Heuristic Search by Particle Swarm Optimization of Boolean Functions for Cryptographic Applications

Luca Mariot[1] and Alberto Leporati[1]

[1]Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano-Bicocca, Viale Sarca 336, 20126 Milano, Italy ,
{luca.mariot, leporati}@disco.unimib.it

Abstract

A Particle Swarm Optimizer for the search of balanced Boolean functions with good cryptographic properties is proposed in this paper. The algorithm is a modified version of the permutation PSO by Hu, Eberhart and Shi which preserves the Hamming weight of the particles positions, coupled with the Hill Climbing method devised by Millan, Clark and Dawson to improve the nonlinearity and deviation from correlation immunity of Boolean functions. The parameters for the PSO velocity equation are tuned by means of two meta-optimization techniques, namely Local Unimodal Sampling (LUS) and Continuous Genetic Algorithms (CGA), finding that CGA produces better results. Using the CGA-evolved parameters, the PSO algorithm is then run on the spaces of Boolean functions from $n = 7$ to $n = 12$ variables. The results of the experiments are reported, observing that this new PSO algorithm generates Boolean functions featuring similar or better combinations of nonlinearity, correlation immunity and propagation criterion with respect to the ones obtained by other optimization methods.

Keywords Particle Swarm Optimization, Boolean Functions, Cryptography, Hill Climbing, Meta-optimization, Local Unimodal Sampling, Continuous Genetic Algorithms

1 Introduction

Boolean functions are fundamental in several symmetric cryptography applications. They are widely used to design *Substitution Boxes* (S-Boxes) for block ciphers, and in certain types of stream ciphers such as the *combiner model* and the *filter model* [2]. In order to withstand specific cryptanalytic attacks, the Boolean functions adopted in these ciphers must meet several

cryptographic properties, some of which include *balancedness*, high *nonlinearity* and *algebraic degree*, low *absolute indicator*, *correlation immunity* and *propagation criterion*.

Most of these properties cannot be satisfied simultaneously, since they induce several theoretical bounds and constraints among them. On the other hand, an exhaustive exploration to find the Boolean functions of n variables achieving the best trade-off with respect to a specific set of properties is not feasible in general, since the cardinality of the corresponding search space is 2^{2^n}. As a consequence, the optimization of the cryptographic properties of Boolean functions is an important open problem in the design of symmetric ciphers.

Several heuristic techniques have been developed in the literature to discover Boolean functions satisfying good combinations of cryptographic properties. Millan, Clark and Dawson [9] proposed a Genetic Algorithm (GA) coupled with Hill Climbing (HC) to evolve the truth tables of highly nonlinear balanced Boolean functions having low deviations from correlation immunity and propagation criterion. Later, Clark, Jacob, Maitra, Stepney and Millan [3] devised a Simulated Annealing (SA) procedure to optimize the nonlinearity and the absolute indicator of Boolean functions as well as their correlation immunity and propagation criteria, which achieved better performances than GA. Aguirre, Okazaki and Fuwa designed in [1] a multi-objective Random Bit Climber which was able to generate more efficiently Boolean functions having good nonlinearity and absolute indicator. Recently, Genetic Programming (GP) has also been used by Picek, Jacobovic and Golub [13] to evolve strong cryptographic Boolean functions of 8 variables.

The aim of this paper is to investigate the application of Particle Swarm Optimization (PSO) to the search of balanced Boolean functions with good cryptographic properties. In particular, we propose a modified version of the discrete PSO algorithm for permutation problems designed by Hu, Eberhart and Shi [4] by adapting it to the case of $\binom{2^n}{2^{n-1}}$ *combinations*, thus limiting the search space to *balanced* Boolean functions. This modification is implemented by a new update method for the positions of the particles, which preserves the Hamming weights of the truth tables. To further improve the nonlinearity and the deviation from correlation immunity of the candidate solutions, the modified PSO algorithm is also integrated with the Hill Climbing procedure by Millan, Clark and Dawson [9]. We address the problem of finding optimal values for the social and cognitive constants, inertia and maximum velocity parameters in the PSO velocity equation by using two meta-optimization techniques: Local Unimodal Sampling (LUS) and Continuous Genetic Algorithms (CGA). While the former has already been adopted in the literature to tune the parameters of PSO [12], to our knowledge CGA have never been applied to this meta-optimization task. We compare the performances of LUS and CGA in tuning the PSO parameters for the case of Boolean functions of $n = 7$ variables with three underlying fitness functions

(each targeting a different set of cryptographic properties), and observe that CGA achieve better results. We finally employ the parameters optimised through CGA to run the PSO algorithm on the spaces of Boolean functions of up to $n = 12$ variables. The results of the experiments are reported and compared with those achieved by other heuristic methods published in the literature, focusing only on the best solutions found. We observe that our PSO algorithm is able to find Boolean functions with similar or better combinations of nonlinearity, correlation immunity and propagation criterion than the ones produced by other methods, especially when the number of variables is less than 10. It is also found that the properties (especially the nonlinearity) get worse as the number of variables increases, suggesting that further parameters tuning is required in this case.

The remainder of this paper is structured as follows. Section 2 gives some basic definitions and facts about Boolean functions and their cryptographic properties. Section 3 describes the modified PSO algorithm, focusing on the update method for the particle positions which preserves their Hamming weights, and defines the fitness functions employed in the experiments. Section 4 deals with the parameter tuning problem, briefly introducing the two meta-optimization techniques used (LUS and CGA) and then reporting their results. Section 5 describes the experiments performed on the PSO algorithm with the CGA-evolved parameters and reports the results, comparing them with those obtained by other optimization methods. Section 6 concludes the paper, and gives some directions for further research on the subject.

2 Basics of Boolean Functions

In this section we recall the basic notions about Boolean functions and their cryptographic properties which will be used throughout the paper. Where not otherwise specified, the reader may refer to [2] for a more detailed discussion of this topic.

Let $\mathbb{F}_2$ be the finite field with two elements, and $\mathbb{F}_2^n$ the $\mathbb{F}_2$-vector space of binary n-tuples. In what follows, given a binary vector x, by $w_H(x)$ we denote the *Hamming weight* of x, that is, the number of nonzero coordinates in x. A *Boolean function* of n variables is a mapping $f : \mathbb{F}_2^n \to \mathbb{F}_2$. The *truth table* of f is the binary string Ω_f of length 2^n which specifies, for all inputs $x = (x_1, \cdots, x_n) \in \mathbb{F}_2^n$, the corresponding output value $f(x)$. The first fundamental cryptographic property which can be defined using the truth table representation is balancedness:

Definition 1. *A Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is* balanced *if $w_H(\Omega_f) = 2^{n-1}$, i.e. the truth table of f is composed of an equal number of zeros and ones.*

Balancedness is a fundamental cryptographic property, since biases in the

distribution of zeros and ones can be exploited for linear and differential cryptanalysis.

Another common representation of Boolean functions is the *Algebraic Normal Form* (ANF). Given $f : \mathbb{F}_2^n \to \mathbb{F}_2$ and $x \in \mathbb{F}_2^n$, the ANF associated to f is a multivariate polynomial $P_f(x)$ of the following form:

$$P_f(x) = \bigoplus_{I \subseteq [n]} a_I \cdot \left(\prod_{i \in I} x_i \right) \ ,$$

where $\oplus$ and $\cdot$ respectively denote sum and product over $\mathbb{F}_2$, and $[n] = \{1, \cdots, n\}$. For all $I \subseteq [n]$, the coefficient a_I is uniquely determined as follows:

$$a_I = \bigoplus_{x \in \mathbb{F}_2^n : \mathcal{S}(x) \subseteq I} f(x) \ ,$$

where $\mathcal{S}(x) = \{i \in [n] : x_i \neq 0\}$ is the *support* of x. The ANF is useful to define the algebraic degree of a Boolean function:

Definition 2. *The* algebraic degree *of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is the degree of the largest nonzero monomial in its ANF $P_f(x)$. Formally, $deg(f)$ is defined as*

$$deg(f) = max\{|I| : I \subseteq [n], a_I \neq 0\} \ .$$

Boolean functions having degree 1 are called *affine functions*. The algebraic degree of Boolean functions employed in stream and block ciphers should be as high as possible, in order to resist attacks based on the *Berlekamp-Massey algorithm* in the former case and higher order differential attacks in the latter.

Several cryptographic properties of Boolean functions can be characterised by means of the *Walsh transform*. Given $f : \mathbb{F}_2^n \to \mathbb{F}_2$, the Walsh transform $\hat{F} : \mathbb{F}_2^n \to \mathbb{R}$ of f is defined for all $\omega \in \mathbb{F}_2^n$ as

$$\hat{F}(\omega) = \sum_{x \in \mathbb{F}_2^n} \hat{f}(x) \cdot (-1)^{x \cdot \omega} \ ,$$

where $\hat{f}(x) = (-1)^{f(x)}$ and $x \cdot \omega = x_1 \cdot \omega_1 \oplus \cdots \oplus x_n \cdot \omega_n$ is the scalar product between x and ω. A Boolean function is balanced if and only if $\hat{F}(\underline{0}) = 0$, where $\underline{0}$ is the null vector of $\mathbb{F}_2^n$. We denote by $W_{max}(f)$ the *spectral radius* of f, which is the maximum absolute value of $\hat{F}(\omega)$ for $\omega \in \mathbb{F}_2^n$. The spectral radius can be used to characterise the nonlinearity of a Boolean function:

Definition 3. *The* nonlinearity *$Nl(f)$ of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ with spectral radius $W_{max}(f)$ is the minimum Hamming distance of f from the set of affine functions, and it is computed as follows:*

$$Nl(f) = 2^{n-1} - \frac{1}{2} W_{max}(f) \ .$$

Boolean functions having high nonlinearity provide better confusion in both stream and block ciphers, making linear cryptanalysis harder. It is known that, if the number of variables n is even, the class of *bent functions* reaches the maximum value of nonlinearity $2^{n-1} - 2^{\frac{n-2}{2}}$. However, such functions are not balanced, thus they cannot be used in the design of symmetric cryptosystems. Determining the maximum nonlinearity for non-bent Boolean functions when n is even, or for generic Boolean functions when n is odd, is still an open problem for all $n > 7$.

A second important cryptographic criterion which can be defined using the Walsh transform is *correlation immunity*:

Definition 4. *Given $k \in \{1, \cdots, n\}$, a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is k-th order* correlation immune *(denoted by $CI(k)$) if by fixing at most k input coordinates the truth tables of the corresponding restrictions of f all have the same Hamming weight. This condition is verified if and only if $\hat{F}(\omega) = 0$ for all $\omega \in \mathbb{F}_2^n$ such that $1 \leq w_H(\omega) \leq k$.*

A balanced Boolean function which is also k-th order correlation immune is called *k-resilient*. Boolean functions used in stream ciphers based on the combiner model should be resilient of high order to resist correlation attacks. In order to define suitable fitness functions in the next section, we also adopt the following *deviation from correlation immunity*, originally introduced in [9]:

Definition 5. *The* deviation from k-th order correlation immunity *of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is defined as*

$$cidev_k(f) = max\{|\hat{F}(\omega)| : \omega \in \mathbb{F}_2^n,\ 1 \leq w_h(\omega) \leq k\}\ .$$

The *autocorrelation function* $\hat{r} : \mathbb{F}_2^n \to \mathbb{R}$ of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is defined for all $s \in \mathbb{F}_2^n$ as:

$$\hat{r}(s) = \sum_{x \in \mathbb{F}_2^n} \hat{f}(x) \cdot \hat{f}(x \oplus s)$$

The maximum absolute value AC_{max} for $s \in \mathbb{F}_2^n \setminus \{\underline{0}\}$ of the autocorrelation function is called the *absolute indicator* of f. This quantity should be low in order to ensure good diffusion both in stream and block ciphers. Another cryptographic property related to the diffusion of cryptosystems which can be characterised by the autocorrelation function is the *propagation criterion*:

Definition 6. *Given $l \in \{1, \cdots, n\}$, a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ satisfies the* propagation criterion *$PC(l)$ if, for all nonzero vectors $s \in \mathbb{F}_2^n$ such that $w_H(s) \leq l$, the function $f(x) \cdot f(x \oplus s)$ is balanced. This condition is met if and only if $\hat{r}(s) = 0$ for all $s \in \mathbb{F}_2^n$ such that $1 \leq w_H(s) \leq l$.*

The propagation criterion $PC(1)$ corresponds to the *Strict Avalanche Criterion* (SAC), which states that by complementing a single input coordinate x_i the probability that the output of f will change is $1/2$. Similarly to correlation immunity, we adopt the following *deviation from propagation criterion*:

Definition 7. *Given $f : \mathbb{F}_2^n \to \mathbb{F}_2$, the* deviation from propagation criterion *$PC(l)$ of f is defined as*

$$pcdev_l(f) = max\{|\hat{r}(s)| : s \in \mathbb{F}_2^n,\ 1 \leq w_h(s) \leq l\} \ .$$

Most of the cryptographic criteria described above are not simultaneously satisfiable. In particular, given a k-resilient Boolean function of n variables $f : \mathbb{F}_2^n \to \mathbb{F}_2$ having algebraic degree $deg(f)$ and nonlinearity $Nl(f)$, the following two bounds hold:

- *Siegenthaler's bound*: $deg(f) \leq n - 1 - k$
- *Tarannikov's bound*: $Nl(f) \leq 2^{n-1} - 2^{k+1}$.

3 PSO Algorithm

3.1 Overview of Discrete PSO

Particle Swarm Optimization (PSO) is a stochastic optimization heuristic originally introduced by Kennedy and Eberhart [5]. PSO basically works by representing a set of candidate solutions of an optimization problem as a *swarm of particles* which move in a coordinated manner through the search space, usually a subset of $\mathbb{R}^m$. At each time step $t \in \mathbb{N}$, the current *position* of the i-th particle $x_i^{(t)} \in \mathbb{R}^m$ is updated using the recurrence equation

$$x_i^{(t+1)} = x_i^{(t)} + v_i^{(t)} \ ,$$

where $v_i^{(t)} \in \mathbb{R}^m$ denotes the *velocity vector* of the i-th particle at time t. The candidate solution represented by the new position is then evaluated by a *fitness function*, which is usually the function to be optimized. Each coordinate $j \in \{1, \cdots, m\}$ of the i-th particle velocity is in turn stochastically updated as follows:

$$v_{ij}^{(t+1)} = w \cdot v_{ij}^{(t)} + R_{ij} \cdot \varphi \cdot (g_j - x_{ij}^{(t)}) + R_{ij} \cdot \psi \cdot (b_{ij} - x_{ij}^{(t)}) \ ,$$

where the current velocity of the particle $v_{ij}^{(t)}$ is weighted by the *inertia* parameter $w \in \mathbb{R}$, the value $R_{ij} \in [0, 1]$ is a random number sampled with uniform probability, and φ and ψ are constants which respectively determine the influence of the *global best* solution $g \in \mathbb{R}^m$ found so far

by the *neighborhood* of the i-th particle and the influence of the *local best* solution $b_i \in \mathbb{R}^m$ found so far by the i-th particle. In order to control the velocity of the particle, each component $v_{ij}^{(t+1)}$ is also limited in absolute value by a global parameter v_{max}. Various topologies can be used to define a neighborhood for the particles, such as the *Von Neumann* topology and the *ring* topology. In this work, however, we focus only on the *fully informed particle* paradigm [7], in which the global best solution g is simply the best solution discovered so far by the whole swarm.

The PSO heuristic has been successfully applied to several continuous optimization problems (see for example [14] for a detailed survey). However, there are no obvious ways to apply it to discrete search spaces.

Kennedy and Eberhart proposed in [6] a variant of their original PSO algorithm in order to solve binary optimization problems. The solutions are represented as vectors of m bits, and the search space is geometrically interpreted as the m-dimensional hypercube $\mathbb{F}_2^m$. Consequently, the particles move through the vertices of this hypercube. The velocity vector becomes a *probability vector*: given the i-th particle in the swarm, for each coordinate $j \in \{1, \cdots, m\}$ the position x_i with respect to dimension j is updated by sampling a Bernoullian random variable with parameter p_{ij}. In particular, if a sampled random number $r \in [0, 1]$ is less than p_{ij} then the value of the j-th coordinate of x_i is updated to 1, otherwise it is updated to 0.

The advantage of this discrete version is that it is possible to use the same velocity equation defined for the basic PSO procedure to update the probability vectors of the particles, provided that their components are normalized on the interval $[0, 1]$. To this end, Kennedy and Eberhart adopted in [6] the *logistic function*, defined for all $x \in \mathbb{R}$ as:

$$S(x) = \frac{1}{1 + \exp(-x)} \ .$$

3.2 Position Update for Balanced Functions

Using the truth table representation, the discrete PSO heuristic described in the previous section can be straightforwadly applied to the optimization problem of finding good cryptographic properties of Boolean functions of n variables. In this case, the particles would move in the space of $m = 2^n$ binary vectors. However, the method proposed in [6] to update the positions of the particles does not give any control over their Hamming weights, since each component in the probability vector is sampled independently from the others. Hence, there are no guarantees that the generated truth tables will be balanced, a fundamental property for cryptographic Boolean functions. A possible solution to this drawback is to add an *unbalancedness* penalty in the fitness function, an approach which has been followed in [13] for Genetic Algorithms and Genetic Programming. Our preliminary experiments however showed that this method is not satisfactory with PSO, since the proportion

of generated balanced functions is really low. Thus, it is necessary to use an update operator which limits the search space to the set of balanced Boolean functions.

Hu, Eberhart and Shi [4] adapted the discrete PSO algorithm in order to apply it to *permutation problems*. Their update method works by stochastically *swapping* the values in the permutation vector which represents the position. In particular, the component x_{ij} of the i-th particle is changed with probability p_{ij} by swapping it with x_{ik}, where k is such that $x_{ik} = g_j$. As a consequence, the permutation represented by vector x_i is adjusted by making it more similar to the global best solution g.

From a combinatorial point of view, the set of balanced Boolean functions of n variables is isomorphic to the set of $\binom{2^n}{2^{n-1}}$ combinations. In fact, a subset of 2^{n-1} out of 2^n objects can be represented by its *characteristic function*, which is basically a balanced binary vector $x \in \mathbb{F}_2^m$, where $m = 2^n$. Starting from this observation, we generalised the update operator proposed by Hu, Eberhart and Shi to the case of balanced combinations. Given the balanced binary vector $x_i \in \mathbb{F}_2^m$ and the corresponding probability vector $p_i \in [0,1]^m$, for each coordinate $j \in \{1, \cdots, m\}$ a random number $r \in [0,1]$ is sampled with uniform probability, and if r is less than p_{ij} then a swap is performed as follows. First, the value of x_{ij} is compared with that of the global best in the same index, g_j. If the two values are equal, then no action is taken. Otherwise, the bit in x_{ij} is swapped with another bit x_{ik}, where $k \neq j$ is such that $x_{ik} \neq g_k$ and $x_{ik} \neq x_{ij}$. These two conditions ensure that, while the Hamming weight of the vector is preserved, its Hamming distance from the global best solution is decreased by 2. In fact, if only $x_{ik} \neq g_k$ is verified, swapping the values of x_{ij} and x_{ik} yields the same Hamming distance between x_i and g. Since there can be more than one index k which satisfies these two conditions, our update operator randomly selects one of them.

The whole update process is then repeated using the local best b_i of the i-th particle instead of the global best g. In this way, x_i is changed by considering both the social attraction of the whole swarm and the cognitive attraction of the particle. Moreover, if the current position of the particle is equal to g, a random pair of bits in x_i is swapped in order to avoid premature convergence, a solution similar to the one proposed in [4].

Algorithm 1 reports the general pseudocode implementing our position update operator. The input parameters x_i and y are balanced binary vectors which respectively represent the position of the i-th particle in the swarm and either the position of the global best g or local best b_i. We assume that $x_i \neq y$. Vector p_i is the probability vector associated to the i-th particle and $m = 2^n$ is the length of x_i. The procedure RAND-UNIF() samples a random number $r \in [0,1]$ with uniform probability, which is used to determine whether a swap is required by comparing it to p_{ij}. The subroutine FIND-CAND-SWAP(), whose details are omitted, performs the search of a suitable index for the

swap, returning 0 if it cannot find one.

Algorithm 1 UPDATE-BAL-POS(x_i, y, p_i, m)

```
for j := 1 to m do
  r := RAND-UNIF()
  if (r ¡ p_ij AND x_ij ≠ y_j) then
    k := FIND-CAND-SWAP(x_i, j)
    if (k ≠ 0) then
      Swap x_ij with x_ik
    end if
  end if
end for
```

3.3 Fitness Functions

We tested our Particle Swarm Optimizer with three fitness functions, all of which have to be maximised. The first function fit_1 considers the three properties of nonlinearity, deviation from first order correlation immunity and deviation from the Strict Avalanche Criterion:

$$fit_1(f) = Nl(f) - \frac{cidev_1(f)}{4} - \frac{pcdev_1(f)}{8} \ .$$

Since the values of the Walsh and autocorrelation spectra of a balanced Boolean function are respectively multiples of 4 and 8, the two deviations in fit_1 are normalized by these two factors. This fitness function closely resembles those defined in [9] for Genetic Algorithms, where it is proposed either to minimise the *normalised deviation* of the Boolean function, defined as the maximum value between $cidev_k(f)/4$ and $pcdev_l(f)/8$, or to maximise the difference between nonlinearity and $cidev_k(f)$. We adopted the latter as our second fitness fuction, with $k = 2$:

$$fit_2(f) = Nl(f) - cidev_2(f) \ .$$

Finally, our third fitness function targets the nonlinearity and the absolute indicator of Boolean functions, two criteria which have been optimized together by several heuristic methods proposed in the literature [3, 1, 13]:

$$fit_3(f) = Nl(f) - AC_{max}(f) \ .$$

It can be observed that none of the above fitness functions takes into account the algebraic degree, as opposed for example to the ones employed in [13]. The motivation for this choice is twofold. First, algebraic degree is a property which is easier to optimize than nonlinearity or correlation immunity, the reason being that as $n \to \infty$, the algebraic degree of a random Boolean

function of n variables is almost surely $n-1$ [2]. Hence, heuristic methods using algebraic degree in their fitness functions are likely to find Boolean functions having maximum degree but which do not satisfy $CI(k)$ or $PC(l)$. Second, as it is shown in Section 5, our PSO algorithm is able to discover Boolean functions reaching Siegenthaler's bound, despite the fact that the algebraic degree is not considered in our fitness functions.

3.4 Overall PSO Algorithm

To further improve the performance of our Particle Swarm Optimizer, we combined it with the Hill Climbing (HC) algorithm designed by Millan, Clark and Dawson [9]. This technique works by swapping a pair of bits in the truth table of a balanced Boolean function in order to increase its nonlinearity and decrease its deviation from $CI(k)$. In what follows, we denote by NL-CI(K)-HC the HC procedure which increases nonlinearity while decreasing $cidev_k(f)$, while NL-HC stands for the HC targeted only at increasing nonlinearity. The reader is referred to [9] for further details about the general HC method.

The type of Hill Climbing performed by our PSO algorithm depends on the underlying fitness function: in the case of fit_1 and fit_2 respectively NL-CI(1)-HC and NL-CI(2)-HC are applied, while for fit_3 NL-HC is used.

We now summarise the overall procedure of our discrete Particle Swarm Optimizer:

1. Given a swarm of size N, for all $i \in \{1, \cdots, N\}$ initialize the i-th particle by randomly creating a balanced binary vector $x_i \in \mathbb{F}_2^m$ and a probability vector $p_i \in [0,1]^m$, where $m = 2^n$ and n is the number of variables of the Boolean functions.

2. Given $k \in \{1,2,3\}$, for all $i \in N$ compute the fitness value fit_k of solution x_i found by particle i.

3. Update the global best solution g and the local best solutions b_i, for all $i \in \{1, \cdots, N\}$.

4. For all $i \in \{1, \cdots, N\}$, update the probability vector v_i using the PSO velocity recurrence, and then normalize each coordinate through the logistic function.

5. For all $i \in \{1, \cdots, N\}$, update the position vector x_i. If $x_i = g$ or $x_i = b_i$ then swap a random pair of bits in x_i, otherwise invoke UPDATE-BAL-POS(x_i, g, p_i, m) and then UPDATE-BAL-POS(x_i, b_i, p_i, m).

6. Depending on the fitness function, apply to all the particles in the swarm the hill climbing optimization step NL-CI(K)-HC or NL-HC described in [9].

7. If the maximum number of iterations has been reached then output the global best solution g, otherwise return to step 2.

4 Parameters Tuning

4.1 Problem Statement

It has been widely shown in the literature that the choice of the velocity parameters greatly influences the performance of PSO [16, 17]. Instead of searching by trial-and-error a good combination of parameters for our PSO algorithm, we tackled the problem in a systematic way using a *meta-optimization* approach.

The main idea behind meta-optimization is to consider the selection of the parameters governing an optimizer O as an optimization problem itself. An *overlaying* meta-optimizer M is then applied to explore the parameters space, using a meta-fitness function to assess the performance of O under a given combination of parameters.

A candidate solution for the meta-optimization problem of our discrete PSO is thus a vector $(w, \varphi, \psi, v_{max}) \in \mathbb{R}^4$ which specifies the four parameters to be used in the velocity equation. Considering the observations reported in [6], we chose to limit the value of each parameter in the interval $[0, 10]$. For the choice of the overlaying meta-optimizer, we decided to test *Local Unimodal Sampling* (LUS) and *Continuous Genetic Algorithms* (CGA, also known as *Real-coded Genetic Algorithms*).

4.2 Local Unimodal Sampling

LUS is a local search technique which iteratively improves the current solution x by sampling with uniform probability a point y in its neighborhood $N(x)$. Considering a maximisation problem, if the fitness value of y is higher than that of x, the current solution is set to y. Otherwise, the size of $N(x)$ is decreased by a discount factor β, the rationale being that by sampling with a constant-size neighborhood the algorithm is not guaranteed to converge to a local optimum. The sampling process is then repeated until a termination criterion is met, which is usually a minimum threshold τ for the size of the neighborhood. Pedersen and Chipperfield [12] employed LUS to tune the velocity parameters of a Particle Swarm Optimizer aimed at training the weights of artificial neural networks.

4.3 Continuous Genetic Algorithms

CGA are a generalisation of Genetic Algorithms to continuous optimization problems, which represent the chromosome of a candidate solution using a vector of real numbers in place of a binary string. To our knowledge, CGA

have never been applied to tune PSO parameters. In the context of our meta-optimization problem, we employed the *flat operator* introduced by Radcliffe [15] as the crossover method of our CGA, while for the mutation procedure we relied on the simple *random operator* proposed by Michalewicz [8]. The reproduction operator is implemented using the *roulette wheel method*, which stochastically selects an individual with a probability proportional to its fitness. Specifically, given a population of P chromosomes, the next generation is created as follows. Using the roulette wheel method, $P/2$ pairs of chromosomes are formed, and for each pair (x, y) an offspring of two chromosomes (c_1, c_2) is created by applying with probability p_c the flat crossover operator (if it is not applied, the chromosome pair (x, y) is simply reproduced unaltered). The random mutation operator is then employed with probability p_m to each locus of the chromosomes in the offspring. We also used an *elitist strategy* to ensure that the best individual is preserved in the next generation.

4.4 Meta-Fitness Function

The *meta-fitness function*, used to drive the search for a good combination of PSO parameters, is clearly the most intensive step from a computational point of view. In fact, given a vector $x \in \mathbb{R}^4$ several runs of our discrete PSO algorithm must be performed, in order to have a statistically significant measure of its performance under the parameters specified by x. In particular, we chose to test the case of balanced Boolean functions defined on $n = 7$ variables, using a swarm of $N = 50$ particles evolved for $I = 100$ iterations. The PSO algorithm is executed for $R = 30$ independent runs, and at each run the fitness of the global best solution g at the last iteration is recorded. According to Pedersen and Chipperfield [12], the average fitness value μ_g of the global best over all the R optimization runs should be used as the meta-fitness function. Since from a cryptographic point of view we are interested in Boolean functions satisfying the best possible properties, we also considered the maximum fitness value max_g achieved by the global best over R runs. It is known that for Boolean functions of 7 variables the maximum value of nonlinearity is $Nl_{max} = 56$ [11]. Hence, the maximum value achievable by max_g is 56 with respect to fitness functions fit_1 and fit_2. No function of 7 variables having absolute indicator $AC_{max} < 16$ has ever been reported in the literature, and it has been conjectured that 16 is the highest lower bound [18]. As a consequence, under the current state of knowledge the maximum value reachable by max_g with respect to fitness function fit_3 is $56 - 16 = 40$.

Given a parameter vector $x \in \mathbb{R}^4$, our meta-fitness function can thus be defined as

$$mfit_k(x) = \mu_g + max_g \ ,$$

where $k \in \{1, 2, 3\}$ indicates the fitness function fit_k which is being optimized

by the PSO algorithm.

4.5 Meta-Optimization Results

Following the methodology described in [12], for each underlying fitness function fit_k we performed $M = 6$ runs to assess the performances of both LUS and CGA, thus carring out a total of 36 meta-optimization experiments. In the case of LUS we adopted the value $\beta = 0.33$ for the discount factor and $\tau = 0.001$ for the minimum threshold of the neighborhood size. On the other hand, for the CGA meta-optimizer we used a population of $P = 20$ individuals evolved for $G = 100$ generations, setting the crossover and mutation probability respectively to $p_c = 0.95$ and $p_m = 0.05$.

Table 1 compares the best parameters combinations found by LUS and CGA over the 6 meta-optimization runs, for each fitness function fit_k.

Table 1: Comparison of Best PSO Parameters

fit_k	Method	μ_g	$\max_g$	$mfit_k(f)$
fit_1	LUS	52.7	56	108.7
	CGA	53	56	109
fit_2	LUS	46	52	98
	CGA	46.27	56	102.27
fit_3	LUS	30.87	40	70.86
	CGA	38.4	40	78.4

It can be observed that CGA outperforms LUS with respect to all three fitness functions. While in the case of fit_1 there is only a slight difference concerning the average fitness values μ_g, for fit_2 the best parameter combination found by LUS did not allow the Particle Swarm Optimizer to reach the maximum fitness value of 56, whereas for fit_3 the mean fitness μ_g of LUS is remarkably lower than that achieved by CGA. However, the higher performance of CGA is associated with a higher computational cost, since Genetic Algorithms are a population-based heuristic. As a matter of fact, in our experimental setting a single CGA meta-optimization run required $P \cdot G \cdot R \cdot N \cdot I = 3.0 \cdot 10^8$ fitness evaluations, which took almost 17 hours to complete on a 64-bit Linux machine, with a Core i5 architecture and a CPU running at 2.8 GHz. On the other hand, with the selected β and τ parameters LUS performed an average of 4971 fitness evaluations per single meta-optimization run before reaching the minimum threshold, roughly corresponding to 1.3 hours of CPU time on the same machine.

5 PSO Experiments

5.1 Experimental Setting

We now describe the experiments performed with our Particle Swarm Optimizer. Regarding the velocity parameters, we adopted the best combination evolved by the CGA meta-optimizer, since it achieved an higher meta-fitness value with respect to the ones obtained by LUS. The values of the selected parameters for each fitness function are reported in Table 2. We applied our

Table 2: CGA-Evolved PSO Parameters

fit_k	w	φ	ψ	v_{max}
fit_1	0.5067	2.8751	1.3587	3.5008
fit_2	0.7614	2.0073	2.0273	2.7183
fit_3	0.2828	2.1824	0.8951	4.2639

PSO algorithm on the spaces of balanced Boolean functions from $n = 7$ to $n = 12$ variables. The number of particles and iterations were set to $P = 200$ and $I = 400$ respectively. Finally, for each value of n and fitness function fit_k, we carried out $R = 100$ PSO runs.

5.2 Best Solutions Found

Tables 3 to 5 show for each fitness function the cryptographic properties of the best balanced Boolean functions discovered by PSO, that is, the properties of the global best solution g which scored the highest fitness value among all the $R = 100$ optimization runs. We reported the algebraic degree as well, even if we did not adopt this criterion in any of the three fitness functions.

Table 3: Best Boolean Functions Found, fit_1

Property	7	8	9	10	11	12
Nl	56	112	236	480	972	1972
deg	5	6	7	8	9	10
$cidev_1$	0	0	0	0	0	0
$pcdev_1$	0	0	8	8	8	8

As a general observation, one can notice in Tables 3 and 4 that the Boolean functions discovered by PSO satisfying $CI(k)$ always have an algebraic degree of $n - 1 - k$, which is the maximum allowed by Siegenthaler's bound. Hence, these results empirically confirm that it is not necessary to consider the algebraic degree in the definition of the PSO fitness functions, as we mentioned in Section 3.4.

Table 4: Best Boolean Functions Found, fit_2

Property	7	8	9	10	11	12
Nl	56	112	232	476	972	1972
deg	4	6	7	8	9	10
$cidev_1$	0	8	8	8	8	16
$cidev_2$	0	8	8	8	8	16

Table 5: Best Boolean Functions Found, fit_3

Property	7	8	9	10	11	12
Nl	56	116	236	480	976	1972
deg	5	6	7	9	10	11
AC_{max}	16	32	48	80	128	208

Looking in particular at Table 3, we can see that our PSO algorithm scales fairly well to higher numbers of variables with respect to the optimization of $cidev_1$, even if the CGA parameters were evolved only for the case $n = 7$. As a matter of fact, all the best Boolean functions found by PSO with fit_1 are first order correlation immune (and thus 1-resilient, since they are also balanced). Moreover, for $n = 7$ and $n = 8$ they also satisfy the Strict Avalanche Criterion $PC(1)$, while for higher values of n they reach the minimum deviation $pcdev_1 = 8$. Nevertheless, our Particle Swarm Optimizer is able to find Boolean functions of up to $n = 11$ variables which satisfy both $CI(1)$ and $PC(1)$, even if their nonlinearity is lower (for a detailed comparison with other heuristic methods, see Section 5.3).

On the other hand, Table 4 shows that by using fitness function fit_2 the Particle Swarm Optimizer does not perform well when the number of variables is higher than 7. In fact, 2-resilient functions are found only for $n = 7$, while in all other cases the deviation from $CI(2)$ is at least 8. However, it worths noting that the best solution of 7 variables, besides satisfying with equality Siegenthaler's bound, achieves Tarannikov's bound on nonlinearity as well, since $56 = 2^{7-1} - 2^{2+1}$.

Finally, another different behaviour of the PSO algorithm can be observed using fitness function fit_3. Indeed, one can see from Table 5 that as the number of variables grows the absolute indicator of the best solution gets worse. Nonetheless, for $n = 8$ and $n = 11$ the nonlinearity values achieved with fit_3 are greater than those obtained using fit_1, while they are equal in all other cases.

5.3 Comparison with other Heuristics

We now compare the results of our Particle Swarm Optimizer with those obtained by other heuristic methods. Due to the great heterogeneity in the experimental settings and the parameters adopted in the relevant literature, a comprehensive comparison is not possible. For this reason, in Tables 6 to 9 we summarise the results separately for each class of cryptographically significant balanced Boolean functions discovered by the PSO algorithm. A dash symbol in the tables indicates that the corresponding data is not available, either because the heuristic failed to discover Boolean functions with those cryptographic properties or because that specific case was not considered for testing.

Table 6 reports the maximum nonlinearity achieved by $CI(1)$ functions. In this case, we used Genetic Algorithms (GA) [9], Directed Search Algorithm (DSA) [10] and Simulated Annealing (SA) [3] for the comparison. It can be seen that for $n = 7$ variables our PSO algorithm manages to find 1-resilient functions having maximum nonlinearity 56, while SA stops at 52. For $8 \leq n \leq 12$, the results achieved by PSO are globally similar to those of the other optimization methods, except in the case of $n = 11$ variables where it reaches a maximum nonlinearity of 972 instead of 976. In particular, our PSO outperforms both Genetic Algorithms and Simulated Annealing for $n = 9$ and $n = 10$ variables.

Table 6: Maximum Nonlinearity Achieved by $CI(1)$ Functions

Method	7	8	9	10	11	12
GA [9]	-	112	232	476	976	1972
DSA [10]	-	112	236	480	976	-
SA [3]	52	112	232	476	-	-
PSO	56	112	236	480	972	1972

In Table 7 the maximum nonlinearity of balanced Boolean functions which satisfy both $CI(1)$ and $PC(1)$ is considered. By comparing the results achieved by PSO and SA, we can see that also in this case the former reaches a higher value of nonlinearity for $n = 7$ variables, while for $n = 8$ it is equal to SA. To our knowledge, no heuristic method has ever been applied to discover functions satisfying both $CI(1)$ and $PC(1)$ of $n > 8$ variables. However, our PSO algorithm managed to find this kind of functions for up to $n = 11$ variables, even though for $n > 8$ they were not the best solutions among all the optimization runs with respect to fitness function fit_1. The nonlinearity of these functions is reported in Table 7 as a reference for future research.

Table 8 reports the maximum nonlinearity achieved by Boolean functions with minimal deviation from second order correlation immunity. In particular, the performances of PSO and GA are compared, since in this case we used

Table 7: Maximum Nonlinearity Achieved by Functions satisfying both $CI(1)$ and $PC(1)$

Method	7	8	9	10	11	12
SA [3]	52	112	-	-	-	-
PSO	56	112	232	476	968	-

the same fitness function defined in [9]. As we already discussed in Section 5.2, we can observe that our PSO algorithm does not generalise well to higher numbers of variables. As a matter of fact, PSO manages to reach the same results achieved by GA only for $n = 8$ variables, while in all other cases either the nonlinearity or the deviation from $CI(2)$ is worse. We remark however that for $n = 7$ the 2-resilient functions found by PSO have the same value of nonlinearity as the ones discovered by SA in [3].

Table 8: Comparison of Nl and $cidev_2$ Values

Method		7	8	9	10	11	12
GA [9]	Nl	-	112	232	480	976	1972
	$cidev_2$	-	4	8	8	8	8
PSO	Nl	56	112	232	476	972	1972
	$cidev_2$	0	8	8	8	8	16

Table 9: Comparison of Nl and AC_{max} Values

Method		7	8	9	10	11	12
RBC [1]	Nl	56	116	-	-	-	-
	AC_{max}	16	24	-	-	-	-
GP [13]	Nl	-	116	-	-	-	-
	AC_{max}	-	32	-	-	-	-
SA [3]	Nl	56	116	238	484	982	1986
	AC_{max}	16	24	40	56	88	128
PSO	Nl	56	116	236	480	976	1972
	AC_{max}	16	32	48	80	128	208

Similar considerations can be made for the comparisons in Table 9, which reports the maximum nonlinearity reached by Boolean functions having minimal absolute indicator. The benchmark heuristics in this case are Multi-Objective Random Bit Climber (RBC) [1], Genetic Programming (GP) [13] and again SA. It can be observed that for $n = 7$ variables PSO obtained the same results as RBC and SA, while for $n = 8$ it discovered the same combination of Nl and AC_{max} featured by GP. However, for $n > 8$ our PSO

scored worse values than SA with respect to both nonlinearity and absolute indicator.

6 Conclusions

In this work, we applied a new PSO algorithm to search for balanced Boolean functions from $n = 7$ to $n = 12$ variables with good cryptographic properties. The performed experiments lead us to conclude that our PSO is able to generate Boolean functions having similar or better combinations of nonlinearity, first order correlation immunity and Strict Avalanche Criterion than those obtained by other optimization methods, while it does not perform well when it minimizes deviation from $CI(2)$ or the absolute indicator. The reason could lie in the fact that the adopted velocity parameters were evolved only for the case of $n = 7$ variables, which suggests that further parameters tuning is required for $n \geq 8$. Considering the high computational cost of our meta-fitness function, it may be preferable to use the LUS meta-optimizer for this task instead of CGA.

There are several venues for future developments on the subject. One possibility is to test our PSO with other fitness functions, such as the one adopted in [3] for Simulated Annealing which measures how flat the Walsh spectrum of a Boolean function is. Another interesting direction of research would be to modify the UPDATE-BAL-POS() procedure in such a way that only the swaps which increase nonlinearity or decrease the deviation from k-th correlation immunity are performed. This could be accomplished, for instance, by integrating the Hill Climbing algorithm inside the update procedure.

References

[1] H. E. Aguirre, H. Okazaki, Y. Fuwa. An evolutionary multiobjective approach to design highly non-linear Boolean functions. In *Proceedings of GECCO '07*, pp. 749-756. ACM, July 2007.

[2] C. Carlet. Boolean Functions for Cryptography and Error-Correcting Codes. In Y. Crama, P.L. Hammer. *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*. Cambridge University Press, New York, May 2011.

[3] J. A. Clark, J. Jacob, S. Stepney, S. Maitra, W.Millan. Evolving Boolean Functions Satisfying Multiple Criteria. In *Proceedings of INDOCRYPT 2002*, LNCS vol. 2551, pp. 246-259. Springer, December 2002.

[4] X. Hu, R. C. Eberhart, Y. Shi. Swarm Intelligence for Permutation Optimization: Case Study of n-Queens Problem. In *Proceedings of the*

2003 IEEE Swarm Intelligence Symposium, pp. 243-246. IEEE Press, April 2003.

[5] J. Kennedy, R. Eberhart. Particle Swarm Optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, pp. 1942-1948. IEEE Press, November 1995.

[6] J. Kennedy, R. C. Eberhart. A Discrete Binary Version of the Particle Swarm Algorithm. In *Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics*. pp. 4104-4108. IEEE Press, 1997.

[7] R. Mendes, J. Kennedy, J. Neves. The Fully Informed Particle Swarm: Simpler, Maybe Better. *IEEE Trans. Evolutionary Computation* 8(3):204-210, June 2004.

[8] Z. Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs. Springer, 1992.

[9] W. Millan, A. J. Clark, E. Dawson. Heuristic Design of Cryptographically Strong Balanced Boolean Functions. In *Proceedings of EUROCRYPT 98*, LNCS vol. 1403, pp. 489-499. Springer, June 1998.

[10] E. Pasalic, T. Johansson. Further Results on the Relation Between Nonlinearity and Resiliency for Boolean Functions. In *Proceedings of the 7th IMA International Conference*, LNCS vol. 1746 pp. 35-44. Springer, December 1999.

[11] N. J. Patterson, D. H. Wiedemann. The Covering Radius of the $(2^{15}, 16)$ Reed-Muller Code is at Least 16276. *IEEE Transactions on Information Theory* IT-29(3):354-356, May 1983.

[12] M. E. H. Pedersen, A. J. Chipperfield. Simplifying Particle Swarm Optimization. *Applied Soft Computing* 10(2):618-628, March 2010.

[13] S. Picek, D. Jakobovic, M. Golub: Evolving cryptographically sound Boolean functions. In *Proceedings of GECCO'13 (Companion)*, pp. 191-192. ACM, July 2013.

[14] R. Poli. Analysis of the Publications on the Applications of Particle Swarm Optimisation. *Journal of Artificial Evolution and Applications* 2008(3):1-10, January 2008.

[15] N. J. Radcliffe. Equivalence Class Analysis of Genetic Algorithms. *Complex Systems* 5(2):183-205, 1991.

[16] Y. Shi, R. C. Eberhart. Parameter Selection in Particle Swarm Optimization. In *Proceedings of Evolutionary Programming VII*, LNCS vol. 1447 pp. 591-600. Springer, March 1998.

[17] I. C. Trelea. The Particle Swarm Optimization Algorithm: Convergence Analysis and Parameter Selection. *Information Processing Letters* 85(6):317-325, March 2003.

[18] X-M. Zhang, Y. Zheng. GAC - The Criterion for Global Avalanche Characteristics of Cryptographic Functions. *Journal of Universal Computer Science* 1(5):316-333, 1995.