \right|$$

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

$$Fitness(f) = \frac{1}{2} \left( 2^n - WH_{\text{max}}(f) \right) + \max_s \left| \sum_x f(x) f(x \oplus s) \right|$$

If further criteria were of interest fitness function components for these would typically be added. Increasing the number of components will generally entail a great deal of experimentation to determine optimal settings of the component weights.

### 4.6 Parameters

The choice of the parameters of the algorithm will be left open. Since the only nontrivial decision is the use of the merge operator, the algorithm has largely the same structure as the one used by Clark et al in [9] and in [10]. The hill climbing procedure is described in [11], it comes down to finding the pairs of truth table positions for which the nonlinearity improves or stays equal when swapping their outputs.

All tests were run with population size between 20-30, and the results are averaged over 20 independent runs. The parameters for using GA are summarized in Table 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population Size</td>
<td>20</td>
</tr>
<tr>
<td>Selection</td>
<td>Tournament method</td>
</tr>
<tr>
<td>Crossover</td>
<td>merge operator</td>
</tr>
<tr>
<td>Crossover Rate</td>
<td>0.7</td>
</tr>
<tr>
<td>Mutation Rate</td>
<td>0.1</td>
</tr>
</tbody>
</table>

### 4.7 Proposed Algorithm

The overall proposed algorithm works as follows:

1. Generate $P$ random balanced functions and calculate their fitness.
2. For $i = 1$ to MAX_GEN do:
   a. For all pairings of the current generation, perform the merge operator to produce $P (P - 1)/2$ children.
   b. Apply mutation to each of the children.
   c. Apply hill climbing to each of the children.
d. Calculate the fitness for each of the children.
e. Select the best $P$ individuals from the current generation and the children, removing doubles. This forms the new generation.

3. Output the best solution from the current generation.

5. Experimental Results

A variety of runs have been carried out. Our interest is primarily in demonstrating the profiles of properties of the functions generated by nonlinearity and autocorrelation methods. The best profiles are recorded in Table 3. The entry $(n, nl, ac)$ indicates that the memetic technique is able to evolve a function on $n$ inputs with nonlinearity ($nl$) and autocorrelation ($ac$).

For $n$ less than or equal to 7 the technique has generated functions with the highest achievable nonlinearity values. For $n=8$, the function with nonlinearity of 116 has been demonstrated. The evolution of functions with profile $(8,116,24)$ is of particular interest.

Table 3: The best profiles are recorded of $n$-variable Boolean

<table>
<thead>
<tr>
<th>$n$</th>
<th>$nl$</th>
<th>$ac$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>26</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>56</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>116</td>
<td>24</td>
</tr>
<tr>
<td>9</td>
<td>238</td>
<td>32</td>
</tr>
<tr>
<td>10</td>
<td>484</td>
<td>56</td>
</tr>
<tr>
<td>11</td>
<td>984</td>
<td>88</td>
</tr>
<tr>
<td>12</td>
<td>1992</td>
<td>96</td>
</tr>
</tbody>
</table>

The number of generations is kept at 100 in every run. To illustrate this choice, consider Figure 5. The figure shows the average and the maximum nonlinearity of the population at each generation. The number of variables was 10, the population size 20. Results were averaged over 20 runs. Most improvement seems to occur in the first 20-30 generations. Afterwards the algorithm occasionally finds a better function. To make sure most of the convergence has occurred, while making the computational load not too high, the number of generations is set at 100.
Figure 5: Nonlinearity as a function of the generation number, \( n = 10 \)

**Example 1:**

For \( n = 5 \)

\[ 0,0,1,1,1,1,0,0,0,0,1,1,1,0,0,0,0,1,1,0,0,1,1,1,1,0,0, \]

Non-Linearity: 12
Auto-Correlation: 8

**Example 2:**

For \( n = 6 \)

\[ 1,1,0,0,0,1,1,1,0,1,1,0,0,1,1,1,0,1,0,0,0,0,0,0,0, \]

Non-Linearity: 26
Auto-Correlation: 16

**Example 3:**

For \( n = 7 \)

\[ 0,0,1,1,1,0,1,1,1,0,1,0,0,0,0,0,1,0,0,1,0,1,0,1,0,0,1,1,0,1, \]

Non-Linearity: 56
Auto-Correlation: 16

**Example 4:**

For \( n = 8 \)

\[ 0,1,0,1,0,0,0,1,1,1,1,1,1,1,1,1,0,1,0,1,1,1,0,0,0,0, \]

Non-Linearity: 102
Auto-Correlation: 16

**Auto-Correlation:**

- For \( n = 5 \): 8
- For \( n = 6 \): 16
- For \( n = 7 \): 16
- For \( n = 8 \): 16
6. Conclusions

The work reported in this paper has shown how memetic algorithm can be used to generate strong Boolean functions for cryptographic applications, and this algorithm can be a good design tool of cryptographically good Boolean functions. The problem of finding cryptographically strong Boolean functions contains some difficulties for GAs. The search space is vast, and increases exponentially with the number of variables. Different, possibly conflicting, criteria have to be incorporated in one fitness function. Further, it is clear how to combine two good functions into one new, with equal or better properties. The algorithms in this paper looked for combinations of nonlinearity and autocorrelation, \((nl, ac)\) profiles.

A memetic algorithm is an extension of the traditional genetic algorithm. It is based on a genetic algorithm extended by a local search technique to further improve individual’s fitness that may keep high population, diversity and reduce the likelihood of premature convergence. The experimental results show that memetic algorithm is slightly superior for finding the main criteria related with this study (balance, high nonlinearity, and low autocorrelation).
References


