Evolving Cryptographically Sound Boolean Functions

ABSTRACT

This paper explores the evolution of Boolean functions for a cryptographic usage. To succeed in that goal, we use two well-known evolutionary computation methods. Additionally, we investigate the influence of selection and mutation operators in the evolution process. Since there are multiple criteria that a Boolean function must satisfy to be appropriate for a cryptographic usage, we also conduct experiments with different combinations of criteria in a fitness function. We also experiment with the new mutation operator and a new kind of initialization process. Results obtained show that those modifications can help in obtaining better solutions. The results indicate that it is possible to obtain highly quality results with algorithms that are not tailor-made for this purpose. Additionally, among the algorithms tested, the best performance was obtained with the variations of the genetic programming algorithm.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*heuristic methods*

General Terms

Algorithms, Experimentation

Keywords

Heuristic Methods, Genetic Algorithms, Genetic Programming, Boolean Functions, Cryptography, Experimental Results

1. INTRODUCTION

Today, evolutionary algorithms (EAs) present an important tool for different optimization tasks. Since the "No Free Lunch" theorem [20] states that there is no single best algorithm for all the problems, in the last decades a plethora of algorithms were (and are) developed, each one with its properties and usages. Some of those evolutionary algorithms are only applicable for a specific class of problems, while other algorithms are applicable to various fields of optimization problems. Of course, evolutionary algorithms would not be so popular heuristic approach if there were not many real-world problems that can be successfully solved by EAs.

One of the interesting real-world problems is the creation of Boolean functions with the properties relevant for the area of cryptography. From this point, when we talk about good Boolean functions, we presume Boolean functions with good cryptographic properties. Boolean functions for the use in cryptography can be designed by random search, algebraic construction or by heuristic methods [10]. Main advantage of the algebraic construction approach is that it is possible to design a function with an optimal property. However, the downside is that due to the trade-off between criteria, one optimal property could mean that some other properties are weak. Furthermore, with the algebraic construction the final result is only one Boolean function, whereas we could possibly want a whole family of good Boolean functions. For the purpose of this paper we do not assume reader's familiarity with the area of cryptography. However, notions not directly related with Boolean functions we define on a intuitive level and for a more detailed information refer to [12].

Boolean functions have widespread use in symmetric cryptography. Basically, ciphers in symmetric cryptography can be divided to stream and block ciphers. Shannon defined [18] two basic principles that a computationally secure cryptosystem should follow to be the confusion and diffusion principles. Diffusion principle serves to propagate the influence of each bit of plaintext and key to as many bits of ciphertext as possible. Confusion principle is used to make the relation between the key and ciphertext as complex as possible. Confusion is obtained by non-linear transformations. In block ciphers, confusion comes from S-boxes where S-box can be regarded as a vectorial Boolean function. In stream ciphers Boolean functions are used to introduce non-linearity (in fact, they are the only non-linear elements in stream ciphers) into otherwise linear systems. For a detailed description of block and stream ciphers refer to [12].

Since the problems of constructing good Boolean functions are well known, there is a significant body of work on heuristic methods [2] [5] [6] . Some of the algorithms used were general, like genetic algorithms or simulated annealing, while others were more specific and developed especially for the evolution of the Boolean functions. It must be mentioned that heuristic methods often evolved Boolean functions with at least comparable results as those obtained

GECCO'13, July 6-10, 2013, Amsterdam, The Netherlands.

by the algebraic constructions. In several cases heuristic methods found Boolean functions with the best known (at the time) set of properties. However, the best results were obtained when heuristic algorithm was coupled with some local search algorithm. If a Boolean function has n inputs, then there are 2^{2^n} possible Boolean functions. For n larger than 4 it is impossible to do an exhaustive search.

The goal of our paper is to explore the possibilities for the development of good Boolean functions. That was firstly done by using Genetic Algorithm (GA) where it is important to state that we also used out-of-the-box genetic algorithm so there would be no need for additional adjustment for someone who wants to reproduce our results. Furthermore, as far as we know, we are the first to use Genetic Programming (GP) in the evolution of good Boolean functions. Additionally, this is the first time that the influence of the evolutionary operators is explored. We also experimented with some operators specifically designed for Boolean functions. Finally, for some combinations of algorithms we used the widest set of criteria of all publicly known work that Boolean functions needed to satisfy. We must state that the algorithms presented here did not find the best solutions in every property when compared with the combination of EA and local search algorithms [7]. We believe that results obtained in this paper can be further augmented if some local search technique is to be used. In this paper we did not follow that direction because we were interested solely in the performance of EA.

In Section 2 we present the necessary background information about Boolean functions, Section 3 defines the experimental environment and presents results, Section 4 gives a discussion about the results, and finally, Section 5 draws a conclusion.

2. BOOLEAN FUNCTIONS

In this section we present only the necessary information about Boolean functions and their cryptographic properties. For additional information refer to [4].

2.1 Basic properties and representations

Boolean function is a mapping from $\{0,1\}^n$ to $\{0,1\}$. Vectorial Boolean function or $n \times m$ S-box is a mapping from $\{0,1\}^n$ to $\{0,1\}^m$. An $n \times m$ S-box can be constructed by combining m Boolean functions.

Set F_2^n represents n-tuples of elements in Galois field F_2 and it will be represented as $\overline{x} = (x_1, ..., x_n)$.

Basic (and most natural) representation of a Boolean function is a truth table (TT). When the total order is assigned, Boolean function f with n inputs has a truth table with 2^n elements, where each element $\in \{0,1\}$. Every Boolean function is uniquely determined by its truth table.

Second unique representation of a Boolean function is Algebraic Normal Form (ANF). ANF is a multivariate polynomial P defined as

$$P(x) = \bigoplus_{\overline{a} \in F_2^n} h(\overline{a}) \cdot \overline{x}^{\overline{a}} \tag{1}$$

where h is defined by Moebius inversion as

$$h(\overline{a}) = \bigoplus_{\overline{x} \leq \overline{a}} f(\overline{x}), for\ any\ \overline{a} \in F_2^n \tag{2}$$

Operation $\bigoplus$ is addition in characteristics 2.

Boolean function is also uniquely determined by its Walsh transform. Walsh transform of a Boolean function f is denoted as W_f, which is a real-valued function defined for all $\overline{\omega} \in F_2^n$ as

$$W_f(\overline{\omega}) = \sum_{\overline{x} \in F_2^n} (-1)^{f(\overline{x}) \oplus \overline{x} \cdot \overline{\omega}} \tag{3}$$

where $\overline{x} \cdot \overline{\omega}$ can be regarded as a linear Boolean function of $\overline{x}$ determined by the $\overline{\omega}$.

The autocorrelation function of a Boolean function f is a real-valued function defined for all $\overline{\omega} \in F_2^n$ as

$$r_f(\overline{\omega}) = \sum_{\overline{x} \in F_2^n} (-1)^{f(\overline{x}) \oplus f(\overline{x} \oplus \overline{\omega})} \tag{4}$$

Autocorrelation function does not uniquely represent a Boolean function. [5] Autocorrelation is a measure of a function's periodicity [17], or the correlation of outputs between inputs that are related to each other through some constant.

2.2 Cryptographic properties

In this section we shortly present the properties we investigated in the experiments and that are important for good Boolean functions.

Definition 1. The Hamming weight $w_h(f)$ of a Boolean function f is the number of ones in its binary truth table.

Definition 2. Boolean function f with n inputs is balanced if its Hamming weight equals 2^{n-1}.

Alternative way of stating that the Boolean function is balanced is with the Walsh coefficients as

$$W_f(\overline{0}) = 0 \tag{5}$$

Boolean functions that are not balanced are not appropriate for the use in cryptography. The higher the magnitude of a Boolean function imbalance, the easier is to approximate it with a linear function [5]. If the imbalance is large enough, it could be possible to approximate Boolean function with a constant function. In formulas we abbreviate balancedness with BAL.

Definition 3. The nonlinearity NL_f of a Boolean function is its minimum Hamming distance to any affine function.

Nonlinearity can be calculated as

$$NL_f = \frac{1}{2}(2^n - max|W_f(\overline{\omega})|) \tag{6}$$

where $maxW_f$ is the maximum value of Walsh transform of f over all vectors $\overline{\omega}$.

Nonlinearity property has been introduced to asses the resistance of a Boolean function to linear and correlation attacks.

Definition 4. The algebraic degree, deg(f), of a function f is the number of variables in the highest order term in its ANF with non-zero coefficient.

If a Boolean function has a high algebraic degree it aids in ensuring that the function is not highly correlated with any particular inputs [17].

Definition 5. Function is correlation immune (CI) of degree q if the output of the function is statistically independent of the combination of any q inputs.

Correlation immunity of degree q can be calculated with the Walsh spectrum as

$$W_f(\overline{\omega}) = 0, 1 \leq w_h(\overline{\omega}) \geq q \quad (7)$$

Definition 6. Function is resilient with order q if it is balanced and correlation immune with degree q.

Correlation immunity and resiliency are used to assess the resistance of a Boolean function to the correlation attacks. [13]

Definition 7. The algebraic immunity (AI) of a Boolean function f on F_2^n is defined as the lowest degree of the Boolean function q from F_2^n into F_2 for which $f \cdot g = \overline{0}$.

Algebraic immunity is used to assess the resistance of a Boolean function to the algebraic attacks based on annihilators [13].

Definition 8. A Boolean function f is said to satisfy the propagation criterion (PC) of order q, if changing any (up to q) bits in the input results in the output of the function being changed for exactly half of the 2^n vectors. [8]

Propagation criterion is important property of Boolean functions to be used in S-box [14].

Zhang and Zheng [22] found that PC have some limitations in identifying desirable cryptographic properties so they introduced the idea of global avalanche characteristics - GAC.

The global avalanche characteristics indicators consist of an absolute indicator and a sum-of-square indicator.

The absolute indicator is used to define the distance between a Boolean function f and the set of functions with linear structures.

Absolute indicator is defined as

$$AC(f) = max_{\overline{\omega} \in F_2^n \setminus \{0\}} |r_f(\overline{\omega})| \quad (8)$$

Sum-of-square indicator is defined as

$$SSI(f) = \sum_{\overline{\omega} \in F_2^n} r_f(\overline{\omega})^2 \quad (9)$$

For a Boolean function to have good cryptographic properties we want it to be balanced, with high nonlinearity, algebraic degree, algebraic immunity, correlation immunity, low absolute indicator and sum-of-square indicator. It is not possible to get the Boolean function with all the optimal properties. Connections between some properties are known, while connections between some others are still not clear. Here we mention only as an example that bent Boolean functions [4] have maximum possible nonlinearity and minimum absolute indicator value but are not balanced and have algebraic degree $deg(f) \leq n/2$ and therefore they are not appropriate for the use in the cryptography. For a more detailed discussion about trade-off between properties and attainable values for those properties refer to [4] [10].

3. ENVIRONMENTAL SETTINGS AND RESULTS

Table 1: Fitness Functions for GA and GP Algorithms

Name	Formula
FIT1	$fit1 = BAL + NL$
FIT2	$fit2 = fit1 + AI + CI + DEG$
FIT3	$fit3 = fit2 + SSI + PC + AC + W_s$

3.1 Fitness function and representation

When conducting experiments we use three different fitness functions - abbreviated FIT1, FIT2 and FIT3 as listed in Table 1. In all the experiments, maximizing the value of a fitness function is the objective.

Since the absolute indicator and sum-of-square indicator give much larger absolute values than the other parameters we normalized them in the following way:

$$AC(f) = AC(f)/(n >> 1) \quad (10)$$

$$SSI(f) = (\sqrt{SSI(f)/n}) >> 1 \quad (11)$$

where n is the number of inputs for a Boolean function.

In FIT3 formula we considered the whole Walsh spectrum because the smaller the maximal value it is, the larger the nonlinearity will be as stated in Equation 6. But, it seems advantageous that whole Walsh spectrum be as flat as possible, and with as small as possible values (with that it is somewhat closer to bent functions that have an ideal spread of Walsh spectrum). Same principles were also used in [2]. The formula to calculate Walsh spectrum is

$$W_s(w) = \sqrt{\sum_w ||W_f(w)| - 2^{\frac{n}{2}}|} \quad (12)$$

In the initial experiments we initialize population with random individuals where we do not require that the individuals are balanced. That is opposite from what is done in most of the literature [10] [15], but we believe that the evolution process would benefit from that additional diversity. To ensure that the best solution is balanced, we use a balance penalty as a part of fitness function, presented in pseudo-code as

Algorithm 1 Calculate balancedness

```
if (h_w(TT) > 2^n/2) then
    penalty ← (2^n − h_w(TT))/2^n · X
else
    penalty ← 2^n/(2^n − h_w(TT)) · X
end if
```

where we experimentally found that X = - 5 scales well for Boolean functions with $n = 8$ inputs when working with FIT1 and FIT2. For FIT3 case we set X to - 50 to match the scale. When experimenting with FIT3 with X parameter set to -5, the pressure of balancedness penalty function was not enough to get balanced functions, which is due to the fact that other properties contributed more to the fitness value. It is interesting, though, that GP found several bent functions when X was -5.

The experiments are divided into three distinctive phases

where in each phase experiments are conducted with one fitness function.

We could have also considered three different fitness functions as three distinctive problems and normalize the values (and analyze as in the Multiple Problems - Multiple Algorithms scenario) but we opted for the aforementioned approach since we believe it will be more expressive in displaying the differences between algorithms.

Additionally, in each of those three phases we give two perspectives: first, from a cryptographic point of view where we are interested in the best solutions and the properties of those solutions, and second, from EA view, where we conduct extensive statistical tests to try to find the best algorithm.

As previously stated, we compare 2 different algorithms: GA and GP. Since GA and GP are in standard form, we will assume reader's familiarity with them. In statistical analysis, each modification of an algorithm (e.g. different mutation or initialization) is treated as a separate algorithm.

The first decision needed was to decide on the appropriate choice for the representation of the individuals. As mentioned in Background section, there are several ways how to uniquely represent Boolean functions. For GA we decided to represent the individuals as binary vectors where values are truth tables of functions, and for GP individuals are binary trees of Boolean functions which are then evaluated according to the truth tables.

Since there is a plethora of possibilities for the different genetic operators we decided to conduct experiments with several combinations of them. First we evaluate the influence of selection where we conduct experiments with three different selection procedures: steady-state tournament (SST), generational roulette-wheel (RW) selection and elimination (generation gap, EL) selection. The same selection mechanisms are applied both to genetic algorithm and genetic programming.

After the selection operator that gave the best results is determined, we conduct further experiments with variations of mutation operators and initialization procedures as follows.

3.2 GA variations

For GA representation, mutation is selected uniformly at random between simple and mixed mutation.

To evaluate the influence of mutation operators, we use a new variation of mutation operator - balanced mutation. Pseudo-code for balanced mutation is

Algorithm 2 Balanced mutation

```
if (h_w(TT) = 2^{n-1}) then
  mutate(p_m) ← 2 bits
else
  mutate(p_m) ← 1 bit
end if
```

Balanced mutation is designed to preserve the balancedness property of a Boolean function. Of course, in the case that the Boolean function is not balanced, there is a 50% chance that this kind of mutation will result in even more unbalanced function. However, this way algorithm can search larger solution space and balancedness penalty will stop highly unbalanced function from propagating through generations.

Additionally, we used adaptive mutation rate for all mutation operators. In the beginning, the mutation is given a fixed probability, but as the evolution starts to stagnate (i.e. no improvement in the best solution), the mutation probability raises. The probability is increased until it reaches a predefined maximum level, or until a new best solution is found, when the mutation rate is reset to initial value.

To explore the influence of initial population initialization, further experiments are conducted with GA where we initialize the algorithm with balanced population: truth tables are either randomly set or initialized with the orthogonal array [21]. GA is initialized with random balanced individuals in the following way

Algorithm 3 Random balanced initialization

```
TT ← 0̄
count ← 0
while count ≠ 2^{n-1} do
  if pos[rnd] = 0 then
    pos[rnd] ← 1
    count ← count + 1
  end if
end while
```

3.3 GP variations

Of all the modifications in the previous section, with GP we employ only the adaptive mutation rate, in the same manner as for the GA. Orthogonal initialization is also used on GA only; GP would be difficult to initialize in this manner since there is no simple way to represent trees in orthogonal array and balanced random initialization would be computationally intensive since it would require a trial and error approach.

Function set for genetic programming in all the experiments is OR, NOT, XOR, AND, IF, and terminals correspond to Boolean variables. Genetic programming has maximum tree depth of 11. For the Boolean functions we are interested only in XOR and AND operators, but it is quite easy to transform it from one notation to other.

3.4 Common parameters

Parameters that are in common for every round of the experiments are the following: the size of Boolean function is 8 (the size of the truth table is 256), number of independent runs for each experiment is 30 and the population size is 500. In steady-state tournament selection, tournament size is equal to 3; crossover probability for roulette-wheel selection is 0.5 and selection pressure (the ratio of best and worst individual fitness) is 10. In elimination selection, the generation gap is 0.6. Mutation probability for non-adaptive variations is set to 0.3 per individual.

Further information regarding experimental setup is listed as needed.

In effort to find the differences in the performance of the algorithms, we also compare the best algorithms with the random search genetic algorithm. Stopping criterion for the random search algorithm is 400 generations. This algorithm is a standard random search algorithm with only the random initialization of GA individuals. We also conducted experiments with random search GP, but due to the more complex process of tree initialization, the resulting solutions

are much worse in average (we do not display the results for the GP random search).

Algorithm names presented in tables are abbreviated in the following way: the first part is the representation (GA or GP), the next is a selection operator (SST, RW or EL), and finally the fitness function (FIT1 to FIT3). For instance, genetic algorithm with roulette-wheel selection and second fitness function would be GA_RW_FIT2. Aside from those basic information, the abbreviation can contain BAL which means that the balanced mutation is used, ALL that means the mutation operator is randomly selected from all possible mutation operators, +VAR represents the adaptive mutation rate, ORT means the initial population is created with an orthogonal array, RAND means that the algorithm is random search.

If it is clear from the context, we omit the fitness function or selection operator from the abbreviation.

As a test suite the Evolutionary Computation Framework (ECF) was used. ECF is a C++ framework intended for the application of any type of the evolutionary computation, developed at the University of Zagreb [11]. For the statistical analysis we use R programming environment [1].

3.5 Best Solutions

With the objective to find the best individuals, stopping criterion is 300 generations without improvement for GA, and 30 generations without improvement for GP. Since from the cryptographic point of view we are interested in obtaining the best possible results, the best solution for GA and GP algorithms for every fitness function are presented in Table 2. As a reference, we also give solutions obtained via random search. All solutions are balanced so we did not specifically write that property in the table. Genetic programming with the adaptive mutation and steady-state tournament selection (GP_SST_ALL+VAR) reached the highest known combination of NL, AC, SSI and W_s, as far as we know. The truth table of that solution in hexadecimal format is denoted as
040B616EFBF4616E5E513B34A1AE3B34C7C8828D3837
222D9D927877626D7877.

3.6 Statistical Analysis

In the Single Problem - Multiple Algorithms scenario that we follow for these experiments, we opted to follow the guidelines about statistical methods from [3] [9] [16]. For details about statistical tests refer to [19].

The stopping criterion for the experiments is the number of generations, here set to 50. We are aware that 50 generations is quite low, but all the algorithms manifested quick convergence and little improvement after 50 generations in average. This is also in accordance with the results presented in [10]. Our opinion is that the analysis of the performance of the algorithms with the number of generations as the stopping criterion yields more objective results then those obtained when stagnation is the stopping criterion.

To try to find the best algorithm from the initial pool of 14 variations of algorithms, we conduct statistical analysis on 6 algorithms that produced the best solutions. As a fitness function we use FIT2 and steady state tournament selection for all the algorithms since the results obtained for that set of parameters are the best.

We use two approaches in the statistical analysis. First one, average-case scenario, takes average values from each of the runs and compares those values. Second kind of analysis is based on the threshold values. Here we set values for different properties (based on the values obtained in the Best Solutions section) and check the percentage of the solutions that reach those values. For all the experiments, level of significance α is equal to 0.05.

3.6.1 Average Case Best Algorithm

Basic statistical data for best algorithms are displayed in Table 3. All algorithms use steady-state tournament selection and fitness 2 (FIT2) function.

To find the best algorithm, first we check necessary conditions to use parametric statistical tests.

To do that, we use Shapiro-Wilk test to examine normality condition and Levene test to examine the heteroscedasticity condition (there is no need to check independence condition because there are independent runs of experiments with random initial populations).

Since the tests showed that the solutions are not normally distributed and the variances of the distributions of the different algorithms are not homogeneities we use nonparametric statistical tests.

First we use Kruskal-Wallis (K-W) test to try to discover whether there are differences between algorithms. Since the chi-square value (75.14) of K-W test is greater then the tabled critical chi-square value (11.07) for a level of significance 0.05 we conclude there are differences between the algorithms. All algorithms use steady-state tournament and second fitness function (FIT2) so those data will be omitted from the abbreviations from now. Algorithm GP is chosen as a control (best) algorithm based on the ranking of the algorithms obtained with the ordinal values. To verify if the GP algorithm is the best, now we conduct pairwise sign test between it and every other algorithm. Results are displayed in Table 4.

Based on the results of the sign test, we can conclude that in average, GP, GA_BAL and GP_ALL+VAR algorithms perform better than the standard GA. Boxplots for the algorithms from Table 3 are displayed in Figure 1.

3.6.2 Threshold Values Best Algorithm

In this experiment, we decide on the desired minimum values that the properties of Boolean functions should have. Of course, that is subjective and depends on the goals of the researchers. We examined here following properties (AI, DEG, NL, W_s, SSI, AC) with the threshold values (4, 7, 111, 156, 147968, 48). Additionally, Boolean functions need to be balanced. Values used here are selected solely to help us determine the performance of the algorithms so they should not be considered as a guideline when searching for the best solutions. Since normality condition is not fulfilled and variances of the distributions are not homogeneities, we use nonparametric statistical tests. To check whether there are differences between algorithms we use Friedman test [9]. Due to the lack of space, we display results for the following algorithms: GA, GP and GP_ALL+VAR. Results for the Friedman test are displayed in Table 5.

The p-value of Friedman test equals 0.04 so we conclude there are differences between the algorithms. For the post-hoc analysis we use Bonferroni and Hochberg methods with the GA as a control algorithm (decided based on the rank values from Friedman test). Results obtained are in Table 6.

Table 2: Best Solutions

Algorithm	Fitness 1 $(NL, DEG, AI, CI, AC, SSI, PC, W_s)$
GA_SST_ALL+VAR_ORT	(114, 7, 4, 0, 56, 126976, 0, 145)
GP_SST_ALL+VAR	(116, 6, 3, 0, 32, 87040, 0, 90)
GA_SST_RAN	(110, 7, 4, 0, 56, 158080, 0, 154)
Algorithm	**Fitness 2 $(NL, DEG, AI, CI, AC, SSI, PC, W_s)$**
GA_SST_BAL	(114, 7, 4, 0, 48, 128128, 0, 143)
GP_SST	(116, 7, 3, 0, 40, 95104, 0, 101)
GA_SST_RAN	(110, 7, 4, 0, 72, 160384, 0, 152)
Algorithm	**Fitness 3 $(NL, DEG, AI, CI, AC, SSI, PC, W_s)$**
GP_SST	(112, 7, 4, 0, 40, 128896, 0, 135)
GA_SST_BAL	(112, 7, 4, 0, 40, 123520, 0, 132)
GA_SST_RAN	(110, 7, 4, 0, 40, 131200, 138)

Table 3: Best Algorithms Statistics

Algorithm	Min	Max	Mean	Stdev
GA	114.494	120.348	115.135	1.065
GP	112.336	119.398	116.565	1.827
GA_BAL	114.784	120.888	116.077	1.903
GA_ALL+VAR	114	115	114.697	0.29
GP_ALL+VAR	113	119	116.421	1.441
GA_ALL+VAR_ORT	114.476	115.002	114.734	0.151

Table 4: Sign Test Analysis

	Control algorithm: GA		
Algorithm	Win	Loss	Sign Test
GP	24	6	0.001
GA_BAL	25	5	0.000
GA_ALL+VAR	11	19	0.2
GP_ALL+VAR	25	5	0.000
GA_ALL+VAR_ORT	12	18	0.36

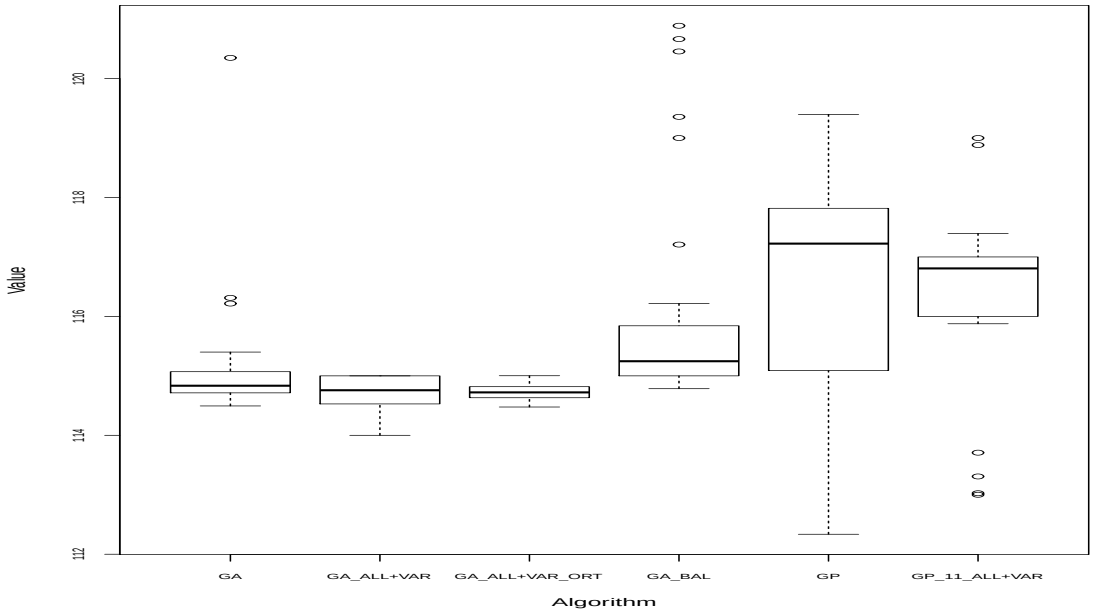

Figure 1: Box plot of the best algorithms

Table 5: Average Rankings of the Algorithms

Algorithm	Ranking
GA	1.2857
GP	2.1429
GP_ALL+VAR	2.5714

Table 6: Post-hoc Analysis

algorithm	unadjusted p	p_{Bonf}	$p_{Hochberg}$
GP_ALL+VAR	0.016157	0.032314	0.032314
GP	0.108809	0.217619	0.108809

Since the level of significance is 0.05 we can conclude that in the threshold experiment GA performs better than the GP_ALL+VAR algorithm.

4. DISCUSSION

Output space of possible Boolean functions is huge and therefore it is impossible to do exhaustive search for Boolean function with the number of inputs relevant in cryptography. However, output space of Boolean functions with good (of course, there is a question: "What is good?") cryptographic properties is also large. In that large space of good Boolean function it is difficult to find Boolean functions with excellent cryptographic properties. This is an obvious example of the convergence of the algorithm towards the local optima. In an attempt to search beyond those local optima we employ different algorithms. As expected, every modification of basic GA or GP algorithms displayed differences in the performance of the algorithm. In the experiments we used 3 different selection methods where we expected that the roulette-wheel selection should be the best one, because preliminary results (also the results from other researchers) showed that all algorithms display very quick convergence. Since steady-state tournament selection has the biggest selection pressure we believed that, when coupled with a problem with the fast convergence it will lead to suboptimal results. However, algorithms with the steady-state tournament selection consistently found the best solutions among all the algorithms. One explanation could be that since the convergence is quick the majority of the population have good properties so extra selection pressure helps in the combining of the best individuals. We also experimented with 3 different fitness functions. Function with the smallest number of variables (FIT1) displayed very good results and some results even outperform any results from existing research. Simple function has the advantage that there are no conflicts between variables, and some high quality properties inherently mean that other properties will also be good. In FIT2 fitness function we tried to control more properties of a Boolean function. That approach showed the best results in average. Here, it could be prudent to use a weighted fitness function approach. In FIT3 function the results are worse than for the first 2 fitness functions. This is due to the fact that some properties reach much higher values so linear grading in fitness function (value of every property is added to the final result and all properties are equally significant) is not the best approach. However, the problem there is that we use many properties so we could first run evolutionary algorithm to find the best weights for those properties. It can also be observed that some of the properties (PC and CI) were constantly 0. The obvious reason for that is that those properties are in conflict with some of the other properties. Additionally, possible values for the PC and CI are not high enough to direct the search from more influential ones (in regards to attainable values). Modifications in the mutation and selection operators displayed good performance and should be considered in future research for additional

analysis. Naturally, modifications in selection procedure are much easier to do for a GA since its representation is easier to control.
The results of statistical analysis show that there exist significant statistical differences in the performance between the algorithms. We performed 2 kinds of analysis where the best algorithms are different. In the analysis based on average values best results are obtained for GA_BAL and GP_ALL+VAR algorithms. The analysis based on the threshold values is heavily influenced by the choice of the threshold levels. We displayed here one example of that analysis where the best algorithms were GA and GP_ALL+VAR. On the basis of several conducted analysis we conclude that there are better algorithms to use than those commonly used [5] [6].

4.1 Further Research

We are interested in the further investigation of the influence of cryptographic properties in the evolution process. There is possibility that some other set of properties would yield better results. Additionally, there are properties we did not consider and they are important for the resilience against some cryptographic attacks (i.e. transparency order that is related with side-channel attacks). From other perspective, we are interested in experimenting with some new algorithms like Cartesian Genetic Programming or Estimation of Distribution Algorithms. Initial experiments with those two algorithms give promising results. Of course, there is always the need to expand the research to the Boolean functions with a various number of inputs.

5. CONCLUSIONS

Finding Boolean functions with good cryptographic properties is a difficult problem. That is especially so because some of the properties are conflicting so it is not even possible to reach all optimal values. Research on this subject was mostly reserved for a cryptographic perspective: to find the best possible solution. In that kind of research, evolutionary computation is just a tool so there is little data on the performance of different algorithms. In this paper we tried to find good Boolean function but with the emphasis on the evolutionary computation algorithms. Results showed that there is a lot of room for improvements since the experiments showed that the best algorithms are not those commonly used.

6. REFERENCES

[1] The R project for statistical computing, 2013.

[2] H. Aguirre, H. Okazaki, and Y. Fuwa. An evolutionary multiobjective approach to design highly non-linear boolean functions. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO'07*, pages 749–756, 2007.

[3] T. Bartz-Beielstein. *Experimental Research in Evolutionary Computation: The New Experimentalism*. Springer, New York, USA, 2006.

[4] A. Braeken. *Cryptographic Properties of Boolean Functions and S-Boxes*. PhD thesis, Katholieke Universiteit Leuven, 2006.

[5] L. Burnett. *Heuristic Optimization of Boolean Functions and Substitution Boxes for Cryptography*. PhD thesis, Faculty of Information Technology, Queensland University of Technology, 2005.

[6] L. Burnett, W. Millan, E. Dawson, and A. Clark. Simpler methods for generating better boolean functions with good cryptographic properties. *Australasian Journal of Combinatorics*, 29:231–247, 2004.

[7] J. A. Clark, J. L. Jacob, S. Stepney, S. Maitra, and W. Millan. Evolving boolean functions satisfying multiple criteria. In *Progress in Cryptology - INDOCRYPT 2002*, pages 246–259, 2002.

[8] T. W. Cusick and P. Stanica. *Cryptographic Boolean Functions and Applications*. Elsevier Inc., San Diego, USA, 2009.

[9] S. Garcia, D. Molina, M. Lozano, and F. Herrera. A study on the use of non-parametric tests for analyzing the evolutionary algorithms? behaviour: a case study on the CEC 2005 special session on real parameter optimization. *Journal of Heuristics*, (15):617–644, 2009.

[10] K. Goossens. Automated creation and selection of cryptographic primitives. Master's thesis, Katholieke Universiteit Leuven, 2005.

[11] D. Jakobovic and et al. Evolutionary computation framework, Dec. 2011.

[12] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC, Boca Raton, 2008.

[13] F. Laffite. *The boolfun Package: Cryptographic Properties of Boolean Functions*.

[14] S. Maitra. Autocorrelation properties of correlation immune boolean functions. In *Progress in Cryptology - INDOCRYPT 2001*, pages 242–253, 2001.

[15] W. Millan, A. Clark, and E. Dawson. Heuristic design of cryptographically strong balanced boolean functions. In *Advances in Cryptology - EUROCRYPT '98*, pages 489–499, 1998.

[16] D. Ortiz-Boyer, C. Hervas-Martinez, and N. Garcia-Pedrajas. Improving crossover operators for real-coded genetic algorithms using virtual parents. *Journal of Heuristics*, (13):265–314, 2007.

[17] M. Read. Explicable boolean functions. Master's thesis, Department of Computer Science, The University of York, 2007.

[18] C. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):656–715, 1949.

[19] D. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman and Hall/CRC, fourth edition, 2007.

[20] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, Apr. 1997.

[21] Q. Zhang and Y.-W. Leung. An orthogonal genetic algorithm for multimedia multicast routing. *IEEE Transactions on Evolutionary Computation*, 3(1):53–62, Apr. 1999.

[22] X. Zhang and Y. Zheng. Gac-the criterion of global avalanche characteristics of cryptographic functions. *Journal of Universal Computer Science*, 1(5):316–333, 1995.