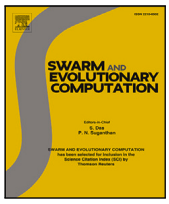
Swarm and Evolutionary Computation 101 (2026) 102287

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

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



A novel bio-inspired encoding for evolving cryptographic Boolean functions

Rocco Ascone [a,*], Giulia Bernardini [b], Luca Manzoni [a], Gloria Pietropolli [a,c]

[a] Department of Mathematics, Informatics and Geosciences, University of Trieste, Italy
[b] Department of Computer Science, University of Milan, Italy
[c] CWI, Amsterdam, The Netherlands

ARTICLE INFO

Dataset link: https://github.com/roccoasc1/evobrs

Keywords:
Reaction system
Evolutionary algorithm
Evolutionary reaction system
Bent function
Balanced function
Boolean function
Cryptographic function
Cryptographic Boolean function

ABSTRACT

Discovering Boolean functions that satisfy properties such as balancedness and nonlinearity is a complex optimization problem, which is crucial to important cryptographic constructions like block and stream ciphers. The difficulty of this problem lies in the search space growing super-exponentially in the number of variables. Evolutionary approaches, including Genetic Algorithms (GAs) and Genetic Programming (GP), have been successfully applied to overcome this difficulty. The major drawback of these methods is that they evolve functions through encodings that are either exponential in the input size or hard to interpret. We address this problem as follows. (i) We propose a new encoding for Boolean functions as reaction systems, a bio-inspired computational model which can be directly translated into the compact and easily interpretable Disjunctive Normal Form (DNF). (ii) We design EvoBRS, an evolutionary optimization framework that exploits this new representation to discover Boolean functions with maximum nonlinearity (bent functions), possibly under the balancedness constraint. (iii) We back up our novel paradigm with a refined theoretical analysis of independent interest. (iv) We conduct a rigorous experimental study, demonstrating that EvoBRS consistently discovers diverse, highly nonlinear Boolean functions with and without the balancedness constraint. EvoBRS proves particularly effective on balanced functions, successfully identifying balanced maximally nonlinear instances and outperforming both GP and state-of-the-art GAs. All the discovered functions are returned in a compact and easily interpretable DNF. A preliminary version of this work appeared in Ascone et al., GECCO 2025.

1. Introduction

Designing Boolean functions satisfying strong – and often conflicting – cryptographic properties like nonlinearity, balancedness, and correlation immunity is a challenging task [1,2]. Functions enjoying these properties serve as key components in block and stream ciphers to resist attacks and ensure secure encryption; their search is a complex combinatorial optimization problem, as the number of candidates increases super-exponentially in the number of variables. Finding new ways to generate cryptographic Boolean functions is thus an active research area, with applications spanning several domains [3–6]. Three main approaches are used in the literature: algebraic constructions [2, Section 6.1.15], random generation, and (meta-) heuristic methods [7]. Algebraic constructions have the merit of working for many different function sizes; however, they produce the same Boolean functions when using the same starting conditions, and finding new constructions can be far from trivial [7]. Random generation is impractical due to the prohibitive size of the search space [8]. Among metaheuristic strategies, Evolutionary Computation (EC) has demonstrated strong potential [9], the two main paradigms being Genetic Algorithms (GAs) [10] and Genetic Programming (GP) [11].

The major drawback of existing EC approaches is represented by the function encodings they rely upon: GAs use a bitstring encoding of the function's truth table, whose size is exponential in the number of variables, while GP methods use complex, hard-to-interpret trees of operators (see Fig. 1 for an example).

The main goal of this work is to address these limitations by proposing a novel encoding to evolve Boolean functions that is at the same time polynomial in size and easily interpretable. Our encoding consists of the direct translation of the Disjunctive Normal Form (DNF) of a Boolean function into a special type of Reaction Systems (RS) [12,13], a computational model inspired by biochemical reactions in living cells which offers an easy yet intuitive way of describing complex constructs. The main advantage of RS-based strategies is that they do not require the definition of a problem-specific set of functional or terminal symbols, and individuals can be directly read and understood by humans.

* Corresponding author.
E-mail address: rocco.ascone@phd.units.it (R. Ascone).

https://doi.org/10.1016/j.swevo.2026.102287
Received 5 September 2025; Received in revised form 8 January 2026; Accepted 11 January 2026
Available online 17 January 2026

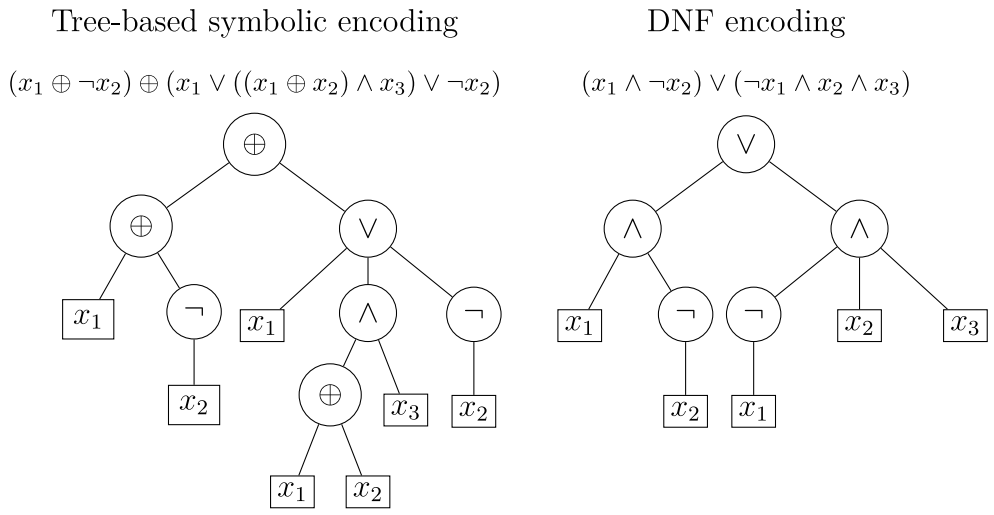


Fig. 1. Three different encodings of the 3-variable function $f(x) = (x_1 \oplus \neg x_2) \oplus (x_1 \vee ((x_1 \oplus x_2) \wedge x_3) \vee \neg x_2)$. In DNF, f can be written as $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2 \wedge x_3)$; 00011100 is the bitstring (of length 2^3) representing its truth table (for inputs in lexicographic order). The DNF of any function f can also be represented via a tree with the root labeled by $\vee$, and its sons always labeled by $\wedge$. Observe that the height of a DNF tree is always bounded by 3; in contrast, the height of a general tree encoding using more operators (like those used in GP methods) is unbounded, and can be particularly large due to phenomena like bloating [7]. The uniformity of DNF trees ensures human-readable solutions without limiting the number of possible Boolean functions they can represent.

The evolution of general RS has been first explored by Manzoni et al. [14], who proposed the paradigm of Evolutionary Reaction Systems (EvoRS) and reported promising results for several real-world binary classification problems. Inspired by this work, we introduce Evolutionary Boolean Reaction Systems (EvoBRS), an original evolutionary framework specifically designed to evolve individuals represented through the newly defined class of Boolean RS, that encode Boolean functions in DNF. To suitably design the key components of EvoBRS, we provide a novel and comprehensive investigation of the theoretical properties of the DNF representation of cryptographic Boolean functions, and use these insights to design the EvoBRS operators.

We apply our paradigm to two well-established problems related to the design of cryptographic Boolean functions: generating highly nonlinear balanced Boolean functions and bent Boolean functions, called the Bal and Bent problems, respectively. To validate our approach, we conducted an extensive set of experiments across multiple problem instances. We compare our results both with GP and with multiple GAs using improved evolutionary operators from a recent study [15]. Beyond performance measures like nonlinearity and balancedness, we analyze the structural properties of the evolved solutions and show that EvoBRS not only generates Boolean functions with strong cryptographic properties but also produces compact and diverse solutions, highlighting the benefits of RS-based representations.

The main goal of our work is to provide a compact, easy-to-interpret model (as well as to unveil nontrivial connections between the seemingly unrelated domains of RS and Boolean functions), rather than maximizing the success rate of our method. Nevertheless, we demonstrate that EvoBRS consistently outperforms state-of-the-art GAs, and in many cases even GP, also in terms of success rate, while offering a compact, polynomially bounded output encoding.

A preliminary version of the EvoBRS paradigm, limited to the Bent problem, was presented in the short paper [16]. Compared to [16], here we (i) extend the EvoBRS framework to solve the Bal problem, that has the additional requirement of balancedness; (ii) provide a novel and comprehensive investigation of theoretical properties of the DNF representation, unveiling fundamental constraints to ease the search, and develop ad-hoc operators; (iii) significantly improve over previous results obtained in [16] for the Bent problem, and achieve known maximum values of nonlinearity for all the Bal instances (Section 7.1); (iv) investigate the diversity and the interpretability of the obtained solutions (Section 7.2); and (v) provide a user-friendly C++ implementation of the algorithm, which allows a faster-execution and a larger fitness evaluation budget.

1.1. Related work

We begin by reviewing the main EC paradigms employed to generate Boolean functions: for a more comprehensive analysis, we refer the reader to the survey [7]. Various representations for Boolean functions have been proposed in the literature, and, building upon them, different metaheuristic approaches have been developed to explore the corresponding search spaces.

Bitstring encoding is the most straightforward and common approach, where a candidate Boolean function over n variables is encoded as a string of bits of length 2^n, corresponding directly to its truth table. This encoding is particularly suitable for many metaheuristics, especially GAs, which can operate directly on bitstrings. The first effort in this direction employed a GAs in 1997 [17], followed by progressively advanced GAs [18–23]. Other EC paradigms, including Particle Swarm Optimization [24], have also been explored in this context [25] relying on bitstring encoding. However, direct mappings to truth tables suffer from the curse of dimensionality as the number of variables increases, making this representation hardly scalable for large n.

Symbolic representations partially mitigate the scalability issues associated with bitstring encoding, although at the cost of reduced interpretability. Evolutionary Algorithms (EA) that employ symbolic representations, such as GP, are commonly applied in this context. A GP individual representing a Boolean function typically takes the form of a syntax tree, where terminal nodes (leaves) correspond to the n Boolean variables, and internal nodes represent binary operators such as AND, OR, NOR, XOR, XNOR, NAND, or other more complex operators. GP and its variants, including Cartesian GP [26] and

linear GP [27], have shown good performances in evolving Boolean functions [28–35]. Another line of research has focused on evolving secondary Boolean constructions rather than designing entirely novel ones [36–40]. Nonetheless, it remains uncertain whether these evolved constructions are genuinely novel, as metaheuristics often produce numerous candidate solutions requiring exhaustive validation to assess their novelty [7]. Additionally, the size of these constructions is often unbounded, leading to non-interpretable results and a considerable bloat in many cases [7,38].

Other representations of Boolean functions proposed in the literature include integer encoding [41] and floating-point encoding [42], though these are rarely used and suffer from similar scalability problems as bitstring encoding. Fig. 1 shows an example of a Boolean function represented through bitstring and symbolic encoding, together with an example of the DNF encoding proposed in this work.

2. Boolean functions

This section covers the preliminary definitions and results related to Boolean functions used throughout the paper. The treatment is, of course, far from being exhaustive. The interested reader is referred to Carlet's book [2] for an in-depth discussion of Boolean functions and their properties relevant to cryptography.

Let $\mathbb{F}_2 = \{0,1\}$ denote the finite field with two elements, where addition corresponds to the XOR operation of two bits $a, b \in \mathbb{F}_2$ (denoted as $a \oplus b$) and multiplication corresponds to the logical AND (denoted by the concatenation ab). For any $n \in \mathbb{N}$, the set of all n-tuples of bits is denoted by $\mathbb{F}_2^n$. The scalar product of two vectors $x, y \in \mathbb{F}_2^n$ is defined as $x \cdot y = \bigoplus_{i=1}^{n} x_i y_i$.

A *Boolean function* of n variables is a mapping $f : \mathbb{F}_2^n \to \mathbb{F}_2$. A natural way to uniquely represent such a function is by its *truth table*, which is the 2^n-bit vector Ω_f that specifies, for each possible input $x \in \mathbb{F}_2^n$ in lexicographic order, the corresponding output value $f(x)$. The Hamming weight $w_H(f)$ of f is defined as the number of ones in Ω_f.

Another unique representation of a Boolean function is the *Walsh transform*, which measures the correlations between a Boolean function f and linear functions, i.e. those that are defined as an XOR of a subset of the input variables. Formally, given $f : \mathbb{F}_2^n \to \mathbb{F}_2$, the Walsh transform is the mapping $W_f : \mathbb{F}_2^n \to \mathbb{Z}$ defined as:

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

for all $a \in \mathbb{F}_2^n$. The coefficient $W_f(a)$ determines the correlation between f and the linear function $a \cdot x$.

A function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is called *balanced* if $w_H(f) = 2^{n-1}$, meaning that its truth table contains an equal number of zeros and ones. Balancedness is a fundamental cryptographic property of Boolean functions used in the combiner and filter model of stream ciphers: unbalanced functions present a statistical bias in their output that can be exploited in distinguishing attacks [2].

Another critical property of Boolean functions in cryptography is their *nonlinearity*, which measures the Hamming distance of the function from the set of all linear functions. The nonlinearity of a Boolean function can be expressed in terms of the Walsh transform:

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

The nonlinearity of any Boolean function with n inputs satisfies:

$$nl(f) \leq 2^{n-1} - 2^{\frac{n}{2}-1} \ . \tag{3}$$

The above inequality is also called the *covering radius bound*.

A Boolean function can be considered highly nonlinear if its nonlinearity is close to the covering radius bound in its class. Functions with high nonlinearity are crucial for resisting fast correlation attacks in stream ciphers [43]. Moreover, functions with higher correlation to linear ones are more susceptible to affine approximation attacks. In particular, the functions whose nonlinearity equals the maximal value $2^{n-1} - 2^{\frac{n}{2}-1}$ are called *bent*. Bent functions exist only for even values of n (since the Walsh transform only gives integer values), and moreover they are not balanced (since $W_f(\underline{0}) = \pm 2^{\frac{n}{2}} \neq 0$, called bent weight). Consequently, bent functions, being unbalanced, cannot be directly used in the design of stream ciphers, as we have remarked above.

Table 1
Best known nonlinearities for balanced Boolean functions up to $n = 10$ variables, along with the search space sizes. For even number of variables, the nonlinearity of bent functions is reported within parentheses.

n	6	7	8	9	10
nl	26 (28)	56	116 (120)	240	492 (496)
$\#\mathcal{B}_n$	$1.8 \cdot 10^{19}$	$3.4 \cdot 10^{38}$	$1.2 \cdot 10^{77}$	$1.3 \cdot 10^{154}$	$1.8 \cdot 10^{308}$

When taking balancedness into account, the question of the maximum nonlinearity achievable by a Boolean function becomes more complicated. The optimal value is known only up to $n = 7$ variables, but beyond that it is still an open question. The problem is further exacerbated by the fact that the number of Boolean functions grows super-exponentially in n. Therefore, it is unfeasible to perform an exhaustive search of all Boolean functions of 6 or more variables.

Table 1 summarizes the best known nonlinearity values for balanced functions, from $n = 6$ up to $n = 10$ variables, along with the search space sizes. For even n, the table reports within parentheses the nonlinearity corresponding to the bent case. These values are drawn from standard references in the literature [2].

Now, we formally define the combinatorial optimization problems investigated in this work:

Problem 1 (Bal). Given $n \in \mathbb{N}$, find a n-variable Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ such that f is balanced and has maximum nonlinearity.

Problem 2 (Bent). Let $n \in \mathbb{N}$ be even. Find a n-variable Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ such that f has maximum nonlinearity, i.e. f is bent.

Disjunctive normal form

We can represent Boolean functions via logical formulas, which combine input variables through the logical operators AND, OR, and NOT. A *literal* refers to either an input variable or its negation, while a *clause* is a conjunction (AND) of literals, with its *size* defined as the number of literals it contains. DNF is a particular representation of a logical formula structured as a disjunction (OR) of clauses. When working with the DNF representation, we use the following notation: $\wedge$ (AND), $\vee$ (OR) and $\neg$ (NOT). Any Boolean function can be represented in DNF, unfortunately not in a unique way, i.e., different logical formulas can give rise to the same Boolean functions. Nevertheless a logical formula in DNF can be easily converted in a Boolean circuit [44].

3. Reaction systems

This section introduces preliminary definitions related to Reaction Systems (RS) and explains how they can be used to represent Boolean functions in DNF form, which is the focus of their use in this work. For a more comprehensive introduction to RS, we refer the interested reader to the survey by Brijder et al. [45].

RS were introduced by Rozenberg and Ehrenfeucht [12,13] in 2004 as a formal model inspired by chemical reactions, and inspired a great deal of research ever since, see e.g. [46–54]. The model represents chemical substances with a finite set of *entities* and the interactions among them happening within a cell as a finite set of rules called *reactions*.

A *reaction* over a finite set S of *entities* is a triple (R_a, I_a, P_a) of subsets of S such that $R_a \cap I_a = \emptyset$, where R_a is the set of *reactants*, I_a the set of *inhibitors*, and P_a the nonempty set of *products*: it models a chemical reaction that transforms the set of chemicals R_a into a new set P_a, and can only take place if none of the substances from I_a is present. The *size* of a reaction is defined as $|R_a \cup I_a|$. Just like in the original definition [12], in this paper we allow both the set of reactants and the set of inhibitors to be empty. The set of all possible reactions over S is denoted by $\mathrm{rac}(S)$. A *Reaction System* (RS) $\mathcal{A} = (S, A)$ consists of the finite set of entities S, called the *background set*, and a set $A \subseteq \mathrm{rac}(S)$ of reactions over S: see Example 1.

Any subset of S is a *state* of the RS. A reaction a is *enabled* in a state T if $R_a \subseteq T$ (all reactants are present in the state) and $I_a \cap T = \emptyset$ (no inhibitors are present). The set of all the reactions of $\mathcal{A}$ enabled in T is denoted by $\mathrm{en}_{\mathcal{A}}(T)$. The *result function* $\mathrm{res}_a : 2^S \to 2^S$ of a reaction a is defined as:

$$\mathrm{res}_a(T) := \begin{cases} P_a & \text{if } a \text{ is enabled in } T \\ \emptyset & \text{otherwise.} \end{cases}$$

The definition of result function extends naturally to sets of reactions: given any $T \subseteq S$ and $A \subseteq \mathrm{rac}(S)$, we define $\mathrm{res}_A(T) := \bigcup_{a \in A} \mathrm{res}_a(T)$. Consistently, the result function $\mathrm{res}_{\mathcal{A}}$ of the whole RS $\mathcal{A} = (S, A)$ is defined to be equal to res_A, i.e. the result function of the whole set of reactions of the RS. In this way, any RS $\mathcal{A} = (S, A)$ induces a discrete dynamical system with state set 2^S and next state function $\mathrm{res}_{\mathcal{A}}$.

Example 1. Consider the RS $\mathcal{A} = (S, A)$ consisting of the background set of three entities $S = \{x_1, x_2, x_3\}$ and the set of reactions $A = \{a, b, c\}$ with $a = (\{x_1, x_2\}, \emptyset, \{x_3\})$, $b = (\{x_2, x_3\}, \{x_1\}, \{x_1\})$, and $c = (\{x_2\}, \{x_3\}, \{x_1\})$. Reaction a models the transformation of two chemicals x_1, x_2 into a third substance x_3 which happens whenever x_1 and x_2 are present at the same time (there are no inhibitors); b transforms x_2 and x_3 into x_1, but it can only happen if x_1 is not already present, because it is an inhibitor for the reaction; and analogously, the reaction c that transforms x_2 into x_1 can only happen if x_3 is not present in the same state.

The set of states of $\mathcal{A}$ is given by $2^S = \{\emptyset, \{x_1\}, \{x_2\}, \{x_3\}, \{x_1, x_2\}, \{x_1, x_3\}, \{x_2, x_3\}, \{x_1, x_2, x_3\}\}$. For instance, the set of reactions enabled in $T = \{x_1, x_2\}$ is $\mathrm{en}_{\mathcal{A}}(T) = \{a, c\}$ (b is not enabled because x_3 is in R_b but not in T); and the state $T' = \{x_1, x_2, x_3\}$ enables only reaction a, because although the reactant sets of both b and c are contained in T', their inhibitors are present as well. The result function of T is thus given by the union of the products of the enabled reactions a and c: $\mathrm{res}_{\mathcal{A}}(T) = \{x_3\} \cup \{x_1\} = \{x_1, x_3\}$.

3.1. Boolean reaction systems

Reaction systems can be used to represent Boolean functions in DNF form [55]. A RS $\mathcal{A}$ univocally represents a function f in DNF if it is of the following form. The background set of $\mathcal{A}$ consists of the set of literals $X = \{x_1, \ldots, x_n\}$ appearing in the DNF plus an extra entity denoted 'True'. Each reaction univocally corresponds to a clause C of the DNF and is of the form $(R, I, \{\mathrm{True}\})$ with $R, I \subseteq X$, i.e. the product set always consists of the single entity True, which in contrast, is never part of the reactants nor the inhibitors. The set of reactants R contains all and only the literals that appear non-negated in C; the set of inhibitors I contains all and only the negated literals of C. We call reaction systems of this form Boolean Reaction Systems (BRS).

The states of a BRS not including the entity True are into a one-to-one correspondence with all possible assignments to the literals in X: in particular, a state $T \subseteq X$ uniquely encodes the assignment such that $x_i = 1$ if and only if $x_i \in T$, for all $i = 1, \ldots, n$. This encoding implies that a reaction is enabled by a state T if and only if the corresponding clause is satisfied by the assignment encoded with T: see also Example 2. Thus, the production of the entity True corresponds to the output of the Boolean function being 1: in other words, f is 1 for an assignment encoded with a state T iff $\mathrm{True} \in \mathrm{res}_{\mathcal{A}}(T)$.

Note that the encoding described above is unique and, as such, the reduction can be reversed so that a Boolean function in DNF can univocally be reconstructed from any BRS. In particular, the requirement $R \cap I = \emptyset$ in reaction systems corresponds to requiring that no clause of the DNF contains $x_i \wedge \neg x_i$ for any $i = 1, \ldots, n$. We will call *balanced* any BRS representing a balanced function; and similarly, a BRS is *bent* if it encodes a bent function.

Example 2. Consider the Boolean function $f : \mathbb{F}_2^3 \to \mathbb{F}_2$ in DNF:

$$f(x) = C_1(x) \vee C_2(x) \vee C_3(x) = (x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (x_2 \wedge \neg x_1 \wedge \neg x_3) \vee (x_2 \wedge x_3 \wedge \neg x_1).$$

Function f is univocally encoded by the BRS $\mathcal{A} = (S, \{r_1, r_2, r_3\})$, where $S = \{x_1, x_2, x_3, \mathrm{True}\}$ and the reactions are

$$r_1 = (\{x_1\}, \{x_2, x_3\}, \{\mathrm{True}\})$$
$$r_2 = (\{x_2\}, \{x_1, x_3\}, \{\mathrm{True}\})$$
$$r_3 = (\{x_2, x_3\}, \{x_1\}, \{\mathrm{True}\}).$$

In this way, any state $T \subseteq \{x_1, x_2, x_3\}$ encodes an input for f where $x_i = 1$ iff $x_i \in T$, e.g., the state $T = \{x_1, x_2\}$ corresponds to the assignment $x_1 = 1, x_2 = 1, x_3 = 0$. The symbol True represents that the output of f is 1. For instance, the reaction $r_1 = (\{x_1\}, \{x_2, x_3\}, \{\mathrm{True}\})$ is not enabled by the state $T = \{x_1, x_2\}$, since $C_1 = (x_1 \wedge \neg x_2 \wedge \neg x_3)$ is not satisfied by the assignment $x_1 = 1, x_2 = 1, x_3 = 0$. In general, each reaction r_i corresponds to the clause C_i, and r_i is enabled by a state T iff C_i is satisfied by the assignment associated with T: in particular, f is 1 with input encoded by state T iff $\mathrm{True} \in \mathrm{res}_{\mathcal{A}}(T)$.

4. Theoretical results

In this section, we present new results regarding the DNF representation of Boolean functions with maximal nonlinearity. These theoretical findings are reported because they form the foundation for the design of the evolutionary algorithm used in our approach, which we detail in the next section.

We start by recalling a useful identity between the Walsh transform and the Hamming weight of a Boolean function. Let $\ell_a(x) = a \cdot x$, for any $a \in \mathbb{F}_2^n$, the following holds

$$W_f(a) = 2^n - 2w_H(f \oplus \ell_a). \tag{4}$$

Eq. (4) tells us that the correlation between f and the linear function ℓ_a can also be expressed in terms of the Hamming distance between their truth tables. Eq. (4) implies in particular the following.

Remark 3. A Boolean function f is balanced if and only if $W_f(\underline{0}) = 0$, where $\underline{0}$ is the zero vector of $\mathbb{F}_2^n$.

Furthermore, the *spectral radius* $W_{max}(f)$ of f is the maximum absolute value assumed by the Walsh transform, i.e. $\max_{a \in \mathbb{F}_2^n} |W_f(a)|$. Thus we can express the nonlinearity as $nl(f) = 2^{n-1} - \frac{1}{2} W_{max}(f)$.

Let $C_{k,i}(x) = x_1 \wedge \cdots \wedge x_i \wedge \neg x_{i+1} \wedge \cdots \wedge \neg x_k$ be a conjunction of the first k variables of x where the first i variables are not negated and the rest are negated.

Remark 4. The following hold:

1. If $i < k$ and x is such that $x_k = 0$, then $C_{k,i}(x) = C_{k-1,i}(x)$.
2. If $i = k$ and x is such that $x_k = 1$, then $C_{k,i}(x) = C_{k-1,i-1}(x)$.

We can compute the Hamming weight of f as $w_H(f) = \sum_{x \in \mathbb{F}_2^n} f(x)$, where we are interpreting f as a function from $\mathbb{F}_2^n$ to $\mathbb{N}$, thus for any $1 \le k \le n$, we can split the summation as

$$w_H(f) = \sum_{x_k = 0} f|_{x_k = 0}(x) + \sum_{x_k = 1} f|_{x_k = 1}(x) \tag{5}$$

where $\sum_{x_k=1}$ (resp. $\sum_{x_k=0}$) denotes the sum over all $x \in \mathbb{F}_2^n$ s.t. $x_k = 1$ (resp. $x_k = 0$). From now on, we assume, like in Eq. (5), that each summation is over integer numbers, thus boolean inputs are identified with 0 and 1 in $\mathbb{Z}$. Given a vector $a \in \mathbb{F}_2^n$ and any $1 \le i \le n$, we define $p_i(a) = \oplus_{j=1}^{i} a_j$, i.e. $p_i(a)$ is the parity of the number of ones in the vector $(a_1, \dots, a_i) \in \mathbb{F}_2^i$.

The following lemma is at the heart of all the next proofs, where we prove that adding a clause to a given function imposes a restriction on the Walsh transform, i.e. a certain linear combination (with coefficients $+1$ and -1) of some values of the Walsh transform must always be equal to -2^n.

Lemma 5. *Given $\varphi : \mathbb{F}_2^n \to \mathbb{F}_2$ and any $1 \le k \le n$, $0 \le i \le k$, let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be defined as $f(x) = \varphi(x) \vee C_{k,i}(x)$. Then*

$$\sum_{a=(a_1,\dots,a_k,0,\dots,0)\in\mathbb{F}_2^n} (-1)^{p_i(a)} W_f(a) = -2^n. \tag{6}$$

Proof. We prove Eq. (6) by full induction on k, that is, we prove that the statement being true for some k and all n implies that it is true for $k+1$ and all n.

Base case: $k = 1$. We consider the case where $i = 0$, i.e. $f(x) = \varphi(x) \vee (\neg x_1)$. We first notice that

$$w_H(f) = \sum_{x_1=0} 1 + \sum_{x_1=1} \varphi(x) = 2^{n-1} + \sum_{x_1=1} \varphi(x). \tag{7}$$

Let $e_1 = (1, 0, \dots, 0) \in \mathbb{F}_2^n$. Notice that in this case, $\ell_{e_1}(x) = 1$ if and only if $x_1 = 1$, implying in particular that $f(x) \oplus \ell_{e_1}(x) = 1$ if either $x_1 = 0$, or $x_1 = 1$ and $\varphi(x) = 0$. Furthermore, note that for any two boolean values a and b, it holds that $a \vee b = a \oplus b \oplus ab$. We thus have

$$\begin{aligned} f(x) \oplus \ell_{e_1}(x) &= (\varphi(x) \vee (1 \oplus x_1)) \oplus x_1 \\ &= (\varphi(x) \oplus 1 \oplus x_1 \oplus (1 \oplus x_1)\varphi(x)) \oplus x_1 = 1 \oplus x_1\varphi(x). \end{aligned}$$

Thus, $w_H(f \oplus \ell_{e_1}) = \sum_{x_1=0} 1 + \sum_{x_1=1} (1 \oplus \varphi(x))$. Combining this with Eq. (7), we obtain that

$$\begin{aligned} w_H(f) + w_H(f \oplus \ell_{e_1}) &= 2 \cdot 2^{n-1} + \sum_{x_1=1} (1 \oplus \varphi(x)) + \sum_{x_1=1} \varphi(x) \\ &= 2 \cdot 2^{n-1} + 2^{n-1} = 3 \cdot 2^{n-1} \end{aligned}$$

since $\sum_{x_1=1} (1 \oplus \varphi(x))$ and $\sum_{x_1=1} \varphi(x)$ together sum up to 2^{n-1}. Finally, by applying Eq. (4) we get:

$$\begin{aligned} \sum_{a=(a_1,0,\dots,0)\in\mathbb{F}_2^n} (-1)^0 W_f(a) &= W_f(\underline{0}) + W_f(e_1) \\ &\overset{(Eq.\ (4))}{=} 2 \cdot 2^n - 2(w_H(f) + w_H(f \oplus \ell_{e_1})) \\ &= 2 \cdot 2^n - 2 \cdot 3 \cdot 2^{n-1} = -2^n. \end{aligned}$$

The case $i = 1$ follows similarly.

Inductive case: $k - 1 \Rightarrow k$. Consider the case $i < k$ (the case $i = k$ follows similarly). We define $A = \{a = (a_1, \dots, a_k, 0, \dots, 0) \in \mathbb{F}_2^n\}$ and we want to evaluate $\sum_{a\in A} (-1)^{p_i(a)} w_H(f \oplus \ell_a)$. By applying Eq. (5) and Remark 4. 1, and since $x_k = 1$ implies that $C_{k,i}(x) = 0$, we have that

$$\begin{aligned} w_H(f \oplus \ell_a) &= \sum_{x\in\mathbb{F}_2^n} f(x) \oplus a \cdot x \\ &\overset{(Eq.\ (5))}{=} \sum_{x_k=0} (\varphi(x) \vee C_{k,i}(x) \oplus a \cdot x) + \sum_{x_k=1} \varphi(x) \oplus a \cdot x \\ &\overset{(Rk.\ 4.1)}{=} \sum_{x_k=0} (\varphi(x) \vee C_{k-1,i}(x) \oplus a \cdot x) + \sum_{x_k=1} \varphi(x) \oplus a \cdot x. \end{aligned}$$

Given any $a \in A$, we denote $a^0 = (a_1, \dots, a_{k-1}, 0, \dots, 0) \in A$ and $a^1 = (a_1, \dots, a_{k-1}, 1, 0 \dots, 0) \in A$, and we split A in the two sets $A_0 = \{a^0 : a \in A\}$ and $A_1 = \{a^1 : a \in A\}$: note that $A_0 \cup A_1 = A$. Then the sum of $w_H(f \oplus \ell_a)$ over A_0 is:

$$\begin{aligned} &\sum_{a^0\in A_0} (-1)^{p_i(a)} w_H(f \oplus \ell_{a^0}) = \\ &= \sum_{a^0\in A_0} (-1)^{p_i(a)} \sum_{x_k=0} (\varphi(x) \vee C_{k-1,i}(x) \oplus a^0 \cdot x) + \\ &\quad + \sum_{a^0\in A_0} (-1)^{p_i(a)} \sum_{x_k=1} \varphi(x) \oplus a^0 \cdot x. \end{aligned}$$

Whereas the sum of $w_H(f \oplus \ell_a)$ over A_1 is:

$$\begin{aligned} &\sum_{a^1\in A_1} (-1)^{p_i(a)} w_H(f \oplus \ell_{a^1}) = \\ &= \sum_{a^1\in A_1} (-1)^{p_i(a)} \sum_{x_k=0} (\varphi(x) \vee C_{k-1,i}(x) \oplus a^1 \cdot x) + \\ &\quad + \sum_{a^1\in A_1} (-1)^{p_i(a)} \sum_{x_k=1} \varphi(x) \oplus a^1 \cdot x \\ &= \sum_{a^0\in A_0} (-1)^{p_i(a)} \sum_{x_k=0} (\varphi(x) \vee C_{k-1,i}(x) \oplus a^0 \cdot x) + \\ &\quad + \sum_{a^0\in A_0} (-1)^{p_i(a)} \sum_{x_k=1} \varphi(x) \oplus a^0 \cdot x \oplus 1 \end{aligned}$$

The last equality holds since when $x_k = 0$ then $a^0 \cdot x = a^1 \cdot x$, and when $x_k = 1$ then $a^1 \cdot x = a^0 \cdot x \oplus 1$. Since

$$\begin{aligned} &\sum_{x_k=1} ((\varphi(x) \oplus a^0 \cdot x \oplus 1) + (\varphi(x) \oplus a^0 \cdot x)) = \\ &= \sum_{x_k=1} \neg(\varphi(x) \oplus a^0 \cdot x) + (\varphi(x) \oplus a^0 \cdot x) = \sum_{x_k=1} 1 = 2^{n-1}, \end{aligned} \tag{8}$$

we obtain that

$$\begin{aligned} &\sum_{a\in A} (-1)^{p_i(a)} w_H(f \oplus \ell_a) = \\ &\overset{(Eq.\ (8))}{=} 2 \sum_{a^0\in A_0} (-1)^{p_i(a)} \sum_{x_k=0} (\varphi(x) \vee C_{k-1,i}(x) \oplus a^0 \cdot x) + \sum_{a^0\in A_0} (-1)^{p_i(a)} 2^{n-1}. \end{aligned} \tag{9}$$

Let $\pi_k : \mathbb{F}_2^n \to \mathbb{F}_2^{n-1}$, $\pi_k(x) = (x_1, \dots, x_{k-1}, x_{k+1}, \dots, x_n)$ and $\iota_k : \mathbb{F}_2^{n-1} \to \mathbb{F}_2^n$, $\iota_k(x) = (x_1, \dots, x_{k-1}, 0, x_k, \dots, x_{n-1})$. Remark that $\pi_k(\iota_k(x)) = x$ for any $x \in \mathbb{F}_2^{n-1}$. Consider the function $f' : \mathbb{F}_2^{n-1} \to \mathbb{F}_2$, $f'(x) = \varphi(\iota_k(x)) \vee C_{k-1,i}(\iota_k(x))$. Since the form of f' is the same form as f but with $C_{k-1,i}$ instead of $C_{k,i}$, the inductive hypothesis applies. Given any $b \in \mathbb{F}_2^{n-1}$, we have that

$$\begin{aligned} w_H(f' \oplus \ell_b) &= \sum_{x\in\mathbb{F}_2^{n-1}} f'(x) \oplus b \cdot x \\ &= \sum_{x\in\mathbb{F}_2^{n-1}} \varphi(\iota_k(x)) \vee C_{k-1,i}(\iota_k(x)) \oplus b \cdot x \\ &= \sum_{x\in\mathbb{F}_2^n : x_k=0} (\varphi(x) \vee C_{k-1,i}(x) \oplus \iota_k(b) \cdot x). \end{aligned}$$

Therefore, since $\iota_k(\pi_k(a^0)) = a^0$ for any $a^0 \in A_0$, we always have that

$$\sum_{x\in\mathbb{F}_2^n : x_k=0} (\varphi(x) \vee C_{k-1,i}(x) \oplus a^0 \cdot x) = w_H(f' \oplus \ell_{\pi_k(a^0)}). \tag{10}$$

Furthermore, since f' is over $n-1$ variables, for any $b \in \mathbb{F}_2^{n-1}$ Eq. (4) gives us

$$2w_H(f' \oplus \ell_b) = 2^{n-1} - W_{f'}(b). \tag{11}$$

By applying Eqs. (9), (10), and (11), we get:

$$\begin{aligned} &\sum_{a\in A} (-1)^{p_i(a)} w_H(f \oplus \ell_a) = \\ &\overset{(Eq.\ (9),(10))}{=} \sum_{a^0\in A_0} (-1)^{p_i(a)} 2w_H(f' \oplus \ell_{\pi_k(a^0)}) + \sum_{a^0\in A_0} (-1)^{p_i(a)} 2^{n-1} \\ &\overset{(Eq.\ (11))}{=} \sum_{a^0\in A_0} (-1)^{p_i(a)} (2^{n-1} - W_{f'}(\pi_k(a^0))) + \sum_{a^0\in A_0} (-1)^{p_i(a)} 2^{n-1} \end{aligned}$$

Table 2
Truth tables of the functions defined in Example 6.

x	φ	f	ℓ_{α_1}	ℓ_{α_2}	ℓ_{α_3}	ℓ_{α_4}
000	0	0	0	0	0	0
001	0	0	0	0	0	0
010	0	0	0	1	0	1
011	1	1	0	1	0	1
100	1	1	0	0	1	1
101	1	1	0	0	1	1
110	0	1	0	1	1	0
111	0	1	0	1	1	0

$$
\begin{aligned}
&= \sum_{a^0\in A_0}(-1)^{p_i(a)}2^n - \sum_{a^0\in A_0}(-1)^{p_i(a)}W_{f'}(\pi_k(a^0))\\
&= \sum_{a^0\in A_0}(-1)^{p_i(a)}2^n - \sum_{b\in\pi_k(A_0)}(-1)^{p_i(b)}W_{f'}(b)\\
&= \sum_{a^0\in A_0}(-1)^{p_i(a)}2^n - \sum_{b=(b_1,\dots,b_{k-1},0,\dots,0)\in\mathbb{F}_2^{n-1}}(-1)^{p_i(b)}W_{f'}(b)\\
&= 2^n\sum_{a^0\in A_0}(-1)^{p_i(a)} - (-2^{n-1}),
\end{aligned}
$$

where the last equality holds by the inductive hypothesis. Finally, we obtain Eq. (6):

$$
\begin{aligned}
&\sum_{a\in A}(-1)^{p_i(a)}W_f(a) =\\
&= \sum_{a\in A}(-1)^{p_i(a)}2^n - 2\sum_{a\in A}(-1)^{p_i(a)}w_H(f\oplus\ell_a)\\
&= 2^n\sum_{a\in A}(-1)^{p_i(a)} - 2\left(\sum_{a^0\in A_0}(-1)^{p_i(a)}2^n + 2^{n-1}\right)\\
&= 2^n\left(2\sum_{a^0\in A_0}(-1)^{p_i(a)}\right) - 2^n\left(2\sum_{a^0\in A_0}(-1)^{p_i(a)}\right) - 2^n\\
&= -2^n,
\end{aligned}
$$

where $\sum_{a\in A}(-1)^{p_i(a)} = 2\sum_{a_0\in A_0}(-1)^{p_i(a)}$ since $i<k$ and thus the value of a_k does not influence the result of $(-1)^{p_i(a)}$. □

Example 6 (*For Lemma 5*). Consider the Boolean function $\varphi:\mathbb{F}_2^3\to\mathbb{F}_2$, $\varphi(x) := (x_1\wedge\neg x_2)\vee(\neg x_1\wedge x_2\wedge x_3)$, and take $k=i=2$. Then $C_{2,2}(x) := x_1\wedge x_2$, thus $f(x) := \varphi(x)\vee C_{2,2}(x) = (x_1\wedge\neg x_2)\vee(\neg x_1\wedge x_2\wedge x_3)\vee(x_1\wedge x_2)$. The truth tables of φ and f are in Table 2.

The summation in the statement of Lemma 5 is taken over the four vectors $\alpha_1=(0,0,0)$, $\alpha_2=(0,1,0)$, $\alpha_3=(1,0,0)$, and $\alpha_4=(1,1,0)$, with $p_2(\alpha_1)=p_2(\alpha_4)=0$, and $p_2(\alpha_2)=p_2(\alpha_3)=1$. The truth tables of the linear functions ℓ_{α_1}, ℓ_{α_2}, ℓ_{α_3}, ℓ_{α_4}, used to compute the Walsh transforms, are shown in Table 2. It follows that $W_f(\alpha_1)=2^3-2w_H(f\oplus\ell_{\alpha_1})=-2$, $W_f(\alpha_2)=2$, $W_f(\alpha_3)=6$, and $W_f(\alpha_4)=2$. Putting all together, the summation of Lemma 5 gives correctly $(-1)^0W_f(\alpha_1)+(-1)^1W_f(\alpha_2)+(-1)^1W_f(\alpha_3)+(-1)^0W_f(\alpha_4)=-2-2-6+2=-8$.

Finally, it is easy to verify (by computing $W_f(\alpha)$ for the rest of $\alpha\in\mathbb{F}_2^3$) that the nonlinearity of f is $nl(f)=4-\frac{1}{2}W_{max}(f)=4-3=1$. Notice that this is consistent with the definition of nonlinearity of f as the minimum Hamming distance between f and any linear function: in this example, the minimum distance is between f and ℓ_{α_3}, which indeed differs from f only on $x=011$ (see Table 2).

We now focus on bent functions, where the number of variables n is even. The following results establish a lower bound on the size of the clauses in the DNF representation of a bent Boolean function, thus providing a necessary condition for any function to be bent.

Theorem 7. *Given any $\varphi:\mathbb{F}_2^n\to\mathbb{F}_2$, n even, and given $C(x)$ a conjunction of at most $k<\frac{n}{2}$ literals, the function $f(x)=C(x)\vee\varphi(x)$ is never bent.*

Table 3
Example of non-bent functions. The last column highlights the types of reactions that must be avoided to construct a BRS capable of representing a bent function.

	function type	associated reaction
$n\geq 6$	$(x_i\wedge x_j)\vee\varphi(x)$	$(\{x_i,x_j\},\emptyset,\{\text{True}\})$
$n\geq 6$	$(x_i\wedge\neg x_j)\vee\varphi(x)$	$(\{x_i\},\{x_j\},\{\text{True}\})$
$n\geq 6$	$(\neg x_i\wedge\neg x_j)\vee\varphi(x)$	$(\emptyset,\{x_i,x_j\},\{\text{True}\})$
$n\geq 8$	$(x_i\wedge x_j\wedge x_k)\vee\varphi(x)$	$(\{x_i,x_j,x_k\},\emptyset,\{\text{True}\})$
$n\geq 8$	$(x_i\wedge x_j\wedge\neg x_k)\vee\varphi(x)$	$(\{x_i,x_j\},\{x_k\},\{\text{True}\})$
$n\geq 8$	$(x_i\wedge\neg x_j\wedge\neg x_k)\vee\varphi(x)$	$(\{x_i\},\{x_j,x_k\},\{\text{True}\})$
$n\geq 8$	$(\neg x_i\wedge\neg x_j\wedge\neg x_k)\vee\varphi(x)$	$(\emptyset,\{x_i,x_j,x_k\},\{\text{True}\})$

Proof. Up to renaming the variables, we can assume w.l.o.g. that $C(x)=C_{k,i}(x)$ for some $0\leq i\leq k$. By Lemma 5 and the triangle inequality, we deduce that

$$\sum_{a=(a_1,\dots,a_k,0,\dots,0)\in\mathbb{F}_2^n}|W_f(a)|\geq 2^n.$$

Recall that $W_{max}(f)=\max_{a\in\mathbb{F}_2^n}|W_f(a)|$, therefore

$$2^n\leq\sum_{a=(a_1,\dots,a_k,0,\dots,0)\in\mathbb{F}_2^n}|W_f(a)|\leq 2^kW_{max}(f)$$

thus $W_{max}(f)\geq 2^{n-k}$. Therefore, by the definition of nonlinearity:

$$nl(f)=2^{n-1}-\frac{1}{2}W_{max}(f)\leq 2^{n-1}-2^{n-k-1}<2^{n-1}-2^{\frac{n}{2}-1}.\quad\square$$

Corollary 8 (*Necessary Condition for Bent Functions*). *In any DNF representation of a bent function $f:\mathbb{F}_2^n\to\mathbb{F}_2$, all clauses have size at least $\frac{n}{2}$.*

In light of the correspondence between BRS and the DNF representation of Boolean functions, Corollary 8 immediately gives a lower bound on the size of reactions required for a BRS to represent a bent function. To increase the likelihood of generating individuals that are bent or close to being bent, we thus impose that the size of reactions during the initialization phase is at least $\frac{n}{2}$. For instance, for $n\geq 6$, a function such as $(x_i\wedge x_j)\vee\varphi(x)$ cannot be bent for any i and j. The results for small values of n are summarized in Table 3.

In the next proposition, we demonstrate that the bound of Corollary 8 is sharp. Specifically, we show the existence of bent functions with clauses of size $\frac{n}{2}$ in their DNF representation.

Proposition 9. *For each even $n\in\mathbb{N}$, there exist bent functions $f:\mathbb{F}_2^n\to\mathbb{F}_2$ with at least a clause of size $\frac{n}{2}$ in their DNF representation.*

Proof. Given an even integer n, we define inductively $q_n(x)=q_{n-2}(x)\oplus x_{n-1}x_n$ where $q_2(x)=x_1x_2$. The function q_n is a quadratic bent function, thus also $f_n(x)=\neg q_n(x)=q_n(x)\oplus 1$ is a quadratic bent function: we prove by induction that f_n has a clause of size $\frac{n}{2}$ in its DNF representation. Remark that $f_n(x)=q_n(x)\oplus 1=q_{n-2}\oplus x_{n-1}x_n\oplus 1=f_{n-2}\oplus x_{n-1}x_n$ and $a\oplus b=(a\wedge\neg b)\vee(\neg a\wedge b)$.

Base case: $n=2$. Since $\neg q_2(x)=x_1x_2\oplus 1=\neg x_1\vee\neg x_2$, in the base case the statement holds.

Inductive case: $n-2\Rightarrow n$. By the definition of f_n, we have

$$
\begin{aligned}
f_n(x) &= f_{n-2}(x)\oplus x_{n-1}x_n\\
&= (f_{n-2}(x)\wedge\neg(x_{n-1}\wedge x_n))\vee(\neg f_{n-2}(x)\wedge x_{n-1}\wedge x_n)\\
&= (f_{n-2}(x)\wedge\neg x_{n-1})\vee(f_{n-2}(x)\wedge\neg x_n)\\
&\qquad\vee(\neg f_{n-2}(x)\wedge x_{n-1}\wedge x_n)\\
&= g_1(x)\vee g_2(x)\vee g_3(x).
\end{aligned}
$$

By the inductive hypothesis we know that f_{n-2} has a DNF representation $C_1\vee\cdots\vee C_m$ with at least a clause C_i of size $\frac{n-2}{2}$. Starting

from this DNF representation, we can build a DNF representation for f_n containing a clause of size $\frac{n}{2}$. Indeed, $g_1(x) = f_{n-2}(x) \wedge \neg x_{n-1} = (C_1 \wedge \neg x_{n-1}) \vee \cdots \vee (C_m \wedge \neg x_{n-1})$ is a DNF representation of g_1 with at least a clause of size $\frac{n-2}{2} + 1 = \frac{n}{2}$. The same argument applies to g_2. We can finally choose any DNF decomposition of g_3 to get a DNF representation of f_n where there exists a clause of size $\frac{n}{2}$. □

Finally, we show that a similar result as Theorem 7 applies to balanced Boolean functions with maximal nonlinearity, for all $n \leq 15$ (odd values of n included).

Example 10 (*Bent Function with Minimum Clause Size*). Consider $n = 4$: we apply the construction of the inductive case of the proof of Proposition 9 to obtain the DNF representation of a bent Boolean function over $\mathbb{F}_2^4$ with at least one clause of minimum size $\frac{n}{2} = 2$. From the base case, we have that $f_2(x) := \neg x_1 \vee \neg x_2$; we can thus obtain $f_4(x) := (\neg x_1 \wedge \neg x_3) \vee (\neg x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge \neg x_4) \vee (\neg x_2 \wedge \neg x_4) \vee (x_1 \wedge x_2 \wedge x_3 \wedge x_4)$. Notice that f_4 is bent by construction: in fact, it is immediate to verify that $|W_f(a)| = 4$ for all $a \in \mathbb{F}_2^4$, implying in particular that $W_{max}(f) = 4$ and thus the nonlinearity of f is $nl(f) = 2^3 - \frac{1}{2} W_{max}(f) = 6$, which is the maximum nonlinearity value for $n = 4$.

Theorem 11. *Given any $\varphi : \mathbb{F}_2^n \to \mathbb{F}_2$ and $n \leq 15$, let $C(x)$ be a conjunction of at most $k < \lceil \frac{n}{2} \rceil$ literals such that the function $f(x) = C(x) \vee \varphi(x)$ is balanced. Then, f does not have the maximal known nonlinearity value.*

Proof. Up to renaming the variables, we can assume w.l.o.g. that $C(x) = C_{k,i}(x)$ for some $0 \leq i \leq k$. Recall that by Remark 3 f is balanced if and only if $W_f(\underline{0}) = 0$. Using Lemma 5, we get that

$$2^n \leq \sum_{a=(a_1,\ldots,a_k,0,\ldots,0) \in \mathbb{F}_2^n} |W_f(a)| \leq (2^k - 1) W_{max}(f)$$

thus $W_{max}(f) \geq \frac{2^n}{2^k - 1}$. Therefore, since $k \leq \lceil \frac{n}{2} \rceil - 1$, we get

$$nl(f) = 2^{n-1} - \frac{1}{2} W_{max}(f) \leq 2^{n-1} - \frac{2^{n-1}}{2^k - 1} \leq 2^{n-1} - \frac{2^{n-1}}{2^{\lceil \frac{n}{2} \rceil - 1} - 1}.$$

To conclude, when we evaluate the right-hand side for $n \leq 15$, we always obtain a smaller value than the best-known nonlinearity value (see [2, Table 3.1]). □

Corollary 12 (*Necessary Condition for Balanced Functions with Maximum Nonlinearity*). *In any DNF representation of a maximally nonlinear balanced function $f : \mathbb{F}_2^n \to \mathbb{F}_2$, $n \leq 15$, all clauses have size at least $\lceil \frac{n}{2} \rceil$.*

Thus, similar to how we improved the generation process for bent functions, we can use these findings to refine the generation process for individuals in the Bal problem. For example, Corollary 12 implies that, for $n = 7$, no balanced BRS containing a reaction of size 3 can have maximal nonlinearity.

5. Evolutionary Boolean reaction systems

In this section, we propose a new algorithm, *Evolutionary Boolean Reaction Systems* (EvoBRS), specifically designed for the representation and evolution of Boolean functions through BRS.

Recall that a framework for evolving general RS was originally proposed in [14]; our method, however, is fundamentally different, as the individuals we evolve are constrained to be BRS (see Section 3.1). To maintain this constraint, we define specialized operators, as well as novel fitness measures which are evaluated on the Boolean functions represented by the BRS. Our algorithm tailors the evolution to the requirements of the Bent and Bal problems (see Section 2) and incorporates the theoretical results of Section 4. In the following subsections, we provide all the details of this new evolutionary paradigm.

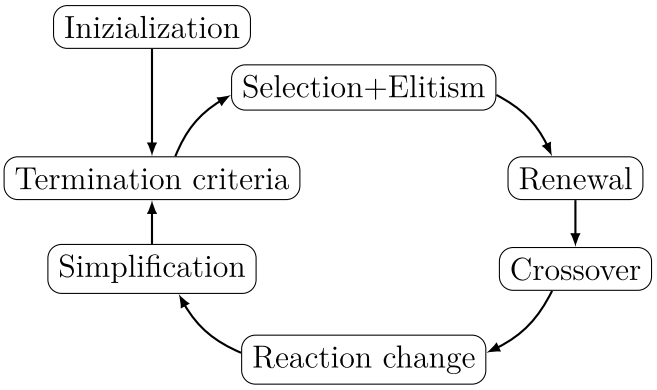


Fig. 2. The execution cycle of EvoBRS.

Initialization

To define the initial population for EvoBRS, each BRS individual requires both a background set and a set of reactions. For a problem of dimension n (i.e., to generate n-variable functions), the background set of each BRS is defined as $S = \{x_1, \ldots, x_n, \text{True}\}$. Each reaction is represented as a triplet of sets of the form $(R, I, \{\text{True}\})$. To initialize the reactions, we rely on the two parameters `init_size_min` and `init_size_max`, which set the lower and upper bounds for the number of reactions allowed in each BRS. Furthermore, we randomly generate reactions of size at least $\lceil \frac{n}{2} \rceil$: this additional constraint is motivated by Theorems 7 and 11.

Crossover

We define a new crossover operator, which is applied with probability p_c. Given two BRS, $\mathcal{A} = (S, A)$ and $\mathcal{B} = (S, B)$, let A_i (resp. B_i) be the set of all reactions of size i of $\mathcal{A}$ (resp. $\mathcal{B}$). For each possible size i, it generates an offspring of two BRS: $\mathcal{A}_i = (S, (A \setminus A_i) \cup B_i)$ and $\mathcal{B}_i = (S, (B \setminus B_i) \cup A_i)$, i.e. it swaps all the reactions of size i of $\mathcal{A}$ and $\mathcal{B}$. An example of this crossover (with $i = 4$) is shown in Fig. 3.

This crossover is particularly effective for the Bal problem, as balanced parents are more likely to produce balanced offspring when only blocks of reactions of the same size are exchanged, as we observed from preliminary tests. Moreover, since we initialize individuals with reactions of size at least $\lceil \frac{n}{2} \rceil$, the amount of possible offspring is bounded by n, and all offspring have reactions of size at least $\lceil \frac{n}{2} \rceil$. Thus, the offspring does not violate Theorem 7 or Theorem 11.

Mutations

We define two types of mutations that operate on BRS. The first, called *renewal*, replaces an existing BRS with a completely new one, generated in the same way as the individuals during the initialization process. Each individual has a probability p_{ren} of undergoing this mutation. Renewal is a strong mutation that introduces new genetic material in each cycle to increase diversity and help explore the search space, especially in the early stages of evolution. To balance this, the second mutation is less disruptive, changing fewer reactions to allow for a more gradual evolution.

The second mutation, called *random reaction change*, modifies an existing BRS $\mathcal{A} = (S, A)$ by replacing up to one third of its reactions with new reactions of the same size. The number of reactions to be replaced is chosen uniformly at random. Each individual has a probability p_m of being affected by this mutation. As with the crossover, this mutation ensures that the resulting BRS remains consistent with Theorems 11 and 7.

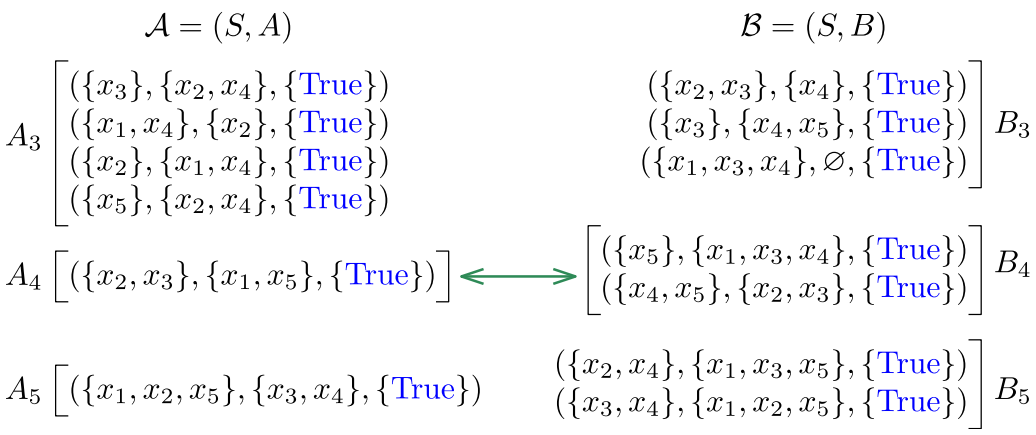


Fig. 3. Example of two balanced BRS $\mathcal{A}$ and $\mathcal{B}$ over $S = \{x_1, \ldots, x_5, \text{True}\}$. The reaction system $\mathcal{A}_4$ whose reactions are $A_3 \cup B_4 \cup A_5$, is balanced.

Simplification

This operator reduces the number of reactions in BRS by removing some reactions from each system. The simplification operator is always applied to the new individuals generated in the evolution cycle. We say that a reaction a *covers* a reaction b when all states T enabling b enable a as well (see [13]). In the context of BRS, it can be proven that $a = (R_a, I_a, \{\text{True}\})$ covers $b = (R_b, I_b, \{\text{True}\})$ if and only if $R_a \subseteq R_b$ and $I_a \subseteq I_b$. Thus if a covers b, then the size of a is smaller than the size of b, and in particular, the products of b are always consistent with the products of a, implying that the presence of a in a RS makes b redundant. To eliminate this redundancy, the simplification operator transforms each BRS $\mathcal{A} = (S, A)$ into a new BRS $\mathcal{B} = (S, B)$, where $B \subseteq A$ is obtained from A by removing the *covering* reactions. Although it would perhaps seem more natural to remove the *covered* reactions instead of the covering ones, we notice that this choice would actually work against the evolutionary process. Instead, by removing covering reactions, we can release space for reactions of a bigger size, thus keeping more diversity in the reactions. For example, fixed $n = 6$ and a reaction $a = (\{x_1, x_2, x_3\}, \{x_4\}, \{\text{True}\})$ of size $k = 4$, a covers $(\{x_1, x_2, x_3, x_5\}, \{x_4\}, \{\text{True}\})$ or $(\{x_1, x_2, x_3\}, \{x_4, x_5\}, \{\text{True}\})$ (and many more). If we kept a instead, all of the covered reactions would never be used.

Fitness

We define two distinct fitness functions, tailored to the specific problem we are investigating. The fitness values are evaluated on the function represented by the BRS individuals. The goal for evolving bent BRS is to maximize nonlinearity. Thus, we define the fitness function as:

$$fit_{\text{bent}}(f) = nl(f) + \frac{2^n - \#maxvalues}{2^n}.$$

where $nl(f)$ represents the nonlinearity of the function f, and the term $1 - \frac{\#maxvalues}{2^n}$ represents the normalized number of instances in which the Walsh transform (see Eq. (1)) of f attains its largest value. A small number of instances attaining this maximum value makes it easier for the algorithm to progress to the next higher nonlinearity value.

For highly nonlinear balanced functions, we use a more complex fitness function [38]:

$$\begin{aligned} fit_{\text{bal}}(f) &= -unbal + \delta_{unbal}\left(nl(f) + \frac{2^n - \#maxvalues}{2^n}\right) \\ &= -unbal + \delta_{unbal}\, fit_{\text{bent}}(f), \end{aligned}$$

where *unbal* is the *unbalancedness*, defined as the smallest number of bits in the truth table that need to be flipped to make f balanced. The negative sign penalizes unbalancedness, ensuring that balancedness is prioritized during the optimization process. The delta function δ_{unbal} takes the value one when $unbal = 0$ (i.e. f is balanced) and zero otherwise. This mechanism ensures the fitness function initially focuses on creating a balanced population before shifting its focus to maximizing nonlinearity.

EvoBRS cycle

At the beginning, the initial population is generated. Then, the evolutionary cycle (illustrated in Fig. 2) begins. The process starts with selection, where elitism preserves the best individuals to maintain high-quality solutions in the population. After selection, the renewal mutation is applied. Next, parents are chosen via tournament selection for crossover, followed by the application of the random reaction change mutation. The cycle ends with simplification. Then, syntactic duplicates (BRS with identical reaction sets) are eliminated to prevent equivalent solutions from persisting in the population. At the end of each generation, we check whether the stopping criteria have been met, either reaching the maximum number of fitness evaluations or fitness achieving the maximal nonlinearity thresholds (reported in Table 1). If not, the process repeats.

6. Experimental methodology

In this section, we present the experimental settings for running the EvoBRS algorithm, as well as the algorithms we use for comparison. Our code is implemented in `C++` to ensure computational efficiency and is publicly available at https://github.com/roccoasc1/evobrs.

Algorithms selected for comparison

We test EvoBRS against the two most commonly EC paradigms to generate Boolean functions for cryptographic applications: GAs and GP. To assess the performance of EvoBRS against GAs, we use the best-performing algorithms identified in [15], where the authors conduct a comprehensive evaluation and comparison of different GAs for solving the Bal and Bent problems. Their study includes a detailed statistical analysis and introduces new balanced crossover operators that improve results for the Bal problem. Based on this work, we selected the three GAs OP, CB, and MoO, named after their crossover strategies: one-point, counter-based, and map-of-ones crossover, respectively. While OP employs a standard crossover commonly used in Boolean frameworks, CB and MoO use balanced crossovers designed to evolve balanced functions. For our comparison, we adopt the same framework used in [15], including the evolution cycle, evolutionary operators, fitness functions, and hyperparameters, which were already fine-tuned by the authors. The implementation is based on the `Java` code used in [15].

To compare our method with a symbolic encoding representation, we use a tree-based GP with the same parameter settings reported in recent studies [38,56], where it achieved very competitive results. Specifically, we use the following function set: `or`, `xor`, `and`, `and2`, `xnor`, `ifthenelse`, where `and2` behaves as the function `and` but with the second input negated, and `ifthenelse` takes three arguments and returns the second if the first one evaluates to true and

Table 4
Problem instances, initialization sizes and number of fitness evaluations.

Problem	Instance	`init_size_min`	`init_size_max`	Fit evaluations
Bal	$n = 6$	80	100	10^5
	$n = 7$	120	160	$5 \cdot 10^5$
	$n = 8$	240	320	$5 \cdot 10^6$
Bent	$n = 6$	80	100	10^5
	$n = 8$	240	320	10^7
	$n = 10$	320	480	$5 \cdot 10^7$

the third one otherwise. We use the following operators: one-point crossover, uniform mutation, and ramped half-and-half initialization. To ensure a fair comparison with EvoBRS, which operates with a maximum depth of 3, we impose a maximum tree depth equal to 3 (rather than 9, which is the value used in previous studies). For completeness, we also report GP results with a maximum depth of 9. The implementation of the GP baseline is based on the `DEAP` [57] library in `Python`.

Experimental settings

Table 4 summarizes the problems evaluated in our experiments. The second column specifies the number n of variables for which we have run experiments for each problem. The third and fourth columns provide `init_size_min` and `init_size_max`, which are the minimum and maximum sizes, respectively, of the individuals generated in the initial step of EvoBRS. These are the only parameters of the algorithms that vary with the input dimension.

In all experiments, we used a population size of $n_{\text{pop}} = 100$ individuals, and a tournament size of $n_t = 4$. We scaled the number of fitness evaluations, specified in the fifth column, based on the input n, since the search space grows superexponentially (see Table 1). For a fair comparison of different methods, we terminated all of them after the same number of fitness evaluations. A random reaction change mutation was applied to each individual with a probability of 0.8, a renewal mutation with a probability of 0.1, and crossover with a probability of 0.2 (see Section 5). Elitism preserved the best five individuals for the next generations. We refer the reader to standard EC literature for the rationale behind these parameter choices [58,59].

We conducted 30 independent runs for each problem dimension to obtain statistically significant results. For comparisons, we used the Wilcoxon rank-sum test to assess statistical significance, with the null hypothesis of equality.

7. Experimental results

This section presents the results of the experimental campaign. The first part (Section 7.1) focuses on performance, while the second (Section 7.2) addresses the diversity and interpretability of the solutions generated.

7.1. Performances and comparison with other algorithms

Figs. 4 and 5 summarize the results for the Bal and Bent problems, respectively. In both figures, instances of different problems are shown in separate rows. The left panels report the progression of the average nonlinearity of the best individuals over fitness evaluations, while the right panels show the final nonlinearity distribution, both based on 30 runs. The boxplot graph for the Bal 6 instance also shows p-values from pairwise statistical tests to highlight the significance of differences. The bold p-values (with a value of 0.046 and 10^{-15}, respectively) highlight the only cases in which EvoBRS significantly outperformed the opponents for this instance. P-values (all on the order of 10^{-11} or smaller) are omitted from the other graphs, as they consistently confirm that EvoBRS significantly outperforms the competitors.

Performances on the Bal problem. Unlike the competitors, our method reaches the known theoretical maximum nonlinearity for Bal (26, 54, and 116 for $n = 6$, 7, and 8, respectively) with all the considered dimensions. This occurs consistently (30 out of 30 runs) for $n = 6$, whereas in the other two cases, such consistency would likely be achieved with additional fitness evaluations. The average nonlinearities achieved by EvoBRS surpass those of all other compared methods. In particular, the other GAs can also reach the maximum nonlinearity for $n = 6$, while for $n = 7$ and $n = 8$ they all fail to do so (with one exception for $n = 7$, see Fig. 4). Across all these instances, EvoBRS and GP start from a lower initial nonlinearity compared to the GAs. The initialization strategy explains this: they generate balanced individuals, a straightforward task since they use the truth table representation (it suffices to create a string with equal numbers of 0 and 1 [15]) but more challenging with a BRS representation and, in general, with compact encodings and tree representations.

EvoBRS performs consistently better than GP when n is even, while for $n = 7$, GP always reaches the maximum nonlinearity across all runs: for this dimension, the difference in the performance of EvoBRS and GP is not statistically relevant, as highlighted by a p-value of 0.686. It is unclear why the performance of GP is much worse when n is even. A possible explanation is that in these cases, Bal functions are rarer and more isolated than with odd values of n; thus, the small structural changes introduced by generic crossovers or mutations used in GP tend to flip many outputs at once, severely degrading balancedness.

To assess whether increasing the maximum tree height mitigates this problem, we ran additional experiments with a height limit of 9. For Bal 6, using the same number of fitness evaluations as in the original setting, GP reached the maximum nonlinearity of 26 in only 2 out of 30 runs. We therefore increased the fitness evaluation budget from 10^5 to 10^6. Under this setting, GP reached a median nonlinearity of 24.83 (std. 0.83), achieving the maximum value of 26 in 10/30 runs, still well below EvoBRS. As shown in Fig. A.8, the resulting individuals are extremely complex and arguably uninterpretable. For Bal 8, using 10^5 fitness evaluations, GP obtained a median nonlinearity of 113.75 (std. 1.84) and an optimal run rate of 12/30, again lower than the 21/30 optimal runs achieved by EvoBRS.

Performances on the Bent problem. Focusing on the Bent problem, for $n = 6$ EvoBRS reaches the theoretical maximum nonlinearity of 28 in 21 out of 30 runs, a result never achieved by any competing GAs, but consistently obtained by GP. For $n = 8$ and $n = 10$, although EvoBRS achieves high nonlinearity, it never reaches the maximum bounds (120 and 496, respectively). However, the left panels of the plots show that the fitness of EvoBRS continues to improve throughout the evolution, suggesting that a higher fitness evaluation budget may yield even better results.

Across all instances, EvoBRS outperforms all the GAs baselines early on and maintains a consistent fitness growth, whereas the baselines tend to stagnate in local optima: see Fig. 5. In contrast, GP achieves the maximum nonlinearity for $n = 6$ and $n = 8$, while for $n = 10$, it converges prematurely to a local optimum, stabilizing at a nonlinearity equal to 480 in all runs, thus underperforming compared to EvoBRS with a statistically significant p-value of 0.0007. This is likely due to the height of the GP trees being limited to 3 in this experiment. In fact, further tests we run evolving trees of height 9 show that in this case, GP reached the maximum nonlinearity value for $n = 10$ in 20/30 runs, although the resulting solutions were no longer human-readable: see Fig. A.9 for an example.

Discussion and runtime. All in all, our experiments show that EvoBRS is particularly effective for the Bal problem, consistently reaching the maximum nonlinearity for all dimensions and outperforming both GP and the GAs baselines. Interestingly, its performance on Bal is even stronger than on the Bent problem, despite the smaller computational budget. The excellent performance of EvoBRS on Bal also aligns with the fact that its operators are specifically designed for this problem.

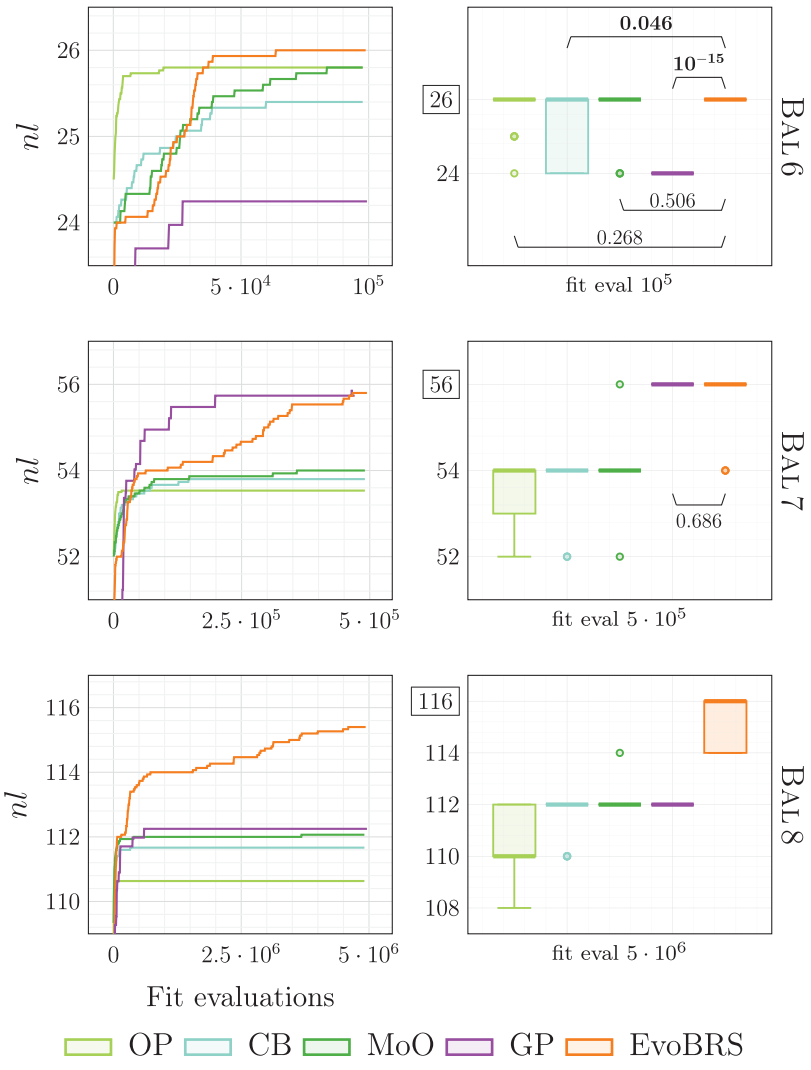


Fig. 4. Progression of the average nonlinearity achieved by the best individual in the population for an increasing number of fitness evaluations (left) and the final distribution (right) represented as a boxplot across 30 independent runs for the Bal problem for $n = 6, 7, 8$. We ran a pairwise Wilcoxon rank-sum test with equality as the null hypothesis between EvoBRS and each of the other methods, and decorated the boxplot with the resulting p-values. The p-values not present in the plots are of the order of 10^{-11} or smaller and were omitted to improve the figure's clarity. The maximum possible nonlinearity value for each of the instances is boxed: EvoBRS reached the maximum value in every single run on Bal6, in 27 out of 30 runs on Bal7, and in 21 out of 30 runs on Bal8.

Table 5
Mean runtime (± standard deviation) for a single cycle of EvoBRS, expressed in seconds. The means are computed over 30 runs for all problems and instances considered.

Time per cycle EvoBRS			
n	Bal	n	Bent
6	$(9.90 \pm 0.0006) \times 10^{-3}$ s	6	$(9.20 \pm 0.0003) \times 10^{-3}$ s
7	$(2.38 \pm 0.0001) \times 10^{-2}$ s	8	$(6.62 \pm 0.0022) \times 10^{-2}$ s
8	$(6.88 \pm 0.0015) \times 10^{-2}$ s	10	$(3.88 \pm 0.0054) \times 10^{-1}$ s

Although GP is often regarded as the most effective EC approach for discovering cryptographic functions, our results reveal some limitations: clear examples are the problems Bal 6 and Bal 8, where GP performs much worse than both EvoBRS and the other GAs. Understanding why GP underperforms on even-dimensional Bal remains an interesting avenue for future investigation.

To assess the computational efficiency of our algorithm, in Table 5 we report the mean time, expressed in seconds, required for a single cycle of EvoBRS for each problem and each dimension. We run each experiment independently on an Intel Xeon Gold 6140 CPU 2.30 GHz. We do not compare execution times with competing algorithms as the use of different programming languages (`C++` for EvoBRS, `Python` for GP, `Java` for GAs) prevents a fair comparison. As expected, the time required for a single cycle increases with n, reflecting the growth of the search space and the higher evaluation cost. Overall, runtime remains small, even for the largest instances (Bent 10), where a full cycle requires less than half a second.

7.2. Interpretability and diversity of the solutions

We now closely examine the functions generated by EvoBRS, focusing on interpretability and diversity, and compare them with those generated with other methods. Interpretability is crucial to understand and validate the solutions, while diversity ensures that different runs produce alternative functions [60], offering multiple equally valid solutions to the same problem.

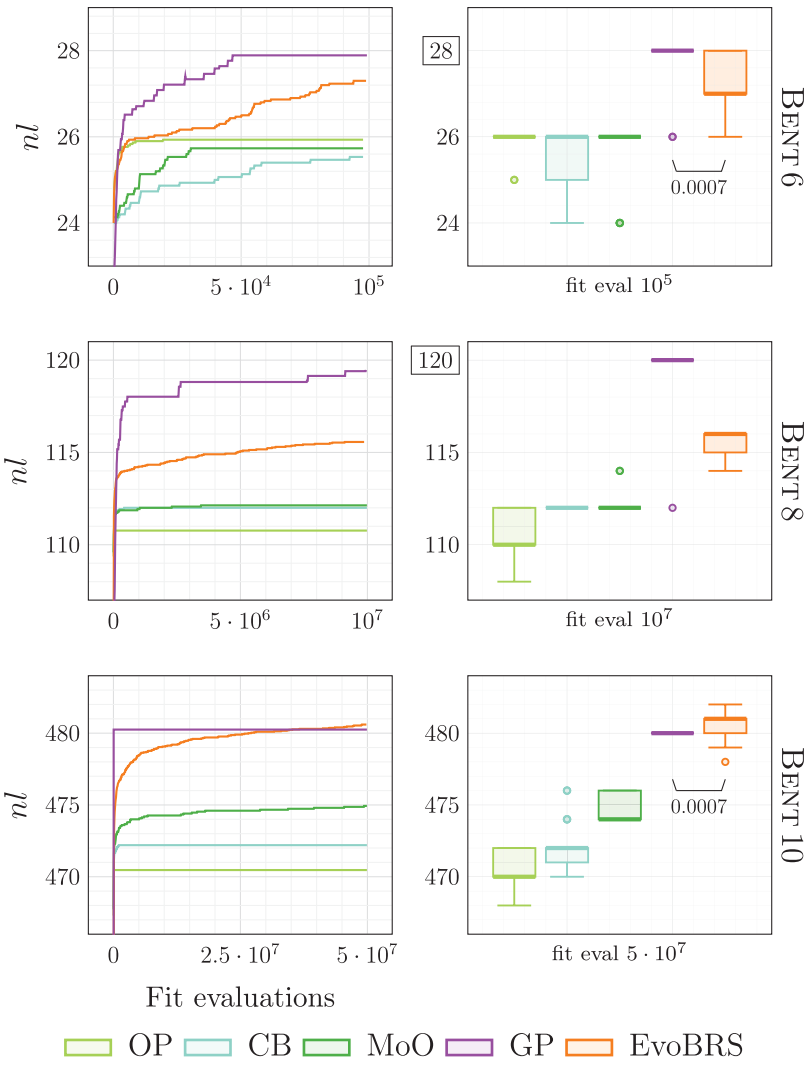


Fig. 5. Average nonlinearity of the best individual per iteration (left) and final distributions across 30 runs (right) for the Bent problem with $n = 6, 8, 10$. We ran a pairwise Wilcoxon rank-sum test with equality as the null hypothesis between EvoBRS and each of the other methods: the resulting p-values in all cases were on the order of 10^{-11}, thus indicating statistically significant differences in the performance of EvoBRS and all the other methods except GP. The maximum possible nonlinearity value for Bent6 is boxed: EvoBRS reached the maximum value in 12 out of 30 runs on this instance. The maximum value for $n = 10$ is not shown as it was never reached by any of the methods, including EvoBRS, which, however, outperformed all the other methods (including GP) for $n = 10$.

We assess interpretability in terms of structural simplicity and semantic transparency [61,62]. A Boolean function in our BRS representation is expressed as a disjunction of reactions, each corresponding to a set of input assignments for which the output is True. This clause-level representation allows a direct, human-readable mapping between the formula and the conditions under which it evaluates to True, without the need to traverse nested logical operators. For example, one evolved solution for Bal with $n = 6$, is

$(\{x_5, x_6\}, \{x_2, x_3\}, \{\text{True}\})$ $(\{x_4, x_5\}, \{x_1, x_6, x_3\}, \{\text{True}\})$
$(\varnothing, \{x_2, x_4, x_5, x_6\}, \{\text{True}\})$ $(\{x_2, x_4\}, \{x_1, x_3, x_5\}, \{\text{True}\})$
$(\{x_1, x_2\}, \{x_5, x_6\}, \{\text{True}\})$ $(\{x_1, x_6, x_3, x_4\}, \{x_2\}, \{\text{True}\})$
$(\{x_5, x_2, x_3\}, \{x_1\}, \{\text{True}\})$ $(\{x_1, x_2\}, \{x_6, x_3, x_4\}, \{\text{True}\})$
$(\{x_1, x_3, x_5\}, \{x_4\}, \{\text{True}\})$ $(\{x_5, x_1, x_3, x_4\}, \{x_6\}, \{\text{True}\})$
$(\{x_2, x_3\}, \{x_4, x_5\}, \{\text{True}\})$

This function contains only 11 reactions, each involving 4–5 variables. For instance, the clause $(\{x_5, x_6\}, \{x_2, x_3\}, \{\text{True}\})$ corresponds to all the assignments where $x_2 = x_3 = 0$, $x_5 = x_6 = 1$ and x_1 and x_4 could take any value. In this way, given any assignment of the variables, it is straightforward to check whether it satisfies any of the reactions above. For example, the assignment $x_1 = x_2 = 1$ and $x_3 = x_4 = x_5 = x_6 = 0$ satisfies both $(\{x_1, x_2\}, \{x_5, x_6\}, \{\text{True}\})$ and $(\{x_1, x_2\}, \{x_6, x_3, x_4\}, \{\text{True}\})$.

Table 6
Mean (± standard deviation) of the semantic distances for the best individuals generated by EvoBRS, across all problems and instances considered.

Semantic distances EvoBRS			
n	Bal	n	Bent
6	20.41 ± 2.14 bits	6	19.86 ± 2.37 bits
7	44.29 ± 2.43 bits	8	94.89 ± 3.10 bits
8	95.37 ± 2.74 bits	10	426.09 ± 4.65 bits

This kind of reasoning is not possible when the function is written using nested logic operators, which is usually the case with solutions obtained via GP (see Fig. 1). Other studies employing GP report that even after logic minimization, expressions for $n = 6$ remain too complex for manual interpretation [38]. To verify this, we examine the solutions obtained by GP for the same problem (Bal 6). As discussed in Section 7.1, we ran two experiments with different maximum tree heights: one aimed at maximizing performance (maximum height = 9) and one

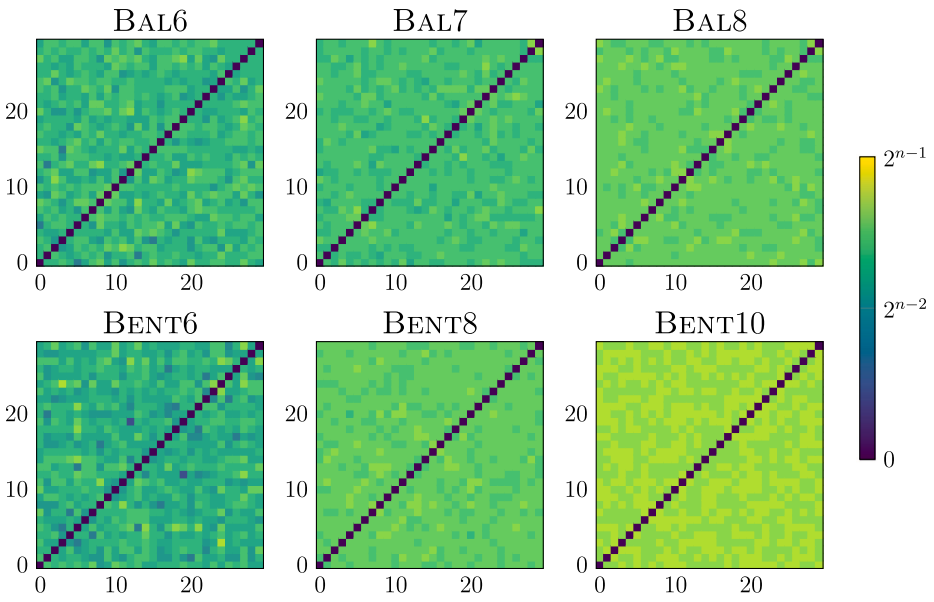


Fig. 6. Heatmaps showing the permutation distances between the best EvoBRS individuals from each experiment for the BAL problem (up) and the BENT problem (bottom). The lighter the color, the more diversity in the solutions.

aimed at obtaining human-readable solutions (maximum height = 3). An example of the former is shown in Fig. A.8: the resulting tree is extremely complex and can hardly be read or interpreted by a human. In contrast, an example of the latter is shown in Fig. A.7. This individual is indeed readable, as it is compact and structured. However, it does not achieve maximum nonlinearity (none of the individuals with height 3 does).

When using GAs, finding the minimal DNF representation from a truth table is an NP-hard problem [63,64]. Our approach circumvents this issue by directly generating compact and easily implementable solutions (thanks to the simplification operator), thus eliminating the need for post-processing. Indeed, the circuit complexity required to implement our Boolean functions physically remains low. Specifically, the circuit depth, defined as the longest path from input to output, is limited to 3, while the circuit size, measured as the number of gates, is at most $\mathcal{O}(n \cdot \texttt{init\_size\_max})$. For a detailed discussion on circuit complexity, we refer to [65].

Since discovering new constructions is a relevant task in cryptography [7], we also crucially investigate whether EvoBRS produces multiple alternative, and equally valid, Boolean functions, or if, on the contrary, it consistently converges to functions similar to each other. To do so, we measure semantic similarities between evolved functions using the Hamming distance. Recall that given two Boolean functions f and g, their Hamming distance $d_H(f,g)$ is $w_H(f \oplus g)$, i.e. the number of input vectors $x \in \mathbb{F}_2^n$ such that $f(x) \neq g(x)$ [2]. We first perform pairwise comparisons of all solutions in each experiment and verify that none of them are identical. To quantify the magnitude of these pairwise differences, we report the mean (± standard deviation) for both problems and all instances in Table 6. In all cases, the mean is close to 2^{n-1}, indicating that the functions differ by about half of their total length (which is 2^n).

While the Hamming distance captures semantic diversity, it does not account for the equivalence of functions under input variable permutations. To address this, we compute the *permutation distance* between each pair of generated functions f and g. The permutation distance is defined as $\min_{\sigma \in S_n} d_H(f, g \circ \sigma)$, where S_n is the group of permutations of n. Results are shown in Fig. 6. Each heatmap can be seen as the adjacency matrix of a weighted undirected graph with 30 nodes, each representing the best solution to one of the 30 instances. The weight on an edge between two nodes is the permutation distance between the two corresponding solutions: the distance is zero only if the BRS are semantically equivalent under any input permutation. These results confirm that none of the solutions generated by EvoBRS are identical and that they remain highly diverse.

8. Conclusions, limitations, and future directions

Conclusions. We presented EvoBRS, an evolutionary framework for the design of cryptographic Boolean functions that provides compact, human-readable solutions without requiring post-processing or logic minimization. We validated EvoBRS on two well-established benchmarks and compared its performance with that of the two evolutionary baselines most commonly used for this task, i.e., GAs and GP. Our experiments demonstrate that EvoBRS consistently evolves highly nonlinear balanced Boolean functions that are also compact and easy to interpret. Our experiments show that EvoBRS is consistently the best-performing method (among the tested ones) for generating highly nonlinear balanced Boolean functions. Not only do the balanced functions discovered by EvoBRS have high, often maximal, nonlinearity scores; their representation is also compact and easy to interpret. Interestingly, our tests have revealed that GP, which is commonly regarded as the best option for generating high-quality cryptographic Boolean functions, struggles to generate highly nonlinear balanced functions with an even number of variables. Although we provide some hypotheses for this behavior of GP, the underlying reason remains unclear and warrants further investigation.

We have also experimentally demonstrated that EvoBRS produces high-quality solutions to the BENT problem, significantly outperforming state-of-the-art GAs. For small dimensions ($n \leq 8$), GP returns even better functions than EvoBRS in terms of nonlinearity, representing them with shallow, human-readable trees of operators; however, when the problem dimension increases to $n = 10$, GP can outperform EvoBRS only at the price of completely losing readability and interpretability. EvoBRS remains the best-performing method when the tree height is limited to guarantee readable results.

Furthermore, our diversity analysis confirmed that EvoBRS consistently finds semantically distinct functions across different runs for all the considered problems and dimensions, even when accounting for input variable permutations. Considering the promising results, our work opens the door to multiple future developments: we next point out its main limitations.

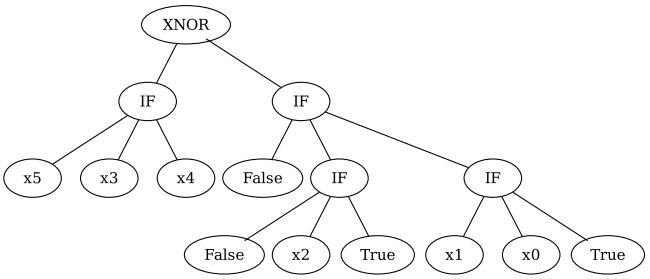


Fig. A.7. Individual generated by GP with maximum height equal to 3 for the BAL 6 problem. This individual reaches a sub-optimal nonlinearity of 24.

Limitations. The main limitation of our framework is that it is not entirely application-agnostic, and to perform at its best, it requires tailoring the crossover and mutation operators depending on the desired function properties. A hint of this issue emerges from our experimental analysis: EvoBRS really shines when it is applied to the BAL problem, for which its operators were originally designed, while it seems that there is room for improvement when it is used to attack the BENT problem.

Another limitation, which EvoBRS shares with GP, is that the function representation it relies on is not unique: there are, in general, multiple possible representations of the same function both in DNF (as well as a tree of operators). This complicates verifying that functions generated in different runs are actually distinct. Finally, both EvoBRS and GP begin the evolutionary process with rather low nonlinearity values: improving the initialization strategy would likely allow EvoBRS to converge faster.

Future work. The first and natural avenue to take is to extend the experimental analysis to higher dimensions, and to assess the performance of EvoBRS and the competing evolutionary methods using new metrics. Extending EvoBRS to work with adaptive mutation rates [66] and designing new mutation and crossover operators [67] optimized for the BENT problem is also a direction worth exploring, as well as rethinking the initialization process to increase the initial nonlinearity values of the individuals. Another interesting direction would be to extend RS-based representations to encode Boolean functions from $\mathbb{F}_2^n \to \mathbb{F}_2^m$. Given the good results presented in this paper for $m = 1$, this extension could provide a novel approach for designing Substitution-boxes (S-boxes), which are key components in modern symmetric-key ciphers [68].

Finally, it would be worth exploring practical applications of our work. In particular, EvoBRS could be applied as a first step in constructing a partially automated pipeline for the generation of new cryptographic systems. By evolving Boolean functions, it is possible to generate a collection of cryptographic primitives that are, in turn, combined in an automated way to create candidate cryptographic systems to be evaluated at later stages of the pipeline. While not fully automated, such a pipeline could be a useful tool for experts in designing new cryptographic systems, possibly increasing the speed at which future secure systems are developed. A concrete example of an advantageous application of EvoBRS is to discover Boolean functions that describe S-boxes, for instance, replacing the use of Cellular Automata (CA) in Keccak [69]. Indeed, unlike CA, the function representation of EvoBRS can encode functions with interactions between inputs that are arbitrarily far away: using EvoBRS in place of CA would thus reduce the need to iterate the update function multiple times to obtain such interactions.

CRediT authorship contribution statement

Rocco Ascone: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Giulia Bernardini:** Writing – review & editing, Writing – original draft, Supervision, Formal analysis. **Luca Manzoni:** Writing – review & editing, Supervision, Resources, Conceptualization. **Gloria Pietropolli:** Writing – review & editing, Writing – original draft, Validation, Supervision, Project administration, Investigation, Conceptualization.

Funding

This research is partially supported by the PRIN 2022 PNRR project "Cellular Automata Synthesis for Cryptography Applications (CASCA)" (P2022MPFRT) financed by the European Union – Next Generation EU.

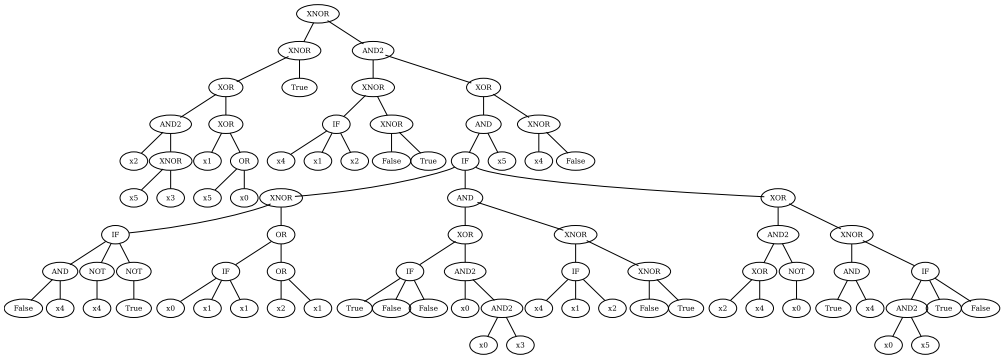


Fig. A.8. Individual generated by GP with maximum height equal to 9 for the BAL 6 problem. This individual reaches the maximum nonlinearity 26.

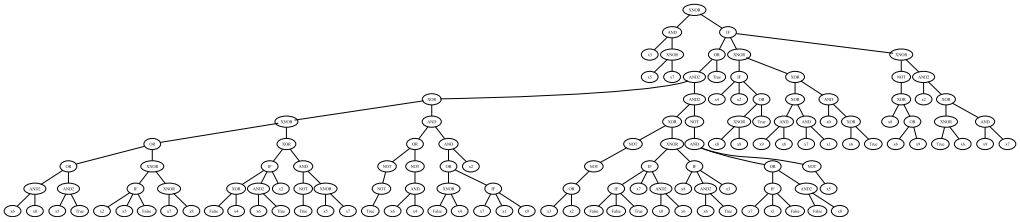


Fig. A.9. Individual generated by GP with maximum height equal to 9 for the BENT 10 problem. This individual reaches the maximum nonlinearity of 496.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix. GP trees

In this section, we present two individuals generated by GP to examine their interpretability and compare it with that of an EvoBRS individual, discussed in Section 7.2. As for EvoBRS, we report examples of individuals obtained for the BAL problem with $n = 6$. We consider two cases. Firstly, we report a solution when we limit the maximum tree height to 3 (Fig. A.7), which corresponds to the setup used for the comparison discussed in Section 7.1. As can be observed in Fig. 4, none of these solutions reaches maximum nonlinearity, so we report a solution with nonlinearity equal to 24. Then, in Fig. A.8, we show a tree solution obtained by limiting the maximum tree height to 9, which achieves the maximum nonlinearity of 26. Finally, under the same tree height limit, we exhibit (Fig. A.9) a solution for the BENT problem with $n = 10$ found by GP.

Data availability

The code is available at https://github.com/roccoasc1/evobrs.

References

[1] R. O'Donnell, Analysis of Boolean Functions, Cambridge University Press, 2014.

[2] C. Carlet, Boolean Functions for Cryptography and Coding Theory, Cambridge University Press, 2020, http://dx.doi.org/10.1017/9781108606806.

[3] S. Kavut, S. Maitra, M.D. Yücel, Search for boolean functions with excellent profiles in the rotation symmetric class, IEEE Trans. Inf. Theory 53 (5) (2007) 1743–1751, http://dx.doi.org/10.1109/TIT.2007.894696.

[4] Z. Zhang, L. Wu, A. Wang, Z. Mu, X. Zhang, A novel bit scalable leakage model based on genetic algorithm, Secur. Commun. Netw. 8 (18) (2015) 3896–3905, http://dx.doi.org/10.1002/SEC.1308.

[5] G.T. Becker, The gap between promise and reality: On the insecurity of XOR arbiter PUFs, in: Cryptographic Hardware and Embedded Systems (CHES), in: Lecture Notes in Computer Science, Vol. 9293, Springer, 2015, pp. 535–555, http://dx.doi.org/10.1007/978-3-662-48324-4_27.

[6] S. Saha, R.S. Chakraborty, S.S. Nuthakki, Anshul, D. Mukhopadhyay, Improved test pattern generation for hardware trojan detection using genetic algorithm and boolean satisfiability, in: Cryptographic Hardware and Embedded Systems (CHES), in: Lecture Notes in Computer Science, Vol. 9293, Springer, 2015, pp. 577–596, http://dx.doi.org/10.1007/978-3-662-48324-4_29.

[7] M. Djurasevic, D. Jakobovic, L. Mariot, S. Picek, A survey of metaheuristic algorithms for the design of cryptographic Boolean functions, Cryptogr. Commun. 15 (6) (2023) 1171–1197, http://dx.doi.org/10.1007/S12095-023-00662-2.

[8] W. Millan, J. Fuller, E. Dawson, New concepts in evolutionary search for boolean functions in cryptology, Comput. Intell. 20 (3) (2004) 463–474, http://dx.doi.org/10.1111/J.0824-7935.2004.00246.X.

[9] S. Picek, D. Jakobovic, M. Golub, Evolving cryptographically sound boolean functions, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), ACM, 2013, pp. 191–192, http://dx.doi.org/10.1145/2464576.2464671.

[10] O. Kramer, Genetic algorithms, in: Genetic Algorithm Essentials, Springer, 2017, pp. 11–19.

[11] W.B. Langdon, R. Poli, Foundations of Genetic Programming, Springer Science & Business Media, 2013.

[12] A. Ehrenfeucht, G. Rozenberg, Basic notions of reaction systems, in: Developments in Language Theory, 8th International Conference, DLT 2004, Auckland, New Zealand, December 13-17, 2004, Proceedings, in: Lecture Notes in Computer Science, Vol. 3340, Springer, 2004, pp. 27–29, http://dx.doi.org/10.1007/978-3-540-30550-7_3.

[13] A. Ehrenfeucht, G. Rozenberg, Reaction systems, Fund. Inform. 75 (1–4) (2007) 263–280, http://dx.doi.org/10.3233/FUN-2007-751-415.

[14] L. Manzoni, M. Castelli, L. Vanneschi, Evolutionary reaction systems, in: Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics - 10th European Conference, EvoBIO 2012, Málaga, Spain, April 11-13, 2012. Proceedings, in: Lecture Notes in Computer Science, Vol. 7246, Springer, 2012, pp. 13–25, http://dx.doi.org/10.1007/978-3-642-29066-4_2.

[15] L. Manzoni, L. Mariot, E. Tuba, Balanced crossover operators in genetic algorithms, Swarm Evol. Comput. 54 (2020) 100646, http://dx.doi.org/10.1016/J.SWEVO.2020.100646.

[16] R. Ascone, L. Mariot, L. Manzoni, G. Pietropolli, Evolving cryptographic boolean functions with reaction systems, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), in: GECCO '25 Companion, Association for Computing Machinery, New York, NY, USA, 2025, pp. 195–198, http://dx.doi.org/10.1145/3712255.3726685.

[17] W. Millan, A.J. Clark, E. Dawson, An effective genetic algorithm for finding highly nonlinear boolean functions, in: Information and Communication Security, First International Conference, in: Lecture Notes in Computer Science, Vol. 1334, Springer, 1997, pp. 149–158, http://dx.doi.org/10.1007/BFB0028471.

[18] W. Millan, A.J. Clark, E. Dawson, Heuristic design of cryptographically strong balanced boolean functions, in: Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, in: Lecture Notes in Computer Science, Vol. 1403, Springer, 1998, pp. 489–499, http://dx.doi.org/10.1007/BFB0054148.

[19] J.A. Clark, J. Jacob, Two-stage optimisation in the design of boolean functions, in: Information Security and Privacy, 5th Australasian Conference, ACISP, in: Lecture Notes in Computer Science, Vol. 1841, Springer, 2000, pp. 242–254, http://dx.doi.org/10.1007/10718964_20.

[20] H.E. 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), ACM, 2007, pp. 749–756, http://dx.doi.org/10.1145/1276958.1277112.

[21] R. Asthana, N. Verma, R. Ratan, Generation of boolean functions using genetic algorithm for cryptographic applications, in: 2014 IEEE International Advance Computing Conference (IACC), IEEE, 2014, pp. 1361–1366, http://dx.doi.org/10.1109/IAdCC.2014.6779525.

[22] L. Mariot, A. Leporati, A genetic algorithm for evolving plateaued cryptographic boolean functions, in: Theory and Practice of Natural Computing - Fourth International Conference, TPNC, in: Lecture Notes in Computer Science, Vol. 9477, Springer, 2015, pp. 33–45, http://dx.doi.org/10.1007/978-3-319-26841-5_3.

[23] P.K. Behera, S. Gangopadhyay, An improved hybrid genetic algorithm to construct balanced boolean function with optimal cryptographic properties, Evol. Intell. 15 (1) (2022) 639–653, http://dx.doi.org/10.1007/S12065-020-00538-X.

[24] F. Marini, B. Walczak, Particle swarm optimization (PSO). A tutorial, Chemometr. Intell. Lab. Syst. 149 (2015) 153–165, http://dx.doi.org/10.1016/j.chemolab.2015.08.020.

[25] L. Mariot, A. Leporati, Heuristic search by particle swarm optimization of boolean functions for cryptographic applications, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), ACM, 2015, pp. 1425–1426, http://dx.doi.org/10.1145/2739482.2764674.

[26] J.F. Miller, A.J. Turner, Cartesian genetic programming, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), ACM, 2015, pp. 179–198, http://dx.doi.org/10.1145/2739482.2756571.

[27] M.F. Brameier, W. Banzhaf, Basic Concepts of Linear Genetic Programming, Springer, 2007.

[28] J.F. Miller, An empirical study of the efficiency of learning boolean functions using a Cartesian Genetic Programming approach, in: Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 2, GECCO '99, Morgan Kaufmann Publishers Inc., 1999, pp. 1135–1142, http://dx.doi.org/10.5555/2934046.2934074.

[29] R. Hrbacek, V. Dvorak, Bent function synthesis by means of cartesian genetic programming, in: Parallel Problem Solving from Nature - PPSN XIII - 13th International Conference, in: Lecture Notes in Computer Science, Vol. 8672, Springer, 2014, pp. 414–423, http://dx.doi.org/10.1007/978-3-319-10762-2_41.

[30] 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, in: Lecture Notes in Computer Science, Vol. 9025, Springer, 2015, pp. 192–204, http://dx.doi.org/10.1007/978-3-319-16501-1_16.

[31] J. Husa, R. Dobai, Designing bent boolean functions with parallelized linear genetic programming, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), ACM, 2017, pp. 1825–1832, http://dx.doi.org/10.1145/3067695.3084220.

[32] J. Husa, Comparison of genetic programming methods on design of cryptographic boolean functions, in: Genetic Programming - 22nd European Conference, EuroGP, in: Lecture Notes in Computer Science, Vol. 11451, Springer, 2019, pp. 228–244, http://dx.doi.org/10.1007/978-3-030-16670-0_15.

[33] C. Carlet, D. Jakobovic, S. Picek, Evolutionary algorithms-assisted construction of cryptographic boolean functions, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), ACM, 2021, pp. 565–573, http://dx.doi.org/10.1145/3449639.3459362.

[34] L. Rovito, A.D. Lorenzo, L. Manzoni, Evolution of walsh transforms with genetic programming, in: Companion Proceedings of the Conference on Genetic and Evolutionary Computation, GECCO, ACM, 2023, pp. 2386–2389, http://dx.doi.org/10.1145/3583133.3596317.

[35] J. Husa, L. Sekanina, Semantic mutation operator for a fast and efficient design of bent Boolean functions, Genet. Program. Evol. Mach. 25 (1) (2024) 3, http://dx.doi.org/10.1007/S10710-023-09476-W.

[36] L. Mariot, S. Picek, D. Jakobovic, M. Djurasevic, A. Leporati, Evolutionary construction of perfectly balanced boolean functions, in: IEEE Congress on Evolutionary Computation, CEC 2022, Padua, Italy, July 18-23, 2022, IEEE, 2022, pp. 1–8, http://dx.doi.org/10.1109/CEC55065.2022.9870427.

[37] L. Mariot, M. Saletta, A. Leporati, L. Manzoni, Heuristic search of (semi-)bent functions based on cellular automata, Nat. Comput. 21 (3) (2022) 377–391, http://dx.doi.org/10.1007/S11047-022-09885-3.

[38] C. Carlet, M. Djurasevic, D. Jakobovic, L. Mariot, S. Picek, Evolving constructions for balanced, highly nonlinear boolean functions, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), ACM, 2022, pp. 1147–1155, http://dx.doi.org/10.1145/3512290.3528871.

[39] S. Picek, D. Jakobovic, Evolving algebraic constructions for designing bent boolean functions, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), ACM, 2016, pp. 781–788, http://dx.doi.org/10.1145/2908812.2908915.

[40] C. Carlet, A wide class of boolean functions generalizing the hidden weight bit function, IEEE Trans. Inf. Theory 68 (2) (2022) 1355–1368, http://dx.doi.org/10.1109/TIT.2021.3126684.

[41] S. Picek, K. Knezevic, L. Mariot, D. Jakobovic, A. Leporati, Evolving bent quaternary functions, in: 2018 IEEE Congress on Evolutionary Computation, CEC 2018, Rio de Janeiro, Brazil, July 8-13, 2018, IEEE, 2018, pp. 1–8, http://dx.doi.org/10.1109/CEC.2018.8477677.

[42] S. Picek, D. Sisejkovic, D. Jakobovic, Immunological algorithms paradigm for construction of Boolean functions with good cryptographic properties, Eng. Appl. Artif. Intell. 62 (2017) 320–330, http://dx.doi.org/10.1016/J.ENGAPPAI.2016.11.002.

[43] W. Meier, O. Staffelbach, Fast correlation attacks on stream ciphers (extended abstract), in: Advances in Cryptology - EUROCRYPT '88, Workshop on the Theory and Application of of Cryptographic Techniques, Davos, Switzerland, May 25-27, 1988, Proceedings, in: Lecture Notes in Computer Science, Vol. 330, Springer, 1988, pp. 301–314, http://dx.doi.org/10.1007/3-540-45961-8_28.

[44] Y. Crama, P.L. Hammer, Boolean functions - theory, algorithms, and applications, in: Encyclopedia of mathematics and its applications, Vol. 142, Cambridge University Press, 2011.

[45] R. Brijder, A. Ehrenfeucht, M.G. Main, G. Rozenberg, A tour of reaction systems, Internat. J. Found. Comput. Sci. 22 (7) (2011) 1499–1517, http://dx.doi.org/10.1142/S0129054111008842.

[46] L. Corolli, C. Maj, F. Marini, D. Besozzi, G. Mauri, An excursion in reaction systems: From computer science to biology, Theoret. Comput. Sci. 454 (2012) 95–108, http://dx.doi.org/10.1016/j.tcs.2012.04.003.

[47] E. Formenti, L. Manzoni, A.E. Porreca, On the complexity of occurrence and convergence problems in reaction systems, Nat. Comput. 14 (1) (2015) 185–191, http://dx.doi.org/10.1007/s11047-014-9456-3.

[48] R. Barbuti, R. Gori, F. Levi, P. Milazzo, Investigating dynamic causalities in reaction systems, Theoret. Comput. Sci. 623 (2016) 114–145, http://dx.doi.org/10.1016/j.tcs.2015.11.041.

[49] A. Dennunzio, E. Formenti, L. Manzoni, A.E. Porreca, Complexity of the dynamics of reaction systems, Inform. and Comput. 267 (2019) 96–109, http://dx.doi.org/10.1016/j.ic.2019.03.006.

[50] R. Ascone, G. Bernardini, L. Manzoni, Fixed points and attractors of reactantless and inhibitorless reaction systems, Theoret. Comput. Sci. 984 (2024) 114322, http://dx.doi.org/10.1016/J.TCS.2023.114322.

[51] A. Leporati, L. Manzoni, G. Mauri, G. Pietropolli, C. Zandron, Inferring P systems from their computing steps: An evolutionary approach, Swarm Evol. Comput. 76 (2023) 101223, http://dx.doi.org/10.1016/J.SWEVO.2022.101223.

[52] R. Ascone, G. Bernardini, L. Manzoni, Fixed points and attractors of additive reaction systems, Nat. Comput. 23 (2) (2024) 205–215, http://dx.doi.org/10.1007/S11047-024-09977-2.

[53] A. Alhazov, R. Freund, S. Ivanov, On the spectrum between reaction systems and string rewriting, Nat. Comput. 23 (2) (2024) 159–175, http://dx.doi.org/10.1007/S11047-024-09986-1.

[54] R. Ascone, G. Bernardini, L. Manzoni, Cycles and global attractors of reactantless and inhibitorless reaction systems, Theoret. Comput. Sci. 1045 (2025) 115300, http://dx.doi.org/10.1016/J.TCS.2025.115300.

[55] D. Genova, H.J. Hoogeboom, Z.G. Prodanoff, Extracting reaction systems from function behavior, J. Membr. Comput. 2 (3) (2020) 194–206, http://dx.doi.org/10.1007/S41965-020-00045-Z.

[56] C. Carlet, M. Đurasević, D. Jakobović, S. Picek, L. Mariot, A systematic evaluation of evolving highly nonlinear boolean functions in odd sizes, in: European Conference on Genetic Programming (Part of EvoStar), Springer, 2025, pp. 18–34.

[57] F.-M. De Rainville, F.-A. Fortin, M.-A. Gardner, M. Parizeau, C. Gagné, Deap: A python framework for evolutionary algorithms, in: Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation, 2012, pp. 85–92.

[58] R. Poli, W.B. Langdon, N.F. McPhee, A Field Guide to Genetic Programming, lulu.com, 2008, URL http://www.gp-field-guide.org.uk/.

[59] A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computing, second ed., in: Natural Computing Series, Springer, 2015, http://dx.doi.org/10.1007/978-3-662-44874-8.

[60] G. Pietropolli, S. Nichele, E. Medvet, The role of the substrate in ca-based evolutionary algorithms, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2024, pp. 768–777, http://dx.doi.org/10.1145/3638529.3654112.

[61] S. Jorgensen, G. Nadizar, G. Pietropolli, L. Manzoni, E. Medvet, U.-M. O'Reilly, E. Hemberg, Policy search through genetic programming and LLM-assisted curriculum learning, ACM Trans. Evol. Learn. (2025) http://dx.doi.org/10.1145/3772718.

[62] F. Marchetti, G. Pietropolli, F.J.C. Verdù, M. Castelli, E. Minisci, Automatic design of interpretable control laws through parametrized genetic programming with adjoint state method gradient evaluation, Appl. Soft Comput. 159 (2024) 111654, http://dx.doi.org/10.1016/j.asoc.2024.111654.

[63] E. Allender, L. Hellerstein, P. McCabe, T. Pitassi, M.E. Saks, Minimizing DNF formulas and ac^0_d circuits given a truth table, in: 21st Annual IEEE Conference on Computational Complexity (CCC 2006), 16-20 July 2006, Prague, Czech Republic, IEEE Computer Society, 2006, pp. 237–251, http://dx.doi.org/10.1109/CCC.2006.27.

[64] C. Umans, The minimum equivalent DNF problem and shortest implicants, J. Comput. System Sci. 63 (4) (2001) 597–611, http://dx.doi.org/10.1006/JCSS.2001.1775.

[65] I. Wegener, The Complexity of Boolean Functions, Wiley-Teubner, 1987.

[66] D. Farinati, G. Pietropolli, L. Vanneschi, A study on the dynamics and effectiveness of the deflate geometric semantic mutation, IEEE Trans. Evol. Comput. (2025) http://dx.doi.org/10.1109/TEVC.2025.3611226.

[67] J. Ferreira, M. Castelli, L. Manzoni, G. Pietropolli, A self-adaptive approach to exploit topological properties of different GAs' crossover operators, in: European Conference on Genetic Programming (Part of EvoStar), Springer, 2023, pp. 3–18, http://dx.doi.org/10.1007/978-3-031-29573-7_1.

[68] K. Nyberg, Perfect nonlinear S-boxes, in: Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, in: Lecture Notes in Computer Science, Vol. 547, Springer, 1991, pp. 378–386, http://dx.doi.org/10.1007/3-540-46416-6_32.

[69] S. Picek, L. Mariot, B. Yang, D. Jakobovic, N. Mentens, Design of S-boxes defined with cellular automata rules, in: Proceedings of the Computing Frontiers Conference, ACM, 2017, pp. 409–414, http://dx.doi.org/10.1145/3075564.3079069.