Generating of homogeneous Boolean functions with high nonlinearity using genetic algorithm

Mohannad Najjar

University of Tabuk, Tabuk, KSA

Summary
In this paper an advanced genetic algorithm is used to find the maximum nonlinearity of *n*-argument Boolean homogenous functions. The best nonlinearity for Boolean functions number of arguments *n* was searched, where *n*= 8, 10, 12, 14 and for the degree *k* of homogeneity such that $1<k<n$. The experimental results that were reached prove that our genetic algorithm is a powerful tool to search for high nonlinearity. Results are presented in diagrams and tables.

Key words:
Cryptography, homogenous Boolean functions, bent functions, hash functions, block ciphers, S-boxes, Genetic Algorithms.

1. Introduction

In the paper the nonlinearity of n-argument Boolean homogenous functions that are used as components of hash functions was tested. The examples of hash functions of this kind are:

1. MD4 and SHA-1 that use two homogenous functions:
   - $g(x,y,z)= yz \oplus xz \oplus xy$ – 3-argument homogenous function of degree 2
   - $h(x,y,z)= x \oplus y \oplus z$ – 3-argument homogenous function of degree 1
2. MD5 use $h(x,y,z)= x \oplus y \oplus z$ – 3-argument homogenous function of degree 1.
3. PETRA hash function

Boolean functions are powerful tools for creating cryptographic algorithms (they are appropriate if they have good cryptographic characteristics). The nonlinearity is the most important characteristic of cryptographically strong Boolean functions. When *n* is even the *n*-argument functions that have the maximum nonlinearity equal to $2^{n-1}-2^{(n/2)-1}$ are called bent (or perfect).

In the design of product ciphers two basic components are used: substitution boxes and permutation boxes. A careful design of both helps to obtain more secure cryptographic algorithms. Each substitution box can be viewed as a set of Boolean functions. On the other hand, for cryptographic algorithms based on the structure used in MD4 and SHA-1 hash functions, homogenous Boolean functions can be useful cryptographic components [2].

Manuscript received March 5, 2010
Manuscript revised March 20, 2010

2. Nonlinearity of Boolean functions

Boolean function can be presented in different ways.
If the function *f* is presented as
$f(x_1, x_2, \dots , x_n) = c_0 \oplus c_1x_1 \oplus c_2 x_2 \oplus \dots \oplus c_nx_n \oplus c_{12}x_1x_2 \oplus c_{13}x_1x_3 \oplus \dots \oplus c_{(n-1)n}x_{n-1}x_n \oplus c_{123}x_1x_2x_3 \oplus \dots \oplus c_{12\dots n} x_1x_2\dots x_n$,
where $c_i \in \{0,1\}$, then we say that it is presented in the algebraic normal form (ANF).

A Boolean function is called affine if its ANF is as follows: $f(x_1, x_2, \dots , x_n) = c_0 \oplus c_1x_1 \oplus c_2x_2 \oplus \dots \oplus c_nx_n$, where $c_i=\{0,1\}$. If $c_0=0$ then the function f is called linear. If the function is not affine then it is called nonlinear.
Let $f, g: \sum n \to \sum$ be Boolean functions. The Hamming distance between f and g, denoted by $d(f,g)$, is defined as follows:

$$(f,g) = \sum_{x\in\{0,1\}^n} f(x) \oplus g(x).$$

Let X_n be a set of all affine n-argument Boolean functions. The integer

$$\delta(f) = \min_{g\in X_n} d(f,g),$$

is called the (Hamming) distance of f from the set of the affine functions. The minimum distance of the function f from the set of all affine functions is called nonlinearity of f. A bent function is a function, whose distance from the set of all affine functions is maximum, i.e., it is equal to $2^{n-1}-2^{(n/2)-1}$ [8].
An n-argument Boolean function is called homogeneous of degree k if the algebraic normal form of the function is as follows: $f(x_1, x_2, \dots , x_n) = c_0 \oplus c_1x_1 \oplus c_2x_2 \oplus \dots \oplus c_nx_n \oplus c_{12}x_1x_2 \oplus c_{13}x_1x_3 \oplus \dots \oplus c_{(n-1)n}x_{n-1}x_n \oplus c_{123}x_1x_2x_3 \oplus \dots\oplus c_{12\dots n} x_1x_2\dots x_n$,
where $c_i\in\{0,1\}$, and $c_i=0$ for every conjunction in which the number of variables is different from k.

Fact: There exist $2^{\binom{n}{k}}$ n-argument homogenous functions of degree k [4].

3. Introduction to Genetic Algorithms

Genetic Algorithms (GAs) were invented by John Holland and developed by him and his students and colleagues. In 1992 John Koza used genetic algorithm to evolve programs to perform certain tasks. He called his method "genetic programming" (GP). Later many scientists used this technique. The algorithm begins with a set of solutions (represented by chromosomes) called population. Solutions from one population are taken and used to form a new population. This is motivated by a hope, that the new population will be better than the old one. Solutions, which are then selected to form new solutions, are selected according to their fitness - the more suitable they are the more chances they have to reproduce [3].

Two important operations are used in genetic algorithm to improve its work [3]:

Crossover operates on selected genes from parent chromosomes and creates new offspring. The simplest way to do that is to choose randomly some crossover point and copy everything before this point from the first parent and then copy everything after the crossover point from the other parent.

Mutation is intended to prevent from falling all solutions in the population into a local optimum of the solved problem. Mutation operation randomly changes the offspring resulted from crossover. In case of binary encoding we used in tests we switch a few randomly chosen bits from 1 to 0 or from 0 to 1.

4. The designed genetic algorithm

In the genetic algorithm presented in this paper as an initial input we used a population of 1000 chromosomes (Boolean functions). These initial chromosomes are calculated by using a random n-argument nonlinear homogenous function of degree k generator. We invented an algorithm that generates random n-argument nonlinear homogenous function of degree k by the using of a random bit generator. These algorithms are represented in this section.

4.1. Random binary digit generator

In the program's application the "RANDOM" function from the Delphi standard 5 combined with real time (the function "Time") expressed in milliseconds was used. In the first step real time is saved to a variable, after this RANDOM(2) starts and samples 0 or 1, and the result is saved to the variable. The sampling result is the addition operation modulo 2 for the first and the second variable. It is a very simple but effective solution.

Algorithm 1 (Random bit generator).
INPUT: real time taken from the system clock, random(2).

Method:
```
  Randomize;
  Present := Now; {Present real time in the system}
  DecodeTime(Present, Hour, Min, Sec, MSec);
  MSec:=MSec div 10; {the value of milliseconds is
  divided by 10}
  j:=MSec mod 2;
  jj:=random(2); {Chosen 0 or one by using the system
  random function}
  result:=(j+jj) mod 2;  {result is equal 0 or 1}   ■
```
OUTPUT: a random bit.

- Function Random (Integer: x);
  The Random function returns a random number within the range 0 to x-1.
  If x=2, the result of function is a random integer equal 0 or 1.
- Procedure DecodeTime(Time: TDateTime; var Hour, Min, Sec, MSec: Word)
  The DecodeTime procedure breaks the value specified as the Time parameter into hours, minutes, seconds, and milliseconds.

4.2. Algorithm of generating random n-argument nonlinear homogenous function of degree k

For an n-argument homogenous functions of degree k there are $\binom{n}{k}$ k-elements groups. It means that the number of all the nonlinear homogenous functions of degree k is equal to $2^{\binom{n}{k}}$ and the number of all affine functions is equal to 2^{n+1}.
In order to test the distance for all n-argument homogenous function of degree k generated as in Algorithm 1, or to find any bent homogenous function the following algorithm was used.

Algorithm 2 (Generation of random n-argument nonlinear homogenous function of degree k).

INPUT: random bit and table $m[1..\binom{n}{k}]$ of all possible monomials.

Method:

```
1. Compute the value $l=\binom{n}{k}$.
2. Construct the table $a[1..l]$:
    for $i$=1 to $l$ do
    $a[i]$:=the value generated with the help of Algorithm 1.
3. $f(x_1,x_2,\ldots,x_n)$:=0.
4. for $i$=1 to $l$ do
    If $a[i]$=1 then $f(x_1,x_2,\ldots,x_n) := f(x_1,x_2,\ldots,x_n) \oplus m[i]$.
5. If $f(x_1,x_2,\ldots,x_n)$:=0 then goto 2. ■
```

OUTPUT: random n-argument nonlinear homogenous function f of degree k.

4.3. Nonlinearity testing algorithm

The nonlinearity of random homogenous functions was tested using Algorithm 3. As input we have n-argument nonlinear homogenous function of degree k generated by the Algorithm 2.

Algorithm 3 (Testing nonlinearity of random homogenous function).

INPUT: random n-argument nonlinear homogenous function of degree k.

METHOD:

1. Specify the distance for the Boolean function f from the set of all affine functions.
2. Choose the minimum distance of the function f from the set of all affine functions.
3. Write all results to the file.

OUTPUT: Boolean function nonlinearity.

4.4. Main Genetic Algorithm

The calculated chromosomes represent the Boolean functions that we want to use as a start for our genetic algorithm. These functions are represented by using binary system. We checked the 1000 chromosomes by using a tool that calculates the nonlinearity for each of them and we selected the best 500 according to the chrematistics of nonlinearity of Boolean functions. Then we do crossover in the middle between these chosen offsprings (chromosomes) the result will be new 500 offsprings. After this we do mutation of 10 bits for all these chromosomes. To the next population the chosen 500 chromosomes from the previous population and the new 500 offsprings are copied. The copying of 500 chromosomes from the previous is an important operation that is called Elitism and it is used to prevent the loss of the best found solutions from the previous population [3]. These operations are repeated 100 times.

The main genetic algorithm:

Algorithm 4 (Genetic algorithm calculation)

Input: Sample of 100 Boolean functions of n-argument and k-degree randomly generated by algorithm 2.

Method:

```
1. Choose 50 Boolean functions (chromosomes) with the best nonlinearity from the 100 functions and write them to $f_1, f_2, f_3, \ldots, f_{50}$
2. For $i$=1 to 100 do
  Begin
    a. Make crossover in the middle between every pair $(f_1,f_2), (f_3,f_4), \ldots (f_{49},f_{50})$ to produce new offsprings $f_{501},f_{502}, \ldots, f_{100}$
    b. Do mutation of 10 bits for the new 50 offsprings: $f_{51},f_{52}, \ldots, f_{100}$.
    c. Add the offsprings to parents and the result will be 1000 function: $f_1, f_2, f_3, \ldots, f_{100}$
    d. Choose 50 Boolean functions with the best maximum nonlinearity from the 100 functions (the original 50 and the 50 offspring) by using algorithm 3 and write them to $f_1, f_2, f_3, \ldots, f_{50}$
  End.
```

Output: 100 Boolean functions of n-argument and k-degree with the maximum nonlinearity.

By using algorithm 3 we tested the nonlinearity for number of arguments n= 8, 10, 12, 14 for k-degree where $1<k<n-1$. For $k= n-1$ all combinations will be tested by using comprehensive method.

5. Experimental results

The results of testing n variables Boolean functions for n= 8, 10, 12, 14 are presented in the following forms:

- diagrams in which the X-axis represents the nonlinearity of functions (the number n of arguments is distinguished by different colors) and the Y-axis presents the percentage of the homogenous functions of degree k in the sample of tested functions,
- aggregated form in tables, where we present the min and the max nonlinearity that we reach by using the genetic algorithm

In Diagrams 1–5 the results of testing n-argument homogenous functions of degree k are presented, n=8, 10, 12, 14 and $1<k<n$. The symbol n_k indicates the n-argument homogenous functions of degree k.

For the typographical purposes the diagrams presenting results for 10-, 12- and 14-argument functions are divided into two parts each, denoted as x/1and x/2, x=2,3,4.

Diagram 1 : 8_k, k=2,3,4,5,6,7

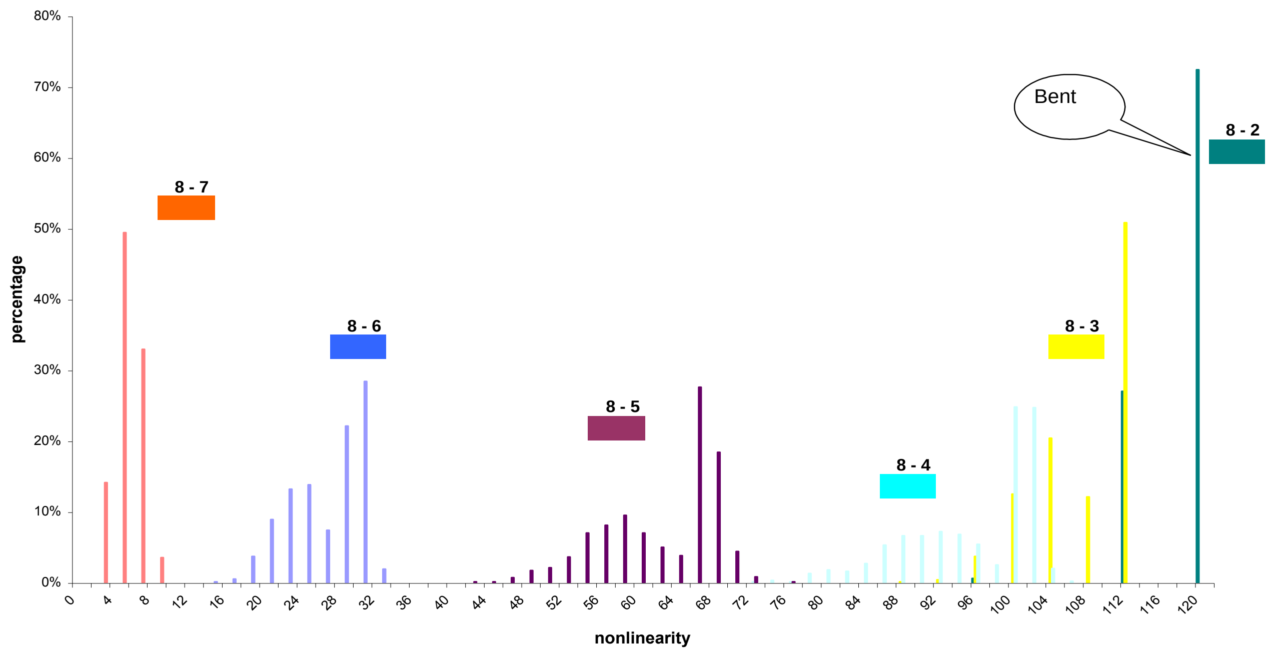


Diagram 2/1:10_k, k=2,3,4,5

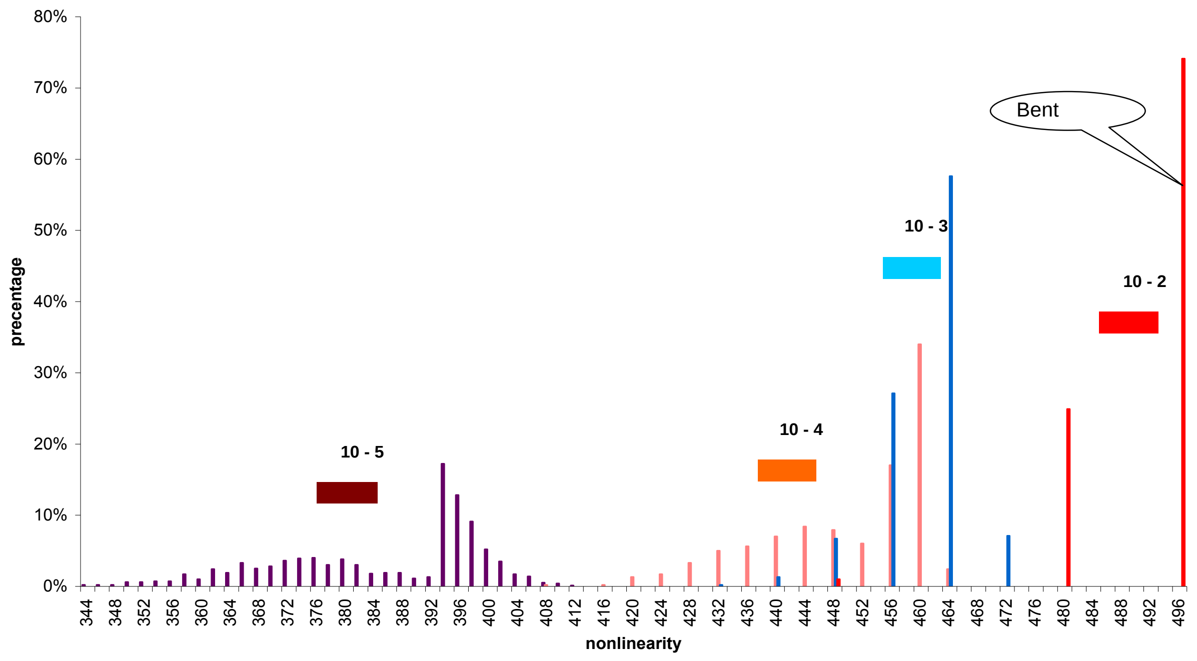


Diagram 2/2 : 10_k, k=6,7,8,9

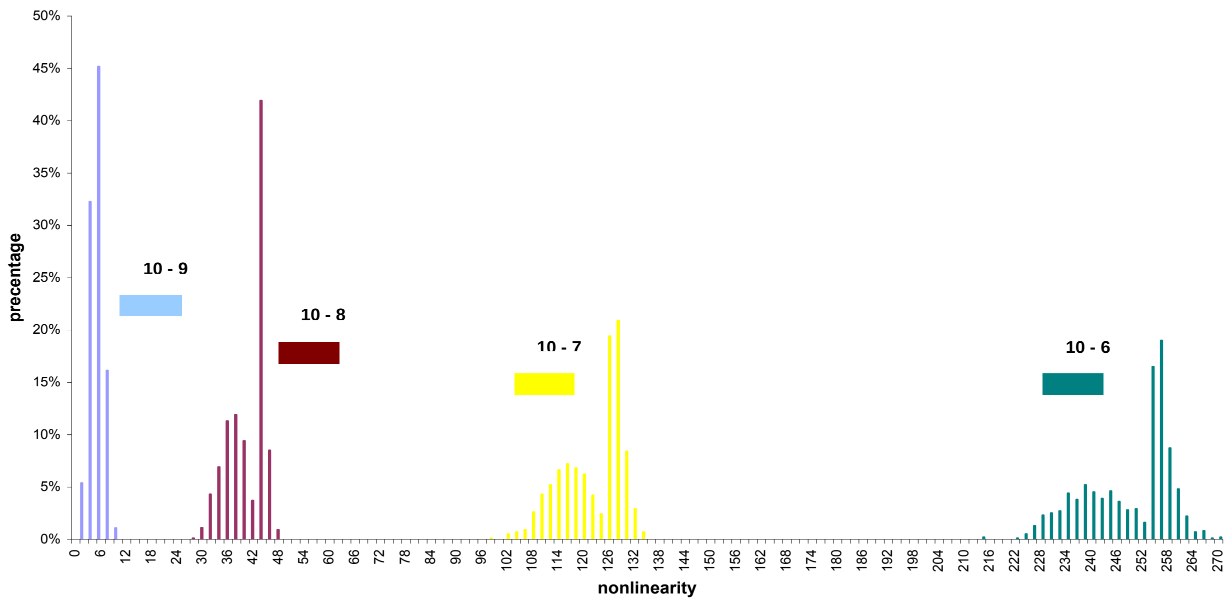
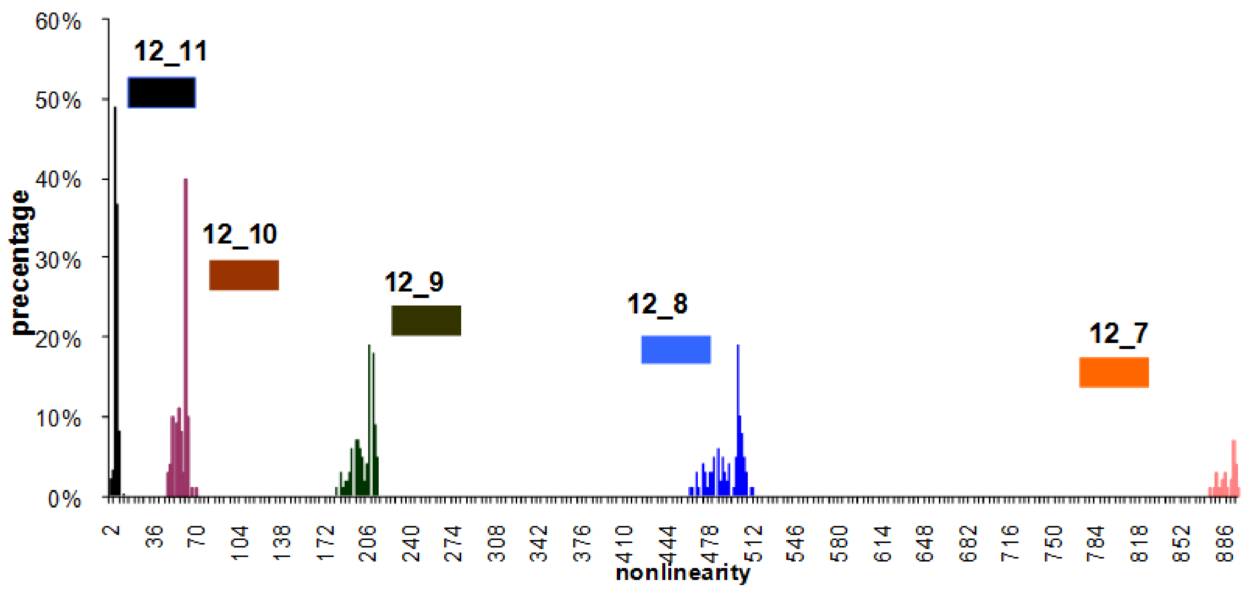


Diagram 3/2 : 12_k, k=7,8,9,10,11



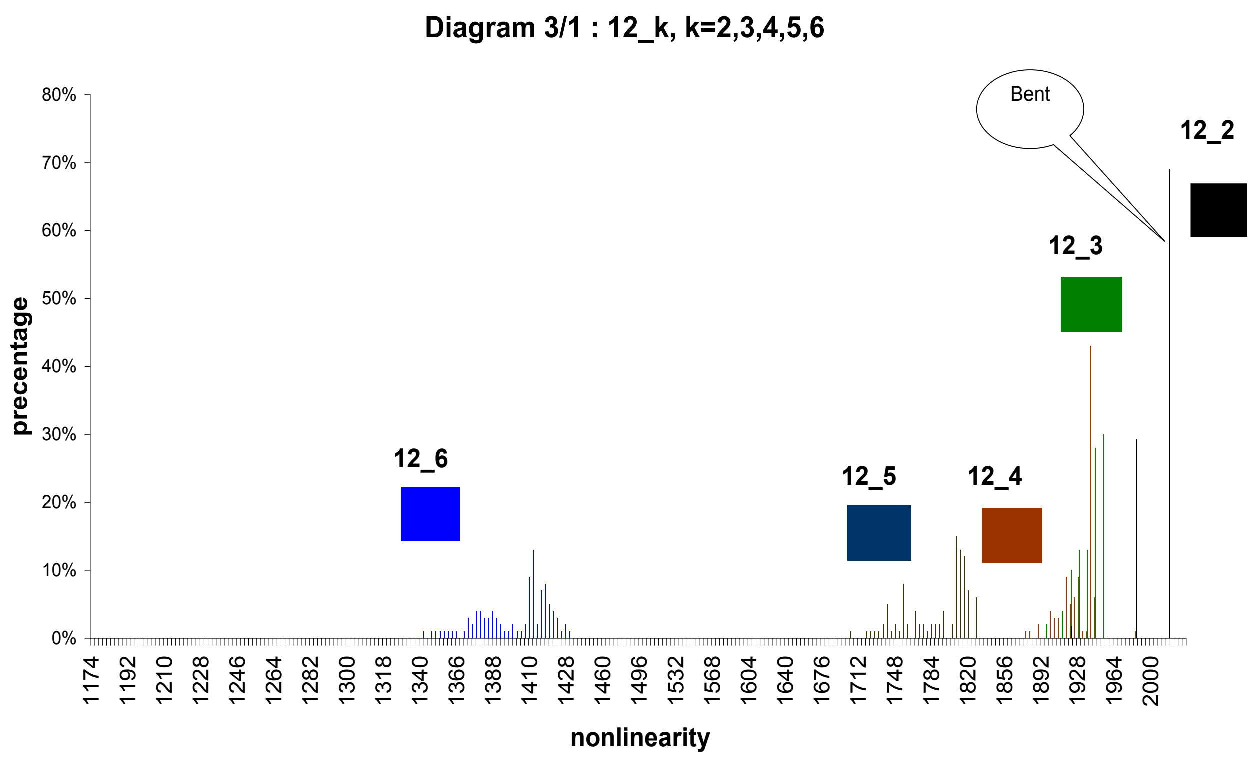


Diagram 3/1 : 12_k, k=2,3,4,5,6
Bent
12_2
12_3
12_6
12_5
12_4
precentage
80%
70%
60%
50%
40%
30%
20%
10%
0%
1174
1192
1210
1228
1246
1264
1282
1300
1318
1340
1366
1388
1410
1428
1460
1496
1532
1568
1604
1640
1676
1712
1748
1784
1820
1856
1892
1928
1964
2000
nonlinearity

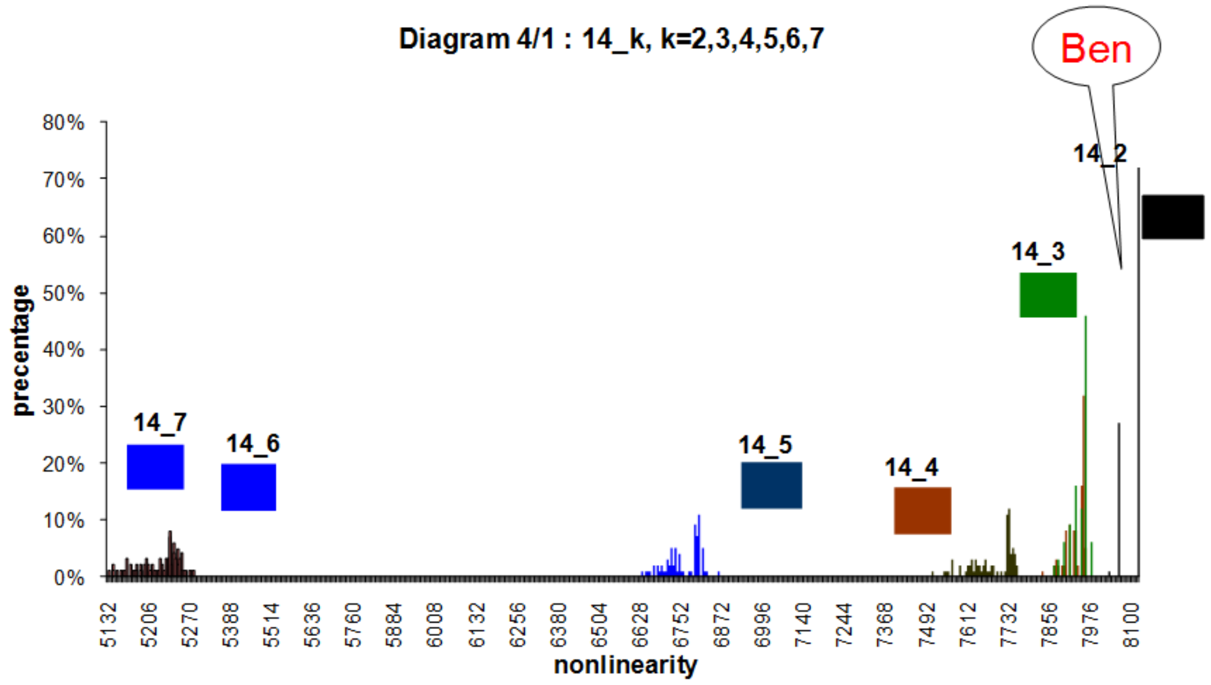
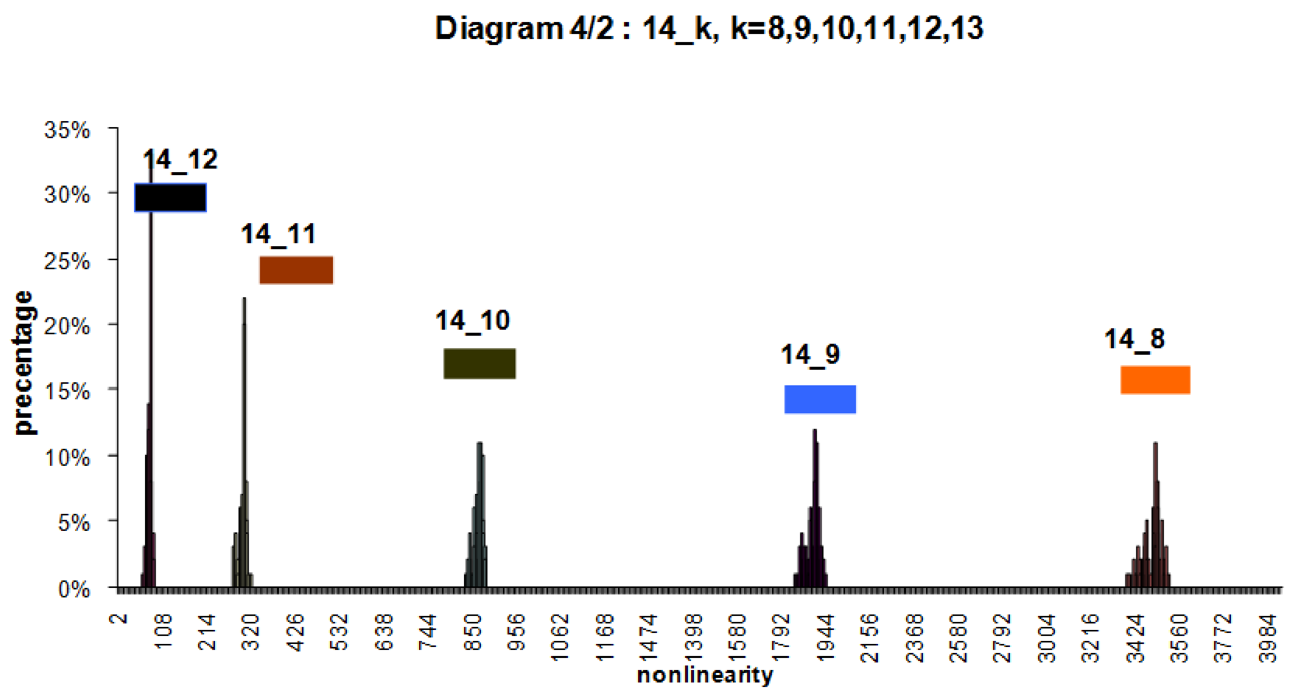


Diagram 4/1 : 14_k, k=2,3,4,5,6,7
Ben
14_2
14_3
14_7
14_6
14_5
14_4
precentage
80%
70%
60%
50%
40%
30%
20%
10%
0%
5132
5206
5270
5388
5514
5636
5760
5884
6008
6132
6256
6380
6504
6628
6752
6872
6996
7140
7244
7368
7492
7612
7732
7856
7976
8100
nonlinearity



Aggregated properties of the set of n-argument (n=8,10,12,14) Boolean homogenous functions of degree k (k=2,3,…,n–1)

In the tables 4.1–4.6 presented below the first row demonstrates the number of arguments of Boolean homogenous functions. The second row, called perfect dist represents the distance of bent function. The third row, called max.dist, represents the maximum nonlinearity that we reached by using the genetic algorithm. The fourth row, called min. dist shows the minimum nonlinearity that we reached.

Table 4.1: Properties of the set of n=8 and k= 2, 3, 4, 5, 6, 7

Degree	**2**	**3**	**4**	**5**	**6**	**7**
Bent	120	120	120	120	120	120
Max	120	112	106	76	32	8
Min	96	92	74	46	16	2

Table 4.2: Properties of the set of n=10 and k= 2, 3, 4, 5, 6, 7, 8, 9

Degree	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Bent	496	496	496	496	496	496	496	496
Max	496	472	464	412	272	134	48	10
Min	448	432	408	344	216	102	28	2

Table 4.3: Properties of the set of n=12 and k= 2, 3, 4, 5, 6

Degree	**2**	**3**	**4**	**5**	**6**
Bent	2016	2016	2016	2016	2016
Max	2016	1952	1944	1828	1430
Min	1920	1904	1892	1720	1344

Table 4.4: Properties of the set of n=12 and k= 7, 8, 9, 10, 11

Degree	**7**	**8**	**9**	**10**	**11**
Bent	2016	2016	2016	2016	2016
Max	948	512	214	70	12
Min	876	462	186	48	2

Table 4.5: Properties of the set of n=14 and k= 2, 3, 4, 5, 6,7

Degree	**2**	**3**	**4**	**5**	**6**	**7**
Bent	8128	8128	8128	8128	8128	8128
Max	8128	7984	7968	7764	6804	5274
Min	7936	7888	7848	7528	6616	5148

Table 4.6: Properties of the set of n=14 and k= 8, 9, 10, 11, 12, 13

Degree	**8**	**9**	**10**	**11**	**12**	**13**
Bent	8128	8128	8128	8128	8128	8128
Max	3522	1948	884	320	86	14
Min	3396	1864	838	280	66	2

6. Final Remarks

The tests in this paper confirm that genetic algorithms are an effective tool of searching for the maximum nonlinearity of homogenous Boolean functions. The results presented in diagrams and tables above show we reached very good results where we found high nonlinearity in efficient time (from 10 seconds to 5 minutes), where we used only 100 functions. In Table 4.7. we represent a comparison with results in [1]:

Table 4.7: BEST NONLINEARITY ACHIEVED AFTER TESTING 100 FUNCTIONS

n	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Method							
Benchmark	110	228	469	958	1946	-	-
RHC	112	232	475	964	1958	-	-
GA	111	229	470	959	1951	-	-
GA HC	113	232	474	969	1962	-	-
GA SM	112	-	472	-	1960	-	7984

As a benchmark the best results obtained from random search for functions with inputs ranging from eight n = 8 to twelve n = 12 were obtained for various search sizes for 100 functions [1]. RHC means a random search utilizing the Hill Climbing 1 algorithm; GA means a basic genetic algorithm with no hill climbing, GA HC means genetic algorithm, which incorporate the Hill Climbing 1 algorithm [1]. GA SM means the simple genetic algorithm designed in this paper. As we can conclude that GA SM is better than GA and Benchmark.

References

[1] Dimovski, A. Gligoroski, D. (2003): Generating highly nonlinear Boolean functions using a genetic algorithm. Telecommunications in Modern Satellite, Cable and Broadcasting Service. TELSIKS 2003. 6th International Conference. Publication Date: 1-3 Oct. 2003. Volume: 2, On page(s): 604- 607 vol.2

[2] C. Qu, J. Seberry, J. Pieprzyk, (1999): On the symmetric properties of homogeneous Boolean functions. In Information Security and Privacy (Wollongong, NSW, Australia, April 7–9). LNCS 1587, Springer, Berlin, 26–35.

[3] M. Obitko, (1998): Genetic Algorithms, http://cs.felk.cvut.cz/~xobitko/ga/, Czech Technical University, Prague

[4] C. Qu, J. Seberry, J. Pieprzyk, (1998): Homogeneous bent functions (preprint). University of Wollongong, NSW, Australia.

[5] William Millan, Andrew Clark, Ed Dawson, (1998): Heuristic Design of Cryptographically Strong Balanced Boolean Functions. Advances in Cryptology — EUROCRYPT'98. Lecture Notes in Computer Science, Springer Berlin / Heidelberg, Volume 1403/1998.

[6] W. Darłowski, (1998): Nonlinear Boolean functions for cryptographic purposes" (in Polish). In VI Międzynarodowa Wojskowa Konferencja Telekomunikacji i Informatyki. Materiały Cz.I. (Jabłonna, Sept. 8–10) WIŁ, Zegrze 1997, 233–240.

[7] William Millan, Andrew Clark and Ed Dawson (1997): An effective genetic algorithm for finding highly nonlinear Boolean Functions". Proceedings of ICICS 1997, pp. 149-158, Springer Lecture Notes in Computer Science, Volume 1334.

[8] W. Meier, O. Staffelbach, (1989): Nonlinearity criteria for cryptographic functions." In Advances in Cryptology – EUROCRYPT '89 (Houthalen, Belgium, April 10–13) LNCS 434, Springer, Berlin, 549–562.

[9] W. Meier and O. Staffelbach (1990): Nonlinearity Criteria for Cryptographic Functions. In Advances in Cryptology - Eurocrypt 89, Proceedings, LNCS, volume 434, pages 549-562. Springer-Verlag.

[10] D.E. Goldberg (1989): Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley, Reading, Massechusetts.

Mohannad Najjar received the B.S. and M.S. degrees in Computer Engineering from Poznan University of Technology, POLAND in 1998. Received Phd. In Telecommunication Engineering – Cryptography in 2002. During 2002-2009, he taught in Applied Science University- Amman, Jordan. Now he is an assistant professor in University of Tabuk in KSA.