Evolving Bent Quaternary Functions

Stjepan Picek*, Karlo Knezevic†, Luca Mariot‡ Domagoj Jakobovic†, Alberto Leporati‡
*Cyber Security Research Group, Delft University of Technology, Delft, The Netherlands
†University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, Croatia
‡Department of Informatics, Systems and Communication,
Università degli Studi di Milano-Bicocca, Viale Sarca 336/14, 20126 Milano, Italy

***Abstract*—Boolean functions have a prominent role in many real-world applications, which makes them a very active research domain. Throughout the years, various heuristic techniques proved to be an attractive choice for the construction of Boolean functions with different properties. One of the most important properties is nonlinearity, and in particular maximally nonlinear Boolean functions are also called bent functions. In this paper, instead of considering Boolean functions, we experiment with quaternary functions. The corresponding problem is much more difficult and presents an interesting benchmark as well as real-world applications. The results we obtain show that evolutionary metaheuristics, especially genetic programming, succeed in finding quaternary functions with the desired properties. The obtained results in the quaternary domain can also be translated into the binary domain, in which case this approach compares favorably with the state-of-the-art in Boolean optimization. Our techniques are able to find quaternary bent functions for up to 8 inputs, which corresponds to obtaining Boolean bent functions of 16 inputs.**

I. Introduction

Boolean functions have a number of real-world applications in various domains. Examples range from usages in combinatorics, such as the construction of *Hadamard matrices* [1] and *strongly regular graphs* [2], or in coding theory, where they are used for constructing certain classes of codes such as *Reed-Muller codes* [3] and *Kerdock codes* [4]. Boolean functions are also important in the construction of codebooks from codes [5] and in cryptography for designing *hash functions* [6] and *symmetric ciphers* [7], [8], as well as in *fully homomorphic encryption* [9].

Boolean functions are basically mappings from $\mathbb{F}_2^n$ to $\mathbb{F}_2$, with $\mathbb{F}_2$ being the finite field of two elements. Although they are among the breeds of *finite functions* that are most often investigated in the literature, one can also consider the more general case of *q-ary functions* defined over the *finite ring* $\mathbb{Z}_q$, for $q > 2$. By doing so, one changes the representation of the problem and the corresponding search space size, as well as possible target applications. As an example, *generalized Boolean functions* (i.e. mappings from $\mathbb{F}_2^n$ to $\mathbb{Z}_q$ with $q > 2$) were first introduced with the aim of finding codes for *Multicode Code-Division Multiple Access* (MC-CDMA) systems [10].

In this paper we concentrate on q-ary functions and their cryptographic applications, remarking however that cryptography is only one among many other domains where they can be applied. As we demonstrate, this problem is very interesting both from the perspective of an evolutionary computation benchmark and as a tool for constructing cryptographic primitives. We study the cryptographic properties of q-ary functions, i.e., mappings from $\mathbb{Z}_q^n$ to $\mathbb{Z}_q$, focusing in particular on the case of *quaternary functions* where $q = 4$. In the rest of this paper, when referring to Boolean functions we consider the standard case where the output belongs to $\mathbb{F}_2$. On the other hand, we call quaternary functions those mapping to $\mathbb{Z}_4$.

The quest for quaternary functions is motivated by the search of new algebraic constructions of cryptographically significant Boolean functions: the idea is to define quaternary functions with good properties, and then derive the associated Boolean functions through projection mappings. Although this technique may sound mathematically very involved, we emphasize that the procedure is straightforward. In particular, the main constraints to be met are the following ones:

1) For both Boolean and quaternary functions there need to exist a way to express the desired properties, i.e., how to write a suitable fitness function. As we see in Sections II-A and II-B, this condition is satisfied for the cryptographic property of *nonlinearity*, since it can be calculated for both types of functions (even though the corresponding equations differ).
2) After evolving Boolean or quaternary functions, there must be a one-to-one mapping between these two classes. The *Gray map* [11], which is one of these possible mappings, is presented in Section V-A.

The problem of finding Boolean functions of n variables with the best possible combinations of cryptographic properties is extremely difficult. This stems from the impossibility of exhaustively exploring the corresponding search space, which grows superexponentially as 2^{2^n}, making complete enumeration unfeasible for $n > 5$. Indeed, for $n = 6$ variables there are already $2^{64} \approx 1.84 \cdot 10^{19}$ possible Boolean functions, while for $n = 8$ there are $2^{256} \approx 1.16 \cdot 10^{77}$ functions, a quantity which almost matches the estimated number of atoms in the observable universe. For this reason, two wide approaches have been developed in the literature to address the problem. The first one uses *algebraic constructions* to generate classes of Boolean functions with good cryptographic properties [12]. The second one employs *heuristic techniques* to explore the space of Boolean functions, by driving the search through specific fitness functions that account for several cryptographic properties. With quaternary functions the search space size equals 4^{4^n}, which is significantly more difficult than for the

Boolean case. Still, from the representation point of view, quaternary functions of n variables can be easily transformed into Boolean functions of $2n$ variables, which means we need to work with only half of the variables.

The aim of this paper is to investigate the application of *evolutionary algorithms* to the search of quaternary functions with maximal nonlinearity, something that to the best of our knowledge has never been done before. In particular, we focus our attention on the use of *Genetic Algorithms* (GA) and *Genetic Programming* (GP) to evolve respectively 1) the truth tables of n-variable quaternary functions, represented as quaternary strings of length 4^n, and 2) quaternary trees that are evaluated through their associated truth tables. Here, by *quaternary tree* we mean a tree whose leaf values are in $\mathbb{Z}_4$, but where every node can only have two children. With both of these encodings, our aim is to evolve *bent* quaternary functions, since they have maximal nonlinearity. Our main contributions can be summarized as follows:

1) A new encoding based on quaternary trees for the genetic programming heuristic. Although used here in a specific example, we are confident that such representation can find its place in other problems as long as the two constraints discussed above about fitness functions and one-to-one mapping are satisfied.
2) The evolution of quaternary functions of different dimensions with specific cryptographic properties through GA and GP. We discuss the difficulty of such a process and compare the results between the two encodings we use. After transforming quaternary functions into Boolean functions, we also compare our results with state-of-the-art works from the relevant literature.
3) The discovery of maximally nonlinear quaternary functions that are *not* bent. As far as we know, the only kind of quaternary functions with maximal nonlinearity that have been described in the literature are bent functions, by analogy with the Boolean case.
4) The observation that evolving non-bent maximally nonlinear quaternary functions is easier than evolving bent quaternary functions. At the same time, all bent quaternary functions can be mapped into bent Boolean functions. Moreover, while all bent quaternary functions can be mapped into bent Boolean functions, we show that this fact does not hold for non-bent maximally nonlinear quaternary functions.

The remainder of this paper is organized as follows. Section II covers the necessary definitions and notions about Boolean and quaternary functions, along with the cryptographic criteria that we take into account. An overview of the literature concerning heuristics for finding Boolean functions with good cryptographic properties is given in Section III. Section IV presents details of the algorithms we use for our experiments, focusing on the representation for the candidate solutions and the genetic operators adopted. Section IV describes the problem instances and the parameters that we consider, while Section V discusses the results obtained by our experiments, as well as possible transformations between Boolean and quaternary functions and future work. Finally, Section VI sums up the key contributions of the paper.

Table I: Walsh-Hadamard transform of a bent Boolean function with 4 inputs.

$x \in \mathbb{F}_2^4$	$f(x)$	$W_f(x)$	$x \in \mathbb{F}_2^4$	$f(x)$	$W_f(x)$
0000	0	4	1000	0	-4
0001	1	4	1001	1	4
0010	0	-4	1010	0	-4
0011	0	4	1011	0	-4
0100	1	4	1100	0	4
0101	1	4	1101	0	-4
0110	0	-4	1110	1	4
0111	1	4	1111	0	4

II. Background

A. Boolean Functions

Let $n \in \mathbb{N}$. A *Boolean function* is a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2$ where $\mathbb{F}_2$ is the *Galois field* (or *finite field*) with two elements. We denote the set of all n-tuples over the field $\mathbb{F}_2$ as $\mathbb{F}_2^n$. The set $\mathbb{F}_2^n$ represents all binary vectors of length n, and it can be viewed as a $\mathbb{F}_2$-vector space [12] . The inner product of vectors a and b over the $\mathbb{F}_2$ field is defined as $a \cdot b = \oplus_{i=1}^n a_i b_i$, with "$\oplus$" denoting addition modulo two. The *Hamming weight* (*HW*) of a vector $a \in \mathbb{F}_2^n$ is the number of non-zero positions in the vector.

A Boolean function f on $\mathbb{F}_2^n$ can be uniquely represented by a *truth table*, which is the vector $(f(0,\cdots,0),...,f(1,\cdots,1))$ that contains the output values of f for the inputs listed in lexicographic order [12].

Another unique representation of a Boolean function is the *Walsh-Hadamard transform* W_f, that measures the correlation between $f(\vec{x})$ and all linear functions of the form $a \cdot x$, for $\vec{a}$ ranging in $\mathbb{F}_2^n$ [13]. An example of the Walsh-Hadamard transform of a Boolean function with 4 inputs is given in Table I. The Walsh-Hadamard transform of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ equals [12]:

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

A Boolean function f is balanced if the Walsh-Hadamard coefficient of the null vector *0* equals zero [14]:

$$W_f(\mathit{0}) = 0. \tag{2}$$

Alternatively, a Boolean function with n inputs is balanced if the Hamming weight of its truth table equals 2^{n-1}, i.e., if it is composed of an equal number of zeros and ones.

A Boolean function f used in the design of stream and block ciphers should lie at a large *Hamming distance* (*HD*) from all affine functions, in order to resist linear cryptanalytic attacks. This distance corresponds to the *nonlinearity* of f, which is defined as the minimum *HD* between f and all affine functions [12]. The nonlinearity N_f of a Boolean function f

expressed in terms of the Walsh-Hadamard coefficients of f is [12]:

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

A natural question is to determine what is the maximum nonlinearity a Boolean function can attain. This can be derived from *Parseval's relation*:

$$\sum_{a \in \mathbb{F}_2^n} W_f(a)^2 = 2^{2n}, \tag{4}$$

which implies that the mean of $W_f(a)^2$ equals 2^n, and $\max_{a \in \mathbb{F}_2^n} |W_f(a)|$ is thus at least equal to the square root of this mean, that is, $2^{n/2}$. Boolean functions whose Walsh coefficients all have absolute value $2^{n/2}$ are called *bent*, and they have the highest possible nonlinearity. Clearly, bent functions only exist for even number of variables, and are never balanced. The expression for nonlinearity of bent functions is as follows: [1], [15]: $N_f = 2^{n-1} - 2^{\frac{n}{2}-1}$. The Boolean function reported in Table I is an example of a bent function, since all its Walsh-Hadamard coefficients are $2^{4/2} = 4$ in absolute value. Hence, its nonlinearity is equal to $2^{4-1} - 2^{4/2-1} = 6$.

B. Quaternary Functions

To generalize the notion of Boolean functions one can take into account the *residual class ring* (*Galois ring*) $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$. Specifically, we consider the case where $q = 4$. The set of all n-tuples of elements in $\mathbb{Z}_4$ is denoted as $\mathbb{Z}_4^n$. A *quaternary function* F of n variables is a mapping from $\mathbb{Z}_4^n$ to $\mathbb{Z}_4$, that is, a $\{0,1,2,3\}$-valued function. Let i denote the complex number such that $i^2 = -1$. There is a group-isomorphism between the set $\{0,1,2,3\}$ and $(\pm 1, \pm i)$ under the standard isomorphism $x \to i^x$ [16], [17]. We denote addition and multiplication modulo 4 with "+" and "·". The Hamming weight (HW) of a function F is the number of values $u \in \mathbb{Z}_4^n$ such that $F(u) \neq 0$.

As noted in [18], a quaternary function $F : \mathbb{Z}_4^n \to \mathbb{Z}_4$ can be represented by its truth table

$$(F(0,\cdots,0), F(0,\cdots,1), \cdots, F(3,\cdots,3)),$$

which is the quaternary vector of length 4^n defining the output values of F in lexicographic order.

The Walsh-Hadamard transform of a quaternary function F is defined as follows:

$$W_F(a) = \sum_{v \in \mathbb{Z}_4^n} i^{a \cdot v + F(v)}. \tag{5}$$

Similarly to the Boolean case, a quaternary function is balanced if and only if $W_F(\vec{0})$ equals 0. Alternatively, it is balanced if and only if for all $j \in \mathbb{Z}_4$ the cardinality of the set $\eta_j(F)$ equals 4^{n-1}, where $\eta_j(F) = \{u \in \mathbb{Z}_4^n | F(u) = j\}$.

To define nonlinearity, we can use either the *Hamming metric* (N_F^H) or the *Lee metric* (N_F^L) as the underlying distance. In this paper, we work with the Lee metric, since there is an isometry (distance preserving bijection) between $\mathbb{Z}_4^n$ equipped with the Lee distance and $\mathbb{F}_2^{2n}$ equipped with the Hamming distance when Gray mapping is used (for details see Section V-A). The nonlinearity of a function F under the Lee distance equals:

$$N_F^L = 4^n - \max_{a \in \mathbb{Z}_4^n, b \in \mathbb{Z}_4} \left\{ Re(i^b W_F(a)) \right\} \tag{6}$$

$$= 4^n - \max_{a \in \mathbb{Z}_4^n} \{ |Re(W_F(a))|, |Im(W_F(a))| \}, \tag{7}$$

where $Re(z)$ and $Im(z)$ denote the real and imaginary part of the complex number z.

A quaternary function is bent if $|W_F(a)| = 2^n \; \forall a \in \mathbb{Z}_4^n$. The nonlinearity of a bent quaternary function with n inputs equals:

$$N_F^L = 2^{2n} - 2^n. \tag{8}$$

Note that in $\mathbb{Z}_4^n$ a function can be bent for any dimension n, while in $\mathbb{F}_2^n$ a function can be bent if and only if n is even. An example of the Walsh-Hadamard transform of a quaternary bent function is given in Table II.

Table II: Walsh-Hadamard transform of a bent quaternary function with two inputs.

$\vec{x} \in \mathbb{Z}_4^2$	$F(z)$	$W_F(z)$	$\vec{x} \in \mathbb{Z}_4^2$	$F(z)$	$W_F(z)$
00	0	-4i	20	0	4i
01	3	4	21	1	4
02	2	-4i	22	2	4i
03	1	4	23	3	4
10	3	4	30	3	-4
11	3	4	31	1	4
12	3	-4	32	3	4
13	3	-4	33	1	-4

III. Related Work

As already mentioned, there are many successful applications of heuristic techniques for constructing Boolean functions usable in cryptography. To the best of our knowledge, there are no examples of heuristic constructions of quaternary functions. Consequently, here we discuss works that consider the evolution of Boolean functions with high (or maximal) nonlinearity.

Millan et al. evolve Boolean functions with high nonlinearity by utilizing a genetic algorithm [19]. Millan, Clark, and Dawson further increase the strength of genetic algorithms by combining them with a hill climbing and a resetting step, with the goal of finding highly nonlinear Boolean functions of up to 12 variables [20]. McLaughlin and Clark experiment with simulated annealing to generate Boolean functions that have optimal values for a number of properties, namely algebraic immunity, fast algebraic resistance, and algebraic degree [21]. In their work, they consider Boolean functions with sizes of up to 16 inputs.

Clark et al. experiment with simulated annealing in order to design Boolean functions using spectral inversion [22]. They observe that several cryptographic properties of interest are defined in terms of the Walsh-Hadamard transform values. On the basis of Parseval's theorem, one can infer what values the Walsh-Hadamard spectrum should have. Note that it is not possible to know how these values should be permuted, since in general the inverse Walsh-Hadamard transform maps to a pseudo-Boolean function. Consequently, when generating a Walsh-Hadamard spectrum containing these values, a necessary step is to verify that it corresponds indeed to a Boolean function. Mariot and Leporati [23] also adopt the spectral inversion method, designing a genetic algorithm where the genotype consists of the Walsh-Hadamard values to permute in order to evolve semi-bent Boolean functions.

Picek, Jakobovic, and Golub experiment with genetic algorithms and genetic programming to find Boolean functions that have several optimal properties [24]. Here, the genetic programming tree (as a genotype) is a posteriori transformed to the truth table representation for evaluation purposes. Mariot and Leporati employ Particle Swarm Optimization (PSO) to find Boolean functions with good trade-offs of cryptographic properties for dimensions up to 12 inputs [25]. Hrbacek and Dvorak use Cartesian genetic programming to evolve bent Boolean functions of sizes up to 16 inputs. In particular, the authors test several configurations of algorithms in order to speed up the evolution process [26]. Additionally, the authors do not limit the number of generations and therefore they succeed in finding bent functions in each run for sizes between 6 and 16 variables.

Picek et al. investigate a number of different evolutionary algorithms and fitness functions for Boolean functions of 8 inputs [27]. They show that genetic programming and Cartesian genetic programming outperform genetic algorithms and evolution strategies in a number of relevant test scenarios. Picek, Sisejkovic, and Jakobovic use immunological algorithms to evolve either bent or highly nonlinear Boolean functions with up to 16 inputs [28]. Finally, Picek and Jakobovic use genetic programming to evolve algebraic constructions that are then used to construct bent Boolean functions [29].

IV. Experimental Settings and Results

A. Representations and Algorithms

The most important decision that needs to be made in a heuristic optimization algorithm is usually the representation of candidate solutions. In this work, we experiment with two different encodings, respectively employed for *Genetic Algorithms* (GA) and *Genetic Programming* (GP), and compare their efficiency.

1) Integer Encoding for GA: Considering both Boolean and quaternary functions, a representation based on the truth table is probably the most natural choice. In this case, a quaternary function of n variables is represented with a string of length 4^n which corresponds to its lexicographically ordered truth table. Each element of the string can assume values in $\{0,1,2,3\}$, and it is initialized by sampling from the uniform distribution.

Appropriate crossover and mutation operators need to be defined in order to be able to use genetic algorithms with this encoding. In our experiments, we use a single mutation operator, which randomly chooses a single gene (element of the string) and generates its new value with a uniform probability over $\mathbb{Z}_4$. For crossover, we adopt three different operators: the first one is a simple single-point crossover, which combines the first part of one parent and the second part of the other parent into a single child. We also use a two-point variant which takes two parts from one and one part from the other parent. For both single-point and two-point crossover operators, crossover points are randomly chosen. Finally, we use the averaging crossover, which constructs each gene in the child by using the average value of the corresponding genes from both parents. The choice of the crossover operator is made randomly each time a crossover is performed.

Table III: Genetic programming functions.

Function	Definition
AND	$(a \cdot b) \mod 4$
OR	$(a+b) \mod 4$

2) Quaternary Tree Encoding for GP: As opposed to the truth table encoding, the other option we consider is to use a symbolic representation of a quaternary function. This is performed in a way such that GP can be used to evolve a quaternary function in the form of a syntactic tree. Here, the terminal set consists of the n input quaternary variables, denoted as $\{v_1,\ldots,v_n\}$. The function set (i.e., the set of inner nodes of a tree) should consist of appropriate operators that allow the definition of quaternary functions with n inputs. In our experiments, we use the AND and OR operations modulo 4, which are defined as shown in Table III. We do not implement the NOT function since it cannot be uniquely expressed in $\mathbb{Z}_4$, due to the fact that the value 2 does not have a multiplicative inverse (recall that $\mathbb{Z}_4$ is not a field but a ring). This means that we are not able to consider all possible functions but only those that can be expressed only through AND and OR operations.

When computing the truth table for the function, the same tree is parsed for every possible input combination of variables. Each result (evaluated at the root node) is written in the corresponding position of the truth table, and the tree is then given its fitness based on the properties of the resulting truth table. 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 [30].

B. Common Parameters

Regardless of the encoding, we use the same selection operator to conduct the search – a *steady-state selection* process, shown in Algorithm 1, where in each iteration only one individual from the population is replaced with a new one. The selection of the individual to be replaced is performed in a tournament of size 3: the algorithm selects 3 individuals at random and eliminates the worst among them. The remaining tournament survivors are then used as parents to create a new individual via crossover. Right after creation, the new individual immediately undergoes mutation, which depends on the mutation rate parameter. In our experiments this parameter equals 0.3, meaning that three out of ten new individuals are mutated on average. This kind of algorithm is convenient since it eliminates the need for specifying the crossover rate, and in our previous experience provides a steady rate of convergence.

In all the experiments the number of independent trials N for each configuration is 30 and the stopping criterion for all algorithms equals 500 000 evaluations or reaching the maximal nonlinearity. The population size for both the GA and GP experiments is 200. When using genetic programming, we experiment with maximal tree depths of 4, 6, and 8. We selected all the above experimental parameters after performing a preliminary tuning phase.

Algorithm 1 Steady-state k-tournament selection

```
randomly select $k$ individuals;
remove the worst of those $k$ individuals;
$child$ = crossover between the best two of the tournament;
perform mutation on $child$, with given probability;
insert $child$ into population;
```

C. Fitness Functions

We start with a fitness function that maximizes the nonlinearity property of an individual:

$$fitness_1 = 4^n - \max_{a \in \mathbb{Z}_4^n} \{|Re(W_F(a))|, |Im(W_F(a))|\}. \quad (9)$$

This fitness function seems a natural choice since it directly looks for the maximal nonlinearity value. A literature survey reveals that such a fitness function has already been used with considerable success when evolving bent Boolean functions. Interestingly, when using this function, we noticed a situation we could not account for by looking at the existing literature. Our algorithms had no problems in finding quaternary functions having maximal nonlinearity, but the Walsh-Hadamard spectrum of such functions was not what one would expect from a bent function. More precisely, the Walsh-Hadamard spectrum of such functions had values of zero in some coefficients, which is not possible for Boolean bent functions.

Consequently, we used an additional fitness function (again, with the goal of maximization), designed to cope with the difference between non-bent maximal nonlinear and bent quaternary functions:

$$fitness_2 = fitness_1 - \frac{2 * (zero\_coeffs)}{4^n}. \quad (10)$$

Here, *zero_coeffs* denotes the number of coefficients in the Walsh-Hadamard spectrum whose values equal 0. We multiply this number by a factor of 2 to account for the two binary components for each element in $\mathbb{Z}_4$.

D. Results

In this section, we present the results obtained in our experiments. First, in Table IV, we give details on search space sizes and maximal possible nonlinearities for all quaternary function dimensions we consider. One can already see that for dimension $n = 3$ the search space size is not amenable to exhaustive search.

Table IV: Search space sizes and the maximal nonlinearity values.

Size	Search space size	Maximal nonlinearity N_F^L
2	4^{16}	12
3	4^{64}	56
4	4^{256}	240
5	4^{1024}	992
6	4^{4096}	4 032
7	4^{16384}	16 256
8	4^{65536}	65 280

Table V: GA results under integer representation, $fitness_1$.

Size	Min	Max	Average	Std dev	Average bent
2	12	12	12.00	0.00	4.83
3	54	55	54.13	0.35	0.00
4	230	233	231.73	0.88	0.00
5	965	970	967.25	1.39	0.00
6	3 962	3 964	3 963.00	1.41	0.00

In Tables V and VI we give results respectively for GA integer and GP tree encodings, both under the first fitness function. The column *Average bent* denotes the average number of bent functions obtained in the last generation of an evolutionary run. This value is averaged over all 30 experimental runs. In Table V we notice that we are able to find quaternary functions with maximal nonlinearity as well as bent functions only for the smallest size we investigated ($n = 2$). All larger sizes result in functions having significantly lower nonlinearity than the one theoretically obtainable (see Table IV). Observe that we did not conduct experiments on integer encoding for $n > 6$, since those sizes resulted in significantly smaller nonlinearities than the maximal ones.

Table VI gives results for GP under tree encoding (recall that in our tree encoding there are 4 possible values, which differentiates our representation from the "standard" one as used in classic GP). As it can be seen, in this case we were able to find non-bent maximally nonlinear functions as well as bent quaternary functions for all considered dimensions. Since GP was able to find maximally nonlinear functions in every run even for the smallest tree depth considered in our investigation, we do not give here results for larger depths. We note however that they are in accordance with the presented results.

The fact that we are always able to find maximal nonlinear functions suggests that this optimization problem, although extremely difficult for GA with integer encoding, is easy for GP under the tree encoding. Still, the average number of bent functions found by GP is small, except for the smallest problem instance with $n = 2$ variables. Hence, GP is not very efficient in generating bent functions under this first fitness function. To cope with this situation, we used the second fitness function.

In Table VII we give results for GA based on integer encoding under the second fitness function. Interestingly, when

Table VI: GP results with tree representation, tree depth 4, $fitness_1$.

Size	Min	Max	Average	Std dev	Average bent
2	12	12	12.00	0.00	30.20
3	56	56	56.00	0.00	0.83
4	240	240	240.00	0.00	2.97
5	992	992	992.00	0.00	0.47
6	4 032	4 032	4 032.00	0.00	2.26
7	16 256	16 256	16 256.00	0.00	0.29
8	65 280	65 280	65 280.00	0.00	2.00

Table VII: GA results with integer representation, $fitness_2$.

Size	Min	Max	Average	Std dev	Average bent
2	12	12	12.00	0.00	0.00
3	54	55	54.03	0.18	0.00
4	231	233	231.93	0.58	0.00
5	964	969	966.33	1.32	0.00
6	3 959	3 959	3 959.00	0.00	0.00

considering maximally nonlinear functions we do not see much difference from the first case (cf. Table V), but we observe that for $n = 2$ we get worse results, since in this case we are not even able to obtain a single bent quaternary function over all experimental runs.

In Figure 1, we give boxplots for the second fitness function for all quaternary function sizes, and for tree depths of 4, 6, and 8. We can observe that the best results are obtained with the depth 8. We observe that when the tree depth equals 6, the average number of bent functions is significantly larger for $n = 7$ and 8 than with the tree depth of 8. Still, on average over all problem sizes, tree depth of 8 performs the best. Consequently, in Table VIII, we report the results of GP performance under the second fitness function using tree depth 8. We notice that for all experiments we were able to find both maximally nonlinear and bent quaternary functions in every run. Actually, for n up to 6 we see that more than half of the population are bent quaternary functions on average. This serves as a strong indication that our improved fitness function works well, if the encoding is appropriate.

V. Discussion

In this section, we first discuss how to map bent quaternary Boolean functions of n variables into bent Boolean functions of $2n$ variables. Next, we compare such results (i.e., constructed bent Boolean functions) with the state-of-the-art results from the relevant literature. Finally, we briefly discuss several possible future research directions.

Table VIII: GP results with tree representation, tree depth 8, $fitness_2$.

Size	Min	Max	Average	Std dev	Average bent
2	12	12	12.00	0.00	160.13
3	56	56	56.00	0.00	134.80
4	240	240	240.00	0.00	143.50
5	992	992	992.00	0.00	93.75
6	4 032	4 032	4 032.00	0.00	111.5
7	16 128	16 256	16 170.67	0.58	13.33
8	65 279	65 280	65 279.50	0.71	1

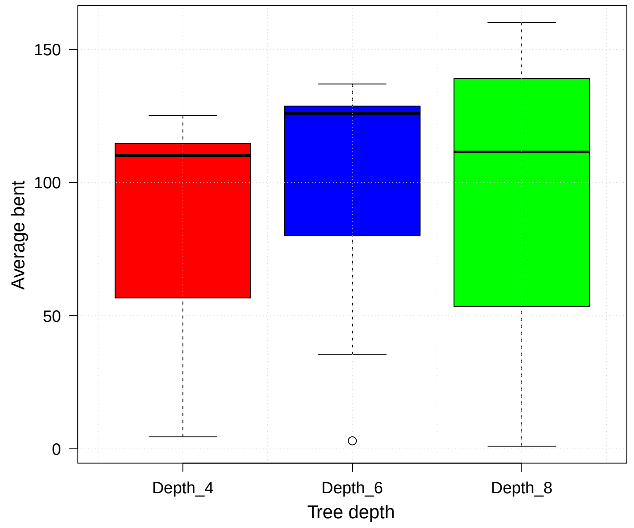

Figure 1: Average number of bent functions over all function sizes.

A. From Quaternary to Binary Functions

Once we obtain quaternary functions, the question is how to use them. One option is to employ them directly as they are, which is the approach taken for instance by Schmidt in [10], where he uses quaternary constant-amplitude codes for multicode CDMA. Another option is to transform quaternary functions into Boolean functions. Here, we follow this line of work. There are several possible mappings to transform a quaternary function into a Boolean one [18]. In our experiments, we use the *Gray mapping* since it is a distance-preserving bijection (i.e. an isometry) between $\mathbb{Z}_4^n$ under the Lee distance and $\mathbb{F}_2^{2n}$ under the Hamming distance. The Gray mapping $\phi : \mathbb{Z}_4 \to \mathbb{F}_2 \times \mathbb{F}_2$ is defined as:

$$0 \to 00,\ 1 \to 01,\ 2 \to 11,\ 3 \to 10.$$

Alternatively, if $u, v \in \mathbb{F}_2$ and $w = u + 2v$ (i.e., by using 2-adic expansion), the Gray mapping equals $\phi(w) = (v, u \oplus v)$. The mapping ϕ can now be extended naturally to $\mathbb{Z}_4^n$. Observe that the same mapping can also be used to transform Boolean functions into quaternary functions.

B. Comparison with the State-of-the-art Results for Boolean Functions

Our results show that whenever we obtain a bent quaternary function it maps to two bent Boolean functions. Conversely, when we obtain a non-bent maximally nonlinear quaternary function, it never maps to a pair of bent Boolean functions (nor to a single bent Boolean function). Consequently, here we only discuss the results obtained with the second fitness function and compare them with the state-of-the-art results.

Picek, Sisejkovic, and Jakobovic investigate the performance of two immunological algorithms, as well as genetic algorithms and evolution strategy (for all algorithms they consider bitstring and floating-point representation). Their results show that they are able to find bent Boolean functions only for the case of $n = 6$ variables. For all larger sizes, the nonlinearity is significantly below the maximal attainable one [28]. In our work, we were able to find bent Boolean functions of up to 16 inputs in every run.

Next, Picek et al. investigate a number of evolutionary algorithms in order to evolve bent Boolean functions with 8 inputs [27]. The results obtained there suggest that Cartesian genetic programming and genetic programming have the best performance, particularly for small tree sizes with which GP is able to reach maximal nonlinearity in 100% of the cases. Naturally, a direct comparison with our work is difficult since they consider only functions with 8 inputs and it remains to be seen how their approach would perform for larger Boolean function sizes. Considering our results, the comparable size is 4 for quaternary functions, where we were able to find bent functions in every run and where more than half of the population is composed of bent functions.

Hrbacek and Dvorak use Cartesian GP to evolve bent Boolean functions in dimensions $[6, \cdots, 16]$. They report success (i.e., they find bent Boolean functions) in each experimental run, which makes their results directly comparable with ours. Still, they use a computer cluster with 112 nodes (Intel E5-2670) and 128 GB of RAM [26] while we use a desktop computer with Intel i5-3470 and 8 GB of RAM. Consequently, our approach seems to be more efficient.

Finally, Picek and Jakobovic use GP in order to design secondary constructions that are then used to obtain bent Boolean functions [29]. There, the authors are able to find bent Boolean functions for much larger sizes than we give here, but let us note that they do not evolve larger Boolean functions directly but use instead a clever trick that enables them to expand small bent Boolean functions into larger ones. The secondary construction method used in that work always generates the same set of bent functions, since it relies on the initial set of bent functions of less variables. The approach presented in this paper allows finding different bent functions in every algorithm run.

C. Future Work

On the basis of the obtained results, there are several possible future directions to explore. The first option is to consider quaternary functions of more variables. Our results indicate that the main issue in pursuing this approach is the computational complexity of the Walsh-Hadamard transformation and not the difficulty of finding bent quaternary functions. In order to speed up the Walsh-Hadamard calculation, we could implement a butterfly divide-and-conquer algorithm analogously to the case of Boolean functions [12], but we did not find any reference for the quaternary case in the literature. The second option is to consider the Hamming distance when calculating nonlinearity. Although then the mapping between $\mathbb{Z}_4^n$ and $\mathbb{F}_2^{2n}$ is less elegant, finding bent Boolean functions is still possible.

Next, it is well known that there exists a connection among bent Boolean, quaternary, and generalized Boolean functions (mappings between $\mathbb{F}_2^n$ and $\mathbb{Z}_q$) [17]. It would be interesting to see how difficult is the evolution of generalized Boolean functions, especially with respect to their application in designing *orthogonal Latin squares* (OLS) via *cellular automata* (CA). As a matter of fact, a recent work by Mariot et al. [31] shows that balanced generalized Boolean functions can be used to define pairs of CA local rules that produce OLS. This observation has been partially exploited in [32] to evolve pairs of binary CA rules that generate OLS via genetic algorithms and genetic programming. An interesting venue for future research would be to investigate how the evolutionary algorithms presented in this paper would perform when evolving generalized Boolean functions that are then mapped to pairs of CA local rules. The fitness function, in this case, would take into account the orthogonality of the Latin squares produced by these pairs of CA rules.

From a more general perspective, here we use quaternary trees. Our results suggest that this representation significantly outperforms the integer encoding with values in $\mathbb{Z}_4$, although it considers only a subset of the search space (recall that there is no NOT function in our experiments). It would be interesting to explore whether this quaternary tree representation would result in high quality results for other problems where the standard GP representation offers good results. We see no obstacle in using it as long as there is a clear notion of the desired properties in both representations, as well as a one-to-one mapping between them.

VI. Conclusions

In this paper, we introduced the problem of evolving quaternary functions with maximal nonlinearity. We experimented with two encodings, namely integer encoding which is employed for GA experiments and tree encoding for GP, and we showed that the latter offers by far superior results. The results for quaternary tree encoding show that we were able to obtain bent functions for all dimensions we experimented with. From there, we used Gray mapping to obtain bent Boolean functions in $2n$ variables (where we went up to 16 variables). Our results are either comparable or better than those obtained with other techniques when evolving bent Boolean functions.

Besides bent quaternary functions, we showed that it is possible to construct maximally nonlinear quaternary functions that are not bent, a fact which to the best of our knowledge was not known before. Finally, we note the efficiency of quaternary tree encoding, which we believe could also be competitive in other problems when Boolean functions are considered.

Acknowledgments

This work has been supported in part by Croatian Science Foundation under the project IP-2014-09-4882.

References

[1] O. Rothaus, "On "bent" functions," *Journal of Combinatorial Theory, Series A*, vol. 20, no. 3, pp. 300 – 305, 1976.

[2] A. Bernasconi, B. Codenotti, and J. M. Vanderkam, "A characterization of bent functions in terms of strongly regular graphs," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 984–985, Sep 2001.

[3] S. Kavut, S. Maitra, and M. D. Yucel, "Search for boolean functions with excellent profiles in the rotation symmetric class," *IEEE Transactions on Information Theory*, vol. 53, no. 5, pp. 1743–1751, May 2007.

[4] A. Kerdock, "A class of low-rate nonlinear binary codes," *Information and Control*, vol. 20, no. 2, pp. 182 – 187, 1972.

[5] C. Xiang, C. Ding, and S. Mesnager, "Optimal codebooks from binary codes meeting the levenshtein bound," *IEEE Trans. Information Theory*, vol. 61, no. 12, pp. 6526–6535, 2015.

[6] Y. Zheng, J. Pieprzyk, and J. Seberry, "HAVAL — a one-way hashing algorithm with variable length of output (extended abstract)," in *Advances in Cryptology — AUSCRYPT '92: Workshop on the Theory and Application of Cryptographic Techniques Gold Coast, Queensland, Australia, December 13–16, 1992 Proceedings*, J. Seberry and Y. Zheng, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 81–104.

[7] M. Hell, T. Johansson, A. Maximov, and W. Meier, "A stream cipher proposal: Grain-128," in *2006 IEEE International Symposium on Information Theory*, July 2006, pp. 1614–1618.

[8] C. M. Adams, "Constructing symmetric ciphers using the cast design procedure," *Designs, Codes and Cryptography*, vol. 12, no. 3, pp. 283–316, Nov 1997.

[9] P. Méaux, A. Journault, F.-X. Standaert, and C. Carlet, "Towards stream ciphers for efficient fhe with low-noise ciphertexts," in *Advances in Cryptology – EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016*, M. Fischlin and J.-S. Coron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 311–343.

[10] K. U. Schmidt, "Quaternary constant-amplitude codes for multicode cdma," *IEEE Transactions on Information Theory*, vol. 55, no. 4, pp. 1824–1832, April 2009.

[11] Z. Jadda, P. Parraud, and S. Qarboua, "Quaternary cryptographic bent functions and their binary projection," *Cryptography and Communications*, vol. 5, no. 1, pp. 49–65, 2013.

[12] C. Carlet, "Boolean functions for cryptography and error-correcting codes," in *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, Y. Crama, , and P. L. Hammer, Eds. New York: Cambridge University Press, 2011, pp. 257–397.

[13] R. Forrié, "The Strict Avalanche Criterion: Spectral Properties of Boolean Functions and an Extended Definition," in *Advances in Cryptology - CRYPTO' 88*, ser. Lecture Notes in Computer Science, S. Goldwasser, Ed. Springer New York, 1990, vol. 403, pp. 450–468.

[14] B. Preneel, W. Van Leekwijck, L. Van Linden, R. Govaerts, and J. Vandewalle, "Propagation characteristics of Boolean functions," in *Proc. workshop on the theory and application of cryptographic techniques on Advances in cryptology*, ser. EUROCRYPT '90. New York, NY, USA: Springer-Verlag New York, Inc., 1991, pp. 161–173.

[15] J. Dillon, "A Survey of Bent Functions*," Reprinted from the NSA Technical Journal. Special Issue, Tech. Rep., 1972, unclassified.

[16] Z. Jadda and P. Parraud, "Z4-nonlinearity of a constructed quaternary cryptographic functions class," in *Sequences and Their Applications – SETA 2010: 6th International Conference, Paris, France, September 13-17, 2010. Proceedings*, C. Carlet and A. Pott, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 270–283.

[17] P. Solé and N. Tokareva, "Connections between quaternary and binary bent functions," Cryptology ePrint Archive, Report 2009/544, 2009.

[18] Z. Jadda, P. Parraud, and S. Qarboua, "Quaternary cryptographic bent functions and their binary projection," *Cryptography and Communications*, vol. 5, no. 1, pp. 49–65, 2013.

[19] W. Millan, A. Clark, and E. Dawson, "An Effective Genetic Algorithm for Finding Highly Nonlinear Boolean Functions," in *Proceedings of the 1st Int Conference on Information and Communication Security*, ser. ICICS '97. London, UK, UK: Springer-Verlag, 1997, pp. 149–158.

[20] W. Millan, A. J. Clark, and E. Dawson, "Heuristic design of cryptographically strong balanced boolean functions," in *EUROCRYPT '98*, ser. LNCS, K. Nyberg, Ed., vol. 1403. Springer, 1998, pp. 489–499.

[21] J. McLaughlin and 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.

[22] J. A. Clark, J. Jacob, S. Maitra, and 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, Dec 2003, pp. 2173–2180 Vol.3.

[23] L. Mariot and A. Leporati, "A Genetic Algorithm for Evolving Plateaued Cryptographic Boolean Functions," in *Theory and Practice of Natural Computing - Fourth International Conference, TPNC 2015, Mieres, Spain, December 15-16, 2015. Proceedings*, 2015, pp. 33–45.

[24] S. Picek, D. Jakobovic, and M. Golub, "Evolving Cryptographically Sound Boolean Functions," in *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2013, pp. 191–192.

[25] L. Mariot and A. Leporati, "Heuristic search by particle swarm optimization of boolean functions for cryptographic applications," in *Genetic and Evolutionary Computation Conference, GECCO, Madrid, Spain, July 11-15, 2015*, 2015, pp. 1425–1426.

[26] R. Hrbacek and V. Dvorak, "Bent Function Synthesis by Means of Cartesian Genetic Programming," in *Parallel Problem Solving from Nature - PPSN XIII*, ser. Lecture Notes in Computer Science, T. Bartz-Beielstein, J. Branke, B. Filipič, and J. Smith, Eds. Springer International Publishing, 2014, vol. 8672, pp. 414–423.

[27] S. Picek, D. Jakobovic, J. F. Miller, L. Batina, and M. Cupic, "Cryptographic boolean functions: One output, many design criteria," *Appl. Soft Comput.*, vol. 40, pp. 635–653, 2016.

[28] S. Picek, D. Sisejkovic, and D. Jakobovic, "Immunological algorithms paradigm for construction of boolean functions with good cryptographic properties," *Eng. Appl. of AI*, vol. 62, pp. 320–330, 2017.

[29] S. Picek and D. Jakobovic, "Evolving Algebraic Constructions for Designing Bent Boolean Functions," in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016*, 2016, pp. 781–788.

[30] R. Poli, W. B. Langdon, and N. F. McPhee, *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008.

[31] L. Mariot, E. Formenti, and A. Leporati, "Enumerating orthogonal latin squares generated by bipermutive cellular automata," in *Cellular Automata and Discrete Complex Systems - 23rd IFIP WG Int Workshop, AUTOMATA 2017, Milan, Italy, June 7-9, 2017*, pp. 151–164.

[32] L. Mariot, S. Picek, D. Jakobovic, and A. Leporati, "Evolutionary algorithms for the design of orthogonal latin squares based on cellular automata," in *Proc. of the Genetic and Evolutionary Computation Conference, GECCO, Berlin, Germany, July 15-19*, 2017, pp. 306–313.