Applied Soft Computing 40 (2016) 635–653

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Cryptographic Boolean functions: One output, many design criteria

Stjepan Picek[a,b,*], Domagoj Jakobovic[b], Julian F. Miller[c], Lejla Batina[d], Marko Cupic[b]

[a] KU Leuven, ESAT/COSIC and iMinds, Kasteelpark Arenberg 10, bus 2452, B-3001 Leuven, Heverlee, Belgium
[b] Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia
[c] Department of Electronics, University of York, UK
[d] Digital Security Group, Radboud University, The Netherlands

ARTICLE INFO

Article history:
Received 17 June 2015
Received in revised form 3 October 2015
Accepted 29 October 2015
Available online 10 December 2015

Keywords:
Evolutionary algorithms
Boolean functions
Cryptography
Comparison
Analysis

ABSTRACT

Boolean functions represent an important primitive in the design of various cryptographic algorithms. There exist several well-known schemes where a Boolean function is used to add nonlinearity to the cipher. Thus, methods to generate Boolean functions that possess good cryptographic properties present an important research goal. Among other techniques, evolutionary computation has proved to be a well-suited approach for this problem. In this paper, we present three different objective functions, where each inspects important cryptographic properties of Boolean functions, and examine four evolutionary algorithms. Our research confirms previous results, but also sheds new insights on the effectiveness and comparison of different evolutionary algorithms for this problem.

© 2015 Elsevier B.V. All rights reserved.

1. Introduction

In the last few decades there has been significant research on Boolean functions, for cryptography as well for other uses like algebraic coding and sequence design [1–4]. Accordingly, there exist many different approaches for the construction of Boolean functions as well as numerous rationales behind the choice of properties relevant for such functions. Although Boolean functions in cryptography have a less prominent position than 20 or more years ago, they still represent an important cryptographic primitive. In this paper, we concentrate on the use of Boolean functions as building blocks in filter and combiner generators [4].

Boolean functions are often the only nonlinear element in stream ciphers and without them a cipher would be trivial to break. Therefore, it is not surprising that there exists a substantial body of work on methods of generating Boolean functions. However, such applications of Boolean functions in cryptography are not the only ones. For instance, they can be also used to resist side-channel attacks. When discussing side-channel countermeasures, one important class contains masking schemes. In masking schemes, one randomizes the intermediate values that are processed by the cryptographic device. One obvious drawback of such an approach is the masking overhead that can be substantial in embedded devices or smart cards. It has been shown that correlation immune Boolean functions that have minimal Hamming weight reduce the masking overhead [5,6]. However, most of the algebraic constructions are designed to produce balanced or bent Boolean functions and are therefore not suitable for this task. Consequently, it would be beneficial to have some other method of constructing Boolean functions.

We distinguish between three main approaches for generating Boolean functions for cryptographic usages: algebraic constructions, random generation and heuristic constructions (and various combinations of these approaches) [7]. Algebraic constructions use some mathematical procedure to create a Boolean function with good cryptographic properties. One example of such a construction is the Maiorana-McFarland construction [4]. Random generation of Boolean functions also has its strong points, the most prominent being that it is easy and fast. However, the resulting Boolean functions usually have suboptimal properties for cryptographic usages [8]. Heuristic methods offer an easy and efficient way of producing a large number of Boolean functions with very good cryptographic properties [2]. Among different heuristic approaches, evolutionary computation (EC) and more specifically evolutionary algorithms (EAs) offer highly competitive results when generating Boolean functions for cryptography [9,10]. It is worth mentioning that EAs can be used either as the primary or the secondary construction method. In primary constructions one obtains new functions without using known ones. In secondary constructions, one uses

* Corresponding author at: KU Leuven, ESAT/COSIC and iMinds, Kasteelpark Arenberg 10, bus 2452, B-3001 Leuven, Heverlee, Belgium. Tel.: +385 98226407.
E-mail address: stjepan@computer.org (S. Picek).

http://dx.doi.org/10.1016/j.asoc.2015.10.066
1568-4946/© 2015 Elsevier B.V. All rights reserved.

already known Boolean functions to construct new Boolean functions (either with different properties or sizes) [4].

In this paper, we experiment with several different EAs in order to explore their efficiency in the evolution of Boolean functions with properties necessary for use in cryptography. More precisely, we investigate Genetic Algorithms (GAs), Genetic Programming (GP), Cartesian Genetic Programming (CGP), and Evolution Strategies (ES). Furthermore, instead of concentrating on only one objective, we investigate three objectives, represented with different fitness functions. Since we experiment with several methods and fitness functions, it is not feasible to conduct experiments with all possible parameter combinations. Therefore, we restrict our attention to combinations which, based on our previous results and usual settings for those algorithms, provide acceptable results.

Since an investigation of any one of the four aforementioned algorithms on a single objective could easily constitute a whole paper, this work should be regarded as a practical guide and not as an in-depth analysis. However, it is worth noting that we perform more than 30,000 independent experimental runs for various EAs in order to conduct statistical analysis. We concentrate here only on Boolean functions with eight inputs. Eight inputs is a relatively small size for Boolean functions for stream ciphers (it is commonly considered that 13 is a strict minimum for resisting algebraic attacks), but we still believe it is an interesting case. Indeed, for instance ciphers RAKAPOSHI [11] and Achterbahn [12] use Boolean functions of that size. Evolving Boolean functions is a challenging task because there exist 2^{2^n} possible Boolean functions of n inputs (for eight inputs this gives 2^{256} candidate solutions). Therefore, with anything more than five inputs it is impossible to do an exhaustive search. Furthermore, with larger sizes of Boolean functions the search space grows, but also the computational complexity of their various properties grows. Thus, it is unrealistic to expect that EAs can work for much larger sizes than 13 inputs. This is because the computation of some properties like the algebraic immunity, and even more the fast algebraic immunity, becomes rapidly prohibitive for large numbers of inputs.

The rest of this paper is organized as follows. In Section 2, we present several applications of stochastic algorithms when generating Boolean functions appropriate for cryptographic usages. Section 3 presents relevant representations and cryptographic properties of Boolean functions. Next, in Section 4 we give fitness functions used in our experiments. Section 5 deals with the experimental setup and EAs we use. In Section 6, we give results for each of the objective functions as well as a short discussion on the results. Finally, we end with a short summary in Section 7.

2. Related work

As noted, there have been many applications of heuristic methods for the generation of Boolean functions for cryptographic usages. Here, we describe previous work directly related to our investigation in a chronological order. As far as the authors know, the first application of GAs to the evolution of cryptographically suitable Boolean functions emerged in 1997 when Millan et al. experimented with GAs to evolve Boolean functions with high nonlinearity [13]. In his thesis, Clark presented several applications of optimization techniques in the field of cryptology [14]. One of the applications is the evolution of Boolean functions with high nonlinearity using GA and hill climbing techniques. Millan, Clark, and Dawson used GAs to evolve Boolean functions that have high nonlinearity [15]. In conjunction with the GA they used hill climbing together with a resetting step in order to find Boolean functions with even higher nonlinearity for sizes of up to 12 inputs. More specifically, when discussing Boolean functions with eight inputs, they found balanced functions with nonlinearity 112 and correlation immunity equal to one.

Millan et al. used variations of a hill climbing method in order to find Boolean functions that have high nonlinearity and low autocorrelation [16].

Clark and Jacob experimented with two-stage optimization to generate Boolean functions with high nonlinearity and low autocorrelation [17]. They used a combination of simulated annealing (SA) and hill climbing with a cost function motivated by Parseval theorem. Clark et al. used SA to generate Boolean functions with cryptographically relevant properties where they considered balanced function with high nonlinearity and with the correlation immunity less and equal to two [18].

Kavut and Yücel developed an improved cost function for a search that combines SA and hill climbing [19]. In their approach, the authors were able to find some functions of eight and nine inputs that have a combination of nonlinearity and autocorrelation values previously unattained. They also experimented with a three-stage optimization method that combines SA and two hill climbing algorithms with different objectives.

Clark et al. experimented with SA in order to design Boolean functions using spectral inversion [20]. They observed that many cryptographic properties of interest are defined in terms of the Walsh–Hadamard transform values. Therefore, they worked in the spectral domain where the cost function punishes those solutions that are not Boolean functions. More precisely, on the basis of Parseval's theorem one can infer what values should be in a Walsh–Hadamard spectrum, but it is impossible to say what the positions should be. Therefore, when generating a Walsh–Hadamard spectrum it is necessary to make an inverse transform to verify that the spectrum indeed maps to a Boolean function. Burnett et al. presented two heuristic methods where the goal of the first method was to generate balanced Boolean functions with high nonlinearity and low autocorrelation. The second method aimed to generate resilient functions with high nonlinearity and algebraic degree that maximizes the Siegenthaler inequality [21]. Millan, Fuller and Dawson proposed a new adaptive strategy for a local search algorithm for the generation of Boolean functions with high nonlinearity [8]. Additionally, they introduced the notion of the graph of affine equivalence classes of Boolean functions. Burnett in her thesis used three heuristic techniques to evolve Boolean functions [2]. The first method aimed to evolve balanced functions with high nonlinearity. The second method was used to find balanced Boolean functions with high nonlinearity that are correlation immune. The last method was used to find balanced functions with high nonlinearity and propagation characteristics different from zero. Aguirre et al. used a multi-objective random bit climber to search for balanced Boolean functions of size up to eight inputs that have high nonlinearity [22]. Their results indicate that the multi-objective approach is highly efficient when generating Boolean functions that have high nonlinearity. Izbenko et al. used a modified hill climbing algorithm to transform bent functions to balanced Boolean functions with high nonlinearity [23]. McLaughlin and Clark experimented with SA to generate Boolean functions that have optimal values of algebraic immunity, fast algebraic resistance, and algebraic degree [24]. In their work, they experimented with Boolean functions with sizes of up to 16 inputs.

Picek, Jakobovic, and Golub experimented with GA and GP to find Boolean functions that possess several optimal properties [9]. As far as the authors know, this is the first application of GP for evolving cryptographically suitable Boolean functions. Hrbacek and Dvorak used CGP to evolve bent Boolean functions of sizes up to 16 inputs [25] where the authors experimented with several configurations of algorithms in order to speed up the evolution process. They did not limit the number of generations and therefore they succeeded in finding bent function in each run for sizes between

6 and 14 inputs. Several EAs are used by Picek et al. to evolve Boolean functions that have better side-channel resistance [10]. This paper presented the first application of optimization techniques for searching for Boolean functions that have improved differential power analysis (DPA) resistance. With the goal of finding maximal nonlinearity values of Boolean functions, Picek et al. experimented with several EAs [26]. Furthermore, they combined optimization techniques with algebraic constructions in order to improve the search. Although they were unable to find eight inputs balanced Boolean function with nonlinearity equal to 118, they presented several possible avenues of work to follow when looking for highly nonlinear balanced Boolean functions. Picek et al. compared the effectiveness of CGP and GP algorithms when looking for highly nonlinear balanced Boolean functions of eight inputs [27]. In this work, the authors showed that CGP performs favorably when compared with some other evolutionary approaches when evolving cryptography relevant functions. Finally, Picek et al. investigated several EAs in order to evolve Boolean functions with different values of the correlation immunity property. In the same paper, the authors also discussed the problem of finding correlation immune functions with minimal Hamming weight where they experimented with Boolean functions that have eight inputs [28].

It is evident from the list above (we emphasize that this is not even a complete list) there has been a multitude of works in the application of optimization techniques to the generation of Boolean functions. Thus, what separates this work from the others? Firstly, although we experiment with the same (or very similar) objectives as in some of the related literature, we use different EAs. Indeed, some of the algorithms we use were never before applied on such problems. Furthermore, we conduct an extensive analysis on the influence of certain algorithm parameters in the effectiveness of the search. Lastly, we give comprehensive comparison of EAs for evolving cryptographically suitable Boolean functions.

3. Boolean functions properties and representations

Let n, m be positive integers, i.e. $n, m \in \mathbb{N}$. The set of all n-tuples of the elements in the field $\mathbb{F}_2$ is denoted as $\mathbb{F}_2^n$ where $\mathbb{F}_2$ is the Galois field with 2 elements. The set $\mathbb{F}_2^n$ represents all binary vectors of length n and it can be viewed as a $\mathbb{F}_2$-vectorspace [4]. The inner product of two vectors $\vec{a}$ and $\vec{b}$ is denoted as $\vec{a} \cdot \vec{b}$ and equals $\vec{a} \cdot \vec{b} = \oplus_{i=1}^{n} a_i b_i$. Here, "$\oplus$" represents addition modulo two (XOR). The Hamming weight (HW) of a vector $\vec{a}$, where $\vec{a} \in \mathbb{F}_2^n$, is the number of non-zero positions in the vector. The support *supp* of a vector $\vec{a}$, where $\vec{a} \in \mathbb{F}_2^n$, is the set containing the non-zero positions in the vector $\vec{a}$. An (n, m)-function is any mapping F from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. If m equals 1 then the function f is called a Boolean function. The set of all Boolean functions is denoted as $\mathcal{BF}_n$ [4].

3.1. Boolean function 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 $(f(\vec{0}), \ldots, f(\vec{1}))$ that contains the function values of f, ordered lexicographically, i.e. $\vec{a} \preceq \vec{b}$ [4].

The *support supp* of a Boolean function f is a set containing the non-zero positions in the truth table (TT) representation, i.e. $supp(f) = \{x : f(x) = 1\}$, see e.g. [4].

The Walsh–Hadamard transform W_f (some authors also call it Walsh transform or Hadamard transform) is a second unique representation of a Boolean function that measures the correlation between $f(\vec{x})$ and the linear function $\vec{a} \cdot \vec{x}$ [4,30]. The Walsh–Hadamard transform of a Boolean function f equals:

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

A third unique representation of a Boolean function f on $\mathbb{F}_2^n$ is a representation by means of a polynomial in $\mathbb{F}_2[x_0, \ldots, x_{n-1}]/(x_0^2 - x_0, \ldots, x_{n-1}^2 - x_{n-1})$. This form is called algebraic normal form (ANF) [4]. ANF is a multivariate polynomial defined as [31]:

$$f(\vec{x}) = \oplus_{\vec{x} \in \mathbb{F}_2^n} h(\vec{a}) \cdot \vec{x}^{\vec{a}}, \tag{2}$$

where $h(\vec{a})$ is defined by the Möbius inversion principle:

$$h(\vec{a}) = \oplus_{\vec{x} \preceq \vec{a}} f(\vec{x}), \text{ for any } \vec{a} \in F_2^n. \tag{3}$$

For a Walsh–Hadamard transform, a Boolean function f is balanced if [29,32]:

$$W_f(\vec{0}) = 0. \tag{4}$$

The nonlinearity N_f of a Boolean function f can be expressed in terms of the Walsh–Hadamard coefficients as [4]:

$$N_f = 2^{n-1} - \frac{1}{2} \max_{\vec{a} \in \mathbb{F}_2^n} |W_f(\vec{a})|. \tag{5}$$

A Boolean function f is bent if it has maximum nonlinearity equal to [4,33,34]:

$$N_f = 2^{n-1} - 2^{\frac{n}{2}-1}. \tag{6}$$

A Boolean function f is correlation immune of order t (in brief, $CI_f(t)$) if the output of the function is statistically independent of the combination of any t of its inputs [35]. For the Walsh–Hadamard spectrum it holds that [36]:

$$W_f(\vec{a}) = 0, \text{ for } 1 \leq HW(\vec{a}) \leq t. \tag{7}$$

A Boolean function f is t-resilient if it is balanced and with correlation immunity of degree t [35].

The algebraic degree *deg* of a Boolean function f is defined as the number of variables in the largest product term of the ANF of the function with a non-zero coefficient [4,37]:

$$deg = \max(HW(\vec{a}) : h(\vec{a}) = 1). \tag{8}$$

Here, $h(\vec{a})$ is defined by the Möbius inversion principle.

The algebraic immunity (AI) of a Boolean function f is the lowest degree of the function g from $\mathbb{F}_2^n$ into $\mathbb{F}_2$ for which $f \cdot g = \vec{0}$ or $(f \oplus \vec{1}) \cdot g = \vec{0}$ where f and g are Boolean functions. Here, $f \cdot g$ denotes the function whose support is the intersection of the supports of f and g. A function g such that $f \cdot g = \vec{0}$ is called an annihilator of f [4,31].

For an in-depth treatment of properties of Boolean functions, we refer interested readers to [4,29,37,38].

4. Fitness functions

The following properties of Boolean functions play an important role in the security of filter and combiner generators and are considered in our experiments: nonlinearity, correlation immunity, balancedness, bentness, algebraic immunity and algebraic degree [4]. Note that there exist other relevant properties, like the fast algebraic immunity [39], but we do not consider them in this paper. We follow this approach for two reasons. Firstly, these are the most researched properties in related literature. Secondly, we aim to experiment with a smaller number of the fitness functions, but with more EA variations within each objective. Furthermore, we define all fitness functions so that it is clearly possible to identify what is the goal, i.e. what properties Boolean functions need to possess in order to say they are the best possible (i.e. to find global optimum).

We start with the simplest fitness function and then gradually make it more complicated. Already the second function proves to be difficult, but this is because the underlying cryptographic problem is difficult and not because of the complexity of the fitness function (e.g. having a large number of terms).

4.1. The first objective

As stated, we start with the simplest fitness function in terms of number of parameters. This function has only one parameter, the nonlinearity property. In order to provide confusion, a Boolean function should lie at a large Hamming distance (HD) from all affine functions. The nonlinearity N_f of a Boolean function f is the minimum HD between the function f and affine functions [4].

The upper bound for the nonlinearity property is often called the covering radius bound and is given as follows:

$$N_f \leq 2^{n-1} - 2^{\frac{n}{2}-1}. \tag{9}$$

The maximum nonlinearity of functions in $\mathbb{F}_2^n$ coincides with the covering radius ρ of the first order binary Reed-Muller code, i.e. $RM(1, n)$ [4]. This bound can be strict only when a Boolean function f has an even number of inputs. If such functions have the maximum nonlinearity they are called bent functions. However, bent functions are not balanced, i.e. their values are not uniformly distributed and are therefore not appropriate for use in filter and combiner generators [4].

Although bent functions are not appropriate as parts of filter and combiner generators, there is nevertheless motivation in their generation. The first reason is that this problem can be used as a benchmark to test various EAs. The second reason lies in the fact that using bent functions it is possible to build highly nonlinear functions that are balanced [4].

How difficult is the problem of evolving bent Boolean functions? The answer to this question depends on the input size of a Boolean function and the number of bent Boolean functions. However, for the latter quantity, there is no known efficient lower bound for an n-variable Boolean function. A naive upper bound equals [4]:

$$upper\_bound = 2^{2^{n-1}+\frac{1}{2}\binom{n}{n/2}}. \tag{10}$$

We can observe that there exist a huge number of bent Boolean functions and therefore we could expect this problem not to be too difficult for the EAs. To evolve bent Boolean functions we use the following fitness function where the goal is *maximization*:

$$fitness_1 = N_f. \tag{11}$$

From Eq. (9) it is apparent that we are looking for a Boolean function with the nonlinearity equal to 120 (i.e. $|W_f(\vec{a})| = 2^{\frac{n}{2}}$ for all $\vec{a} \in \mathbb{F}_2^n$).

We note that here we work with a difficult fitness function from the EC perspective. This is apparent since we look for the maximal nonlinearity value only on a basis of the worst Walsh–Hadamard coefficient. This implies that there is no gradient information that can help lead the search from one nonlinearity level to the other. For example, when having nonlinearity 116, the next better one is 118 and between those two values there is no information that can help in the search, which means that at that moment we effectively do a random search. All the fitness functions presented here may be redefined to account for all the Walsh–Hadamard values, which in turn gives much more information about the quality of a certain solution.

4.2. The second objective

Sarkar and Maitra showed that if a $CI(t)$ (see Eq. (7)) Boolean function f has an even number of inputs n and $k \leq \frac{n}{2} - 1$, then its nonlinearity N_f has an upper bound as follows [40]:

$$N_f \leq 2^{n-1} - 2^{\frac{n}{2}-1} - 2^k, \tag{12}$$

where k equals $t+1$ if f is balanced or has Hamming weight divisible by 2^{t+1} and k equals t otherwise.

In the case when $k > \frac{n}{2} - 1$ then the nonlinearity has the upper bound:

$$N_f \leq 2^{n-1} - 2^k. \tag{13}$$

There exist bounds that are more precise, but the previously mentioned one is sufficiently precise to define our search objective [4]. In the second set of experiments we concentrate only on Boolean functions where $t + 1 \leq \frac{n}{2} - 1$. This nontrivial bound is called Sarkar-Maitra divisibility bound [4] and it implies that the maximum nonlinearity for $n = 8$ and $t = 0$ equals 118. Here, $t = 0$ means that the function is balanced, but with correlation immunity equal to zero.

In the second experiment, we search for a balanced function with the maximum possible nonlinearity and correlation immunity equal to zero. In doing so we use the following fitness function where the objective is *maximization*:

$$fitness_2 = BAL + N_f. \tag{14}$$

We use the balancedness property (BAL) as a penalty. As evident from the Algorithm 1, when the function is balanced, it receives a value (reward) of one while unbalanced functions get penalty proportional to their unbalancedness. We need to include the balancedness property in the fitness since, when working with representations different from bitstrings, it is difficult to create random initial population comprised only of balanced Boolean functions.

Algorithm 1 *(Calculate balancedness).*

```
ones = HW(TT)
zeros = 2^n - ones
if zeros == ones then
  return 1
else
  return |ones − zeros| · X
end if
```

We experimentally find that $X = -5$ scales well for Boolean functions with $n = 8$ inputs. We experimented with several algorithms for calculating balancedness penalty where results show similar performance as long as the penalty is calculated gradually.

Eq. (14) may also be restated as a two-stage fitness function. In that design configuration, only the balancedness is tested at first, and nonlinearity value is added only if the function is balanced. This represents an option commonly used in EAs when one is trying to optimize one quantity under a strict constraint. However, our previous results show that when evolving balanced Boolean functions with maximum nonlinearity, the one stage fitness (as in Eq. (14)) performs better than the two-stage approach [27].

4.3. The third objective

For a Boolean function to be useful in filter generators, it needs to be balanced, with high nonlinearity, high algebraic immunity, and a high algebraic degree. Furthermore, to be used in a combiner generator it also needs to have a good correlation immunity value [4,24,35].

Notice that in previous sentence we speak of "high" and "good" values. First, we discuss what the good values are, and then what are the upper bounds for the relevant properties. Next, we give a short overview of the security roles of each property in the cipher security context.

Boolean function needs to be balanced (for sufficiently large n) to prevent being able to distinguish its output from some random sequence [4]. Boolean functions need to have high nonlinearity value in order to resist against fast correlation attacks [41]. We already discussed what are the maximum possible values on nonlinearity for t-resilient Boolean functions with even number of inputs in Eqs. (12) and (13). In our second objective we search

for Boolean function with correlation immunity equal to zero and in our third objective we search for Boolean functions that have correlation immunity larger than zero.

Recall that earlier we mentioned that we are looking for Boolean functions that have good correlation immunity values. First, we need to answer what are good values. When a Boolean function is used in a filter generator, correlation immunity value of one is usually considered sufficient [42]. To obtain this value, often it is possible to replace a Boolean function that has correlation immunity equal to zero (but, with otherwise good properties) with a linearly equivalent one that has the correlation immunity equal to one [42]. However, when using Boolean functions in a combiner generator, high values of correlation immunity are necessary to resist the Siegenthaler correlation attack [35]. Therefore, in our experiments we look for Boolean functions that have correlation immunity equal to two which we believe is sufficient for a Boolean function with eight inputs.

Next, a Boolean function needs to have high enough value of algebraic degree in order to resist against Berlekamp-Massey attack [43] and Rønjom-Helleseth attack [44]. The correlation immunity and the algebraic degree are conflicting properties and it is not possible to obtain a Boolean function with both properties optimal. For a resilient Boolean function where $t > 1$ and $t \neq n-1$, the Siegenthaler inequality holds [35]:

$$t \leq n - deg - 1. \tag{15}$$

Otherwise, the inequality is as follows:

$$t \leq n - deg. \tag{16}$$

Lastly, Boolean functions need to have high values of algebraic immunity in order to resist against algebraic attacks [45]. The algebraic immunity is upper bounded by the number of input variables, more precisely [46]:

$$AI \leq \left\lceil \frac{n}{2} \right\rceil. \tag{17}$$

Therefore, in the third objective we aim to find Boolean functions that are suitable for combiner generators. Here, the goal is *maximization* of the following fitness function:

$$fitness_3 = BAL + t + \delta_{t,2}(N_f + AI + deg). \tag{18}$$

Our previous results show that if both correlation immunity and algebraic degree properties are present in the fitness function, algebraic degree value dominates over correlation immunity [9]. Here we use a two stage fitness in which a fitness bonus equal to the sum of nonlinearity N_f, algebraic immunity AI, and algebraic degree deg is awarded only to a genotype that is 2-resilient (i.e when BAL = 0 and t = 2); otherwise, the fitness is only the sum of balancedness penalty and the correlation immunity (Eq. (18)). The delta function $\delta_{t,2}$ takes the value one when $t = 2$ and is zero otherwise.

Based on the previous equations, in the third objective we look for balanced Boolean functions that have the nonlinearity 112, correlation immunity 2, algebraic degree 5, and algebraic immunity 4.

4.4. Modifications of fitness functions

We note that all fitness functions could have been stated differently. Here we discuss two most prominent possibilities. The first option is to change the way how the nonlinearity is calculated as already mentioned in Section 4.1. Instead of looking just the extreme value of the Walsh–Hadamard transform, one can use the following cost function [47]:

$$cost(f) = \sum_{\vec{a}} \left| \left| W_f(\vec{a}) \right| - X \right|^R, \tag{19}$$

where X and R are real valued parameters. We acknowledge that using the aforementioned function can result in significant improvements of results, but at the cost of two more additional parameters that need to be carefully chosen (which is not necessary trivial). The second option is concerned with the third objective and the way how we calculate the correlation immunity property. Instead of accepting only those Boolean functions that have the correlation immunity property equal to 2, we could work with the correlation immunity deviation as defined in [15]:

$$cidev_f(t) = \max(|W_f(\vec{a})|; 1 \leq |\vec{a}| \leq t). \tag{20}$$

Alternatively, one can use an even more complex function as given in [18]. However, here we do not follow those directions since our experiments [27,28] show that at least CGP and GP algorithms are powerful enough to find Boolean functions with correlation immunity larger than one. Therefore, once more, we can consider our experiments as the worst case scenario. Naturally, by using the correlation immunity deviation, one should expect the results to be better than with our fitness function. We leave such modifications of fitness functions for future work.

We conclude this section with the quote from [4]: "But designing constructions leading to large numbers of functions achieving good trade-offs between the nonlinearity, the algebraic degree and the resiliency order (if possible, on any numbers of variables) are still necessary for permitting to choose in applications cryptographic functions satisfying specific constraints."

5. Experimental setup

Here, we give details about experimental setup for each of the algorithms as well as the common parameters for all algorithms. For each of the algorithms we conduct a two parameter sweep in order to investigate the best combination of parameters. For evolutionary algorithms test suite we use the Evolutionary Computation Framework (ECF) [48]. Since there are more than two parameters for every algorithm, we try to look only at the most obvious choices for tuning. For that reason, with GA and GP, we use the k-tournament steady-state selection as given in Algorithm 2 (with $k = 3$) to avoid the need for crossover probability. Additionally, in order to have a baseline against which all algorithms efficiency can be compared, we also experiment with the random search.

Algorithm 2 *(Steady-state tournament selection).*

randomly select k individuals;
remove the worst of k individuals;
child = crossover (best two of the tournament);
perform mutation on *child*, with given individual mutation probability;
insert *child* into population;

5.1. Random search

In random search, solutions are generated by creating uniformly at random solutions in a form of a truth table. We generate 500,000 random solutions (Boolean functions) and test their cryptographic properties. Random search is constrained only to balanced solutions for the second and third objective.

5.2. Genetic algorithm

We use a simple GA with tournament selection where its size equals 3 [49]. A mutation is selected uniformly at random between a simple mutation, where a single bit is inverted, and a 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. For each of the fitness functions we experiment with population sizes

of 50, 100, 200, 500, 1000, 2000, 5000, and 10000 and individual mutation probabilities of 0.1, 0.3, 0.5, 0.7, and 0.9. It is important to note that we use the mutation probability to select whether an individual would be mutated or not, and the mutation operator is executed only once on a given individual; e.g. if the mutation probability is 0.9, then on average 9 out of every 10 new individuals will be mutated (see Algorithm 2), and one mutation will be performed on that individual.

5.3. Evolution strategy

When experimenting with ES we use $(\mu + \lambda)$-ES. In this algorithm, in each generation, parents compete with offspring and from their joint set μ fittest individuals are kept. In our experiments offspring population size λ has values of 5, 10, 15, 20, 50, and 100. Parent population size μ has values of 1, 2, 5, 10, 20, 50, 100, 200, and 500.

5.4. Genetic programming

We use tree-based GP in which a Boolean function is represented by a tree of nodes [50]. The function set for GP in all the experiments is OR, XOR, AND, XNOR, and AND with one input inverted. Terminals correspond to n Boolean variables. Boolean functions can be represented only in XOR and AND operators, but it is quite easy to transform it from one notation to the other. GP uses tournament selection with tournament size 3.

The crossover is performed with five different tree-based crossover operators selected at random: a simple tree crossover with 90% bias for functional nodes, uniform crossover, size fair, one-point and context preserving crossover [51]. We use a single mutation type, a subtree mutation, and experiment with tree depth sizes of 5, 7, 8, 9, and 11 and population sizes of 50, 100, 200, 500, 1000, 2000, 5000, and 10,000.

5.5. Cartesian genetic programming

Cartesian Genetic Programming (CGP) represents computational functions as a directed graph [1,52]. The function set n_f for the CGP is the same as for the GP. Setting the number of rows to be one and levels-back parameter to be equal to the number of columns is regarded as the best and most general choice [53]. This choice should be used when there is no specialist knowledge about the problem.

However, this still leaves us to decide the number of columns. Furthermore, CGP usually uses small population sizes and has no crossover operator [53]. Determining the best combination of maximum number of nodes (gates in this case) and mutation rate is an important step in hitting the parameter sweet spot for CGP. Indeed, it has been shown that generally very large genotypes and small mutation rates perform very well [54]. For this reason we experiment with genotype sizes 100, 300, 500, 700, 900, 1100, 1300, 1500, 1700, and 1900 and for mutation rates with values of 0.01, 0.03, 0.05, 0.07, 0.09, 0.11, and 0.13.

The number of input connections n_i for each node is two and the number of program output connections n_o is one. The population size for CGP equals five in all our experiments. For CGP individual selection we use a (1 + 4)-ES in which offspring are favored over parents when they have a fitness better than or equal to the fitness of the parent. The mutation operator is one-point mutation where the mutation point is chosen with a fixed probability. The number of mutated genes is defined as fixed percentage of the total number of genes. Note, the single output gene is not mutated and is taken from the last node in the genotype. The genes chosen for mutation might be a node representing an input connection or a function.

5.6. Representations

Recall, there are several unique representations of a Boolean function. GA and ES represent the individuals as strings of bits whose values define the TT of the Boolean function. GP uses a representation where individuals are trees of Boolean primitives which are then evaluated according to the TT they produce. For CGP, solutions are directed graphs that are also evaluated in accordance to the TT they produce.

5.7. Common parameters

The following parameters of the experiments are common for every objective. The size of Boolean function is eight (the size of the truth table is 256) and the number of independent trials M for each experiment is 50. Stopping criterion for all algorithms is 500,000 evaluations. One can argue that 500,000 evaluations is insufficient, but our previous results [27,28] as well as the results presented here show it is enough to reach desired values. Naturally, allowing more evaluations can only improve the results so we additionally present the results for all fitness functions and best performing algorithm combinations with 2,500,000 evaluations.

6. Results

In all experiments, we follow the same guidelines when presenting results. We give statistics related with the average fitness value for each of the algorithms and objectives. That average value is calculated as the mean value of the best obtained values over all independent experimental runs. However, that is not enough due to a fact that an algorithm can reach a high average fitness value, but still not be able to find optimal value. Therefore, we give statistics related to the number of times that the algorithm obtains the best possible value, which is either the global optimum or the best obtained value over all algorithms. In the cases where an algorithm could not find any global optimum, we skip such statistics.

For all algorithms, we run a parameter sweep of two parameters we consider to play a crucial role in their efficiency. Naturally, some other choice of parameters (or parameter values) could change the results. However, since adding the potential third parameter of interest would significantly increase the number of combinations, we choose fixed settings for the remaining parameters. Some future work could investigate the influence of other parameters in the algorithm behavior. When presenting heatmap figures, in the cases when there exists a single best average solution, we show it enclosed in a rectangular for better visibility.

6.1. The first objective

Recall that we set for each objective a fitness function where we a priori know what the global optimum value is. For the first objective as presented in Eq. (11), it is easy to check that the maximal value is 120 for a function with eight inputs (see Eq. (9)). Furthermore, as already said, the space of bent Boolean functions is still very large although much smaller than the space of Boolean functions of a certain input size. Therefore, we could expect this problem to be of moderate difficulty for EAs. With the random search algorithm, the average nonlinearity value equals 103.85 with no solutions that reached the maximal nonlinearity value.

6.1.1. Cartesian genetic programming

We present a heatmap representation of the average fitness values for CGP for all mutation rates and genotype sizes in Fig. 1. As evident, a large number of parameter combinations produce results with high average fitness value. The highest result (119.04) is achieved for a genotype size of 1500 and a mutation rate of 13%.

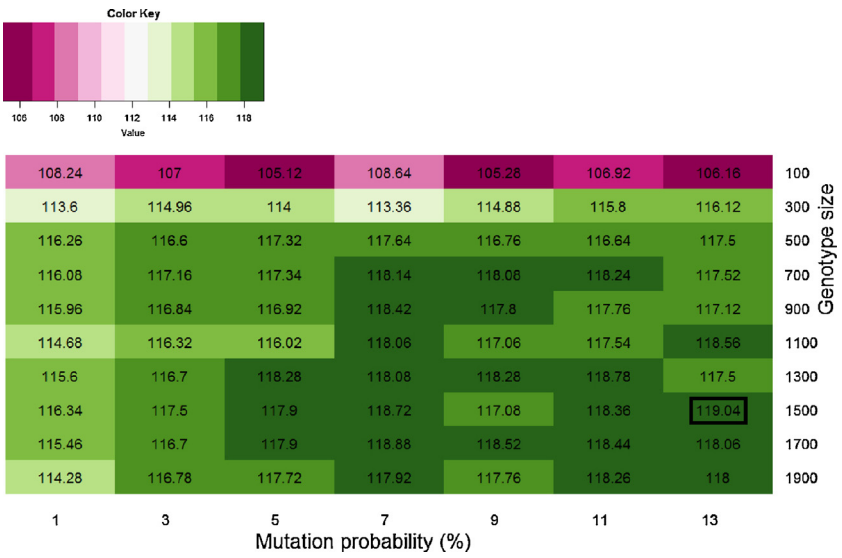


Fig. 1. Heatmap of the average values for CGP, fitness function 1.

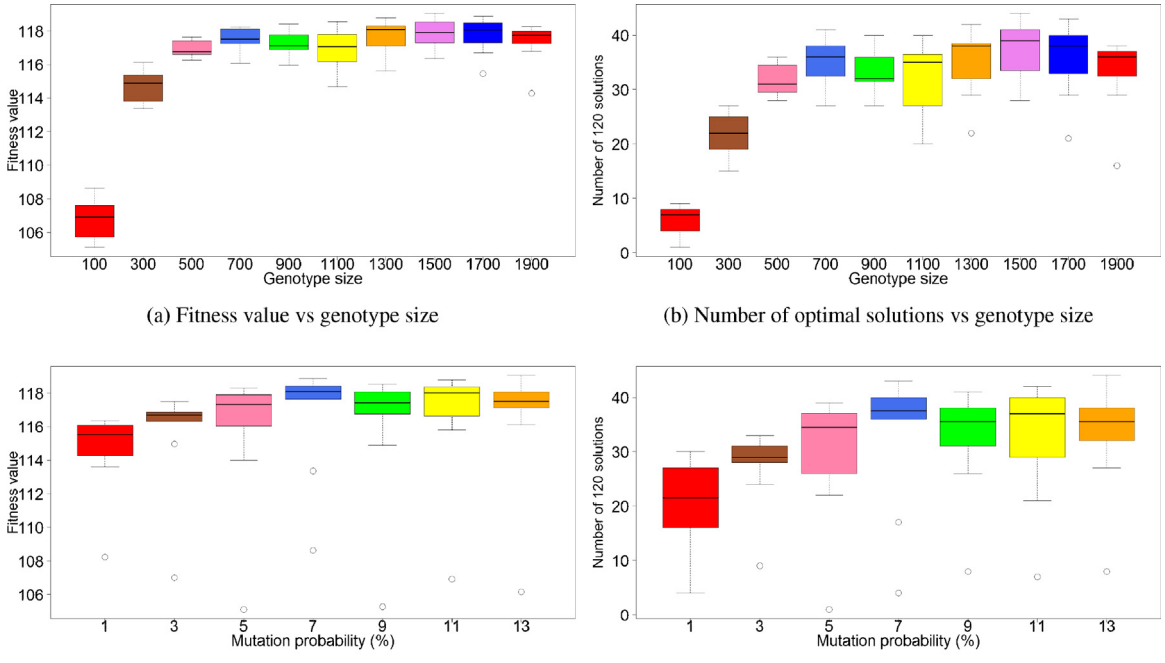
This translates to 44 out of 50 runs that finished with the nonlinearity of 120. In Fig. 2a–d, we show the influence of genotype size and mutation probability on the effectiveness of CGP.

From the figures it can be observed that the genotype size of 1500 and mutation probability of 7% give the best results on average. In Fig. 3, we display the average number of active nodes for all mutation rates and genotype sizes. Our results show that the average size goes from around 14 nodes to 90 nodes. The parameter combination that achieved the best results has on average 80 active nodes.

6.1.2. Genetic programming

We present a heatmap representation of the average fitness values for GP in Fig. 4. Here, we observe a somewhat surprising result; smaller tree depths perform much better than the larger ones. This behavior is especially obvious for tree depths of 3 and 5 where the average fitness value equals 120. This means that the GP found the global optimum in all 50 independent runs. Therefore, the first objective seems to be very easy for GP with small tree depths. However, a more detailed analysis of the solutions reveals that the situation is less straightforward. With a tree depth of 3, all solutions are extremely short (average tree size is 17.8) and it can be observed that a significant number of solutions repeats.

In Fig. 5a–d, we display the influence of the population size and tree depth on the algorithm effectiveness. From the results, it is apparent that the population size does not have a large influence while the tree depth influences the algorithm results dramatically. Even when disregarding the smallest tree depths, we see that for this problem smaller depths behave favorably. As an example, we note that when having the tree depth of 11, most of the solutions reach tree size of several thousand, while having a relatively low nonlinearity value.



(a) Fitness value vs genotype size

(b) Number of optimal solutions vs genotype size

(c) Fitness value vs mutation probability

(d) Number of optimal solutions vs mutation probability

Fig. 2. Mutation probability and genotype size dependency, CGP, function 1.

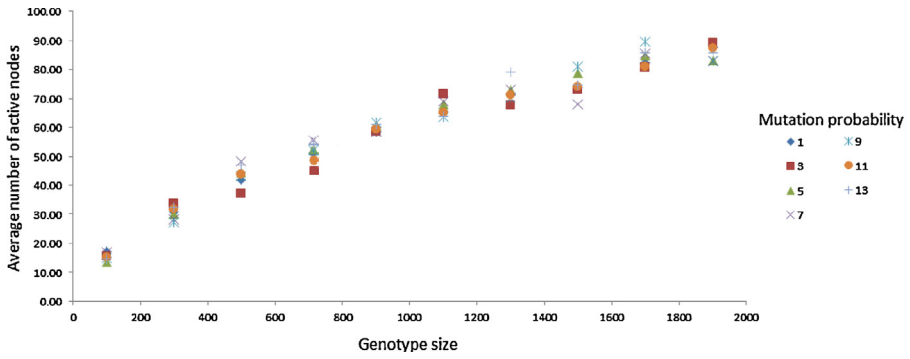


Fig. 3. Average number of active nodes for all mutation rates, fitness function 1.

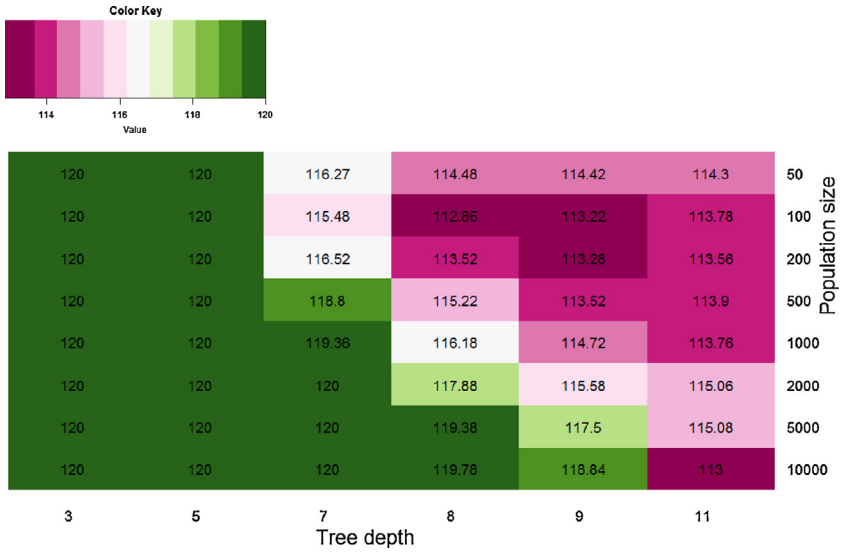
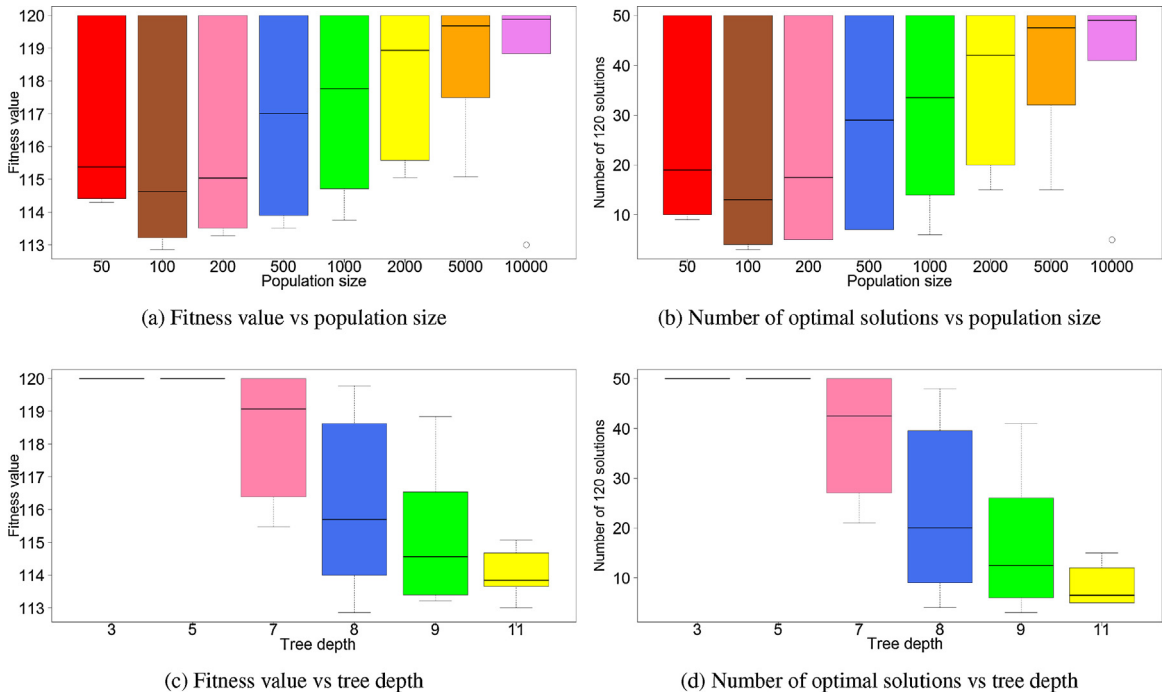


Fig. 4. Heatmap of the average values for GP, fitness function 1.



(a) Fitness value vs population size

(b) Number of optimal solutions vs population size

(c) Fitness value vs tree depth

(d) Number of optimal solutions vs tree depth

Fig. 5. Population size and tree depth dependency, GP, fitness function 1.

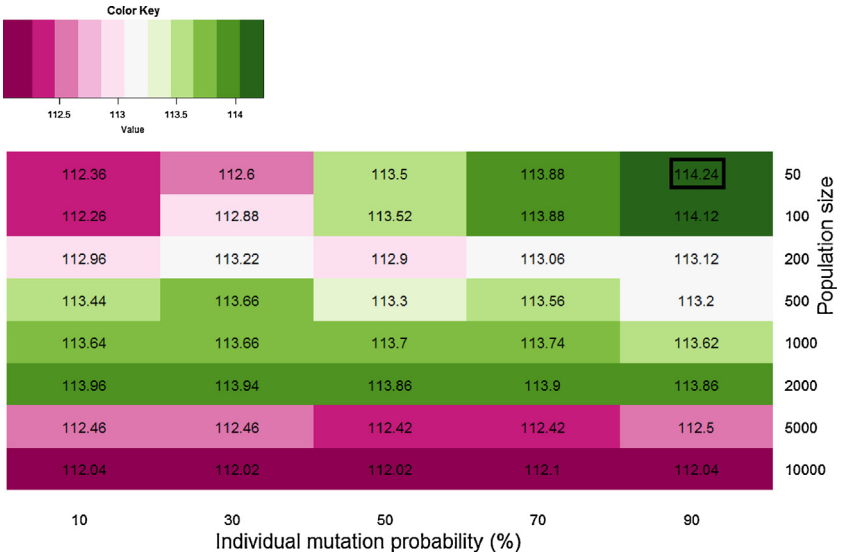


Fig. 6. Heatmap of the average values for GA, fitness function 1.

6.1.3. Genetic algorithm

After the previous two algorithms that perform extremely well on the first objective, we come to the first algorithm for which this problem can be regarded difficult. In Fig. 6 we see that GA reaches average fitness values between 112 and 115. The best average score has the algorithm with 0.9 mutation rate and population size of only 50, where the average value is 114.24. This can be explained by the fact that the algorithm simply needs more evaluations to reach good values. Furthermore, since the population size is directly proportional to the number of evaluations, the algorithm does not have enough time to converge.

In Fig. 7a and b, we observe the algorithm behavior for different mutation probabilities and population sizes, respectively. We see that the best results on average are for high mutation probabilities. When examining the influence of the population size on the algorithm effectiveness, we observe that larger population sizes have less diversity, which indicates that most of the solutions get stuck in local optima. Furthermore, the biggest population sizes perform the worst due to the lack of time (evaluations) to converge. These results are in accordance with those from [15] where the authors also found that GA performance worsens with the population growth. We emphasize that no combination of GA parameters resulted in reaching the optimal value and therefore we do not present graphs showing number of optimal solutions found.

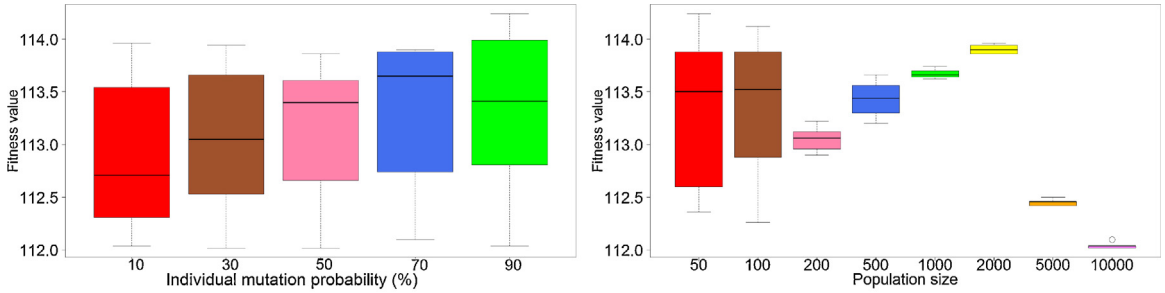
6.1.4. Evolution strategy

As evident from Fig. 8, the effectiveness of ES is similar to that of GA. The best average fitness value of 114 is reached for the two biggest parent population sizes - 200 and 500 individuals, regardless of the offspring population size.

Fig. 9a and b display the influence of offspring and parent population sizes on the average fitness value. We see from the figures that the offspring population size does not have importance while the parent population size influences the algorithm effectiveness drastically. As with the GA, there are no parameter combinations that result in reaching a global optimum. There could be some doubt whether the parameters are chosen smartly (i.e. could we chose better parameter ranges). However, since this is the first time, as far as the authors know, that the ES is used in order to evolve bent Boolean functions, these results should be regarded as a basic guideline for some future work.

6.1.5. Best algorithms for the first objective – long run

In this section, we compare best performing algorithm configurations when setting the number of evaluations to 25,00,000. Here, we experiment with CGP with genotype size of 1500 and mutation rate of 13%, GP with tree size 3 and population size 50, GA with mutation probability 9% and population size 50, and finally, ES with offspring size 10 and parent size 100. For GP and ES cases we determined the optimal parameter combination as the smallest (and therefore computationally easiest) of those that reached the maximum value for a certain algorithm. For CGP and GA, we simply use the one parameter combination that obtained the best result. In Fig. 10, we display convergence of algorithms during the evolution process. Since for the GP case already 500,000 evaluations were enough to reach the optimal value in all runs, there cannot be any



(a) Fitness value vs mutation probability

(b) Fitness value vs population size

Fig. 7. Population size and mutation probability dependency, GA, fitness function 1.

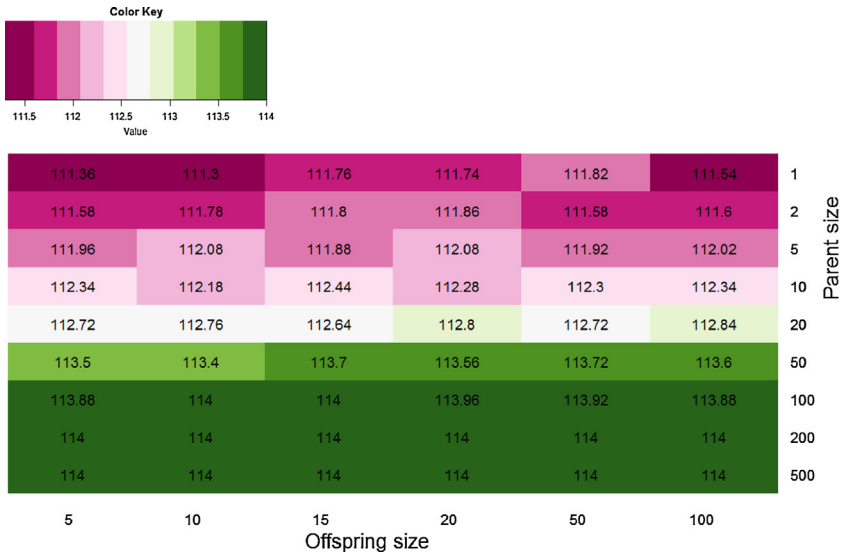
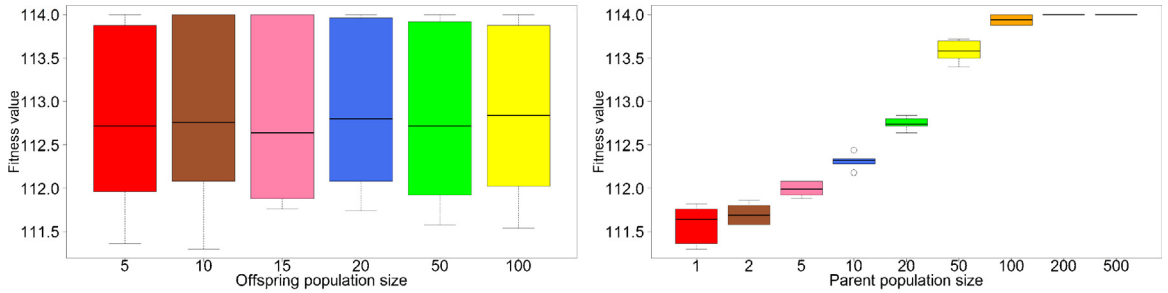


Fig. 8. Heatmap of the average values for ES, fitness function 1.



(a) Fitness value vs offspring size

(b) Fitness value vs parent size

Fig. 9. Offspring and parent population sizes dependency, ES, fitness function 1.

further improvement. For CGP we observe a slight increase in the average value, but still in several runs the algorithm did not reach the global optimum. For both GA and ES, we see no improvement in the average value with the increase of the number of evaluations.

6.1.6. *Discussion – the first objective*

How hard is it to evolve bent Boolean functions? This naturally depends on the size of the Boolean function and the algorithm we use. We see that CGP and GP perform well for functions with 8 inputs. If we consider all parameter options, we can conclude that GP is better than the CGP. However, if we compare our results with the results from Hrbacek and Dvorak [25], we see that CGP can reach 100% success rate as GP did here, but it requires more evaluations. Since Hrbacek and Dvorak give results based on the synthesis time, it is hard to make direct comparisons due to the differences in algorithms and their configurations as well as the hardware configurations.

GA and ES perform much worse on average, not being able to find any optimal solution. One can observe in Fig. 7a that the performance of GA improves with the increase of the mutation rate. Our results for GP and GA are in accordance with the results from [26]. As a future work, it would be interesting to see how far can we go in the evolution of bent Boolean functions. For now, Hrbacek and Dvorak went up to 16 inputs [25], but there remains the question: is that the maximum for the EAs? We believe it is not, since they achieved 100% success rate. Naturally, larger sizes would require even

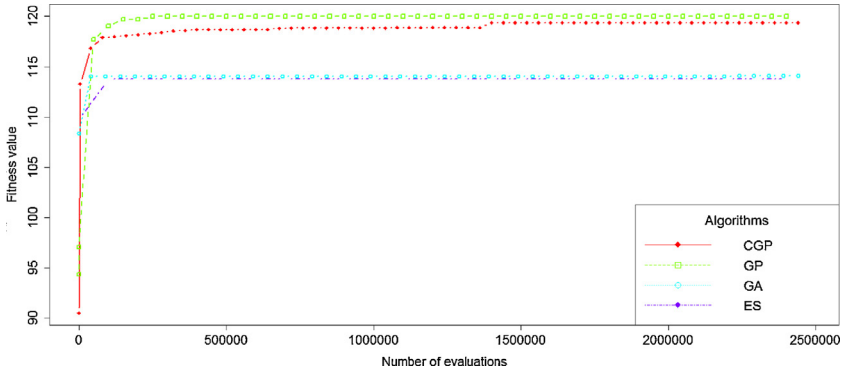


Fig. 10. Long run for the best performing algorithms, fitness function 1.

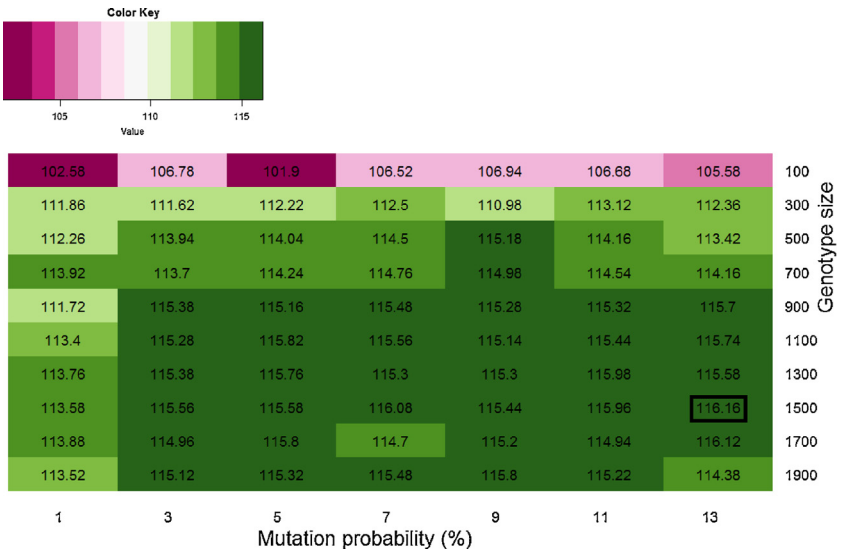


Fig. 11. Heatmap of the average values for CGP algorithm, fitness function 2.

further improvements/optimizations in the parameter settings and algorithm configurations.

A second interesting research avenue could be examining the structure of evolved bent Boolean functions; more precisely, to investigate whether it is possible to use EAs to evolve bent functions that are not normal. A bent Boolean function with $2n$ inputs is called normal if it is constant on an n-dimensional affine subspace [55]. Since until now no one has been able to find non normal bent functions of $n < 10$ [56] and weakly normal bent functions of $n < 10$ [57], this could also potentially lead to new, undiscovered bent functions. Therefore, experiments examining whether some of the evolved functions are non normal or weakly normal would be of high interest and would also help to determine which EAs find "more random" bent functions.

6.2. The second objective

In the second objective, we look for balanced Boolean functions with eight inputs that have nonlinearity 118. Unfortunately, we were not able to find any function with that nonlinearity value; thus, finding such a function is still an open question. We note that although the theory supports the existence of those functions [40], there is also an opinion in the cryptographic community that there are no balanced functions with 8 inputs and nonlinearity 118. If that is true, then the upper bound given in Eq. (12) is not strict. In accordance to the fact that we could not find Boolean functions with nonlinearity 118, we concentrate on the next best attainable nonlinearity – 116. Therefore, in the presented statistics we show the effectiveness of the algorithms when generating balanced Boolean functions with nonlinearity 116. After experiments with the random search, we observe that the average fitness equals 104.51, with no solutions that have nonlinearity value 116.

6.2.1. Cartesian genetic programming

In the second objective CGP outperforms all the other algorithms by far. The best average value is reached for a parameter combination of 1500 genotype size and mutation probability of 13% and it amounts to 116.16. Recall that the fitness function is the sum of balancedness and nonlinearity so the maximum fitness value is larger by 1 than the targeted nonlinearity value. We present the averaged results in Fig. 11.

Fig. 12 displays the average number of active nodes for all mutation rates and genotype sizes. Similar to the first fitness function, we observe a linear increase in the average number of nodes with the increase in the genotype size. The best parameter combination has 79 active nodes on average. In Fig. 13a–d, we display the effectiveness of CGP relative to the genotype size and mutation probability, respectively. We observe that CGP performs similarly for all genotype sizes except the smallest one (100). The best genotype size for this problem seems to be 1500. Again, when considering mutation probability we see that the algorithm variations perform similarly (with the exception of the smallest mutation probability of 1%). The best mutation probabilities are on average 5% and 13%.

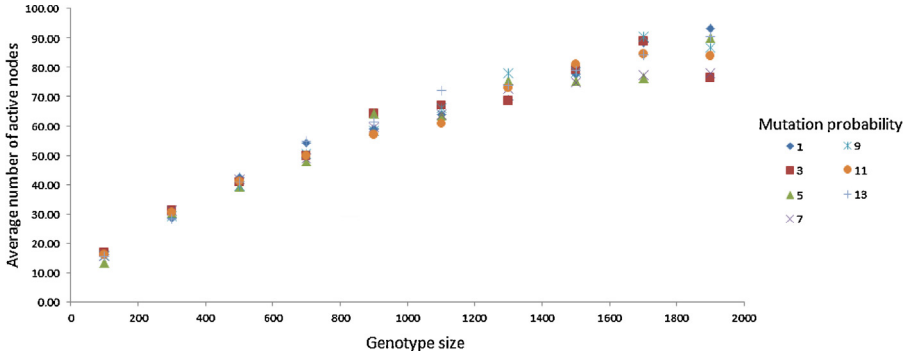
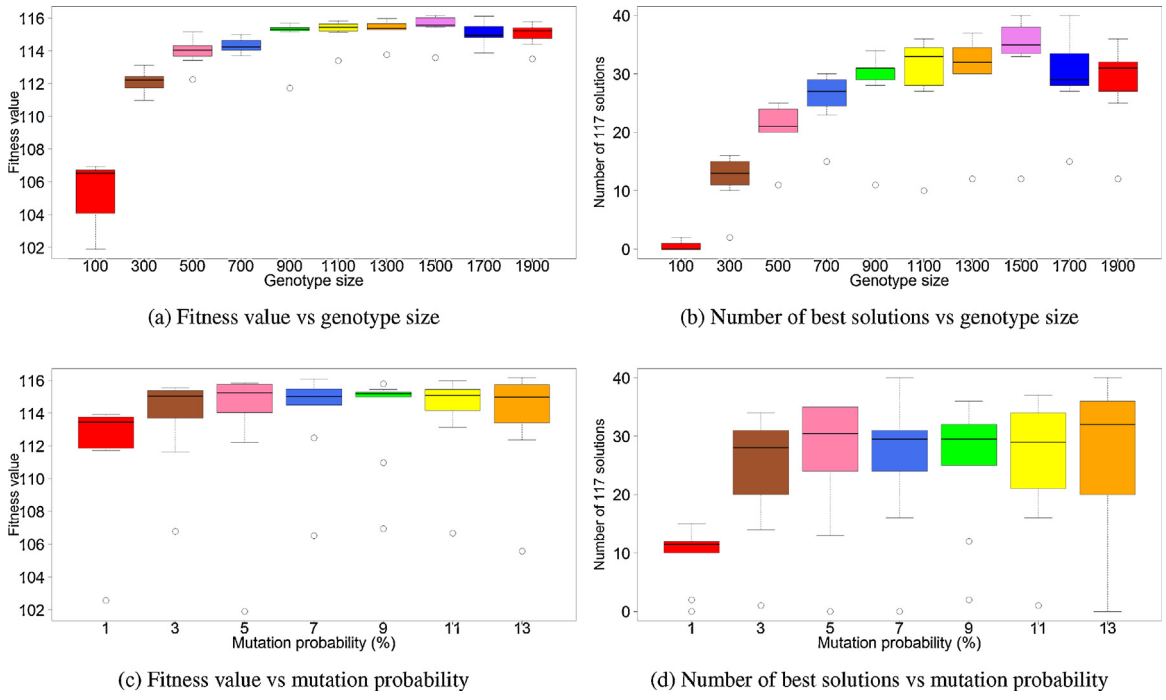


Fig. 12. Average number of active nodes for all mutation rates, fitness function 2.



(a) Fitness value vs genotype size

(b) Number of best solutions vs genotype size

(c) Fitness value vs mutation probability

(d) Number of best solutions vs mutation probability

Fig. 13. Mutation probability and genotype size dependency, CGP, fitness function 2.

6.2.2. Genetic programming

GP performs significantly worse than the CGP and it obtains the best average fitness value of 114.12 for a population size of 10,000 and a tree depth of seven. In Fig. 14, we give average fitness values for GP. These results are also in accordance to the results presented in [26].

We display results for the algorithm performance with respect to different tree depths and population sizes in Fig. 15a–d. For this problem, we can observe that the larger population sizes behave favorably when compared to the smaller ones. A sub-linear increase can be observed in the algorithm efficiency with increase in population size. When considering tree depths, we see that a value of seven seems to be the optimal choice out of those tested.

6.2.3. Genetic algorithm

When considering average fitness value for GA we observe in Fig. 16 that the best average fitness value is 115.08 for a population size of 100 and mutation probability of 0.9. Furthermore, a combination of population size 50 and mutation probability 0.9 gives only a slight worse result with an average fitness value of 115. Fig. 17a and b display fitness values for different mutation probabilities and population sizes. The two best performing variations also succeed in finding several balanced solutions with nonlinearity 116. Namely, the variation with the population size 50 found one solution with nonlinearity 116 and the combination with population 100 found four solutions. Nevertheless, we opted not to show graphs with the relationship of number of optimal solutions to the algorithm performance since there are only a few such solutions.

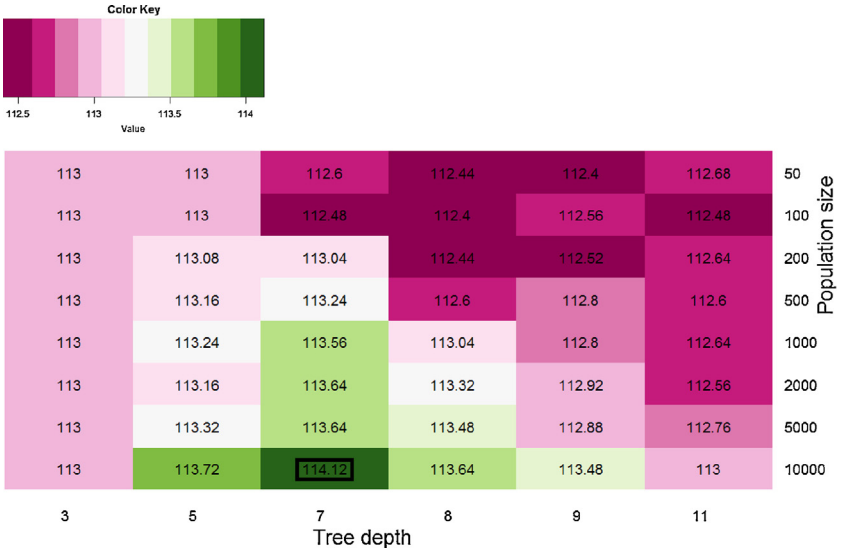
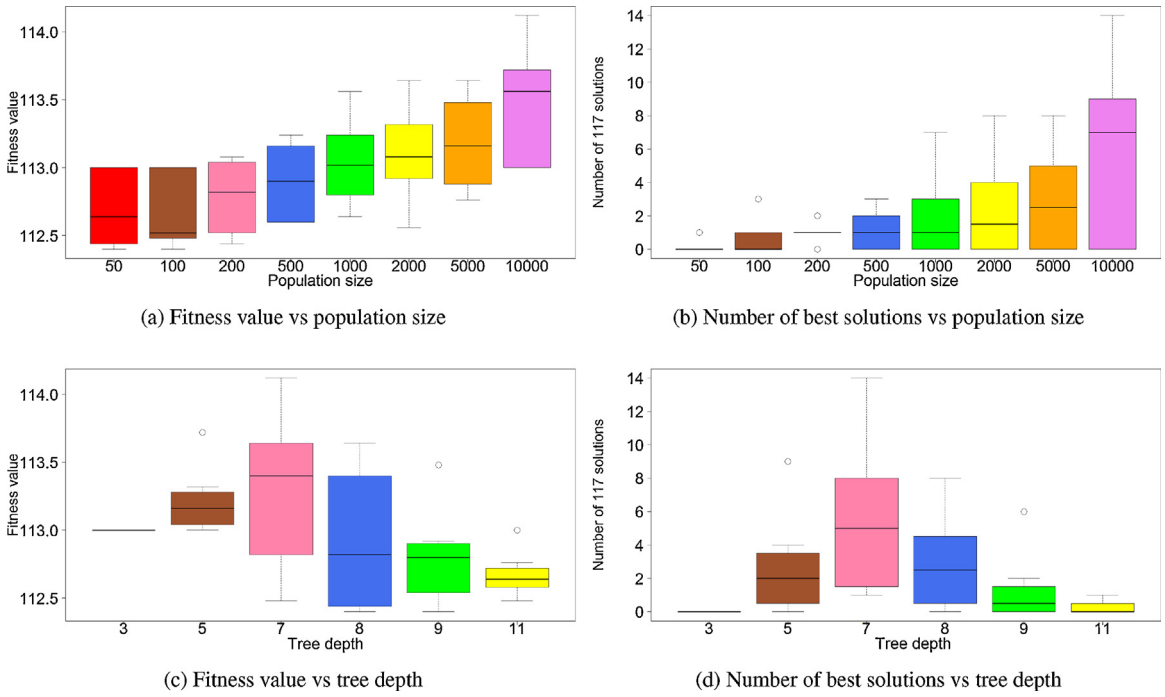


Fig. 14. Heatmap of the average values for GP, fitness function 2.



(a) Fitness value vs population size

(b) Number of best solutions vs population size

(c) Fitness value vs tree depth

(d) Number of best solutions vs tree depth

Fig. 15. Population size and tree depth dependency, GP, fitness function 2.

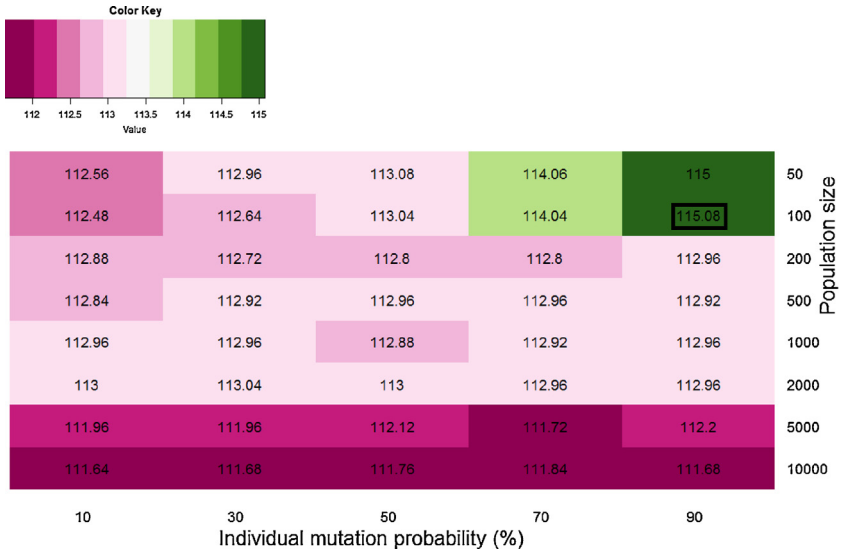
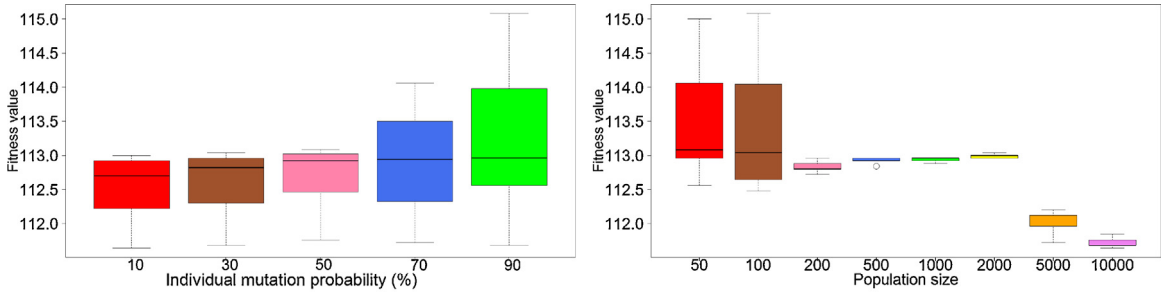


Fig. 16. Heatmap of the average values for GA, fitness function 2.



(a) Fitness value vs mutation probability

(b) Fitness value vs population size

Fig. 17. Population size and mutation probability dependency, GA, fitness function 2.

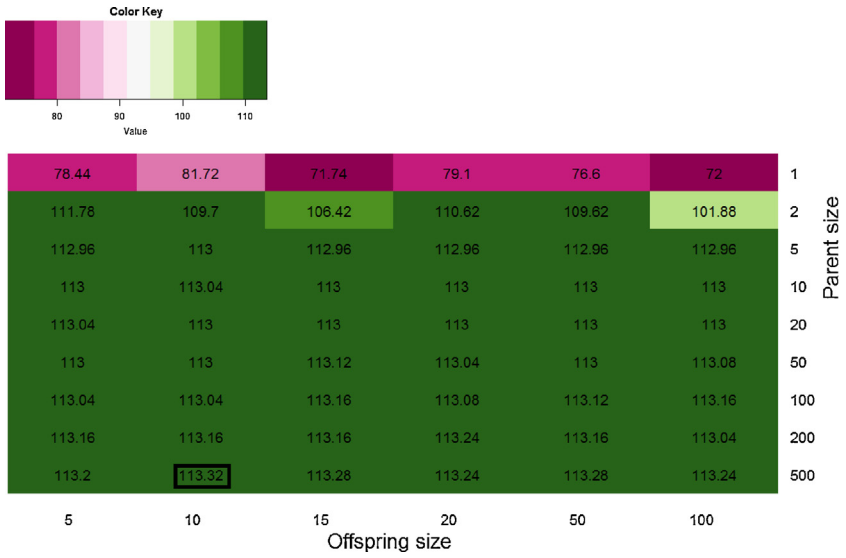


Fig. 18. Heatmap of the average values for ES, fitness function 2.

From the figures below, we see that for this problem the GA favors small population sizes and large mutation probabilities. The observation about the better performance of smaller population sizes is in accordance with the findings in [15].

6.2.4. Evolution strategy

Similarly to the first fitness function, ES performs better for larger parent population sizes where offspring population size has a small influence. In Fig. 18, we see that the best average fitness value is obtained for the parent size of 500 and offspring size of 10 and that value equals 113.32. As is the case with the first objective, ES is the worst out of the four examined EAs. We give the algorithm effectiveness for various offspring and parent sizes in Fig. 19a and b, respectively. The graphs show virtually no effect caused by the number of offspring and no effect caused by parent size except for two smallest parent sizes which perform significantly worse. From the results, we can also observe that there is little (or none) diversity in final solutions of independent runs. This indicates that the ES is stuck in some local optimum value.

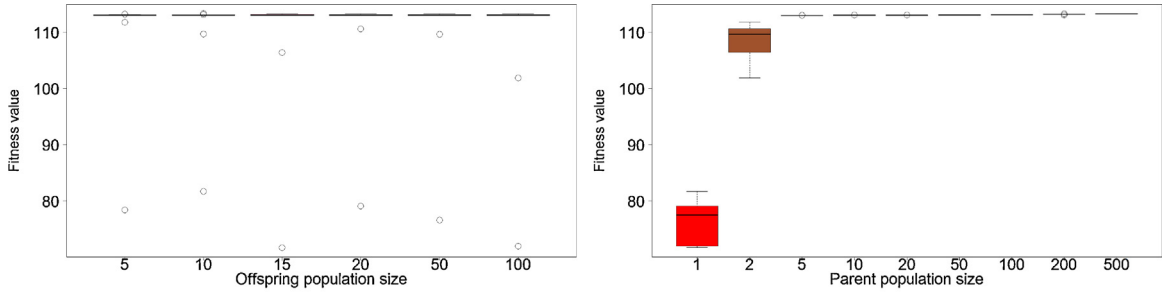
6.2.5. Best algorithms for the second objective – long run

As the best performing algorithms for the second objective, we use CGP with genotype size 1500 and mutation rate 13%, GP with tree depth 7 and population size 10,000, GA with population size 100 and individual mutation probability 90%, and ES with parent size 500 and offspring size 10. We display the results for 50 runs and 2,500,000 evaluations in Fig. 20. It is easy to see that the biggest winner is GP since it benefits the most from the additional evaluations. The average value after 500,000 evaluations was 114.12 and after 2,500,000 evaluations goes up to 116.68. CGP improves only slightly, from 116.16 to 116.32 and finally, GA and ES do not benefit at all from the additional evaluations.

6.2.6. Discussion – the second objective

Finding highly nonlinear balanced Boolean functions with eight inputs is at the same time easy and difficult. It all depends what one considers high nonlinearity. In the case that 112 or 114 represent sufficiently high nonlinearity value, it seems that all of the evaluated algorithms could be used (granted, with more or less success). However, if we want to reach the nonlinearity of 116 the situation starts to change. Immediately ES and even GA do not pose a viable choice. When comparing averaged results for GP and GA, we observe that the GA reaches a higher average value (115.08) than the GP (114.12). However, the GA finds 116 nonlinearity only rarely, but ends in most of the cases with nonlinearity 114. On the other hand, GP often reaches 116 nonlinearity, but also gets stuck with nonlinearity of only 112. When counting the number of times that the algorithm reached 116 nonlinearity, CGP performs by far the best. However, finding nonlinearity value of 118 seems to be impossible with algorithms like these (that is, in the case that such a function exists), especially without any improvements like adding hill climbing or seeding the initial population.

As evident from [26], seeding population with good solutions and running hill climbing (two-stage search) improves drastically the behavior of GP. It would be interesting to check what happens with CGP when coupled with those improvements. When



(a) Fitness value vs offspring size

(b) Fitness value vs parent size

Fig. 19. Offspring and parent population sizes dependency, ES, fitness function 2.

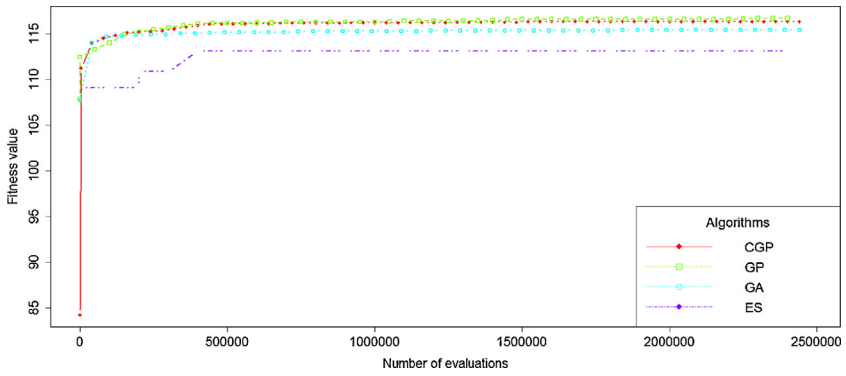


Fig. 20. Long run for the best performing algorithms, fitness function 2.

considering CGP we also see that the fitness function as given in Eq. (14) outperforms two-stage fitness function that is often used in CGP. This is a surprising result since that is often used in CGP, ever since it was first described [3,53]. This implies that more work should be done on a variety of problems to establish the relative merits of the two approaches. In addition, in [54] it was suggested that optimal mutation rates should decrease as genotype length increases. However, the results here confirm the findings from [27] that for the cryptographic problem we study this is not the case. Indeed fairly high mutation rates produce the best results. This is also surprising and merits further investigation.

Looking on a more general level, algorithms with the tree and graph representations outperform algorithms with the TT representation significantly. It would be beneficial to study why TT representation performs worse. One possible explanation is that there exist value and variable symmetries [58,59]. One interesting research avenue would be to check the fitness landscapes of Boolean functions with different nonlinearities [60]. Additionally, experiments with different representations (e.g. the Walsh–Hadamard transform) could provide insights in the problem behavior.

6.3. The third objective

This is the shortest section due to a fact that ES and GA are not able to find any balanced function with correlation immunity equal to two, which was our objective. In fact, only sporadically those two algorithms are able even to find a balanced Boolean function with correlation immunity that equals one. Random search (similarly to GA and ES) is not able to find any solutions that have correlation immunity equal to two. Naturally, the conclusions apply with a fitness function we chose to use. Therefore, in this section we give results only for CGP and GP.

6.3.1. Cartesian genetic programming

Fig. 21 gives the heatmap representation where the best average fitness value equals 119.26. This is obtained for the genotype size of 1700 and mutation probability of 11%. We note that very similar average value of 119.24 is achieved for genotype size of 1500 and mutation probability of 7%. Unlike the first two objectives, we can also observe there are a number of results where CGP is not able even to reach some high (e.g. more than 100) nonlinearity value. This points us to a fact that CGP often needs many evaluations just to find a solution that has correlation immunity equal to two and therefore it does not have enough time to converge to higher nonlinearity values.

Fig. 22 shows the average number of active nodes for fitness function 3. The best parameter combination has the average length of 81.5 active nodes.

When addressing the algorithm performance with respect to the genotype size or mutation probability, we display the corresponding graphs in Fig. 23a–d. We observe that the best genotype size is 1500 and by far the worst one is 100. Furthermore, we see that the mutation probability does not affect significantly average

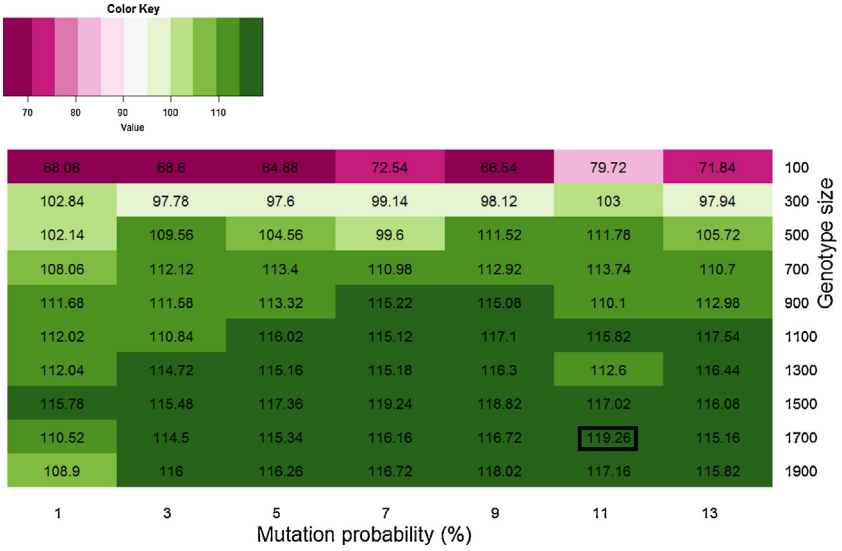


Fig. 21. Heatmap of the average values for CGP, fitness function 3.

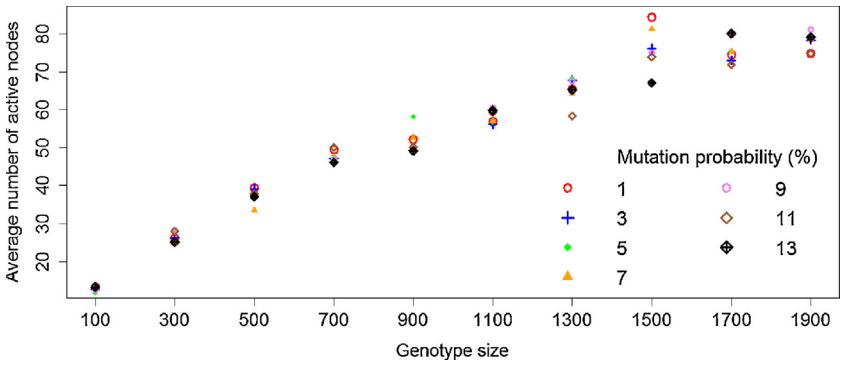


Fig. 22. Average number of active nodes for all mutation rates, fitness function 3.

fitness value while the probability of 5% gives the highest number of best obtained values. These results suggest that the algorithm performs well, but just does not have enough time to converge (another possibility is that it is stuck in some local optima).

6.3.2. Genetic programming

Finally, we present results for GP and fitness function 3. In Fig. 24, we see that the highest average fitness value is 121.3 which is achieved for the population of 10,000 and a tree depth 7.

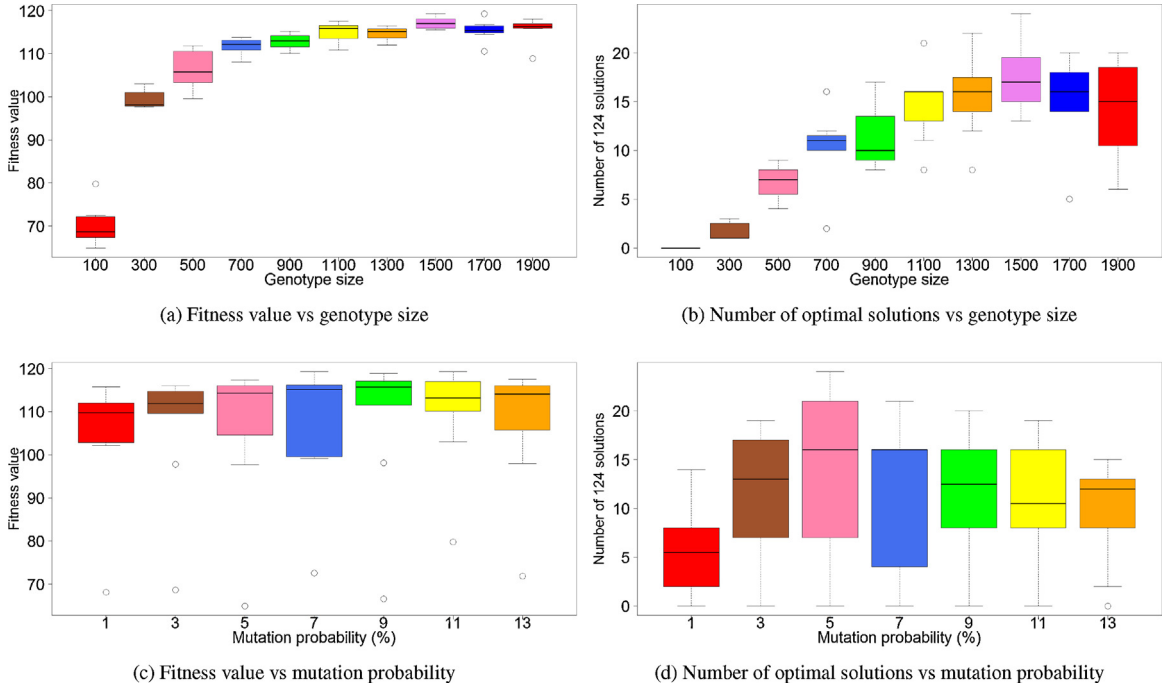
Fig. 25a–d give the average fitness value and the number of optimal solutions for each genotype size and tree depth. From the results we see that population sizes of 5000 and 10,000 reach very high average fitness values while populations 2000 and 5000 give higher number of optimal solutions. Medium sized tree depths (approximately seven to nine) perform the best.

6.3.3. Best algorithms for the third objective - long run

For the third objective and long run, we experiment with CGP with genotype size 1700 and mutation rate 11%, and GP with the population of 10,000 and a tree depth of 7. Since both GA and ES were unable to find any correct solution with 500,000 evaluations, we tried to estimate appropriate parameter values on the basis of the results from the first two objectives. In accordance with that, we investigate the performance of GA with population of 100 and individual mutation probability of 90%, and ES with parent size of 500 and offspring size of 10. However, even those extra evaluations did not help GA and ES to evolve any solutions that are balanced and with correlation immunity equal to 2. Therefore, in Fig. 26 we display the results only for GP and CGP. Similarly as in the second objective, the GP benefits more where it improves the average fitness value from 121.3 to 123.36. On the other hand, CGP did not profit from the additional evaluations and its average fitness value changed only marginally.

6.3.4. Discussion – the third objective

After comparing all the results, it appears that the fitness function three is the most difficult one to solve. This is especially apparent with CGP where we observe that with some genotype lengths CGP has difficulties to evolve optimal solutions in “only” 500,000 evaluations. However, we note that both GP and CGP are able to find balanced Boolean functions with nonlinearity equal to 112, algebraic immunity 4, algebraic degree 5, and correlation immunity 2 and therefore reach the global optimum. Furthermore, we observe that by increasing the number of evaluations, GP benefits more than CGP.



(a) Fitness value vs genotype size

(b) Number of optimal solutions vs genotype size

(c) Fitness value vs mutation probability

(d) Number of optimal solutions vs mutation probability

Fig. 23. Mutation probability and genotype size dependency, CGP, fitness function 3.

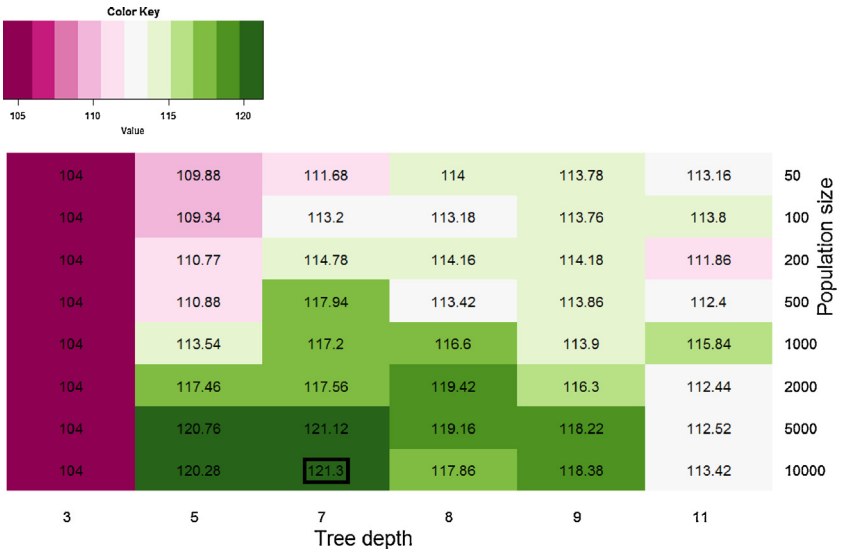
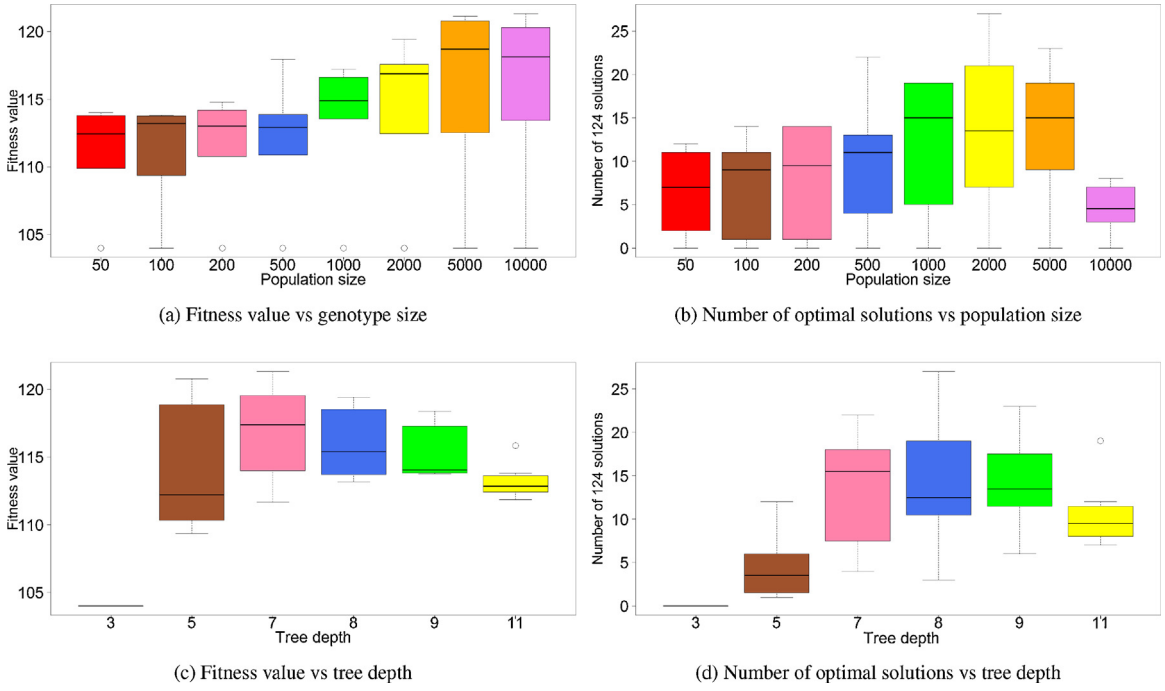


Fig. 24. Heatmap of the average values for GP, fitness function 3.



(a) Fitness value vs genotype size

(b) Number of optimal solutions vs population size

(c) Fitness value vs tree depth

(d) Number of optimal solutions vs tree depth

Fig. 25. Population size and tree depth dependency, GP, fitness function 3.

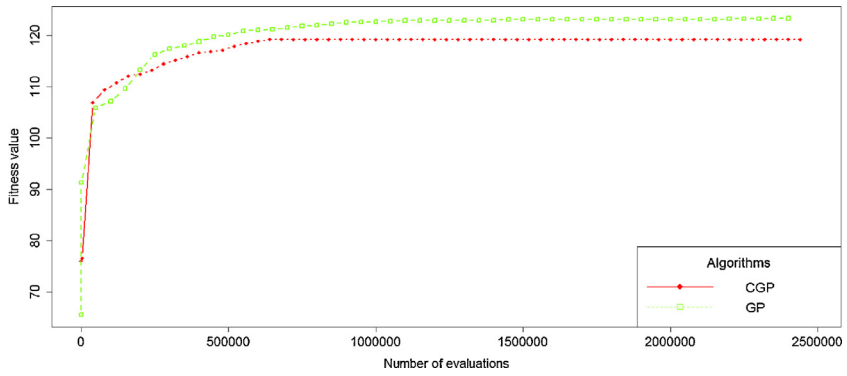


Fig. 26. Long run for the best performing algorithms, fitness function 3.

A fitness function as given in Eq. (18) could have been written differently. Since we know that we want to obtain the nonlinearity value of 112, we could have set that condition in the delta function we use. However, that option seems to bring some undesired consequences. All our experiments show that it is much easier to evolve Boolean function with high algebraic degree (recall Eq. (15) where the algebraic degree and the correlation immunity are conflicting properties). Indeed, if experiments do not specify the correlation immunity value, the results show that all algorithms tend to evolve solutions with algebraic degree equal to seven and correlation immunity equal to zero [9]. In some cases, GP and CGP find functions with algebraic degree value that equals six and correlation immunity equal to one.

Therefore, in the case when the delta function is the nonlinearity condition, we believe that an additional constraint should be put on the algebraic degree property in order not to allow it to step over a certain threshold value. By doing so, we force the algorithm to look for functions with the maximum correlation immunity values considering the Siegenthaler inequality. One more possible research avenue would be to work with the deviation from the desired correlation immunity, instead of the immunity value itself [15].

As a future work it would be interesting to evolve Boolean functions for each correlation immunity value that still ensures large enough algebraic degree value (for a Boolean function of eight inputs that would constitute the correlation immunity values from zero to four [12]). Naturally, there have been previous works on the evolution of Boolean functions with a certain correlation immunity values (i.e. resilient Boolean functions) as enumerated in Section 2. Still, we believe there is a certain lack of methodicalness in those works since the correlation immunity is often either not of primary interest or the researchers are interested in the correlation immunity with values up to two (for Boolean functions with eight inputs).

7. Conclusions

In this paper, we discuss how to use EAs to evolve cryptographically suitable Boolean functions. We concentrate on three fitness functions and we compare four EAs and the random search as a baseline case.

In all the cases, GP and CGP perform the best, which indicate that the TT representation is not the most appropriate one when evolving cryptographically suitable Boolean functions. Naturally, that observation is valid with fitness functions we used in the experiments. We also show that for these fitness functions and algorithm configurations, 500,000 evaluations presents a reasonable stopping criterion. We can observe certain improvements with additional evaluations, but only in the case of GP that improvement could justify significantly longer running time. We give a short discussion for each of the fitness functions where we also note some possible research avenues. Since here we consider only functions of eight inputs, one natural research avenue would be to work with larger sizes (this applies for all objectives we investigate in this paper). Another option is to experiment with other unique representations of Boolean functions (e.g. Walsh–Hadamard transform). As evident from the Section 2, this is one extremely well researched area when considering the total number of applications of EC to cryptology-related problems. However, we still believe there is much room for improvement, especially in the systemization of the current knowledge.

Acknowledgements

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. The authors wish to thank the reviewers for their helpful remarks and suggestions.

References

[1] J.F. Miller, An empirical study of the efficiency of learning Boolean functions using a Cartesian genetic programming approach, in: W. Banzhaf, J.M. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M.J. Jakiela, R.E. Smith (Eds.), GECCO, Morgan Kaufmann, 1999, pp. 1135–1142.
[2] L.D. Burnett, Heuristic Optimization of Boolean Functions and Substitution Boxes for Cryptography, Ph.D. thesis, Queensland University of Technology, 2005.
[3] T. Kalganova, J.F. Miller, Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness, in: Proc. NASA/DoD Workshop on Evolvable Hardware, IEEE Computer Society, 1999, pp. 54–63.
[4] C. Carlet, Boolean functions for cryptography and error correcting codes, in: Y. Crama, P.L. Hammer (Eds.), Boolean Models and Methods in Mathematics, Computer Science, and Engineering, 1st ed., Cambridge University Press, New York, NY, USA, 2010, pp. 257–397.
[5] C. Carlet, S. Guilley, Correlation-immune Boolean functions for easing counter measures to side-channel attacks, in: H.E. Niederreiter, O.E. Alina, e.a. Daniel, Panario (Eds.), Algebraic Curves and Finite Fields. Cryptography and Other Applications, De Gruyter, Berlin, Boston, 2014, pp. 41–70.
[6] C. Carlet, J.-L. Danger, S. Guilley, H. Maghrebi, Leakage Squeezing of Order Two, in: S. Galbraith, M. Nandi (Eds.), Progress in Cryptology – INDOCRYPT, Vol. 7668 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2012, pp. 120–139.
[7] K. Goossens, Automated Creation and Selection of Cryptographic Primitives, Master's thesis, Katholieke Universiteit Leuven, 2005.
[8] W. Millan, J. Fuller, E. Dawson, New concepts in evolutionary search for Boolean functions in cryptology, Comput. Intell. 20 (3) (2004) 463–474.
[9] S. Picek, D. Jakobovic, M. Golub, Evolving cryptographically sound Boolean functions, in: Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation, GECCO '13 Companion, ACM, New York, NY, USA, 2013, pp. 191–192.
[10] S. Picek, L. Batina, D. Jakobovic, Evolving DPA-resistant Boolean functions, in: T. Bartz-Beielstein, J. Branke, B. Filipič, J. Smith (Eds.), Parallel Problem Solving from Nature – PPSN XIII, Vol. 7668 of Lecture Notes in Computer Science, Springer International Publishing, 2014, pp. 812–821.
[11] C. Cid, S. Kiyomoto, J. Kurihara, The RAKAPOSHI Stream Cipher, in: S. Qing, C. Mitchell, G. Wang (Eds.), Information and Communications Security, Vol. 5927 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2009, pp. 32–46.
[12] B.M. Gammel, R. Gottfert, O. Kniffler, The Achterbahn Stream Cipher, 2005.
[13] W. Millan, A. Clark, E. Dawson, An Effective Genetic Algorithm for Finding Highly Nonlinear Boolean Functions, in: Proceedings of the First International Conference on Information and Communication Security, ICICS '97, Springer-Verlag, London, UK, 1997, pp. 149–158.
[14] A.J. Clark, Optimisation heuristics for cryptology, Ph.D. thesis, Queensland University of Technology, 1998.
[15] W. Millan, A. Clark, E. Dawson, Heuristic design of cryptographically strong balanced Boolean functions, in: Advances in Cryptology – EUROCRYPT, '98, 1998, pp. 489–499.
[16] W. Millan, A. Clark, E. Dawson, Boolean Function Design Using Hill Climbing Methods, in: J. Pieprzyk, R. Safavi-Naini, J. Seberry (Eds.), Information Security and Privacy, Vol. 1587 of Lecture Notes in Computer Science, Berlin, Heidelberg, Springer, 1999, pp. 1–11.
[17] J. Clark, J. Jacob, Two-stage optimisation in the design of Boolean functions, in: E. Dawson, A. Clark, C. Boyd (Eds.), Information Security and Privacy, Vol. 1841 of Lecture Notes in Computer Science, Berlin, Heidelberg, Springer, 2000, pp. 242–254.
[18] J.A. Clark, J.L. Jacob, S. Stepney, S. Maitra, W. Millan, Evolving Boolean Functions Satisfying Multiple Criteria, in: Progress in Cryptology – INDOCRYPT, 2002, 2002, pp. 246–259.
[19] S. Kavut, M. Yücel, Improved Cost Function in the Design of Boolean Functions Satisfying Multiple Criteria, in: T. Johansson, S. Maitra (Eds.), Progress in Cryptology – INDOCRYPT, 2003, Vol. 2904 of Lecture Notes in Computer Science, Berlin, Heidelberg, Springer, 2003, pp. 121–134.
[20] J.A. Clark, J. Jacob, S. Maitra, P. Stănică, Almost Boolean functions: the design of Boolean functions by spectral inversion, in: Evolutionary Computation, 2003. CEC '03, The 2003 Congress on, Vol. 3, 2003, pp. 2173–2180.
[21] L. Burnett, W. Millan, E. Dawson, A. Clark, Simpler methods for generating better Boolean functions with good cryptographic properties, Australas. J. Comb. 29 (2004) 231–247.
[22] H. Aguirre, H. Okazaki, Y. Fuwa, An evolutionary multiobjective approach to design highly non-linear Boolean functions, in: Proceedings of the Genetic and Evolutionary Computation Conference GECCO'07, 2007, pp. 749–756.
[23] Y. Izbenko, V. Kovtun, A. Kuznetsov, The design of Boolean functions by modified hill climbing method, Cryptology ePrint Archive, Report 2008/111, 2008 http://eprint.iacr.org/.
[24] J. McLaughlin, J.A. Clark, Evolving balanced Boolean functions with optimal resistance to algebraic and fast algebraic attacks, maximal algebraic degree,

and very high nonlinearity, Cryptology ePrint Archive, Report 2013/011, 2013 http://eprint.iacr.org/.
[25] R. Hrbacek, V. Dvorak, Bent Function Synthesis by Means of Cartesian Genetic Programming, in: T. Bartz-Beielstein, J. Branke, B. Filipič, J. Smith (Eds.), Parallel Problem Solving from Nature – PPSN XIII, Vol. 8672 of Lecture Notes in Computer Science, Springer International Publishing, 2014, pp. 414–423.
[26] S. Picek, E. Marchiori, L. Batina, D. Jakobovic, Combining evolutionary computation and algebraic constructions to find cryptography-relevant Boolean functions, in: T. Bartz-Beielstein, J. Branke, B. Filipič, J. Smith (Eds.), Parallel Problem Solving from Nature – PPSN XIII, Lecture Notes in Computer Science, Springer International Publishing, 2014, pp. 822–831.
[27] S. Picek, D. Jakobovic, J.F. Miller, E. Marchiori, L. Batina, Evolutionary methods for the construction of cryptographic boolean functions, in: Genetic Programming – 18th European Conference, EuroGP 2015, Copenhagen, Denmark, April 8–10, 2015, Proceedings, 2015, pp. 192–204.
[28] S. Picek, C. Carlet, D. Jakobovic, J.F. Miller, L. Batina, Correlation Immunity of Boolean Functions: An Evolutionary Algorithms Perspective, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11–15, 2015, 2015, pp. 1095–1102.
[29] A. Braeken, Cryptographic Properties of Boolean Functions and S-Boxes, Ph.D. thesis, Katholieke Universiteit Leuven, 2006.
[30] R. Forrié, The Strict Avalanche Criterion: Spectral Properties of Boolean Functions and an Extended Definition, in: S. Goldwasser (Ed.), Advances in Cryptology – CRYPTO' 88, Vol. 403 of Lecture Notes in Computer Science, New York, Springer, 1990, pp. 450–468.
[31] W. Meier, E. Pasalic, C. Carlet, Algebraic Attacks and Decomposition of Boolean Functions, in: C. Cachin, J. Camenisch (Eds.), Advances in Cryptology – EUROCRYPT, 2004, Vol. 3027 of Lecture Notes in Computer Science, Berlin, Heidelberg, Springer, 2004, pp. 474–491.
[32] B. Preneel, W. Van Leekwijck, L. Van Linden, R. Govaerts, J. Vandewalle, Propagation characteristics of Boolean functions, in: Proceedings of the workshop on the theory and application of cryptographic techniques on Advances in cryptology, EUROCRYPT, '90, New York, NY, USA, Springer-Verlag New York, Inc., 1991, pp. 161–173.
[33] O. Rothaus, On "bent" functions, J. Comb. Theory, Ser. A 20 (3) (1976) 300–305.
[34] J. Dillon, A Survey of Bent Functions*, Tech. rep., Reprinted from the NSA Technical Journal. Special Issue, unclassified, 1972.
[35] T. Siegenthaler, Correlation-immunity of nonlinear combining functions for cryptographic applications (Corresp.), IEEE Trans. Inf. Theor. 30 (5) (2006) 776–780.
[36] X. Guo-Zhen, J. Massey, A spectral characterization of correlation-immune combining functions, IEEE Trans. Inf. Theory 34 (3) (1988) 569–571.
[37] T.W. Cusick, P. Stănică, Cryptographic Boolean Functions and Applications, Elsevier Inc., San Diego, USA, 2009.
[38] C. Carlet, Vectorial Boolean functions for cryptography, in: Y. Crama, P.L. Hammer (Eds.), Boolean Models and Methods in Mathematics, Computer Science, and Engineering, 1st ed., Cambridge University Press, New York, NY, USA, 2010, pp. 398–469.
[39] Q. Wang, T. Johansson, A note on fast algebraic attacks and higher order nonlinearities, in: X. Lai, M. Yung, D. Lin (Eds.), Information Security and Cryptology, Vol. 6584 of Lecture Notes in Computer Science, Berlin, Heidelberg, Springer, 2011, pp. 404–414.
[40] P. Sarkar, S. Maitra, Nonlinearity Bounds and Constructions of Resilient Boolean Functions, in: M. Bellare (Ed.), Advances in Cryptology – CRYPTO 2000, Vol. 1880 of Lecture Notes in Computer Science, Berlin, Heidelberg, Springer, 2000, pp. 515–532.
[41] W. Meier, O. Staffelbach, Fast correlation attacks on certain stream ciphers, J. Cryptol. 1 (3) (1989) 159–176.
[42] C. Carlet, K. Feng, An infinite class of balanced functions with optimal algebraic immunity, good immunity to fast algebraic attacks and good nonlinearity, in: J. Pieprzyk (Ed.), Advances in Cryptology – ASIACRYPT, 2008, Vol. 535, of Lecture Notes in Computer Science, Berlin, Heidelberg, Springer, 2008, pp. 425–440.
[43] J. Massey, Shift-register synthesis and BCH decoding, IEEE Trans. Inf. Theory 15 (1) (1969) 122–127.
[44] S. Rønjom, T. Helleseth, A new attack on the filter generator, IEEE Trans. Inf. Theory 53 (5) (2007) 1752–1758.
[45] N. Courtois, W. Meier, Algebraic attacks on stream ciphers with linear feedback, in: E. Biham (Ed.), Advances in Cryptology – EUROCRYPT, 2003, Vol. 2656 of Lecture Notes in Computer Science, Berlin, Heidelberg, Springer, 2003, pp. 345–359.
[46] N. Courtois, Fast algebraic attacks on stream ciphers with linear feedback, in: D. Boneh (Ed.), Advances in Cryptology – CRYPTO 2003, Vol. 2729 of Lecture Notes in Computer Science, Berlin, Heidelberg, Springer, 2003, pp. 176–194.
[47] J. Clark, J. Jacob, Two-Stage Optimisation in the Design of Boolean Functions, in: E. Dawson, A. Clark, C. Boyd (Eds.), Information Security and Privacy, Vol. 1841 of Lecture Notes in Computer Science, Berlin, Heidelberg, Springer, 2000, pp. 242–254.
[48] D. Jakobovic, et al., Evolutionary Computation Framework, 2015 http://gp.zemris.fer.hr/ecf/.
[49] A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computing, Springer-Verlag, Berlin, Heidelberg/New York, USA, 2003.
[50] J.R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, MA, USA, 1992.
[51] R. Poli, W.B. Langdon, N.F. McPhee, A field guide to genetic programming, 2008, Published via http://lulu.com and freely available at http://lulu.com and freely available at http://www.gp-field-guide.org.uk, (With contributions by J. R. Koza).
[52] J.F. Miller, P. Thomson, Cartesian Genetic Programming, in: R. Poli, W. Banzhaf, W.B. Langdon, J.F. Miller, P. Nordin, T.C. Fogarty (Eds.), EuroGP, Vol. 1802 of Lecture Notes in Computer Science, Springer, 2000, pp. 121–132.
[53] J.F. Miller (Ed.), Cartesian Genetic Programming, Natural Computing Series, Springer, Berlin, Heidelberg, 2011.
[54] J. Miller, S. Smith, Redundancy and computational efficiency in Cartesian genetic programming, IEEE Trans. Evol. Comput. 10 (2) (2006) 167–174.
[55] H. Dobbertin, Construction of bent functions and balanced boolean functions with high nonlinearity, in: B. Preneel (Ed.), Fast Software Encryption, Vol. 1008 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 1995, pp. 61–74.
[56] A. Canteaut, M. Daum, H. Dobbertin, G. Leander, Finding nonnormal bent functions, Discrete Appl. Math. 154 (2) (2006) 202–218, coding and Cryptography.
[57] G. Leander, G. McGuire, Construction of bent functions from near-bent functions, J. Comb. Theory, Ser. A 116 (4) (2009) 960–970.
[58] R. Santana, R.I. McKay, J.A. Lozano, Symmetry in evolutionary and estimation of distribution algorithms, in: Proceedings of the IEEE Congress on Evolutionary Computation, CEC, Cancun, Mexico, June 20–23, 2013, 2013, pp. 2053–2060.
[59] S. Picek, R.I. McKay, R. Santana, T.D. Gedeon, Fighting the symmetries: the structure of cryptographic Boolean function spaces, in: Proceedings of the 2015 on Genetic and Evolutionary Computation Conference, GECCO '15, ACM, New York, NY, USA, 2015, pp. 457–464.
[60] S. Picek, D. Jakobovic, From fitness landscape to crossover operator choice, in: Proceedings of the 2014 Conference on Genetic and Evolutionary Computation, GECCO '14, ACM, New York, NY, USA, 2014, pp. 815–822.