

On Counts and Densities of Homogeneous Bent Functions: An Evolutionary Approach

Claude Carlet¹, Marko Đurasevic², Domagoj Jakobovic², Luca Mariot³, Stjepan Picek², and Alexandr Polujan⁴

¹ University of Bergen, Bergen, Norway

claude.carlet@gmail.com

² Faculty of Electrical Engineering and Computing, University of Zagreb, Unska 3, Zagreb, Croatia ,

marko.durasevic@fer.hr, domagoj.jakobovic@fer.hr, stjepan@computer.org

³ Semantics, Cybersecurity and Services Group, University of Twente, 7522 NB Enschede, The Netherlands

1.mariot@utwente.nl

⁴ Otto-von-Guericke University Magdeburg, Germany

alexandr.polujan@ovgu.de

November 18, 2025

Abstract

Boolean functions with strong cryptographic properties, such as high nonlinearity and algebraic degree, are important for the security of stream and block ciphers. These functions can be designed using algebraic constructions or metaheuristics. This paper examines the use of Evolutionary Algorithms (EAs) to evolve homogeneous bent Boolean functions, that is, functions whose algebraic normal form contains only monomials of the same degree and that are maximally nonlinear. We introduce the notion of density of homogeneous bent functions, facilitating the algorithmic design that results in finding quadratic and cubic bent functions in different numbers of variables.

Keywords Boolean functions, Cryptography, Genetic Algorithm, Genetic Programming

1 Introduction

Boolean functions with specific cryptographic features have been studied extensively within symmetric cryptography for many years [1]. Designs for both stream ciphers and block ciphers typically require functions that are balanced, highly nonlinear,

of a prescribed algebraic degree, or efficient to evaluate, among other desirable attributes. Such functions can be obtained with different approaches. One possibility is to rely on algebraic constructions, where entire families of Boolean functions are generated with guaranteed properties. However, deriving these constructions is often challenging, and for several classes of functions, no algebraic method is currently known, making empirical search a viable alternative.

Empirical strategies usually fall into three broad categories: random search, ad-hoc heuristics, and metaheuristics. Random sampling of the search space is straightforward but seldom competitive except in trivial instances. Specialised heuristics can perform remarkably well, sometimes producing state-of-the-art results [11], but their success depends strongly on the problem structure and the availability of expert knowledge for their design. Metaheuristics provide a middle ground, offering good performance across a variety of problems while typically requiring only limited problem-specific tuning [6].

In this work, we consider a particularly demanding problem for which the number of known solutions is very limited. More precisely, our focus is on evolving *homogeneous* Boolean functions, i.e., functions whose algebraic normal form (ANF) contains only monomials of the same degree. Such functions are interesting from an implementation perspective, as their uniform degree can simplify and speed up evaluation.

Only a small number of studies have addressed the construction of homogeneous Boolean functions so far. The earliest contribution, dating back to 2001, used combinatorial techniques to obtain balanced and bent homogeneous functions in six variables [30]. Quadratic homogeneous bent functions (of degree 2) are well understood and characterised in classical literature [12]. For the cubic case (of degree 3), however, there is essentially a single known construction [31]. In contrast to the homogeneous case, the broader problem of evolving bent functions has been examined in numerous works employing a range of methods, e.g., [7, 8, 23, 13, 17]. Related research has also demonstrated that evolutionary approaches can successfully produce Boolean functions with high algebraic degree and other cryptographically relevant criteria [22, 18, 27].

In this paper, we investigate the efficiency of several Boolean function representations based on symbolic, truth table, and algebraic normal form encodings. We experiment with Boolean function sizes from dimension 6 to dimension 12 and try to evolve homogeneous bent Boolean functions. Previous results indicate the problem is relatively easy for degree 2, but very difficult for degree 3 [2]. Our contributions are twofold: first, we offer theoretical insights about the notion of the density of homogeneous bent functions, which predicts the likelihood of finding these functions depending on the number of terms in the ANF form. Second, with a custom algorithm design and careful analysis, we can find cubic bent functions, a result that has, up to now, not been achieved with EAs.

2 Preliminaries

We denote by $\mathbb{F}_2 = \{0, 1\}$ the finite field with two elements, equipped with XOR and logical AND, respectively, as the sum and multiplication operations. The n -dimensional vector space over \mathbb{F}_2 is denoted as \mathbb{F}_2^n , consisting of all 2^n binary vectors of length n . Given $a, b \in \mathbb{F}_2^n$, their inner product equals $a \cdot b = \bigoplus_{i=1}^n a_i b_i$ in \mathbb{F}_2^n . A Boolean function of n variables is a mapping $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$. Interested readers can find detailed information about Boolean functions in [12, 1].

2.1 Boolean Function Representations

Truth Table Representation. The most basic means to uniquely represent a Boolean function $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$ is by using its truth table. The truth table of a Boolean function f is the list of pairs $(x, f(x))$ of input vectors $x \in \mathbb{F}_2^n$ and function outputs $f(x) \in \mathbb{F}_2$. Once a total order has been fixed on the input vectors of \mathbb{F}_2^n (most commonly, the lexicographic order), the truth table can be identified only by the 2^n -bit function output vector.

Walsh-Hadamard Transform. The Walsh-Hadamard transform $W_f : \mathbb{F}_2^n \rightarrow \mathbb{Z}$ is another commonly used unique representation of a Boolean function $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$, which allows us to characterize several interesting cryptographic properties. Formally, the Walsh-Hadamard transform measures the correlation between f and the linear functions $a \cdot x$, for all $a \in \mathbb{N}$, as follows:

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

with the sum calculated in \mathbb{Z} . The Walsh-Hadamard transform is an involution up to a normalization by a constant. Therefore, one can retrieve the truth table representation of f from the spectrum of its Walsh-Hadamard coefficients $W_f(a)$.

Algebraic Normal Form. A third unique representation of a Boolean function $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$ is as a multivariate polynomial in the quotient ring $\mathbb{F}_2[x_1, \dots, x_n] / (x_1^2 \oplus x_1, \dots, x_n^2 \oplus x_n)$. This polynomial is the Algebraic Normal Form (ANF) of f , and it is defined as:

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

where $h(a)$ is given by the binary Möbius transform:

$$h(a) = \bigoplus_{x \preceq a} f(x), \text{ for any } a \in \mathbb{F}_2^n, \quad (3)$$

with \preceq denoting the covering relation between vectors of \mathbb{F}_2^n , i.e., a covers x means that $x_i \leq a_i, \forall i \in \{0, \dots, n-1\}$. Similarly to the Walsh-Hadamard transform, the Möbius transform is also an involution, so one can use it to switch between the truth table and the algebraic normal form representations of a function.

2.2 Properties and Bounds

Nonlinearity The minimum Hamming distance between a Boolean function f and all affine functions is the nonlinearity of f , which is calculated from the Walsh-Hadamard spectrum as follows [1]:

$$nl_f = 2^{n-1} - \frac{1}{2} \max_{a \in \mathbb{F}_2^n} \{ |W_f(a)| \}. \quad (4)$$

For every n -variable Boolean function, f satisfies the covering radius bound:

$$nl_f \leq 2^{n-1} - 2^{\frac{n}{2}-1}. \quad (5)$$

Notice that Eq. (5) cannot be tight when n is odd.

Bentness. The functions whose nonlinearity equals the maximal value from Eq. (5) are called bent. Bent functions exist for n even only.

Algebraic Degree. The algebraic degree \deg_f of a Boolean function f is defined as the number of variables in the largest product term of the function's ANF having a non-zero coefficient, see [12]:

$$\deg_f = \max\{w_H(a) : a \in \mathbb{F}_2^n, h(a) = 1\}. \quad (6)$$

The algebraic degree of a bent function cannot exceed $n/2$. A Boolean function is affine if and only if it has an algebraic degree at most 1. We will call quadratic functions the Boolean functions of algebraic degree at most 2, and cubic functions those of algebraic degree at most 3. This means that an affine function is a particular quadratic function (in the same sense that a constant function is a particular affine function).

Homogeneity. A Boolean function is called homogeneous if all the monomials in its algebraic normal form have the same algebraic degree. The only known homogeneous bent functions are quadratic and cubic ones. Furthermore, it is not known whether homogeneous bent functions of higher degrees exist [29].

3 Related Work

Research on the use of evolutionary algorithms and other metaheuristics to design Boolean functions with strong cryptographic properties dates back to the 90s. In this section, we give only a brief overview of the most significant work in this field, referring the reader to the survey [6] for a more complete account.

Millan et al. [20] were the first to pioneer the use of Genetic Algorithms (GA) to evolve highly nonlinear Boolean functions, focusing on the truth table as a genotype representation for the candidate solutions, and performing experiments on functions

with 8 to 16 inputs. Later works in the same direction explored more refined genetic operators—such as crossover operators that preserve the balancedness property [21, 13])—or the optimization of other cryptographic properties beyond nonlinearity, such as autocorrelation and correlation immunity [5]. Concerning other representations, Clark et al. [4] proposed the spectral inversion method, in which the Walsh spectra corresponding to specific cryptographic profiles are optimized using a Simulated Annealing algorithm to obtain Boolean functions with optimal properties. Later, Mariot and Leporati [15] devised a GA to design semi-bent functions through the spectral inversion approach.

Besides GA and Simulated Annealing, other metaheuristics have been considered for the design of Boolean functions for cryptographic purposes. Picek et al. [24] were the first to experiment with GP to evolve 8-variable Boolean functions with a good combination of high nonlinearity and other desirable cryptographic properties, observing that it substantially outmatched GA’s performance. Later, Picek et al. also proposed to use CGP to evolve Boolean functions with good cryptographic properties [25]. Mariot and Leporati [16] considered a discrete version of Particle Swarm Optimization (PSO) to evolve functions from 7 to 12 variables having a good trade-off of nonlinearity, correlation immunity, and propagation criteria.

A related research strand focuses on the use of EA to evolve bent functions specifically. In this regard, Picek and Jakobovic applied GP to evolve algebraic constructions, which are, in turn, used to generate bent functions of up to 24 variables [23]. Hrbacek and Dvorak [8] considered the use of Cartesian GP to synthesize bent functions up to 16 variables, while Husa and Dobai investigated linear GP for the same problem, obtaining bent functions up to 24 inputs [9]. Picek et al. adapted various EAs, including GA and GP, to evolve quaternary bent functions, which are a generalization of classic bent functions over the alphabet \mathbb{Z}_4 [26]. Mariot et al. investigated the use of EA to design hyper-bent Boolean functions [14], a subclass of binary bent functions that lie at maximum distance from all bijective monomial functions. Further, Mariot et al. [17] applied evolutionary strategies to design algebraic constructions based on cellular automata to generate quadratic bent functions.

As mentioned in Section 1, there are very few works in the literature addressing the construction of homogeneous Boolean functions, such as [30], which provides a combinatorial classification of bent and balanced homogeneous functions in 6 variables. As far as we know, the only work applying EA to the design of homogeneous bent functions is [2], which provides preliminary results on the use of GA and GP to evolve this type of function. Interestingly, the authors managed to evolve quadratic homogeneous bent functions, but no examples of cubic homogeneous functions were found. This empirical finding serves as the starting point for the present work, in which we aim to improve on the results of [2].

4 Methodology

4.1 Counts and Densities of Homogeneous Bent Functions

In this section, we present the exact counts and empirical densities of bent quadratic and cubic homogeneous Boolean functions, focusing on the two completely characterized cases $n = 6$ and $n = 8$. We then show that these results highlight the increasing difficulty of evolving cubic homogeneous bent functions compared to quadratic ones, thus addressing the question recently raised in [2]. We use these insights to inform our evolutionary approach in the next sections.

To this end, we introduce the notion of the *density* of homogeneous bent functions of algebraic degree d in n variables, defined as

$$\delta_{n,d} = \frac{|\mathcal{HB}_{n,d}|}{2^{\binom{n}{d}}}, \quad (7)$$

where $|\mathcal{HB}_{n,d}|$ denotes the number of homogeneous bent functions of degree d in n variables, and $2^{\binom{n}{d}}$ is the total number of homogeneous Boolean functions of degree d . A more detailed density measure is obtained by restricting the count to functions containing exactly k terms, namely

$$\delta_{n,d,k} = \frac{|\mathcal{HB}_{n,d,k}|}{\binom{\binom{n}{d}}{k}}, \quad (8)$$

where $|\mathcal{HB}_{n,d,k}|$ denotes the number of homogeneous bent functions of degree d in n variables with k terms, and $\binom{\binom{n}{d}}{k}$ is the number of all homogeneous Boolean functions of degree d with k terms.

In the following two subsections, we compare these density values for quadratic ($d = 2$) and cubic ($d = 3$) homogeneous bent functions in $n = 6$ and $n = 8$ variables, since only for these cases are all such functions completely known.

4.1.1 Quadratic Homogeneous Bent Functions

For quadratic homogeneous bent functions in n variables, the value $\delta_{n,2}$ defined by Eq. (7) can be computed theoretically, since the value $|\mathcal{HB}_{n,2}|$ is well-known (see, e.g., [12, Chapter 15]) and is given by:

$$|\mathcal{HB}_{n,2}| = 2^{k^2-k} \prod_{i=0}^{k-1} (2^{2i+1} - 1). \quad (9)$$

Now, we are interested both in the asymptotic value of $\delta_{n,2}$ and in the exact behavior for small n (here $n = 6, 8$) of the values $\delta_{n,2,k}$. To compute $\lim_{n \rightarrow \infty} \delta_{n,2}$, one can use Wolfram Mathematica [10], which gives

$$\lim_{n \rightarrow \infty} \delta_{n,2} = \left(\frac{1}{2}; \frac{1}{4} \right)_\infty \approx 0.419422, \quad (10)$$

where $(a; q)_\infty$ denotes the q -Pochhammer symbol. This result means that roughly 42% of all quadratic homogeneous functions in n variables are bent when $n \rightarrow \infty$.

For $n = 6, 8$, all quadratic homogeneous bent functions can be enumerated within minutes. Using this data, we compute the values of $\delta_{n,2,k}$ and present them as histograms in Figure 1. For both $n = 6$ and $n = 8$, we observe that for most values of k , the densities $\delta_{n,2,k}$ are close to the asymptotic value $\delta_{n,2}$ given by Eq. (7). From an evolutionary perspective, this indicates that a large part of the search space can be effectively explored: functions with a broad range of term counts can be evolved without a substantial drop in success probability, as bent functions are relatively evenly distributed across most values of k .

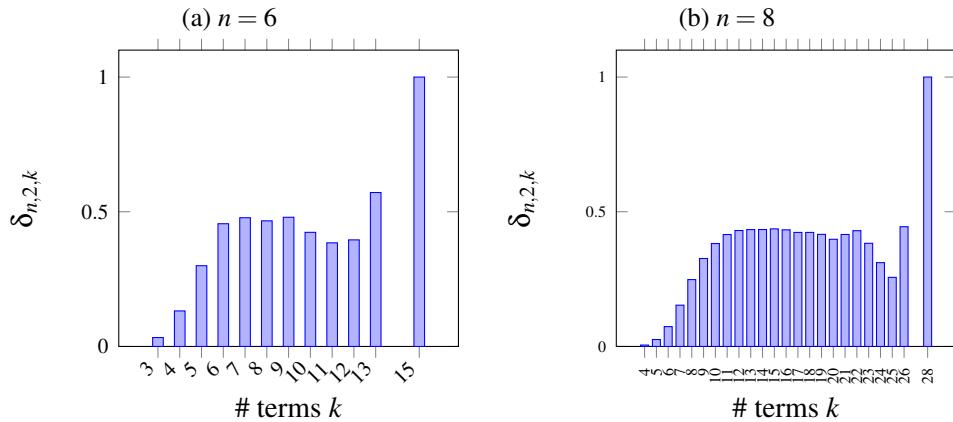


Figure 1: Non-zero density values $\delta_{n,2,k}$.

4.1.2 Cubic Homogeneous Bent Functions

Unlike the quadratic case, the number of homogeneous cubic bent functions in n variables is not known theoretically. Therefore, the asymptotic value of $\delta_{n,3}$ cannot be determined. However, for $n = 6$ and $n = 8$, the exact counts of homogeneous cubic bent functions are known, which allows us to compute the corresponding values of $\delta_{n,3}$. Specifically, it was shown in [3] that the total number of homogeneous cubic bent functions in six variables is $|\mathcal{HB}_{6,3}| = 30$. For eight variables, the total number is $|\mathcal{HB}_{8,3}| = 293,760$; see [19] for details. Using these values, it is straightforward to compute:

$$\delta_{6,3} = \frac{30}{2^{\binom{6}{3}}} \approx 2.86102 \times 10^{-5} \quad \text{and} \quad \delta_{8,3} = \frac{293,760}{2^{\binom{8}{3}}} \approx 4.07674 \times 10^{-12}. \quad (11)$$

These results show that, unlike in the quadratic case, cubic homogeneous bent functions are extremely rare among all cubic homogeneous functions. Moreover, they are not evenly distributed across functions with a fixed number of terms k . For instance, in the case of $n = 6$ variables, all 30 homogeneous cubic bent functions

contain exactly 16 terms in their ANF [3], hence $\delta_{6,3,k} = 0$ for all $k \neq 16$. Similarly, for $n = 8$, only a few values of k satisfy $\delta_{8,3,k} \neq 0$; these are listed in Table 1.

Table 1: Counts and densities for cubic homogeneous bent functions in $n = 8$.

# terms k	$ \mathcal{HB}_{8,3,k} $	$\delta_{n,3,k}$
24	6,720	1.54304×10^{-12}
27	13,440	1.81992×10^{-12}
28	5,760	7.53070×10^{-13}
32	6,720	1.54304×10^{-12}
34	13,440	6.27280×10^{-12}
35	19,200	1.42564×10^{-11}
36	80,640	1.02646×10^{-10}
37	67,200	1.58246×10^{-10}
39	40,320	4.11439×10^{-10}
41	40,320	2.48073×10^{-9}

For larger values of n , a complete enumeration of cubic homogeneous bent functions remains infeasible. Nevertheless, the known values of k for which such functions exist can be used. These values, derived from previous studies [3, 29], are summarized in Table 2.

Table 2: Known values of k s.t. homogeneous cubic bent functions with k terms in the ANF in n variables exist.

n	Values of k
10	39, 49, 53, 57, 58, 61, 65, 66, 69, 70, 72, 75, 78
12	60, 90, 100, 110, 130, 140, 150
16	168

To highlight the difference between quadratic and cubic homogeneous bent functions, as well as their distributions among homogeneous functions with a fixed number of terms k in the ANF, we present in Figure 2 a histogram summarizing the results from Table 1.

In contrast to Figure 1, the viable k -region for cubic homogeneous bent functions is very narrow, indicating that most of the cubic search space contains almost no bent functions. From an evolutionary perspective, this extreme sparsity, combined with the enormous size of the cubic search space, explains the practical difficulty of evolving cubic homogeneous bent functions.

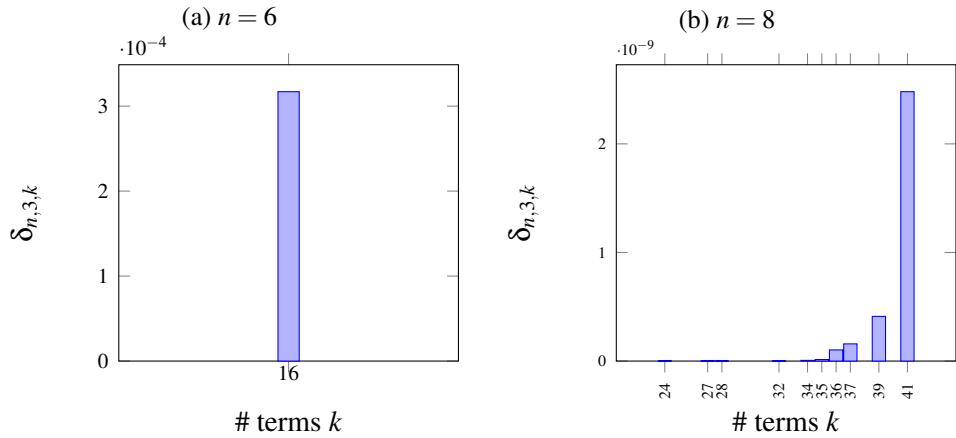


Figure 2: Non-zero density values $\delta_{n,3,k}$.

4.2 Encodings

4.2.1 Symbolic Encoding (GP).

The first encoding in our experiments uses tree-based genetic programming (GP) to represent a Boolean function in its symbolic form. This encoding usually achieves the best results when dealing with the evolution of Boolean functions with cryptographic properties [6]. In this case, we represent a candidate solution with a tree whose leaves correspond to the input variables $x_1, \dots, x_n \in \mathbb{F}_2$. The internal nodes are Boolean operators that combine the inputs received from their children and forward their output to the respective parent nodes. We use the following function set: OR, XOR, AND, AND2, XNOR, IF, and the NOT function that takes a single argument. This function set is common in previous applications and is based on our tuning results. The output of the root node is the output value of the Boolean function. The truth table of the function $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$ is determined by evaluating the tree over all possible 2^n assignments of the inputs at the leaves.

To restrict the search to homogeneous functions of a given degree, we perform the following additional transformations: the truth table is first converted to ANF via the Möbius transform, then all the elements of the ANF corresponding to monomials of the "wrong" degrees are set to zero. The obtained *corrected ANF* is transformed back into a truth table and finally into the Walsh-Hadamard spectrum, based on which we calculate nonlinearity.

4.2.2 Truth Table Encoding (TT).

The most common option for encoding a Boolean function is the truth table (TT) encoding [6], represented with a bitstring. For a Boolean function with n inputs, the truth table is encoded as a bitstring of length 2^n . The bitstring represents the Boolean function upon which the algorithm operates directly. Therefore, the algorithm, in

this case, explores the full space of n -variable Boolean functions, which has size 2^{2^n} . As in the previous encoding, in each evaluation, the truth table is transformed into the ANF form with the Möbius transform, and monomials not corresponding to a desired degree are reset to force homogeneity. The corrected ANF is converted back to a truth table, after which the nonlinearity properties are evaluated.

4.2.3 Reduced ANF Encoding (rANF).

The truth table encoding introduced above does not guarantee that the function the evolutionary algorithm is working with is always homogeneous. Hence, we also considered an encoding based on the ANF, where an individual is represented as a bitstring of length $\binom{n}{d}$, with d being the target degree. Each component in this bitstring specifies whether the corresponding d -degree monomial occurs (1) in the ANF of the function or not (0). Standard genetic operators can be applied to this bitstring, and the individual is then converted to the full ANF representation by mapping only to the bits of the appropriate d -degree monomials. In this way, the evolutionary algorithm only explores the restricted search space of homogeneous Boolean functions, which has size $2^{\binom{n}{d}}$. Each individual is evaluated by first retrieving its truth table and then by computing its nonlinearity through the Walsh-Hadamard transform.

4.2.4 Weighted ANF Encoding (wANF).

Finally, based on the densities of cubic (of degree 3) bent homogeneous functions from Section 4.1, we adopt the approach in which we use the reduced ANF encoding, but also restrict the number of monomials to a specific value. All the previous encodings may be viewed as "unrestricted", since the number of monomials may vary. On the other hand, in *weighted ANF*, encoding this number is preselected to one of the values for which bent functions of degree 3 are shown to exist. With this encoding, we perform a separate experiment for each of the predicted values of the number of monomials for a given function size. Rather than penalizing solutions with "wrong" number of monomials, the evolutionary genotype is designed so that the designated number of ones in the bitstring remains the same during evolution, with customized genetic operators that preserve this property.

4.3 Fitness Functions

4.3.1 (Homogeneous) Bent Functions.

Several objective functions can be defined to find bent functions, regardless of the representation and search algorithm. The approach used here is based on common choices in related works [6] and the authors' previous experience. Apart from maximizing the nonlinearity value, the applied objective value considers the whole Walsh-Hadamard spectrum and not only its extreme value (see Eq. (4)); hence, we count the number of occurrences of the maximal absolute value in the spectrum,

denoted as `#max_values`. As higher nonlinearity corresponds to a lower maximal absolute value, we aim for as few occurrences of the maximal value as possible, hoping it gets easier for the algorithm to reach the next nonlinearity value. The algorithm is thus provided with additional information, making the objective space more gradual. The fitness function to optimize for bent functions is defined as:

$$fit_{bent} = nl_f + \frac{2^n - \#max_values}{2^n}. \quad (12)$$

The second term never reaches the value of 1 since, in that case, we effectively reach the next nonlinearity level. Since all encodings are either constrained or repaired to represent a homogeneous function of a given degree, the homogeneity property is not explicitly included in the fitness function.

4.3.2 Homogeneous Bent Functions with k Monomials.

The previous fitness function optimizes for bent functions with any number of monomials in the ANF form; since it may be beneficial to restrict the number of monomials in the cubic case, we employ the additional function that penalizes the distance to the desired monomial number k . Only if the number of monomials is equal to k , the nonlinearity property is evaluated and awarded:

$$fit_{bent,k} = \begin{cases} -|num_monomials - k|, & \text{if } num_monomials \neq k; \\ nl_f + \frac{2^n - \#max_values}{2^n}, & \text{otherwise.} \end{cases} \quad (13)$$

Obviously, this fitness function is applicable only to the first three encodings, since the weighted ANF (wANF) already has the preselected number of monomials.

4.4 Algorithms and Parameters

Common Experimental Parameters. We employ the same evolutionary algorithm for all encodings: a steady-state selection with a 3-tournament elimination operator (denoted SST). In each iteration of the algorithm, three individuals are chosen at random from the population for the tournament, and the worst one in terms of fitness value is eliminated. The two remaining individuals in the tournament are used with the crossover operator to generate a new child individual, which then undergoes mutation with individual mutation probability $p_{mut} = 0.5$. The mutated child replaces the eliminated individual in the population. All experiments use a population size of 500 individuals and the same stopping criterion of 10^6 evaluations.

Genetic Operators. Concerning the genetic operators for both the bitstring and restricted encodings, we use the simple bit mutation and the shuffle mutation. For crossover, we employ one-point and uniform crossover operators. Each time

the evolutionary algorithm invokes a crossover or mutation operation, one of the previously described operators is randomly selected.

In the case of weighted ANF encoding, the individual bitstrings are always initialized with k bits set to one. The crossover and mutation operators used in this work are based on [13], so that all genetic operators preserve the number of ones throughout the evolution. For the crossover, we used the balanced weighted crossover operator, and the mutation used a two-bit inversion (which flips two random bits) and a mixing mutation, which shuffles the genes between two randomly chosen positions in the bitstring.

For the symbolic encoding, the genetic operators used in our experiments with tree-based GP are simple tree crossover, uniform crossover, size fair, one-point, and context preserving crossover [28] (selected at random), and subtree mutation. The option to use multiple genetic operators was based on previous results indicating better convergence when using a diverse set of operators.

Local Search. Finally, all three encodings can also be used with a generic local search operator, which works in the following way: the operator acts on a single solution and performs a predefined number of mutations. If a better solution is found, the new solution immediately replaces the original one, and the operator is applied again. If no better solution is found after the given number of mutations, the operator terminates. The operator is applied after each generation and acts upon the current best solution and a number of random solutions. In our experiments, the number of solutions undergoing local search was set to 1% of the population size, and the number of trials (random mutations per individual) was set to 30.

5 Experimental Results

In the experiments, all configurations were executed in 30 runs, and relevant statistical values are reported. For the quadratic case (degree 2), we used only the "unrestricted" encodings (GP, TT, rANF), which do not limit the number of monomials in the ANF form. This problem turns out to be relatively simple: all encodings managed to find a bent homogeneous function in all problem sizes (number of variables from 6 to 12), and in all runs. The results for this case are therefore not elaborated any further.

A much more difficult (and interesting) problem is the evolution of cubic bent functions (of degree 3). Previously used evolutionary approaches [2] were unable to find even a single cubic bent function in the sizes we consider. In six variables, all the encodings managed to find cubic bent functions, with most encodings succeeding in every run; the overall results are concisely presented as success rates over 30 runs for each configuration and are shown in Table 3.

The column "restricted weight" denotes whether the first fitness function fit_{bent} was used, which corresponds to the entry "unrestricted". If the second fitness function $fit_{bent,k}(13)$ was used, the entry in the column shows the preselected weight

vars	weight	GP	TT	rANF	wANF	rANF/LS	wANF/LS
6	unrestricted	21	30	30	—		
	16	24	30	30	30		
	unrestricted	0	0	0	—	0	—
	24	0	0	0	0	0	0
	27	0	0	0	0	0	0
	28	0	0	0	0	0	0
	32	0	0	0	0	0	0
	34	0	0	0	0	0	0
	35	0	0	0	0	0	0
	36	0	0	0	0	0	0
8	37	0	0	0	1	0	1
	39	0	0	1	4	0	2
	41	0	0	1	4	0	2

Table 3: Number of successful runs (out of 30) for each encoding and configuration.

(the number of monomials), based on analysis in section 4.1. For cubic functions of 6 variables, this number can only be 16. Indeed, even in the unrestricted experiments (with fit_{bent}), all the solutions contained exactly 16 monomials, which was shown by analyzing the functions after the evolution.

The GP encoding exhibited the worst performance, which is not in accordance with previous results in the literature when evolving cryptographically relevant Boolean functions. However, the main reason for this is most likely the indirect relationship between the genotype (symbolic function form) and the resulting truth table; since in the decoding process all extraneous monomials (not corresponding to the selected degree) are removed, the function that is evaluated for nonlinearity is not identical to the one used internally in GP.

For larger sizes, the algorithms only managed to find cubic bent functions in 8 variables (success rates for 10 and 12 are zero); the number of hits over 30 runs is presented in the lower part of Table 3. Only the reduced ANF and weighted ANF encodings managed to find bent functions in a small number of runs. Notably, the successful runs were only achieved when bitstring weight was restricted, either by penalization in rANF or by design in wANF. Apart from the number of successful runs, we may also observe absolute fitness values of different encodings over 30 runs; the accumulated best-of-run fitness values (over unrestricted and all restricted configurations) for $n = 8$ are shown in Figure 3a.

We may investigate encoding efficiency by examining fitness values directly. The results were analyzed using the Kruskal-Wallis test to determine the differences and a Dunn post-hoc test with Bonferroni correction and significance level of 0.05. Furthermore, pairwise comparisons were conducted with Mann-Whitney U tests. The results of the statistical tests show significant differences among all the encodings; subsequent pairwise comparisons suggest that TT is the worst performer, followed by GP, then by reduced ANF, and finally the weighted ANF as the best one.

The results for sizes 10 and 12 are presented in Figures 3b and 3c. In 10 variables, the tests show statistically significant differences in the same order of efficiency, with TT being the worst and wANF the best method. However, these differences are inconsequential, since all fitness values are grouped very tightly (note the scale in Figure 3b); disregarding the differences in fitness value, all the obtained results correspond to Boolean functions with the same nonlinearity value (which is the integer part of the fitness). Finally, for 12 variables, the statistical differences are again present, but they are of little importance since no encoding is even close to the nonlinearity of bent functions in 12 variables (2016).

5.0.1 Experiments with Local Search.

To try to improve results, we included the local search operator for the best two encodings (rANF and wANF) and repeated the experiments for 8, 10, and 12 variables in the hope of finding cubic bent functions. The statistical tests show that there are no significant differences between the local search and the original variants for both encodings; this observation also holds for all problem sizes. Unfortunately, the number of successful runs with local search is even lower, which is evident from Table 1, and there were no hits for larger sizes.

5.0.2 Effect of Restricting the Number of Monomials.

As mentioned before, the only configurations in which cubic bent functions in 8 variables are found are those that restrict the bitstring weight k . In accordance with findings in Section 4.1, bent functions are indeed found for weights that exhibit the largest densities of those functions (see Figure 2). In other words, it is very likely that without this restriction, those results would not be obtained. The statistical tests also confirm significant differences in pairwise comparison between the unrestricted and all accumulated restricted configurations, showing the advantage of restricting the number of monomials.

6 Conclusions

The results show that evolving homogeneous quadratic bent functions is relatively straightforward: all encodings reached optimal solutions across all tested sizes. In contrast, the cubic case was substantially more challenging. While all encodings succeeded in finding cubic bent functions in six variables, only the reduced ANF and weighted ANF found solutions for eight variables, and none succeeded for larger sizes. Statistical analysis confirmed performance differences between encodings, with the weighted ANF consistently outperforming all others, with the TT and GP encodings showing the weakest results. The inclusion of a local search operator did not improve success rates and, in some cases, reduced them. An important observation is that successful runs in the cubic case occurred only when the number of monomials was explicitly restricted, consistent with the theoretical distribution

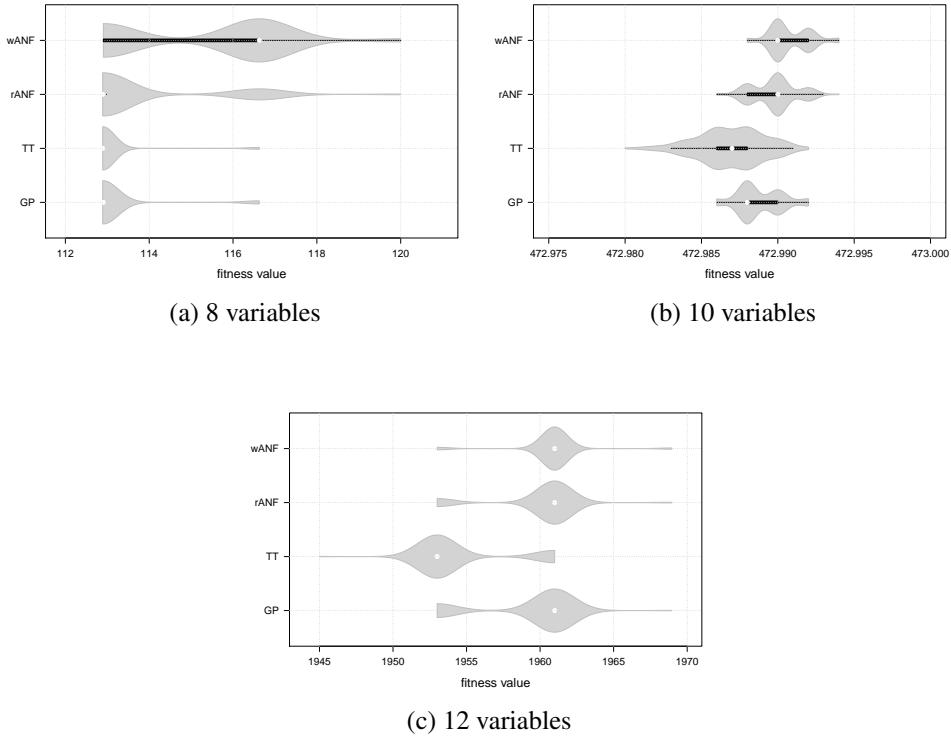


Figure 3: Accumulated fitness values for all encodings, sizes 8-12.

of bent functions over different weights. These results suggest that both the choice of encoding and the enforcement of structural constraints are crucial for guiding EAs towards feasible regions of the search space.

References

- [1] C. Carlet. *Boolean Functions for Cryptography and Coding Theory*. Cambridge University Press, Cambridge, 2021.
- [2] C. Carlet, M. Durasevic, D. Jakobovic, L. Mariot, and S. Picek. Degree is important: On evolving homogeneous boolean functions. GECCO '25 Companion, pages 795–798, New York, NY, USA, 2025. Association for Computing Machinery.
- [3] C. Charnes, M. Rötteler, and T. Beth. Homogeneous bent functions, invariants, and designs. *Des. Codes Cryptogr.*, 26(1-3):139–154, 2002.
- [4] J. A. Clark, J. L. Jacob, S. Maitra, and P. Stănică. Almost boolean functions: the design of boolean functions by spectral inversion. In *The 2003 Congress*

on Evolutionary Computation, 2003. CEC '03., volume 3, pages 2173–2180 Vol.3, 2003.

- [5] J. A. Clark, J. L. Jacob, S. Stepney, S. Maitra, and W. Millan. Evolving boolean functions satisfying multiple criteria. In A. Menezes and P. Sarkar, editors, *Progress in Cryptology — INDOCRYPT 2002*, pages 246–259, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.
- [6] M. Djurasevic, D. Jakobovic, L. Mariot, and S. Picek. A survey of metaheuristic algorithms for the design of cryptographic boolean functions. *Cryptography and Communications*, 15(6):1171–1197, July 2023.
- [7] J. Fuller, E. Dawson, and W. Millan. Evolutionary generation of bent functions for cryptography. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2003, Canberra, Australia, December 8-12, 2003*, pages 1655–1661. IEEE, 2003.
- [8] R. Hrbacek and V. Dvorak. Bent function synthesis by means of cartesian genetic programming. In T. Bartz-Beielstein, J. Branke, B. Filipič, and J. Smith, editors, *Parallel Problem Solving from Nature – PPSN XIII*, pages 414–423, Cham, 2014. Springer International Publishing.
- [9] J. Husa and R. Dobai. Designing bent boolean functions with parallelized linear genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '17, page 1825–1832, New York, NY, USA, 2017. Association for Computing Machinery.
- [10] W. R. Inc. Mathematica, Version 14.3. Champaign, IL, 2025.
- [11] 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*, 53(5):1743–1751, 2007.
- [12] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. Elsevier, Amsterdam, North Holland, 1977. ISBN: 978-0-444-85193-2.
- [13] L. Manzoni, L. Mariot, and E. Tuba. Balanced crossover operators in genetic algorithms. *Swarm Evol. Comput.*, 54:100646, 2020.
- [14] L. Mariot, D. Jakobovic, A. Leporati, and S. Picek. Hyper-bent boolean functions and evolutionary algorithms. In L. Sekanina, T. Hu, N. Lourenço, H. Richter, and P. García-Sánchez, editors, *Genetic Programming*, pages 262–277, Cham, 2019. Springer International Publishing.
- [15] L. Mariot and A. Leporati. A genetic algorithm for evolving plateaued cryptographic boolean functions. In A.-H. Dediu, L. Magdalena, and C. Martín-Vide, editors, *Theory and Practice of Natural Computing*, pages 33–45, Cham, 2015. Springer International Publishing.

- [16] L. Mariot and A. Leporati. Heuristic search by particle swarm optimization of boolean functions for cryptographic applications. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO Companion ’15, page 1425–1426, New York, NY, USA, 2015. Association for Computing Machinery.
- [17] L. Mariot, M. Saletta, A. Leporati, and L. Manzoni. Heuristic search of (semi-)bent functions based on cellular automata. *Nat. Comput.*, 21(3):377–391, 2022.
- [18] 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. <https://eprint.iacr.org/2013/011>.
- [19] Q. Meng, H. Zhang, M. Yang, and Z. Wang. Analysis of affinely equivalent Boolean functions. *Science in China Series F: Information Sciences*, 50(3):299–306, 2007.
- [20] W. Millan, A. Clark, and E. Dawson. An effective genetic algorithm for finding highly nonlinear boolean functions. In Y. Han, T. Okamoto, and S. Qing, editors, *Information and Communications Security*, pages 149–158, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.
- [21] W. Millan, A. Clark, and E. Dawson. Heuristic design of cryptographically strong balanced boolean functions. In K. Nyberg, editor, *Advances in Cryptology — EUROCRYPT’98*, pages 489–499, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.
- [22] S. Picek, C. Carlet, S. Guilley, J. F. Miller, and D. Jakobovic. Evolutionary algorithms for boolean functions in diverse domains of cryptography. *Evolutionary Computation*, 24(4):667–694, 2016.
- [23] S. Picek and D. Jakobovic. Evolving algebraic constructions for designing bent boolean functions. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO ’16, page 781–788, New York, NY, USA, 2016. Association for Computing Machinery.
- [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*, GECCO ’13 Companion, page 191–192, New York, NY, USA, 2013. Association for Computing Machinery.
- [25] S. Picek, D. Jakobovic, J. F. Miller, E. Marchiori, and L. Batina. Evolutionary methods for the construction of cryptographic boolean functions. In P. Machado, M. I. Heywood, J. McDermott, M. Castelli, P. García-Sánchez, P. Burelli, S. Risi, and K. Sim, editors, *Genetic Programming*, pages 192–204, Cham, 2015. Springer International Publishing.

- [26] S. Picek, K. Knezevic, L. Mariot, D. Jakobovic, and A. Leporati. Evolving bent quaternary functions. In *2018 IEEE Congress on Evolutionary Computation, CEC 2018, Rio de Janeiro, Brazil, July 8-13, 2018*, pages 1–8. IEEE, 2018.
- [27] S. Picek, E. Marchiori, L. Batina, and D. Jakobovic. Combining evolutionary computation and algebraic constructions to find cryptography-relevant boolean functions. In T. Bartz-Beielstein, J. Branke, B. Filipič, and J. Smith, editors, *Parallel Problem Solving from Nature – PPSN XIII*, pages 822–831, Cham, 2014. Springer International Publishing.
- [28] R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming*. lulu.com, 2008.
- [29] A. A. Polujan and A. Pott. Cubic bent functions outside the completed maiorana-mcfarland class. *Designs, Codes and Cryptography*, 88(9):1701–1722, Feb. 2020.
- [30] C. Qu, J. Seberry, and J. Pieprzyk. On the symmetric property of homogeneous boolean functions. In J. Pieprzyk, R. Safavi-Naini, and J. Seberry, editors, *Information Security and Privacy*, pages 26–35, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.
- [31] J. Seberry, T. Xia, and J. Pieprzyk. Construction of cubic homogeneous boolean bent functions. *Australas. J Comb.*, 22:233–246, 2000.