Combining Evolutionary Computation and Algebraic Constructions to find Cryptography-relevant Boolean Functions

Stjepan Picek[1,2], Elena Marchiori[1], Lejla Batina[1] and Domagoj Jakobovic[2]

[1] Radboud University Nijmegen, Institute for Computing and Information Sciences
Postbus 9010, 6500 GL Nijmegen, The Netherlands
[2] Faculty of Electrical Engineering and Computing, University of Zagreb
Unska 3, Zagreb, Croatia

Abstract. Boolean functions play a central role in security applications because they constitute one of the basic primitives for modern cryptographic services. In last decades research on Boolean functions has been boosted due to the importance of security in many diverse public systems relying on such technology. A main focus is to find Boolean functions with specific properties. An open problem in this context is to find a balanced Boolean function with an 8-bit input and nonlinearity 118. Theoretically, such a function has been shown to exist, but it has not been found yet. In this work we investigate the use and integration of algebraic constructions and evolutionary computation (EC) to tackle this problem. Results indicate that various combinations of methods give better results although not reaching 118 nonlinearity.

Keywords: Boolean Functions, Nonlinearity, Evolutionary Computation, Bent Functions, Cryptographic Properties

1 Introduction

Cryptography is crucial in most security applications by helping cryptographic services to achieve secure communication through unsecured channels. The main goal is to secure messages so that only the relevant parties can read them. A message (plaintext) is transformed into an incomprehensible form (ciphertext) by a process called encryption, while an encrypted message is mapped to its original form through a process called decryption. Encryption and decryption are performed using symmetric key algorithms. Boolean functions constitute one of the basic primitives for symmetric key algorithms, most notably in stream ciphers [1]. Stream ciphers, based on an input key, generate a sequence of random bits which is used as keystream that will never be used again during the run of the cipher. Boolean functions for cryptography must satisfy various possibly contrasting properties, such as being balanced, highly nonlinear, correlation immune, t-resilient. Therefore, it is typically hard if not impossible to find an optimal function [1, 2]. Finding Boolean functions for cryptography is clearly a

challenging problem, because there are 2^{2^n} possible Boolean functions of n inputs (for instance, when n equals 8 which is usual size in today cryptography, this gives 2^{256} candidate solutions). There are three main approaches to generate Boolean functions for cryptography: algebraic construction, random generation and heuristic construction. Algebraic construction employs well established mathematical procedures that give very good results [3]. Random generation of Boolean functions has its advantages, most prominent one being easy and fast construction, but the resulting Boolean functions usually have suboptimal properties for cryptography [4]. Heuristic methods provide a relatively easy and efficient way of producing large number of Boolean functions with very good cryptographic properties [2]. In particular, Evolutionary computation (EC) has been successfully applied to evolve Boolean functions for cryptography as listed below. So far, researchers on Boolean function generation for cryptography have focused on the optimization of some cryptographic properties. In this work we focus on specific classes of Boolean functions, and analyze the "landscape" of results generated using and integrating algebraic and EC based approaches, in an effort to gain deeper insights on their individual and joint performance.

Specifically we focus on an open problem in cryptography: find a balanced 8-bit Boolean function with nonlinearity 118. Although theoretically it has been proved that such a function exists, no one has succeeded in finding it. Our goal is to investigate properties of EC methods based on the integration of the above mentioned approaches, in particular how difficult is to find Boolean functions with certain set of properties (for instance bent Boolean functions of a specific size) and what is the influence of a specific initial population (consisting of previously evolved bent Boolean functions).

1.1 Our Contributions

Besides evolving balanced function with 118 nonlinearity, our experiments investigate hardness of evolving bent Boolean functions and functions that can be used in algebraic constructions. As far as the authors know, we are the first to investigate the evolution of 7-bit functions and to give insights about the hardness of that problem and distribution of resulting functions. Our hybrid methods prove to be much more successful in evolving highly nonlinear balanced functions than simple evolution methods.

The remainder of this paper is organized as follows: after a short overview of related works, in Section 2 we describe the problem and relevant properties of Boolean functions. In Section 3 the considered methods are described, and in Section 4 experimental setup and results are given. Finally, Section 5 concludes with some suggestions for future work.

1.2 Related Work

We distinguish two main categories of methods for generating Boolean functions with properties of interest: those dealing with algebraic constructions based on

mathematical results and those dealing with heuristic methods. For each of these categories we give only a short overview of work related to our investigation.

Algebraic Constructions. Sarkar and Maitra propose new construction methods which were used to obtain functions that were not known earlier [5]. In the same year, those authors also presented a theorem that states stricter upper bound on nonlinearity of resilient Boolean functions [6]. Zheng and Zhang improved upper bound of the nonlinearity of high order correlation immune functions [7]. Pasalic et al [8] constructed several functions that were not known before that have upper bound on nonlinearity. Those functions were stipulated to exist due to the paper of Sarkar and Maitra [6].

Evolutionary Computation. Aguirre et al. used evolutionary multiobjective approach to evolve Boolean functions that have high nonlinearity [9]. Clark et al. used simulated annealing to find Boolean functions that satisfy several properties desired for cryptographic usage [10]. Burnett et al. created two heuristic methods to evolve Boolean functions for usage in cryptography. Picek et al. used genetic programming and genetic algorithms to find Boolean functions with cryptographic properties [11].

2 Preliminaries

In this section we describe relevant cryptographic properties of Boolean functions and theoretical results that inspired our investigation. In the sequel $\boldsymbol{a} \cdot \boldsymbol{b}$ denotes the inner product of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ defined as $\oplus_{i=1}^{n} a_i b_i$, where " $\oplus$ " denotes addition modulo 2.

2.1 Representations and Properties

A Boolean function f on $\mathbb{F}_2^n$ can be uniquely represented by a truth table (TT), which is a vector $(f(\mathbf{0}), ..., f(\mathbf{1}))$ that contains the function values of f, ordered lexicographically [1].

Walsh transform is a second type of unique representation of a Boolean function. It measures the similarity between $f(\boldsymbol{x})$ and the linear function $\boldsymbol{a} \cdot \boldsymbol{x}$ [1]. Walsh transform of a Boolean functions f equals

$$W_f(\boldsymbol{a}) = \sum_{\boldsymbol{x} \in \mathbb{F}_2^n} (-1)^{f(\boldsymbol{x}) \oplus \boldsymbol{a} \cdot \boldsymbol{x}}. \tag{1}$$

Third unique representation of an Boolean function f on $\mathbb{F}_2^n$ is by means of a polynomial in $\mathbb{F}_2 [x_0, ..., x_{n-1}] / (x_0^2 - x_0, ..., x_{n-1}^2 - x_{n-1})$. This form is called algebraic normal form (ANF) [1]. Algebraic normal form is the multivariate polynomial P defined in [1] as:

$$P(\boldsymbol{x}) = \oplus_{\boldsymbol{a} \in \mathbb{F}_2^n} h(\boldsymbol{a}) \cdot \boldsymbol{x}^{\boldsymbol{a}}. \tag{2}$$

The following properties of Boolean functions play an important role in security algorithms: nonlinearity, bent, correlation immunity, balancedness, t-resiliency, algebraic degree.

The **nonlinearity** NL_f of a Boolean function f can be expressed in terms of the Walsh coefficients as [1]

$$NL_f = 2^{n-1} - \frac{1}{2} max_{\boldsymbol{a} \in \mathbb{F}_2^n} |W_f(\boldsymbol{a})|. \tag{3}$$

Boolean function f is **bent** if it has maximum nonlinearity equal to [1]

$$NL_f = 2^{n-1} - 2^{\frac{n}{2}-1}. \tag{4}$$

Boolean function f is **correlation immune** of order t - $CI(t)$ if the output of the function is statistically independent of the combination of any t of its inputs [1]. Boolean function f is ***t*-resilient** if it is balanced and with correlation immunity of degree t [1].

A Boolean function is **balanced** (BAL) if its weight is equal to 2^{n-1} [1].

From this point on, when we talk about Boolean functions we consider them t-resilient (i.e. balanced). In the cases when that is not true we specify that.

Algebraic degree $deg(f)$ of a Boolean function f, is defined as the number of variables in the largest product term of the functions' algebraic normal form (ANF) having a non-zero coefficient [2].

2.2 Theoretical Background

Sarkar and Maitra showed that if a t-resilient Boolean function f has an even number of inputs n and $t + 1 \leq \frac{n}{2} - 1$ then its nonlinearity NL_f is bounded as follows [6]:

$$NL_f \leq 2^{n-1} - 2^{\frac{n}{2}-1} - 2^{t+1}. \tag{5}$$

From the formula it follows that the maximum nonlinearity for $n = 8$ and $t = 0$ equals 118.

In that same paper the authors gave a second theorem that inspired one of our experiments.

Suppose f is a Boolean function with 8 inputs, resilience 0 and nonlinearity 118. Then degree deg of f must be 7 and it is possible to write

$$f = (1 \oplus X_8) f_1 \oplus X_8 f_2 \tag{6}$$

where f_1 and f_2 are Boolean functions with 7 inputs, nonlinearity 55 and algebraic degree 7 [6].

An interesting implication of this result is that if it is not possible to construct an 8-bit Boolean function with nonlinearity 118, degree 7 and resilience 0 by concatenating two 7 inputs Boolean functions with nonlinearity 55 and degree 7 then the maximum nonlinearity of 8 inputs Boolean function is 116 [6].

3 Approach and Methods

Recall that we focus on the open problem of finding an 8-bit Boolean function with nonlinearity 118. We adopt a methodology consisting of two main phases.

In the first phase, we consider EC methods and their hybridization with algebraic constructions.

- **Simple evolution.** To analyze the effectiveness of EC methods we apply them directly to evolve 8-bit Boolean function with nonlinearity 118.
- **Bent functions.** Because bent functions have high maximum nonlinearity (see Equation (4)), in this setting we search for 8-bit bent functions. We do this in two ways: (a) using EC methods to directly evolve 8-bit bent functions, and (b) using EC methods to evolve 6-bit bent Boolean functions and use them to construct 8-bit bent functions by means of algebraic technique [12]. Furthermore, we use the resulting bent functions to seed the initial population of a EC algorithm for finding an 8-bit Boolean function with nonlinearity 118. This allows us to can gain insight in the influence of the bent functions as initial population.
- **Algebraic concatenation.** In this setting we evolve 7-bit functions with nonlinearity 55 and degree 7. Then we combine and concatenate those functions using Equation (6). Note that 7-bit functions are not balanced.

In the second phase, we select the best performing algorithm and investigate algorithmic hybridizations based on combinations of the algorithms and bent Boolean functions.

The above methodology allows us to address the following questions related to the effectiveness of EC for evolving Boolean functions.

- How difficult is to find bent Boolean functions of various sizes?
- What is the influence of the initial population in the evolution of balanced Boolean functions?
- How hard is to find Boolean functions with certain set of properties that are useful in further search process?

We describe below the techniques employed in our investigation.

3.1 Algorithms, Representations, and Fitness Functions

We consider three EC methods, namely genetic algorithms (GAs), genetic programming (GP) and genetic annealing (GAn). All the employed methods are a part of the Evolutionary Computation Framework (ECF) [13].

Genetic Algorithm. We use a simple genetic algorithm with 3-tournament selection [14], where in each iteration 3 solutions are selected randomly, and the worst of those is replaced with the crossover offspring of the remaining two. Mutation is performed on the offspring, using randomly simple mutation, where a single bit is inverted, and mixed mutation, which randomly shuffles the bits in a randomly selected subset. The crossover operators are one-point and uniform crossover, performed at random for each new offspring.

Genetic Programming.

The function set for genetic programming in all the experiments consists of Boolean functions OR, XOR, AND (taking two arguments), NOT (one argument) and IF, which takes three arguments and returns the second argument if the first one evaluates to 'true', and the third one otherwise. The terminals correspond to n Boolean variables. Genetic programming has maximum tree depth

of 11. Boolean functions can be expressed only in XOR and AND operators, but it is quite easy to transform from one notation to other. Genetic programming uses the same selection as the GA, with simple subtree crossover and subtree mutation.

Genetic Annealing. Genetic annealing is an evolutionary extension of the Simulated Annealing algorithm (SA) [15]. The SA operates on a single potential solution, which is locally changed in each iteration and its new fitness value is recorded. The new solution is accepted if it is an improvement, but a worse solution can also be accepted provided a certain level of *global energy bank*, which depletes with worse and increases with better solutions. The GAn is a simple extension in which the whole process is performed on a population of individuals.

Local Search Algorithm. For local search algorithm we chose strong hill climbing (HC) algorithm where all possible combinations of changes of 2 complementary bits are investigated. This process is repeated until there is no more improvement for every individual in population. Hill climbing algorithm uses truth table representation which means when using GP, individuals need to be transformed from tree representation to truth table representation. Additionally, we change 2 bits since initial solutions are balanced and we need to preserve it by changing one bit from 0 to 1, and other bit from 1 to 0.

Representations. As mentioned in Section 2, there are several ways how to uniquely represent Boolean functions. For genetic algorithms and genetic annealing we decided to represent the individuals as strings of bits where values are truth tables of functions, and for genetic programming, individuals are trees of Boolean primitives which are then evaluated according to the truth table they produce.

Fitness Functions. To evolve bent functions of different input sizes we consider the *maximization* of the following simple fitness function.

$$fitness = NL_f. \tag{7}$$

When looking for 7-bit functions with nonlinearity 55 and degree 7 we consider the *maximization* of the fitness function:

$$fitness = NL_f + deg. \tag{8}$$

In the case when we look for 8-bit balanced function with as good as possible nonlinearity we use following fitness function where the objective is *maximization*:

$$fitness = BAL + NL_f. \tag{9}$$

When calculating balancedness property, we assign it to a value 1 when it is balanced, otherwise we assign it the difference up to the balancedness multiplied with constant 5 (based on the results from tuning phase).

4 Experimental Setup and Results

Parameters that are common for every algorithm are the following: the size of Boolean function is 8 (the size of the truth table is 256) and the population size

is 500. Mutation probability is set to 0.3 per individual. The number of independent runs for each experiment is 2000; we are aware that 30-50 is sufficient for statistical purposes, but our main concern is not a statistical comparison of algorithms, but finding the best possible solutions. Furthermore, we use the rate of successful runs (finding the optimum), rather than the average fitness, as a metric for algorithm comparison.

4.1 Results

Due to the lack of space we give only short overview of results from phase 1. We mention that some of those results deserve additional experiments on its own; EC methods are not often used to generate bent functions and there is a clear lack of literature of how difficult it is or what methods perform the best. Furthermore, we are not aware that anyone before tried to evolve 7-bit functions with nonlinearity 55 and degree 7. In Table 1 we give results for phase I experiments. Column SUCCESS represents the percentage of runs that the algorithm reaches maximum value and column NL_f represents maximum nonlinearity reached by the algorithm. Naturally, we tested a random search method on all these problems but got no SUCCESS results. Based on this results we selected GP as the algorithm of choice for phase 2 as we can see that GP has best results when looking for functions with 8 inputs.

Table 1. Phase 1 experiments

Algorithm, fitness, size	NL_f	Min	Max	Mean	Stdev	SUCCESS(%)
GA, (8), 6-bit	28	26	28	26.141	0.512	7.1
GAn, (8), 6-bit	28	28	28	28	0	100
GP, (8), 6-bit	28	26	28	27.382	0.924	69.1
GA, (8), 8-bit	116	112	116	113.157	0.975	0
GAn, (8), 8-bit	114	112	114	113.9	0.1	0
GP, (8), 8-bit	120	112	120	114.5	2.318	13.8
GA, (9), 7-bit	55	60	62	61.97	0.22	98
GAn, (9), 7-bit	55	60	62	61.63	0.77	81.6
GP, (9), 7-bit	55	60	62	60.06	0.33	3
GA, (10), 8-bit	114	113	115	113.524	0.879	52.4
GAn, (10), 8-bit	114	113	115	113.03	0.25	3.2
GP, (10), 8-bit	116	109	117	112.23	1.01	0.5
HC, (10), 8-bit	112	103	113	108.02	1.43	0.5

From the results for Equation (9) we see that evolving functions with nonlinearity 55 and degree 7 is easy. However, our experiments show that the level of unbalancedness differ significantly as shown in Figure 1. Therefore, it is not easy to find two unbalanced 7-bit functions that produce a balanced 8-bit function. For example, if we find a function with desired properties that has 69 zeros

and 59 ones, then the other function needs to have 59 zeros and 69 ones (this function should not be affine equivalent to the first one since then it will have nonlinearity of 110) to produce a balanced 8-bit function. Since there were no prior experiments on this topic, these levels of unbalancedness present significant guideline for future work. With the concatenation method we obtained a balanced function with maximum nonlinearity of 110.

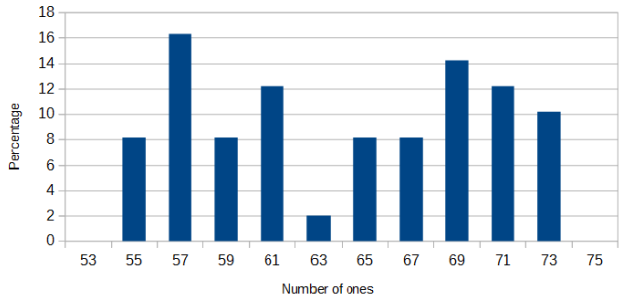

Fig. 1. Distribution of 7-bit unbalanced Boolean functions

In the second phase we use the best algorithm (GP) from the first phase when looking on the ways to further improve solutions. In this phase we concentrate only on Eq. (10) and function size 8. As the first experiment (I) in the second phase we run GP algorithm that has initial population made from bent functions from phase one. We do not distinguish here between bent functions that are directly evolved from those that are evolved and then algebraically expanded to 8-bit size. The second experiment (II) uses results from the first experiment and performs the hill climbing algorithm. In the last, third experiment (III) we run GA on the results from the first experiment. In Table 2 we give results for hybridizations of algorithms where we can see that these approaches improve algorithm behavior and percentage of the obtained maximum values. The approach that combines solutions from GP with bent functions as initial population and GA gives the best result.

Table 2. Phase 2 - algorithm combinations

Experiment	NL_f	Min	Max	Mean	Stdev	SUCCESS(%)
I (GP+seed)	116	113	117	113.55	1.379	13.8
II (GP+seed+HC)	116	113	117	113.8	0.91	19.6
III (GP+seed+GA)	116	113	117	115.32	0.43	67.1

Using EC methods to find 8-bit balanced Boolean function with nonlinearity 118 did not succeed in our experiments which is somewhat expected since this problem is very hard. When looking at the results from the evolutionary process in our opinion GP performs best. Not only did it find solutions with highest nonlinearity but it also resulted in final population with greatest diversity. Bent functions are not suitable for the use in cryptography and therefore not usually defined as the goal of evolutionary search. Previous works that evolved bent Boolean functions include [4,16]. Our results show that evolving 6-bit bent Boolean functions is relatively easy where each of the algorithms obtained numerous global optimums. Here, GAn behaves interestingly since it always reached optimum but we mention that many of those solutions are same. When considering 8-bit bent functions, only GP finds them successfully.

When looking at 7-bit unbalanced functions and Eq. (9), we see that GA performs best. However, it is not possible to concatenate every of those solutions since constructed functions need to be balanced. Prior to this research, as far as the authors know, there were no reports on the level of unbalancedness these functions can reach or on their distribution. From that perspective, we expect that this research will help in future work.

When setting bent Boolean functions as the initial population for GP, we see that we are still not able to reach nonlinearity 118 but the percentage of the population with the best currently known nonlinearity (116) significantly increases. Also, with larger population we also have a larger number of unique solutions. We recommend this procedure for the researchers that need to use functions that are not previously known with nonlinearity 116.

The rationale behind HC algorithm was not only to show the feasibility of improving solutions previously obtained with GP, but also to confirm that there is no 118 nonlinearity in "proximity" of solutions with nonlinearity 116. We see that average value of the final population slightly improved. When repeating same experiment, but using GA instead of HC we see that this setting was also unable to find 118 nonlinearity function. However, from statistical values it is obvious that this setting performs much better. Therefore, it seems prudent to combine multiple EC algorithms where the initial population for one algorithm is the final population of other EC algorithm.

5 Conclusions and Future Work

In this research we had two goals; one was to find balanced Boolean function with 118 nonlinearity and the second one was to investigate the strength of EC methods when producing high quality solutions. Although we did not find an 8-bit function with nonlinearity 118 our results present significant new insights in the area of evolving Boolean functions. As future work we plan to repeat the experiments with 7-bit functions with several different fitness functions and algorithms. Additionally, we are interested in fitness functions where variable is also Hamming distance between the solutions.

Acknowledgments

This work was supported in part by the Technology Foundation STW (project 12624 - SIDES), The Netherlands Organization for Scientific Research NWO (project ProFIL 628.001.007) and the ICT COST action IC1204 TRUDEVICE.

References

1. Braeken, A.: Cryptographic Properties of Boolean Functions and S-Boxes. PhD thesis, Katholieke Universiteit Leuven (2006)
2. Burnett, L.D.: Heuristic Optimization of Boolean Functions and Substitution Boxes for Cryptography. PhD thesis, Queensland University of Technology (2005)
3. Cid, C., Kiyomoto, S., Kurihara, J.: The RAKAPOSHI Stream Cipher. In Qing, S., Mitchell, C., Wang, G., eds.: Information and Communications Security. Volume 5927 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2009) 32–46
4. Millan, W., Fuller, J., Dawson, E.: New concepts in evolutionary search for boolean functions in cryptology. Computational Intelligence **20**(3) (2004) 463–474
5. Sarkar, P., Maitra, S.: Construction of nonlinear boolean functions with important cryptographic properties. In Preneel, B., ed.: EUROCRYPT. Volume 1807 of Lecture Notes in Computer Science., Springer (2000) 485–506
6. Sarkar, P., Maitra, S.: Nonlinearity bounds and constructions of resilient boolean functions. In Bellare, M., ed.: CRYPTO. Volume 1880 of Lecture Notes in Computer Science., Springer (2000) 515–532
7. Zheng, Y., Zhang, X.M.: Improved upper bound on the nonlinearity of high order correlation immune functions. In Stinson, D.R., Tavares, S.E., eds.: Selected Areas in Cryptography. Volume 2012 of Lecture Notes in Computer Science., Springer (2000) 262–274
8. Pasalic, E., Maitra, S., Johansson, T., Sarkar, P.: New constructions of resilient and correlation immune boolean functions achieving upper bound on nonlinearity. Electronic Notes in Discrete Mathematics **6** (2001) 158–167
9. Aguirre, H., Okazaki, H., Fuwa, Y.: An Evolutionary Multiobjective Approach to Design Highly Non-linear Boolean Functions. In: Proceedings of the Genetic and Evolutionary Computation Conference GECCO'07. (2007) 749–756
10. Clark, J.A., Jacob, J.L., Stepney, S., Maitra, S., Millan, W.: Evolving Boolean Functions Satisfying Multiple Criteria. In: Progress in Cryptology - INDOCRYPT 2002. (2002) 246–259
11. Picek, S., Jakobovic, D., Golub, M.: Evolving Cryptographically Sound Boolean Functions. In: GECCO (Companion). (2013) 191–192
12. Adams, C., Tavares, S.: Generating and counting binary bent sequences. Information Theory, IEEE Transactions on **36**(5) (Sep 1990) 1170–1173
13. Jakobovic, D., et al.: Evolutionary computation framework. http://gp.zemris.fer.hr/ecf/ (December 2013)
14. Eiben, A.E., Smith, J.E. In: Introduction to Evolutionary Computing. Springer-Verlag, Berlin Heidelberg New York, USA (2003)
15. Yao, X.: Optimization by genetic annealing. In: Proc. of 2nd Australian Conf. on Neural Networks. (1991) 94–97
16. Fuller, J., Dawson, E., Millan, W.: Evolutionary generation of bent functions for cryptography. In: Evolutionary Computation, 2003. CEC '03. The 2003 Congress on. Volume 3. (Dec 2003) 1655–1661 Vol.3