Immunological Algorithms Paradigm for Construction of Boolean Functions with Good Cryptographic Properties

Stjepan Picek[a], Dominik Sisejkovic[b], Domagoj Jakobovic[b]

[a] *KU Leuven, ESAT/COSIC and iMinds, Kasteelpark Arenberg 10, bus 2452, B-3001 Leuven-Heverlee, Belgium*
[b] *Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia*

Abstract

In this paper we investigate the efficiency of two immunological algorithms (CLONALG and opt-IA) in the evolution of Boolean functions suitable for use in cryptography. Although in its nature a combinatorial problem, we experiment with two representations of solutions, namely, the bitstring and the floating point based representation. The immunological algorithms are compared with two commonly used evolutionary algorithms - genetic algorithm and evolution strategy. To thoroughly investigate these algorithms and representations, we use four different fitness functions that differ in the number of parameters and difficulty. Our results indicate that for smaller dimensions immunological algorithms behave comparable with evolutionary algorithms, while for the larger dimensions their performance is somewhat worse. When considering only immunological algorithms, opt-IA outperforms CLONALG in most of the experiments. The difference in the representation for those algorithms is also clear where floating point works better with smaller problem sizes and bitstring representation works better for larger Boolean functions.

Keywords: Artificial Immune Systems; Evolutionary algorithms; Boolean functions; Cryptography; Comparison; Efficiency analysis

1. Introduction

Cryptography, in its core, is a science (and art) of secret writing with the goal of hiding the meaning of a message [1]. As such, it plays a tremendous role in people's everyday life. To ensure the secrecy of information (but also authentication and data integrity among other relevant goals [2]) we rely on strong, well-designed cryptographic algorithms –commonly reffered as **ciphers**.

When the ciphers use keys, we can divide them on the basis whether all communicating parties use the same key or not [3]. If all entities use the same key for encryption and decryption operations then we talk about **symmetric-key cryptography** or **secret-key cryptography**. Assume that two parties (commonly denoted as Alice and Bob) want to exchange some message and they want it to remain secret, i.e., that an attacker (commonly denoted as Eve) cannot read it. Alice could encrypt her message and send it over an insecure (public) channel to Bob. If Bob has the same key as Alice, he can then decrypt and read the message. Eve cannot decrypt the message if she does not know the key. Therefore, if Alice and Bob want to keep their communication private they need either to keep the key secret or the algorithm secret. However, in 19th century Kerchoff stated that a cryptosystem should be secure even if everything about the system, except the key, is publicly known [4]. A classical division of symmetric-key cryptography is on **block ciphers** and **stream ciphers** [3]. A common trait for all those ciphers is that they are designed in accordance with a number of cryptographic criteria they need to fulfill. Those criteria enable ciphers to resist various cryptanalysis attacks where to resist linear [5] cryptanalysis, we require that the cipher possess enough nonlinearity.

In both block and stream ciphers one common source of nonlinearity (although not the only one) are **Boolean functions**. In block ciphers, one usually uses vectorial Boolean functions or Substitution boxes (S-boxes) [3] where input and output dimensions are comparable (e.g. the same as in the AES cipher [6] or similar like in the DES cipher [7]). On the other hand, in stream ciphers more common are either Boolean functions or S-boxes where output dimension is strictly smaller than the input dimension [8].

To build such nonlinear elements, one has three main options on his disposal: algebraic constructions, random search, and heuristics [9]. Heuristic techniques are well visited with works spanning from simulated annealing [10]

Table 1: The search space size for various input size n.

n	6	8	10	12	14	16
#	2^{64}	2^{256}	$2^{1\,024}$	$2^{4\,096}$	$2^{16\,384}$	$2^{65\,536}$

and evolutionary algorithms [11], to particle swarm optimization [12]. All those methods have in common that they are highly successful and give results comparable to algebraic constructions.

In the rest of this paper, we concentrate only on Boolean functions, i.e., where the output dimension of a function equals one. As the construction principle, we use heuristics, more precisely immunological inspired computation where we experiment with the Clonal Selection Algorithm and the optimization Immune Algorithm. Both algorithms are a special class of Immune Algorithms and are inspired by the clonal selection principle of the human immune system [13, 14]. From the evolutionary algorithms paradigm, we experiment with genetic algorithms (GAs) and evolution strategies (ES). We note that this work presents the first step when investigating the effectiveness of immunological algorithms for the construction of Boolean functions with good cryptographic properties. By doing so, we explore two different representations of solutions, namely the bitstring and the floating point representation and compare the aforesaid algorithms. As a benchmark suite, we use four fitness functions and a number of Boolean function sizes.

1.1. Motivation and Contributions

Historically, Boolean functions were mostly used in combination with Linear Feedback Shift Registers (LFSRs) in two models – the filter generator and the combiner generator. In a filter generator, the output is obtained by a nonlinear combination of a number of positions in one longer LFSR while in a combiner generator, several LFSRs are used in parallel and their output is the input for a Boolean function. Such Boolean functions need to fulfill a number of properties: to be balanced, with high nonlinearity, large algebraic degree, large algebraic immunity, large fast algebraic immunity, and large correlation immunity (in the case of combiner generators) [15]. However, obtaining large enough Boolean functions with good values of the aforesaid properties is not a trivial task. First, to illustrate the size of the problem, we give the corresponding search space sizes in Table 1.

However, in our experiments we concentrate only on two properties out of those listed above - balancedness and nonlinearity. By doing so, we are able to concentrate more on the comparison between different heuristic techniques as well as with related work. Therefore, although talking about constructing Boolean functions with good cryptographic properties, we emphasize that here we consider these problems as benchmarks where the end goal are Boolean functions possessing certain properties. Such obtained functions could, but will not in general, have all necessary properties of a sufficient quality.

In this paper, we give two main contributions and a number of smaller ones. The first contribution represents an experimental investigation on the efficiency of immunological algorithms when designing Boolean functions with good cryptographic properties. Although a simple application of a new algorithm to a well-researched problem does not necessarily constitute a significant contribution, we believe here to be relevant since the whole area of applying immunological algorithms to cryptography is a new one. Moreover, by switching the paradigm from evolutionary algorithms to immunological algorithms we should be able to give an insight whether further improvements are possible by simply changing the algorithms. Indeed, by doing so, we try to give knowledge that is not domain specific and tells us whether for performance increase is more important the choice of algorithm, representation, or fitness function. The second contribution is that we are the first, as far as we know, to experiment with the floating point representation of solutions to evolve cryptographic Boolean functions. Finally, in this work we investigate larger sizes of Boolean functions than can usually be found in the literature, where we go up to Boolean functions with 16 variables. We note that by doing so, we experiment with sizes that have practical importance since 13 inputs is considered the minimal size of a Boolean function to be useful in cryptography [15]. To strengthen our experimental results, we compare two immunological algorithms with two well-researched evolutionary algorithms for both bitstring and floating point representation. Naturally, in order to examine the relevance of algorithms' parameters, we conduct a detailed tuning phase. Finally, all experiments are conducted on four fitness functions where two are well-known ones from the literature, and two are modifications that provide more gradient in the search process.

1.2. Outline of the Paper

This paper is divided into two main parts: the description of our experimental setup in Section 5, and the obtained results in Section 6. However, first we start with a short introduction to Boolean functions, their properties, representations, and notation we use in Section 2. Then, after we introduced the readers with basic notions and properties, we give a selection or relevant works in Section 3. Next, Section 4 introduces fitness functions we use in our experiments. Detailed explanation about the representation perspectives for heuristic algorithms is presented in Section 5.1. The parameter tuning phase is discussed in Section 6.1 and the results for the first and second fitness functions in Sections 6.2 and 6.3, respectively. In Section 6.4, we give a short discussion about the efficiency of the algorithms used as well as some potential future research directions. Finally, we conclude in Section 7.

2. Boolean Functions Properties and Representations

Let $n, m \in \mathbb{N}$. We denote the set of all n-tuples of the elements in the field $\mathbb{F}_2$ as $\mathbb{F}_2^n$, where $\mathbb{F}_2$ is the Galois field with two elements. The set $\mathbb{F}_2^n$ represents all binary vectors of length n and it can be viewed as a $\mathbb{F}_2$-vectorspace [15]. The inner product of vectors $\vec{a}$ and $\vec{b}$ is denoted as $\vec{a} \cdot \vec{b}$ and it equals $\vec{a} \cdot \vec{b} = \oplus_{i=1}^{n} a_i b_i$. Here, "$\oplus$" represents addition modulo two. The Hamming weight (HW) of a vector $\vec{a}$, where $\vec{a} \in \mathbb{F}_2^n$, is the number of non-zero positions in the vector. An (n, m)-function is any mapping F from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ where Boolean functions represent $m = 1$ case.

2.1. Boolean Function Representations

A Boolean function f on $\mathbb{F}_2^n$ can be uniquely represented by a truth table (TT), which is a vector $(f(\vec{0}), ..., f(\vec{1}))$ that contains the function values of f, ordered lexicographically [15].

The second unique representation of a Boolean function is the Walsh-Hadamard transform W_f that measures the correlation between $f(\vec{x})$ and all linear functions $\vec{a} \cdot \vec{x}$ [15, 16]. The Walsh-Hadamard transform of a Boolean function f equals:

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

There exist other unique representations of Boolean functions like the algebraic normal form or numerical normal form [15]. However, since we do not work with those representations nor we need properties that are usually derived from those representations, we omit the details and refer interested readers to [15].

2.2. Boolean Function Properties and Bounds

A Boolean function f is **balanced** if the Walsh-Hadamard spectrum of a vector $\vec{0}$ equals zero [17]:

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

Alternatively, in the truth table representation, a Boolean function with n inputs is balanced if its Hamming weight equals 2^{n-1}.

A Boolean function should lie at a large Hamming distance (HD) from all affine functions and the nonlinearity N_f of a Boolean function f is the minimum HD between the function f and affine functions [15]. The **nonlinearity** N_f of a Boolean function f expressed in terms of the Walsh-Hadamard coefficients equals [15]:

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

The Parseval's relation equals:

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

and it implies that the mean of $W_f(a)^2$ equals 2^n, and $\max_{a \in \mathbb{F}_2^n} |W_f(a)|$ is then at least equal to the square root of this mean.

From Eq. (4), it follows that the maximal value of the Walsh-Hadamard spectrum equals at least $2^{n/2}$ which occurs in the case of **bent** Boolean functions. These bent functions can have only even number of variables and are never balanced. After presenting the general expression for nonlinearity in Eq. (3), we give the equation for nonlinearity of bent functions: [18, 19]:

$$N_f = 2^{n-1} - 2^{\frac{n}{2}-1}. \tag{5}$$

Since the values of bent functions are not uniformly distributed, they are not appropriate for direct usage in cryptography. However, it is possible to use bent functions in secondary constructions that result in balanced Boolean functions with high nonlinearity [15].

Before proceeding to examine balanced Boolean functions, we pause in an effort to provide an insight on the number of bent Boolean functions, which directly translates to the difficulty of the problem of obtaining bent functions. In general, there is no known efficient bounds for an n-variable Boolean function. A naive upper bound equals [15]:

$$upper\_bound = 2^{2^{n-1}+\frac{1}{2}\binom{n}{n/2}}, \tag{6}$$

and the lower bound is:

$$lowwer\_bound = 2^{2^{\frac{n}{2}+log_2(n-2)-1}}. \tag{7}$$

From the previous expressions, we can deduce that the number of bent functions is huge, but still only a tiny fraction of the whole search area.

Sarkar and Maitra showed that if a Boolean function f has the correlation immunity property of order t, an even number of inputs n, and $k \leq \frac{n}{2} - 1$ then its nonlinearity N_f has an upper bound as follows [20]:

$$N_f \leq 2^{n-1} - 2^{\frac{n}{2}-1} - 2^k, \tag{8}$$

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

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

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

There exist more strict bounds, but the one presented here is sufficient for our purposes [15]. For an in-depth treatment of properties of Boolean functions as well as their applications in cryptography, we refer interested readers to [15, 21].

3. Related Work

As already mentioned in Section 1, there are many successful applications of heuristics techniques when constructing Boolean functions usable in cryptography. However, as far as the authors are aware, there are no examples in the literature that involve immunological algorithms and the design of cryptographic Boolean functions. Therefore, in this section we enumerate a subset of works that we consider relevant for this topic, where we investigate the representation and algorithmic perspectives.

When discussing representations, note there are two perspectives; one encompassing representations of Boolean functions and the second one dealing with the genotype representations in optimization algorithms. However, as it will be seen, there exist a certain correlation between those two perspectives.

When considering the truth table representation of Boolean functions, it is easy to see that the bitstring genotype representation lends itself naturally for this task. The first paper, as far as the authors know, that explores the evolution of Boolean functions for cryptography is by Millan et al. [22] where the authors try to evolve Boolean functions with high nonlinearity. Millan, Clark, and Dawson further increase the strength of genetic algorithms by combining them with the hill climbing and a resetting step with a goal to find highly nonlinear Boolean functions of up to 12 variables [23].

Clark and Jacob experiment with two-stage optimization to generate Boolean functions with high nonlinearity and low autocorrelation [10]. They use a combination of simulated annealing and hill climbing with a cost function motivated by the Parseval's theorem.

Aguirre et al. use a multi-objective random bit climber to search for balanced Boolean functions of size up to eight inputs that have high nonlinearity [24]. Their results indicate that the multi-objective approach is highly efficient when generating Boolean functions that have high nonlinearity.

McLaughlin and Clark experiment with simulated annealing to generate Boolean functions that have optimal values of a number of properties, namely, algebraic immunity, fast algebraic resistance, and algebraic degree [25]. In their work, they experiment with Boolean functions with sizes of up to 16 inputs.

Next, there exists a number of papers that uses the truth table representation of Boolean functions, but with the genotype different from the bitstring representation. Picek, Jakobovic, and Golub experiment with genetic algorithms and genetic programming to find Boolean functions that possess several optimal properties [26]. Here, the evolved tree (genotype) is a posteriori transformed to the truth table representation for the evaluation purposes.

Hrbacek and Dvorak use Cartesian genetic programming to evolve bent Boolean functions of sizes up to 16 inputs where the authors experiment with several configurations of algorithms in order to speed up the evolution process [27]. There, the authors do not limit the number of generations and therefore they succeed in finding bent function in each run for sizes between 6 and 16 variables.

Picek et al. compare the effectiveness of Cartesian genetic programming and genetic programming when looking for highly nonlinear balanced Boolean functions with eight inputs [11]. In this work, the authors show that Cartesian genetic programming performs favorably when compared with some other evolutionary approaches when evolving cryptography relevant Boolean functions. Picek et al. investigate several evolutionary algorithms in order to evolve Boolean functions with different values of the correlation immunity property. In the same paper, the authors also discuss the problem of finding correlation-immune functions with minimal Hamming weight where they experiment with Boolean functions that have eight inputs [28]. Picek et al. investigate a number of different evolutionary algorithms and fitness functions for Boolean functions of 8 inputs [29]. They show that genetic programming and Cartesian genetic programming outperform genetic algorithms and evolution strategies in a number of relevant test scenarios. Next, Picek et al. investigate evolution of balanced Boolean functions of up to 16 inputs fulfilling a number of cryptographic properties as well as the evolution of minimal Hamming weight and different correlation immunity order Boolean functions [30]. Finally, Picek and Jakobovic use genetic programming to evolve algebraic constructions that are then used to construct bent Boolean functions [31].

Stepping away from the truth table representation, there are several works that consider the Walsh-Hadamard spectrum representation. There, the genotype representation is a list of integer values. Clark et al. experiment with simulated annealing in order to design Boolean functions using spectral inversion [32]. They observe that many cryptographic properties of interest are defined in terms of the Walsh-Hadamard transform values. Therefore, they work in the spectral domain where on the basis of Parseval's theorem one can infer what values could be in a Walsh-Hadamard spectrum (note it is not possible to know the positions where those values should be). Therefore, when generating a Walsh-Hadamard spectrum a necessary step is to make an inverse transform to verify that the spectrum indeed maps to a Boolean function.

Mariot and Leporati use Particle Swarm Optimization (PSO) to find Boolean functions with good trade-offs of cryptographic properties for dimensions up to 12 inputs [12]. The same authors use genetic algorithm where the genotype consists of the Walsh-Hadamard values in order to evolve semibent (plateaued) Boolean functions [33]. Plateaued functions have only three values in the Walsh-Hadamard spectrum and therefore represent somewhat easier task to evolve when working with the Walsh-Hadamard representation than in the case when considering balanced Boolean functions.

4. Fitness Functions

In this section, we present the used fitness functions that represent relevant objectives in the Boolean function design. In total, there are four fitness functions, two following established lines of work and two that represent our modifications that we consider to be more powerful and providing more gradient in the search. We denote those functions as the fitness one and the modified fitness one, and the fitness two and the modified fitness two.

4.1. Fitness Function One

The first fitness function concerns the objective of finding bent Boolean functions, i.e. Boolean functions with the maximal nonlinearity. Here, we start with the simplest fitness function that has only one parameter - the nonlinearity

Table 2: The nonlinearity of bent functions.

n	6	8	10	12	14	16
N_f	28	120	496	2 016	8 128	32 640

property. To construct bent Boolean functions we use the following fitness where the goal is *maximization*:

$$fitness_1 = N_f. \tag{10}$$

As already stated, bent Boolean functions exist only for even number of variables and the maximal obtainable nonlinearity for each size we examine is given in Table 2.

One can observe that here we use a fitness function that does not provide a lot of gradient information. This is apparent since we look for the nonlinearity value only on a basis of the worst Walsh-Hadamard coefficient and we simply disregard the rest of the values in the spectrum. In other words, two Boolean functions that have a Hamming distance equal to one can have either the same nonlinearity, or a difference of a two in the nonlinearity value. However, since we calculate the nonlinearity property via the Walsh-Hadamard transformation, every change in the truth table representation can change a significant number of values in the spectrum. Unfortunately, that information is not visible with the current fitness function.

4.2. Modified Fitness Function One

In order to increase the strength of our fitness function, we present a modified version that considers the whole Walsh-Hadamard spectrum. Since we know that bent Boolean functions have flat Walsh-Hadamard spectrum with a value equal to $2^{n/2}$, we can use that information to lead our search:

$$fitness_{1A} = N_f + \frac{count\_spectrum\_values}{2^{n-1}}, \tag{11}$$

where $count\_spectrum\_values$ equals the number of occurrences where the Walsh-Hadamard value is $2^{n/2}$. Furthermore, we normalize this value to a range $[0, 2]$ by dividing it by a value of 2^{n-1}. By doing so, we ensure that the second part of the fitness function has a smaller weight than the minimal improvement of the nonlinearity value (since as we stated the minimal step in N_f equals two).

We note one could calculate the nonlinearity differently where one can use the following cost function [34]:

$$cost(f) = \sum_{\vec{a}} \left| \left| W_f(\vec{a}) \right| - X \right|^R, \tag{12}$$

where X and R are real valued parameters. Although we acknowledge that using the above function can result in improvements of results, we believe that the cost of two more additional parameters that need to be tuned is too great. This is especially evident in a setting like ours where we work with many different sizes of Boolean functions. We note that as far as we are aware, there is no straightforward way to choose those parameters nor some obvious scaling rule between parameter values and the Boolean function size.

4.3. Fitness Function Two

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

$$fitness_2 = BAL + N_f. \tag{13}$$

The balancedness property (BAL) we use as a penalty parameter. As evident from the Algorithm 1, when the function is balanced, it receives a value (reward) of one, while imbalanced functions get a penalty proportional to their imbalancedness.

We experimented with several variants for imbalancedness penalty, but the results show similar performance as long as the penalty is calculated gradually. Note that Eq. (13) may also be restated as a two-stage fitness function.

Algorithm 1 Calculate balancedness (BAL)

ones = $HW(TT)$
zeros = 2^n - ones
if zeros == ones **then**
  return 1
else
  return $|ones - zeros|$
end if

Table 3: The maximal nonlinearity of balanced functions.

n	6	8	10	12	14	16
N_f	26	118	494	2 014	8 126	32 638

In that configuration, only the balancedness is tested at first, and nonlinearity value is added only if the function is balanced. In Table 3, we give maximal obtainable values for nonlinearity of balanced Boolean functions.

Finally, since we stated that we look for Boolean functions that have correlation immunity equal to zero, one could ask why not to include that information in the fitness function. We opted not to follow that line of research since we believe it would serve as an additional constraint on the search that could result in inability to visit certain parts of the search space.

4.4. Modified Fitness Function Two

Similarly as with the first fitness function, in Eq. (13) we use only the worst value of the Walsh-Hadamard spectrum to calculate nonlinearity. Again, that can be remedied with a simple modification of the fitness function:

$$fitness_{2A} = BAL + N_f + \frac{2^n - count\_spectrum\_values}{2^{n-1}}. \tag{14}$$

Here, *count_spectrum_values* equals the number of times that the maximal Walsh-Hadamard value occurs. Since the goal is maximization of the nonlinearity property and the lower the number of such maximal values the better, we subtract it from the value 2^n (the total number of values in the Walsh-Hadamard spectrum). Finally, we normalize this expression by dividing it with 2^{n-1} as in the modified fitness function one.

5. Experimental Setup

In this section, we give details about experimental setup for each of the algorithms as well as the common parameters. For each of the algorithms we conduct a two parameter sweep in order to investigate the best combination of parameters. Since there are more than two parameters for every algorithm, we try to look only at the most obvious choices for tuning.

5.1. Representations

For all algorithms, we experiment with two different representations for encoding a Boolean function: bitstring and floating point representation.

As a Boolean function can be represented with a truth table, which is actually a sequence of bits, the bitstring genotype comes naturally as one possible representation that can be easily used without need for transformations. It is important to notice that even though the usage of a bitstring genotype is a natural choice, it can become problematic to use such a representation due to the genotype size exponentially depending upon the number of variables of a Boolean function. In other words, for a problem with a Boolean function with n variables, the corresponding bitstring genotype must be of size 2^n to completely cover the solution search space. As shown in Table 1, for larger values of n the search space and therefore the bitstring size becomes of noticeable and problematic dimensions.

The second approach used for representing Boolean functions is the floating point genotype, defined as a vector of continuous variables. With this representation one needs to define the translation of a vector of floating point numbers into the corresponding truth table (binary values). The idea behind this translation is that each continuous variable (a real number) of the floating point genotype represents a *subsequence of bits* from the truth table.

All the real values in the floating point vector are constrained to the interval $[0, 1]$. As mentioned, the number of elements of a truth table is 2^n, where n is the number of Boolean variables. The number of bits represented by a single continuous variable of the floating point vector can vary, and is defined as:

$$decode\_by = \frac{2^n}{dimension}, \tag{15}$$

where the parameter *dimension* denotes the floating point vector size (number of real values). This parameter can be modified, as long as the size of the truth table is divisible with this value (i.e. each floating point number represents the same number of bits). The first step of the translation is to convert each floating point number to an integer value. Since each real value must represent *decode_by* bits, the size of the integer interval is given as:

$$interval = \frac{1}{2^{decode\_by}}. \tag{16}$$

To obtain a distinct integer value for a given real number, every element d_i of the floating point vector is divided by the calculated interval size, generating a sequence of integer values:

$$int\_value_i = \left\lfloor \frac{d_i}{interval} \right\rfloor. \tag{17}$$

The second translation step consists of decoding the integer values to a binary sequence that can finally be used for evaluation. For this purpose we experiment with two coding systems - binary and Gray coding, and further evaluate their influence.

As an example, consider a Boolean function of 3 variables, with the truth table of 8 bits. Suppose we want to represent the function with 4 real values; in this case, each real value encodes 2 bits from the truth table. A string of two bits may have 4 distinct combinations, therefore a single real value must be decoded into an integer value from 0 to 3. Since each real value is constrained to $[0, 1]$, the corresponding integer value is obtained by dividing the real value with $2^{-2} = 0.25$ and truncating to nearest smaller integer. Finally, the integer values are translated into the sequence of bits they encode, using either binary or Gray encoding scheme. Following the above parameters, a floating point vector $[0.71, 0.93, 0.13, 0.48]$ would be decoded into the integer vector $[2, 3, 0, 1]$, which translates into the truth table "10110001" using the standard binary encoding.

5.2. Clonal Selection Algorithm

In this paper, we use the Clonal Selection Algorithm (CLONALG) version for optimization tasks proposed by de Castro and Von Zuben in [35]. A fixed population size of 50 is used in all experiments. For the cloning phase all antibodies (problem solutions) are chosen and the number of clones for each antibody is selected proportionally to its fitness resulting in a total number of clones defined by [35]:

$$N_C = \sum_{i=1}^{n} round(\frac{\beta \cdot n}{i}), \tag{18}$$

where we experiment with β values of 2, 4, 6, and 8 for each fitness function. In general, an inversely proportional hypermutation is used in the mutation phase. As two different representations are used (recall Section 5.1), a genotype dependent hypermutation is applied.

For the floating point representation the number of mutated values M (mutation intensity) is selected by:

$$M = 1 - \frac{1}{k} \cdot (c \cdot dimension) + (c \cdot dimension), \tag{19}$$

where k is the index of a specific antibody in the population generated in the selection phase (assuming the population is sorted by fitness), c is the mutation parameter set to a value of 0.2 and *dimension* is the floating point dimension

(number of distinct values). Note that the given expression corresponds to the formula used for calculating the number of clones for each antibody resulting in the same value M for each clone of a specific antibody. After M is calculated, the current clone is mutated by adding a randomly generated number in selected bounds to a randomly selected element of the floating point vector.

For the bit string representation a similar approach was used. The mutation intensity value M is calculated by:

$$M = (1 + l \cdot \rho \cdot (1 - e^{\frac{-r}{\tau}})), \tag{20}$$

where l is the bit string length, r is the rank of the clone to be mutated in the current population and ρ is the mutation parameter. It is worth noting that ρ directly affects the mutation intensity. Therefore ρ is varied in experiments by the values of 0.2, 0.4, 0.6, and 0.8. The number of newly generated antibodies in each generation is set to 5. Elitism is achieved by never mutating the first clone of the best antibody in the given population. For additional information about the CLONALG algorithm, we refer interested readers to [36].

5.3. Optimization Immune Algorithm

Alongside CLONALG, we additionally use the optimization Immune Algorithm (opt-IA) [37] with elitism. As before, a constant population size of 50 antibodies is chosen. The number of clones for each antibody is set to 5. A static pure aging strategy is used for which we experiment with τ_B values of 5, 10, 15, and 20. The mutation rate is set to the value of 0.2. Because of the two representations used (Sec. 5.1), specific mutation operators are applied accordingly. For the floating point representation the same hypermutation is used as for CLONALG, while for the bitstring representation a hypermacromutation is used for which the mutation intensity is independent from the fitness value, but partially depends on the mutation rate. It is executed by first randomly choosing two indexes i and j such that $0 \leq i \leq j \leq l$ where l is the bitstring length, and secondly flipping the value of the bits from index i to j in the bit string with probability c. In experiments the values of 0.2, 0.4, 0.6 and 0.8 are used for the parameter c. For additional information about opt-IA algorithm, we refer readers to [38].

5.4. Genetic Algorithm

With the genetic algorithms (GAs), we use the k-tournament steady-state selection [39] as given in Algorithm 2 (with $k = 3$), since this selection avoids the need to use the crossover probability parameter.

Algorithm 2 Steady-state tournament selection

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

Mutation is selected uniformly at random between a simple mutation, where a single bit is inverted, and a mixed mutation, which shuffles the bits in a randomly selected subset. The crossover operators are one-point and uniform crossover, performed at random for each new offspring. For each of the fitness functions we experiment with population sizes of 50, 100, 200, and 300 and individual mutation probabilities of 0.3, 0.5, and 0.7. It is important to note that we use the mutation probability to select whether an individual would be mutated or not, and the mutation operator is executed only once on a given individual; e.g. if the mutation probability is 0.7, then on average 7 out of every 10 new individuals will be mutated (see Algorithm 2), and one mutation will be performed on that individual. For further information about GAs, we refer to [39, 40].

5.5. Evolution Strategy

When experimenting with evolution strategy (ES) we use $(\mu + \lambda)$-ES. In this algorithm, in each generation, parents compete with offspring and from their joint set μ fittest individuals are kept. In our experiments offspring population size λ has values of 5, 10, and 20. Parent population size μ has values of 50, 100, and 200. For further information on ES, we refer interested readers to [41, 42, 43].

5.6. Common Parameters

In all the experiments the number of independent trials M for each configuration is 50 and the stopping criterion for all algorithms is 500 000 evaluations. We decided to work with "only" 500 000 evaluations in an effort to make the comparison with a number of related works easier [11, 28].

6. Results

In this section we present the results separately for the first objective (using fitness functions 1 and 1A) and the second objective (with fitness functions 2 and 2A), preceded with the tuning phase for all the algorithms.

6.1. Parameter optimization

Since the number of possible combinations is prevailing, we perform the tuning phase only with fitness function 2A as in Eq. (14), because it offers a greater diversity in the resulting values. Once optimized, the same parameters are applied to all the fitness variants. The tuning procedure was performed starting with an initial parameter set for each algorithm.

For each observed parameter value, 30 independent runs were performed and 30 best solutions were recorded (a separate set for each parameter value). The choice of the best value for a given parameter was made based on the median values of the recorded sets, where the parameter value producing the set with the best (highest) median was selected. It is important to note that in many cases there were no significant statistical differences between the sets, but we based our choice on the median measure in any case. We performed the tuning experiments for all dimensions of Boolean functions, but a number of combinations did not yield any significant statistical difference. In Table 4, we present results for tuning phase for Boolean functions with 8 and 12 inputs. We note that for every Boolean function size we found the best set of parameters and we applied those parameters for all 4 fitness functions.

Before presenting the results for fitness functions, we note here several questions we are interested in:

- what is the efficiency of the immunological algorithms considering the increasing number of Boolean variables;
- how does the choice of the fitness function influences the results;
- what representation produces better results for which algorithm.

For every algorithm, Boolean function size and fitness function we run 50 independent runs and we report maximal obtained results over all runs. When presenting results in boxplot charts, we use letters $\_b$ to the denote bitstring encoding and $\_f$ to denote the floating point encoding.

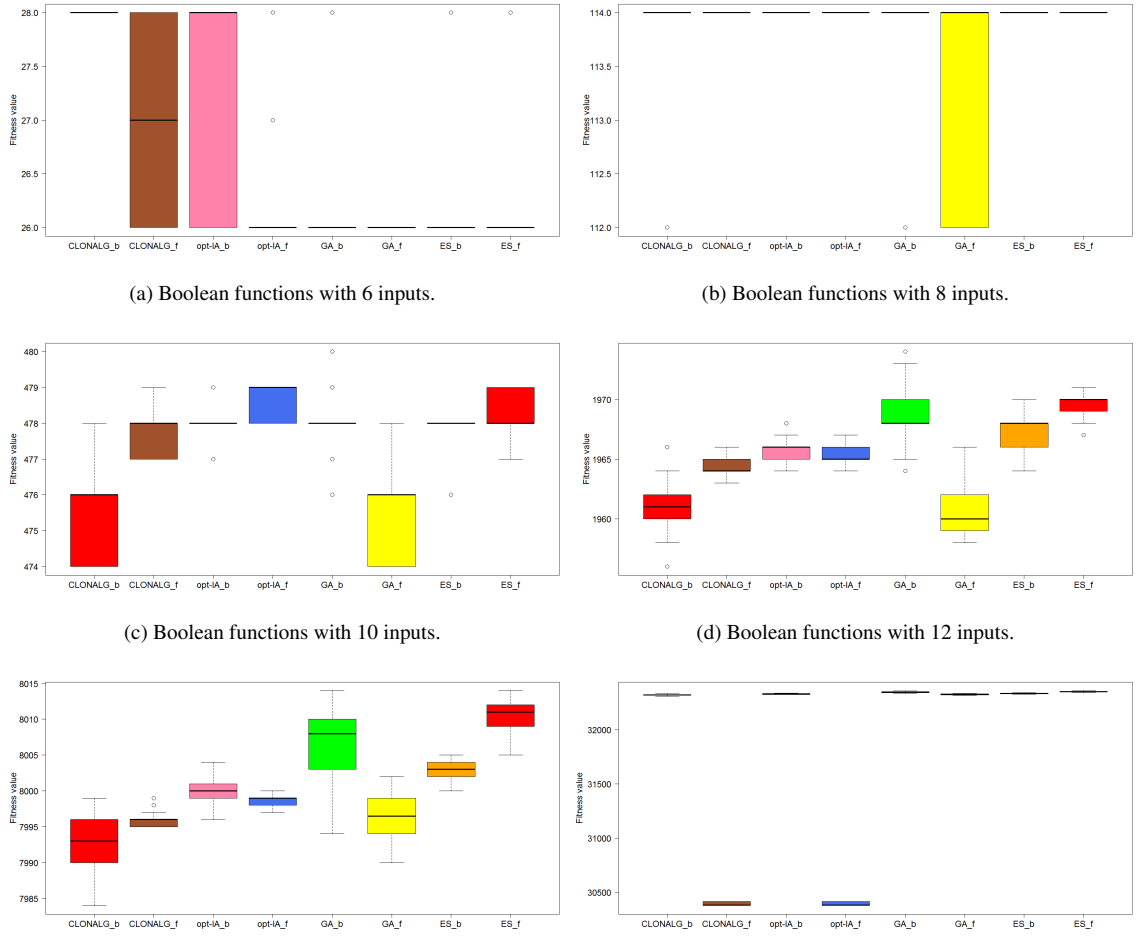
6.2. The first objective

Recall that here the goal is to find bent Boolean functions. After the parameter optimization, all the algorithms are compared by conducting the same number of independent runs with fitness functions 1 and 1A. Figures 1a to 1f show the efficiency of the algorithms for the first fitness function as given in Eq. (10) with both the floating point and the bitstring representation. From the results we observe that for smallest size of Boolean function (6 inputs), all algorithms manage to find the global optimum. However, only CLONALG with the bitstring representation manages to find global optimum in all runs. This size is also the only one where algorithms succeed in finding global optimum. When evaluating the results for Boolean functions with 8 inputs, we see that all but GA with floating point encoding always reach 114 value.

For Boolean function sizes between 10 and 14 inputs we see that generally speaking GA and ES outperform CLONALG and opt-IA algorithms. Furthermore, the difference between CLONALG and opt-IA algorithms is quite visible where opt-IA algorithm outperforms CLONALG. Finally, when considering Boolean functions with 16 inputs, the worst performing algorithms are CLONALG and opt-IA with floating point representation. It is hard to estimate the differences among other algorithms, but we note that the best found value equals 32 363 which is obtained with ES with floating point representation.

Table 4: Results of the parameter optimization phase

Boolean variables	8		12	
Parameter name	Initial value	Optimized value	Initial value	Optimized value
		CLONALG / floating point		
FP vector size	8, 16, 32, 128, 256	256	32, 64, 128, 256, 512, 1 024, 2 048, 4 096	4 096
coding	Gray, binary	Gray	Gray, binary	Gray
		opt-IA / floating point		
FP vector size	8, 16, 32, 64, 128, 256	64	32, 64, 128, 256, 1 024, 2 048, 4 096	4 096
coding	Gray, binary	Gray	Gray, binary	Gray
		GA / floating point		
FP vector size	8, 16, 32, 128, 256	128	32, 64, 128, 256, 2 048, 4 096	2 048
coding	Gray, binary	Gray	Gray, binary	Gray
		ES / floating point		
FP vector size	32, 64, 128, 256	32	256, 512, 1024, 2 048, 4 096	256
λ	5, 10, 20	20	5, 10, 20	10
		CLONALG / bitstring		
β	2,4,6,8	6	2, 4, 6, 8	8
ρ	0.2, 0.4, 0.6, 0.8	0.2	0.2, 0.4, 0.6, 0.8	0.2
		opt-IA / bitstring		
τ_B	5, 10, 15, 20	20	5, 10, 15, 20	20
c	0.2, 0.4, 0.6, 0.8	0.6	0.2, 0.4, 0.6, 0.8	0.2
		GA / bitstring		
population size	50, 100, 200, 300	50	50, 100, 200, 300	300
mutation rate	0.3, 0.5, 0.7	0.7	0.3, 0.5, 0.7	0.7
		ES / bitstring		
μ	50, 100, 200	200	50, 100, 200	200
λ	5, 10, 20	5	5, 10, 20	5



(a) Boolean functions with 6 inputs.

(b) Boolean functions with 8 inputs.

(c) Boolean functions with 10 inputs.

(d) Boolean functions with 12 inputs.

(e) Boolean functions with 14 inputs.

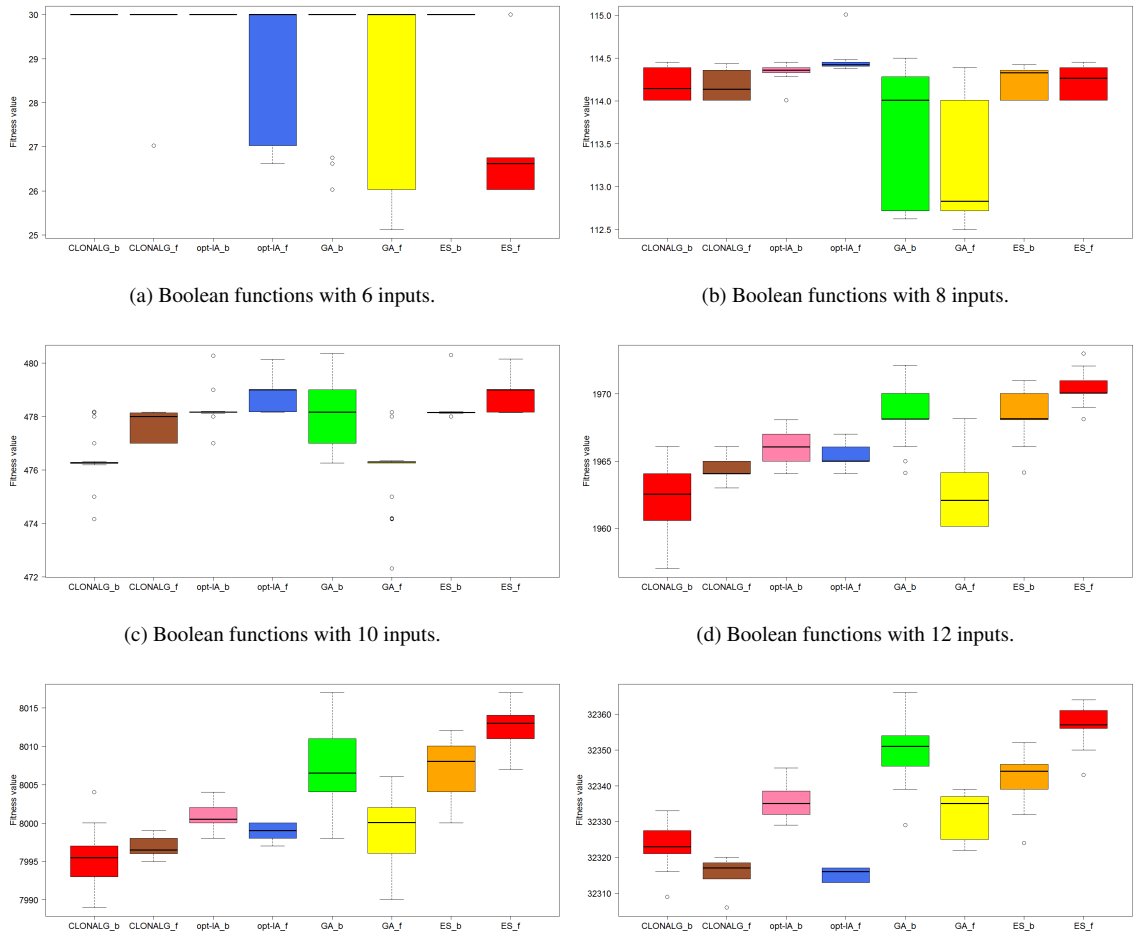
(f) Boolean functions with 16 inputs.

Figure 1: Boxplot results for all algorithms and Boolean function sizes, fitness function 1.

Next, in Figures 2a to 2f, we give results for fitness function denoted as 1*A*. Recall, here we add reward proportional to the number of times the Walsh-Hadamard spectrum has the maximal value and the maximal fitness value for 6 inputs equals 30. Again as for the fitness function 1, we see that the global optimum value is obtained only for the smallest Boolean function size. The results for larger sizes are similar as for the previous fitness function.

However, after analyzing the results we observe that the number of required Walsh-Hadamard coefficients improves over time which points us that for this more complicated fitness function we need more evaluations than for the fitness function 1. The starkest difference in algorithms' efficiency for fitness functions 1 and 1*A* can be observed for Boolean functions with 16 inputs where the fitness 1*A* gives much better results than fitness function 1. As in the fitness function 1, opt-IA outperforms CLONALG.

To conclude, we see that the efficiency of all algorithms quickly deteriorates with the increase of the problem size which points us that larger dimensions would require more than 500 000 evaluations. The differences between the clonal selection algorithms and the evolutionary algorithms are generally speaking small, but with an advantage for evolutionary algorithms. Finally, we can observe there is a clear benefit when adding the extra parameter (the evaluation of the whole Walsh-Hadamard spectrum) into the fitness functions.



(a) Boolean functions with 6 inputs.

(b) Boolean functions with 8 inputs.

(c) Boolean functions with 10 inputs.

(d) Boolean functions with 12 inputs.

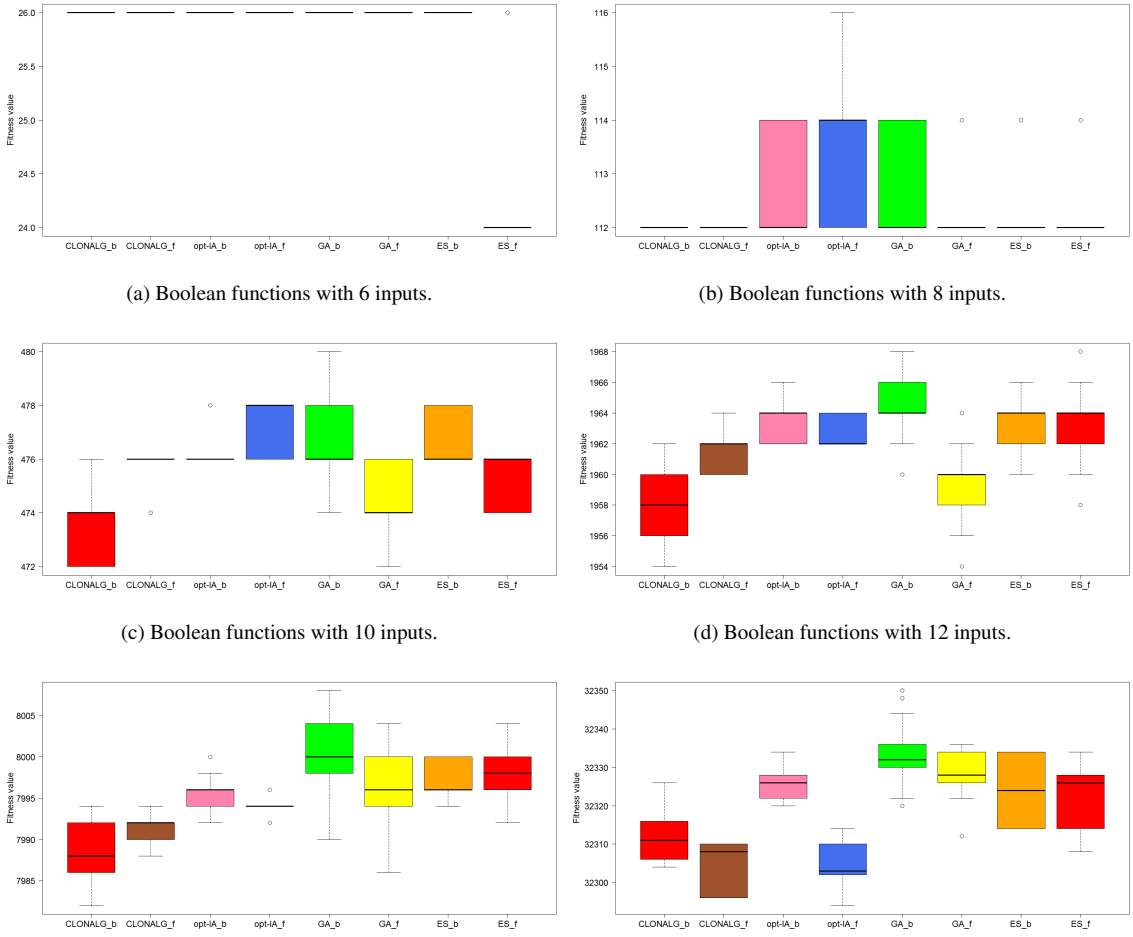
(e) Boolean functions with 14 inputs.

(f) Boolean functions with 16 inputs.

Figure 2: Boxplot results for all algorithms and Boolean function sizes, fitness function 1A.

6.3. The second objective

The goal of the second objective is to find balanced Boolean functions with as high as possible nonlinearity. In Figures 3a to 3f, we give results for fitness function 2. As in the first objective, the smallest Boolean function size does not represent a difficult problem since all algorithms succeed in reaching the global optimum value. For Boolean functions with 8 inputs we observe that the best performing algorithm is opt-IA where it is important to note that the best obtained value equals 116 which also represents the best currently known nonlinearity value for balanced Boolean functions with 8 inputs. For all larger sizes (i.e. sizes larger than 8 inputs), genetic algorithm with the bitstring representation outperforms all other algorithms. When considering only the clonal selection algorithms, we see that opt-IA outperforms CLONALG. Furthermore, we observe that opt-IA with bitstring encoding works better for larger dimensions while opt-IA with the floating point encoding works better for smaller Boolean function sizes.



(a) Boolean functions with 6 inputs.

(b) Boolean functions with 8 inputs.

(c) Boolean functions with 10 inputs.

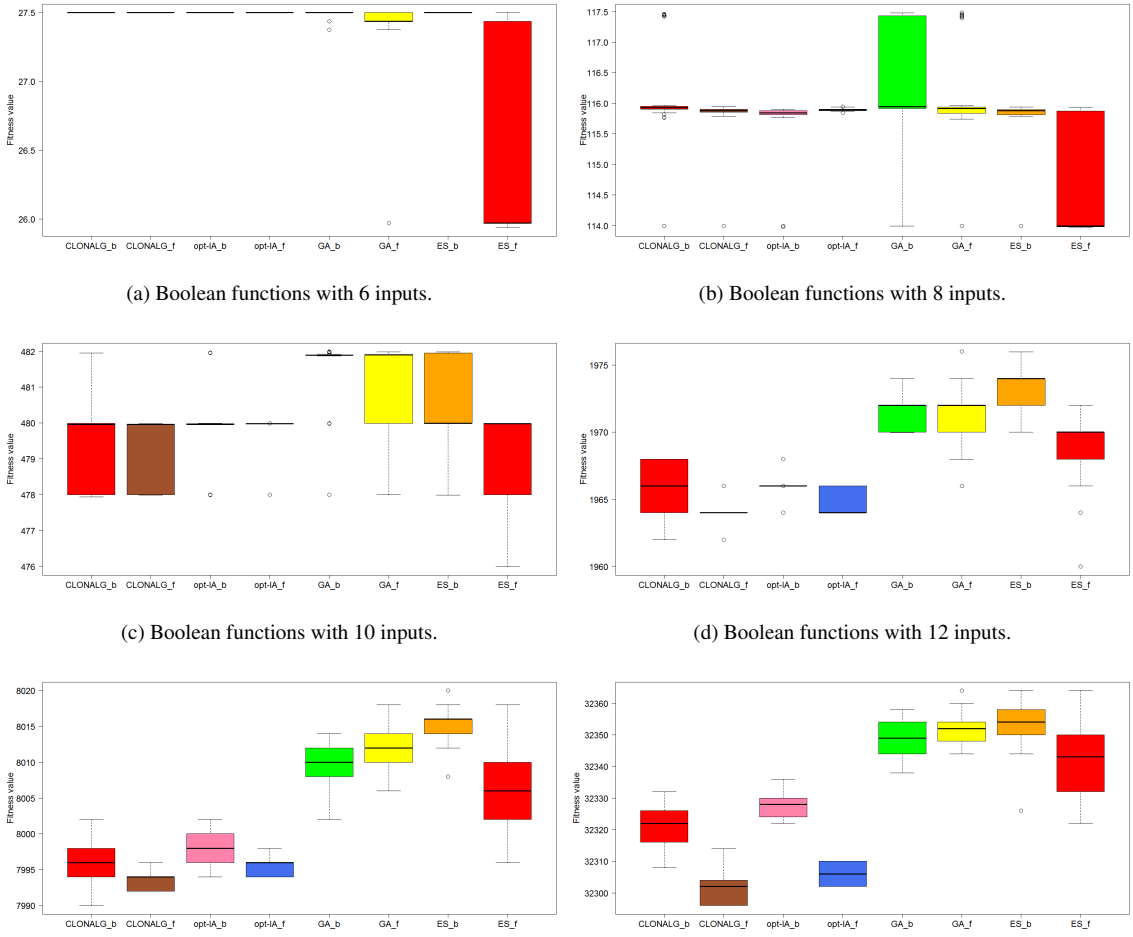
(d) Boolean functions with 12 inputs.

(e) Boolean functions with 14 inputs.

(f) Boolean functions with 16 inputs.

Figure 3: Boxplot results for all algorithms and Boolean function sizes, fitness function 2.

Finally, in Figures 4a to 4f, we give results for fitness function 2*A*. In this set of experiments, for smaller sizes the best performing algorithm is the GA with the bitstring encoding. For larger sizes the ES with the bitstring encoding gives the best results. Unfortunately, it seems that the clonal selection algorithms benefit less from the extra information in the fitness function when compared with the evolutionary algorithms. However, that does not mean that information is wasted, only that the clonal selection algorithms require more evaluations to converge.



(a) Boolean functions with 6 inputs.

(b) Boolean functions with 8 inputs.

(c) Boolean functions with 10 inputs.

(d) Boolean functions with 12 inputs.

(e) Boolean functions with 14 inputs.

(f) Boolean functions with 16 inputs.

Figure 4: Boxplot results for all algorithms and Boolean function sizes, fitness function 2A.

6.4. Discussion and Future Work

In this section, we try to give some general remarks regarding the experiments conducted. The first conclusion we can reach is that the truth table representation of Boolean functions performs worse when compared with the tree or graph representation as used in [29]. The authors there report similar results for fitness functions 1 and 2 as we obtain here. Since we now experiment with a completely different paradigm (clonal selection algorithms), a natural conclusion is that the difficulty of these problems lies in the Boolean function representation and not in the choice of the optimization algorithm. Since there is apparent connection between the truth table representation and the bitstring representation, it is hard to expect significant breakthroughs without the shift from the currently used truth table Boolean function representation. Therefore, as a future research direction we see the investigation of other unique Boolean function representations like the algebraic normal form or the numerical normal form [15]. With those representations, one can continue working with the bitstring encoding, but can also change far more straightforwardly to the floating point encoding.

Next, here we work with 500 000 evaluations in an effort to make the comparison with related works as easy as possible. However, our findings show that the clonal selection algorithms would benefit from more evaluations which therefore represents one apparent future research direction. To further investigate the convergence properties, we display the rate of convergence for the opt-IA algorithm for Boolean function with 12 inputs and fitness 2*A*. Figure 5 shows the values of best population individuals over all 50 independent runs. We can observe that the algorithm

converges before the evaluation limit, which indicates that there would be little progress with a larger number of evaluations.

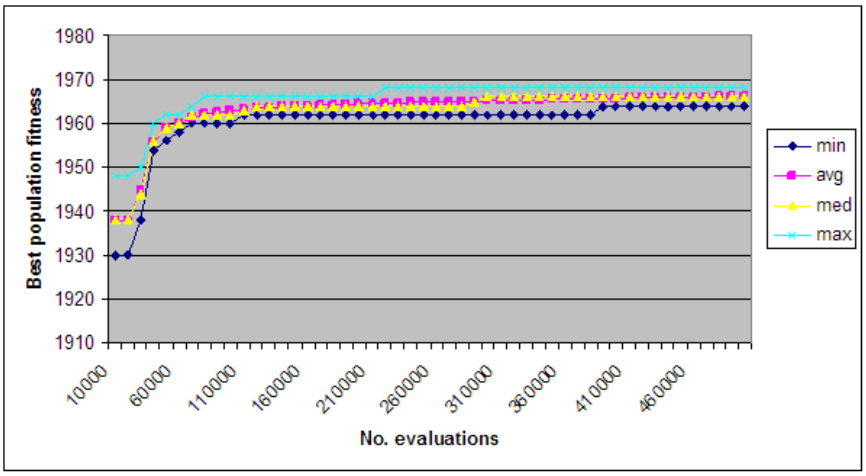


Figure 5: Convergence graph, opt-IA, fitness function 2A, 12 inputs

Of course, the question then could be whether those algorithms are truly comparable in the efficiency with e.g. genetic programming and Cartesian genetic programming since those algorithms succeed in obtaining superior results with 500 000 evaluations. Our intuition says no, but that does not apply only for the clonal selection algorithms, but for all heuristics we tested that rely on the bitstring representation.

Furthermore, our experiments show that there is some merit in using floating point representation for these problems as there are problem instances where that encoding reaches the best results. Naturally, it is hard to compare those results with related work since up to now floating point encoding for cryptographically suitable Boolean functions was completely neglected. Finally, we note that since here we work with dimension of Boolean functions of up to 16 inputs, there is also a gap in the literature concerning those larger Boolean functions sizes where we could find only a few papers considering such dimensions [27, 25].

When comparing CLONALG and opt-IA algorithms, the situation is far more straightforward; except for the smallest dimension where both algorithms consistently reach global optimum, opt-IA performs better. We note that the opt-IA with the bitstring representation works better for larger sizes and for smaller sizes the floating point representation behaves favourably. This is somewhat surprising, since with the increase of the number of inputs, the bitstring genotype size rises exponentially (up to 65536 in our experiments), which considerably expands the search space. The same increase applies to the floating point representation, but with a significantly smaller rate, so this difference in performance could be attributed to the modification operators used in corresponding representations.

Another interesting research avenue would be to investigate more relevant cryptographic properties and see how the aforesaid algorithms behave in those situations. With this step we would not only experiment with more difficult objectives, but also concentrate to functions that are really usable in practice, as we said that here we consider only a small subset of necessary properties for Boolean functions that are used in cryptography.

7. Conclusions

This paper concerns the efficiency of two well known immunological algorithms when applied to a combinatorial optimization problem from cryptographic domain. Furthermore, we compare different representations of solutions which allows the algorithms to be applied in both discrete and continuous domain. Considering the encodings, it is impossible to reach any definitive conclusion whether the bitstring or the floating point encoding should be used. However, we note that the results for the floating point encoding are sufficiently positive to deserve further investigation. By using this encoding, we believe it could be also easier to switch to some other unique Boolean function representation which would then open completely new research directions.

The results show that the opt-IA algorithm performs better than the CLONALG algorithm, but both perform slightly worse than the evolutionary algorithms we investigate. It can also be observed that the immunological algorithms are more invariant to the design of the fitness function, which may prove beneficial in case of lacking gradient information.

Acknowledgments

This work has been supported in part by Croatian Science Foundation under the project IP-2014-09-4882. In addition, this work was supported in part by the Research Council KU Leuven (C16/15/058) and IOF project EDA-DSE (HB/13/020).

[1] C. Paar, J. Pelzl, Understanding Cryptography - A Textbook for Students and Practitioners, Springer, 2010.
[2] J. Katz, Y. Lindell, Introduction to Modern Cryptography, 2nd Edition, Chapman and Hall/CRC, Boca Raton, 2015.
[3] L. R. Knudsen, M. Robshaw, The Block Cipher Companion, Information Security and Cryptography, Springer, 2011.
[4] A. Kerckhoffs, La cryptographie militaire, Journal des Sciences Militaires (1883) 161–191.
[5] M. Matsui, A. Yamagishi, A new method for known plaintext attack of FEAL cipher, in: Proceedings of the 11th annual international conference on Theory and application of cryptographic techniques, EUROCRYPT'92, Springer-Verlag, Berlin, Heidelberg, 1993, pp. 81–91.
[6] J. Daemen, V. Rijmen, The Design of Rijndael, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.
[7] FIPS 46-3, Data Encryption Standard (DES), National Institute for Standards and Technology (NIST), Gaithersburg, MD, USA (1999).
[8] C. Carlet, Vectorial Boolean Functions for Cryptography, in: Y. Crama, P. L. Hammer (Eds.), Boolean Models and Methods in Mathematics, Computer Science, and Engineering, 1st Edition, Cambridge University Press, New York, NY, USA, 2010, pp. 398–469.
[9] S. Picek, E. Marchiori, L. Batina, D. Jakobovic, Combining Evolutionary Computation and Algebraic Constructions to Find Cryptography-Relevant Boolean Functions, in: T. Bartz-Beielstein, J. Branke, B. Filipič, J. Smith (Eds.), Parallel Problem Solving from Nature - PPSN XIII, Lecture Notes in Computer Science, Springer International Publishing, 2014, pp. 822–831.
[10] J. Clark, J. Jacob, Two-Stage Optimisation in the Design of Boolean Functions, in: E. Dawson, A. Clark, C. Boyd (Eds.), Information Security and Privacy, Vol. 1841 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2000, pp. 242–254.
[11] S. Picek, D. Jakobovic, J. F. Miller, E. Marchiori, L. Batina, Evolutionary methods for the construction of cryptographic boolean functions, in: Genetic Programming - 18th European Conference, EuroGP 2015, Copenhagen, Denmark, April 8-10, 2015, Proceedings, 2015, pp. 192–204.
[12] L. Mariot, A. Leporati, Heuristic search by particle swarm optimization of boolean functions for cryptographic applications, in: Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015, Companion Material Proceedings, 2015, pp. 1425–1426.
[13] F. Burnet, The Clonal Selection Theory of Acquired Immunity, Cambridge University Press, Cambridge, UK, 1959.
[14] L. R. d. Castro, J. Timmis, Artificial Immune Systems: A New Computational Intelligence Paradigm, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.
[15] C. Carlet, Boolean Functions for Cryptography and Error Correcting Codes, in: Y. Crama, P. L. Hammer (Eds.), Boolean Models and Methods in Mathematics, Computer Science, and Engineering, 1st Edition, Cambridge University Press, New York, NY, USA, 2010, pp. 257–397.
[16] R. Forrié, The Strict Avalanche Criterion: Spectral Properties of Boolean Functions and an Extended Definition, in: S. Goldwasser (Ed.), Advances in Cryptology - CRYPTO' 88, Vol. 403 of Lecture Notes in Computer Science, Springer New York, 1990, pp. 450–468.
[17] B. Preneel, W. Van Leekwijck, L. Van Linden, R. Govaerts, J. Vandewalle, Propagation characteristics of Boolean functions, in: Proceedings of the workshop on the theory and application of cryptographic techniques on Advances in cryptology, EUROCRYPT '90, Springer-Verlag New York, Inc., New York, NY, USA, 1991, pp. 161–173.
[18] O. Rothaus, On bent functions, Journal of Combinatorial Theory, Series A 20 (3) (1976) 300 – 305.
[19] J. Dillon, A Survey of Bent Functions*, Tech. rep., Reprinted from the NSA Technical Journal. Special Issue, unclassified (1972).
[20] P. Sarkar, S. Maitra, Nonlinearity Bounds and Constructions of Resilient Boolean Functions, in: M. Bellare (Ed.), Advances in Cryptology - CRYPTO 2000, Vol. 1880 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2000, pp. 515–532.
[21] T. W. Cusick, P. Stănică, Cryptographic Boolean Functions and Applications, Elsevier Inc., San Diego, USA, 2009.
[22] W. Millan, A. Clark, E. Dawson, An Effective Genetic Algorithm for Finding Highly Nonlinear Boolean Functions, in: Proceedings of the First International Conference on Information and Communication Security, ICICS '97, Springer-Verlag, London, UK, UK, 1997, pp. 149–158.
[23] W. Millan, A. Clark, E. Dawson, Heuristic design of cryptographically strong balanced Boolean functions, in: Advances in Cryptology - EUROCRYPT '98, 1998, pp. 489–499.
[24] H. Aguirre, H. Okazaki, Y. Fuwa, An Evolutionary Multiobjective Approach to Design Highly Non-linear Boolean Functions, in: Proceedings of the Genetic and Evolutionary Computation Conference GECCO'07, 2007, pp. 749–756.
[25] J. McLaughlin, J. A. Clark, Evolving balanced Boolean functions with optimal resistance to algebraic and fast algebraic attacks, maximal algebraic degree, and very high nonlinearity, Cryptology ePrint Archive, Report 2013/011, `http://eprint.iacr.org/` (2013).
[26] S. Picek, D. Jakobovic, M. Golub, Evolving Cryptographically Sound Boolean Functions, in: Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation, GECCO '13 Companion, ACM, New York, NY, USA, 2013, pp. 191–192.
[27] R. Hrbacek, V. Dvorak, Bent Function Synthesis by Means of Cartesian Genetic Programming, in: T. Bartz-Beielstein, J. Branke, B. Filipič, J. Smith (Eds.), Parallel Problem Solving from Nature - PPSN XIII, Vol. 8672 of Lecture Notes in Computer Science, Springer International Publishing, 2014, pp. 414–423.
[28] S. Picek, C. Carlet, D. Jakobovic, J. F. Miller, L. Batina, Correlation Immunity of Boolean Functions: An Evolutionary Algorithms Perspective, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015, 2015, pp. 1095–1102.

[29] S. Picek, D. Jakobovic, J. F. Miller, L. Batina, M. Cupic, Cryptographic boolean functions: One output, many design criteria, Applied Soft Computing 40 (2016) 635 – 653.
[30] S. Picek, C. Carlet, S. Guilley, J. F. Miller, D. Jakobovic, Evolutionary algorithms for boolean functions in diverse domains of cryptography, Evolutionary computation.
[31] S. Picek, D. Jakobovic, Evolving Algebraic Constructions for Designing Bent Boolean Functions, in: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016, 2016, pp. 781–788.
[32] J. A. Clark, J. Jacob, S. Maitra, P. Stănică, Almost Boolean functions: the design of Boolean functions by spectral inversion, in: Evolutionary Computation, 2003. CEC '03. The 2003 Congress on, Vol. 3, 2003, pp. 2173–2180 Vol.3.
[33] L. Mariot, A. Leporati, A genetic algorithm for evolving plateaued cryptographic boolean functions, in: Theory and Practice of Natural Computing - Fourth International Conference, TPNC 2015, Mieres, Spain, December 15-16, 2015. Proceedings, 2015, pp. 33–45.
[34] J. Clark, J. Jacob, Two-Stage Optimisation in the Design of Boolean Functions, in: E. Dawson, A. Clark, C. Boyd (Eds.), Information Security and Privacy, Vol. 1841 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2000, pp. 242–254.
[35] L. de Castro, F. Von Zuben, Learning and optimization using the clonal selection principle, Evolutionary Computation, IEEE Transactions on 6 (3) (2002) 239–251.
[36] L. N. D. Castro, F. J. V. Zuben, The Clonal Selection Algorithm with Engineering Applications, in: In GECCO 2002 - Workshop Proceedings, Morgan Kaufmann, 2002, pp. 36–37.
[37] V. Cutello, G. Nicosia, M. Pavone, Exploring the Capability of Immune Algorithms: A Characterization of Hypermutation Operators, in: G. Nicosia, V. Cutello, P. Bentley, J. Timmis (Eds.), Artificial Immune Systems, Vol. 3239 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2004, pp. 263–276.
[38] V. Cutello, G. Narzisi, G. Nicosia, M. Pavone, Clonal Selection Algorithms: A Comparative Case Study Using Effective Mutation Potentials, in: C. Jacob, M. Pilat, P. Bentley, J. Timmis (Eds.), Artificial Immune Systems, Vol. 3627 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2005, pp. 13–28.
[39] A. E. Eiben, J. E. Smith, Introduction to Evolutionary Computing, 2nd Edition, Springer-Verlag, Berlin Heidelberg New York, USA, 2015.
[40] R. L. Haupt, S. E. Haupt, Practical Genetic Algorithms, John Wiley & Sons, Inc., New York, NY, USA, 1998.
[41] T. Bäck, D. Fogel, Z. Michalewicz (Eds.), Evolutionary Computation 1: Basic Algorithms and Operators, Institute of Physics Publishing, Bristol, 2000.
[42] G. Rozenberg, T. Bäck, J. N. Kok, Handbook of Natural Computing, 1st Edition, Springer Publishing Company, Incorporated, 2011.
[43] H.-G. Beyer, H.-P. Schwefel, Evolution Strategies A Comprehensive Introduction, Natural Computing 1 (1) (2002) 3–52.