The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-319-26841-5_3

A Genetic Algorithm for Evolving Plateaued Cryptographic Boolean Functions

Luca Mariot and Alberto Leporati

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi Milano - Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
luca.mariot@disco.unimib.it alberto.leporati@unimib.it

Abstract. We propose a genetic algorithm (GA) to search for plateaued boolean functions, which represent suitable candidates for the design of stream ciphers due to their good cryptographic properties. Using the spectral inversion technique introduced by Clark, Jacob, Maitra and Stanica, our GA encodes the chromosome of a candidate solution as a permutation of a three-valued Walsh spectrum. Additionally, we design specialized crossover and mutation operators so that the swapped positions in the offspring chromosomes correspond to different values in the resulting Walsh spectra. Some tests performed on the set of pseudoboolean functions of $n = 6$ and $n = 7$ variables show that in the former case our GA outperforms Clark et al.'s simulated annealing algorithm with respect to the ratio of generated plateaued boolean functions per number of optimization runs.

Keywords: evolutionary computing · cryptography · genetic algorithms · simulated annealing · boolean functions · Walsh transform · spectral inversion · nonlinearity · resiliency

1 Introduction

The security of a symmetric cryptosystem often depends on the choice of the underlying *boolean functions*. For instance, in the *combiner model* for stream ciphers the boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ used to combine the outputs of n *Linear Feedback Shift Registers* (LFSRs) should satisfy several cryptographic criteria in order to resist to specific attacks. These properties include, among others, *balancedness*, high *algebraic degree* and *nonlinearity*, and high order of *resiliency*. When searching for good cryptographic boolean functions, a way to overcome the combinatorial explosion resulting from increasing the number of variables is to use *heuristic techniques*, such as *Genetic Algorithms* [7], *Simulated Annealing* [2], *Genetic Programming* (both basic and *Cartesian* GP [8,9]) and *Particle Swarm Optimization* [5]. These methods usually represent a candidate boolean function by either its *truth table* or by a tree encoding one of its possible algebraic expressions.

Clark, Jacob, Maitra and Stanica [3] proposed the *spectral inversion technique* for designing good cryptographic boolean functions, in which a candidate solution is represented as a *permutation* of a *Walsh spectrum* that encodes a particular set of cryptographic properties. In general, by applying the *inverse Walsh transform* to such

a spectrum, a *pseudoboolean function* $f : \mathbb{F}_2^n \to \mathbb{R}$ is obtained. Thus, the objective function becomes the *deviation* of f from being a boolean function, and in [3] Simulated Annealing (SA) was adopted to minimize it.

The goal of this paper is to investigate the application of *permutation-based Genetic Algorithms* for evolving cryptographic boolean functions by spectral inversion, a method which was conjectured to be more efficient than SA in [3].

In particular, we design a GA in which the chromosomes of the evolved solutions are Walsh spectra of *plateaued pseudoboolean functions*. The motivation for this choice is twofold. First, spectra of plateaued pseudoboolean functions are three-valued, hence they have an easy combinatorial characterization. Second, plateaued *boolean functions* are considered suitable candidates for cryptographic applications, since they satisfy with equality the bounds on maximum achievable algebraic degree and nonlinearity for a given resiliency order, respectively proved by Siegenthaler [10] and Tarannikov [11]. Since our GA manipulates permutations of *repeated* values, we propose a crossover and a mutation operator which ensure that the modified genes in the offspring correspond to different values in the Walsh spectrum.

Let us note that, as the number of boolean functions of n variables is 2^{2^n}, exhaustively searching for plateaued boolean functions (or, more in general, cryptographically relevant boolean functions) becomes unfeasible for $n > 5$. For this reason, we assess the performance of our GA in generating plateaued boolean functions of $n = 6$ and $n = 7$ variables. The results show that our GA outperforms Simulated Annealing in finding plateaued boolean functions of $n = 6$ variables, while for $n = 7$ SA still yields better average fitness values, even if neither technique was able to generate a plateaued boolean function in this case. This would seem to suggest that combining the global search capabilities of our GA and the local exploration of SA could lead to better results for higher numbers of variables.

The remainder of this paper is organised as follows. Section 2 gives a brief introduction about boolean functions and their cryptographic properties. Section 3 describes our permutation-based Genetic Algorithm, defining the solution encoding, the fitness function and the adopted genetic operators. Section 4 presents the results obtained by our GA on the optimization of pseudoboolean plateaued functions for $n = 6$ and $n = 7$ variables, and compares them with the results achieved by the SA algorithm described in [3]. Finally, Section 5 summarises the contributions of this paper and points out some possible future developments on the subject.

2 Preliminaries on Boolean Functions

In this section, we recall some basic definitions and facts about boolean functions and their cryptographic properties. For further details on the subject, we refer the reader to Carlet [1].

2.1 Representations of Boolean Functions

A *boolean function of n variables* is a mapping $f : \mathbb{F}_2^n \to \mathbb{F}_2$, where $\mathbb{F}_2 = \{0,1\}$ is the finite field of two elements. Once an ordering of the input variables $x_1, \cdots, x_n$ has

been fixed, the *truth table representation* of f is a vector $\Omega_f \in \mathbb{F}_2^{2^n}$ which specifies for all $i \in \{0,\cdots,2^n-1\}$ the value of $f(bin(i))$, where $bin(i)$ denotes the n-bit binary expansion of i. The *algebraic normal form* (ANF) represents a boolean function f as a sum of products over $\mathbb{F}_2$. Specifically, given $f : \mathbb{F}_2^n \to \mathbb{F}_2$, $N = \{1,\cdots,n\}$ and $\mathcal{P}(N)$ the power set of N, the ANF of f is defined by the following polynomial:

$$f(x) = \bigoplus_{I \in \mathcal{P}(N)} a_I \left(\prod_{i \in I} x_i \right) \ . \tag{1}$$

The *polar form* $\hat{f} : \mathbb{F}_2^n \to \{-1,1\}$ of f is the function defined as $\hat{f}(x) = (-1)^{f(x)}$ for all $x \in \mathbb{F}_2^n$. Denoting by $\omega \cdot x = \omega_1 x_1 \oplus \cdots \oplus \omega_n x_n$ the *scalar product* modulo 2 of the two vectors $\omega, x \in \mathbb{F}_2^n$, the *Walsh transform* of $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is the function $\hat{F} : \mathbb{F}_2^n \to \mathbb{R}$ defined for all $\omega \in \mathbb{F}_2^n$ as

$$\hat{F}(\omega) = \sum_{x \in \mathbb{F}_2^n} \hat{f}(x) \cdot (-1)^{\omega \cdot x} \ . \tag{2}$$

The vector of Walsh coefficients $\mathcal{S}_f = \left(\hat{F}(bin(0)),\cdots,\hat{F}(bin(2^n-1))\right) \in \mathbb{R}^{2^n}$ of f is also called the *Walsh spectrum* of f, while the maximum absolute value among all Walsh coefficients $W_{max}(f)$ is called the *spectral radius* of f. One important property of the Walsh spectrum is *Parseval's identity*, which states that the sum of all squared Walsh coefficients is constant for every boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$:

$$\sum_{\omega \in \mathbb{F}_2^n} \hat{F}(\omega)^2 = 2^{2n} \ . \tag{3}$$

Given a spectrum $\mathcal{S}_f$ of a boolean function f, it is possible to recover the original (polar) function by applying the *inverse Walsh transform*:

$$\hat{f}(x) = 2^{-n} \sum_{\omega \in \mathbb{F}_2^n} \hat{F}(\omega) \cdot (-1)^{\omega \cdot x} \ . \tag{4}$$

Notice that not all possible real-valued Walsh spectra correspond to boolean functions: in general, by applying the inverse Walsh transform to a random spectrum $\mathcal{S} \in \mathbb{R}^{2^n}$ the outcome will be the polar truth table of a *pseudoboolean function* $f : \mathbb{F}_2^n \to \mathbb{R}$.

2.2 Cryptographic Properties of Boolean Functions

The Walsh spectrum is used to characterize several cryptographic properties of boolean functions. In particular, given $f : \mathbb{F}_2^n \to \mathbb{F}_2$ along with is Walsh spectrum $\mathcal{S}_f$ and spectral radius $W_{max}(f)$, we considered the following cryptographic properties for our optimization problem:

- *Balancedness*: f is balanced if its truth table Ω_f is composed of an equal number of 0s and 1s. Equivalently, f is balanced if and only if $\hat{F}(\underline{0}) = 0$, where $\underline{0}$ denotes the null vector of $\mathbb{F}_2^n$.
- *Algebraic degree*: the algebraic degree d of f is defined as the degree of its ANF. Functions of degree 1 are also called *affine functions*.

- *Nonlinearity*: the nonlinearity nl of f is the Hamming distance of f from the set of all affine functions. Equivalently, the nonlinearity of f can also be computed as $nl = \frac{1}{2}(2^n - W_{max}(f))$.
- *Resiliency*: function f is said to be *m-resilient* if, by fixing at most m input variables, the resulting restrictions of f are all balanced. Equivalently, f is m-resilient if and only if $\hat{F}(\omega) = 0$ for all $\omega \in \mathbb{F}_2^n$ having *Hamming weight* at most m, that is, the Walsh transform vanishes for all vectors $\omega \in \mathbb{F}_2^n$ which have at most m nonzero coordinates. Notice that 0-resiliency corresponds to balancedness.

In order to be suitable for cryptographic purposes (for instance, to be used in the combiner model), a boolean function f should be balanced, have high algebraic degree and nonlinearity, and be resilient of high order. However, functions which fully satisfy all these four criteria simultaneously do not exist. In particular, *Siegenthaler's bound* [10] and *Tarannikov's bound* [11] respectively limit the reachable values of algebraic degree and nonlinearity for a given order of resiliency:

- *Siegenthaler's bound*: $d \leq n - m - 1$
- *Tarannikov's bound*: $nl \leq 2^{n-1} - 2^{m+1}$

In what follows, by (n, m, d, nl) we denote the *profile* of a balanced boolean function of n variables having resiliency order m, algebraic degree d and nonlinearity nl.

Among the various classes of boolean functions which can be characterized in terms of cryptographic properties, *bent functions* are the ones reaching the highest possible values of nonlinearity. Specifically, $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is bent if $\hat{F}(\omega) = \pm 2^{\frac{n}{2}}$ for all $\omega \in \mathbb{F}_2^n$. However, these functions exist only for even values of n, and moreover they are not balanced since $\hat{F}(0) = \pm 2^{\frac{n}{2}}$. Hence, bent functions are not suitable for cryptographic applications.

A broader class which includes bent functions is the set of *plateaued functions*, originally introduced by Zhang and Zheng [13]. Formally, a boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is plateaued if $\hat{F}(\omega) \in \{-W_{max}(f),\ 0,\ +W_{max}(f)\}$ for all $\omega \in \mathbb{F}_2^n$. Thus, Walsh spectra of plateaued functions take at most three values. Plateaued functions are especially interesting for cryptography, since they satisfy with equality both Siegenthaler's and Tarannikov's bounds, a feature which makes them optimal with respect to all four properties mentioned above. The profile of a plateaued boolean function is of the form $(n, r-2, n-r-3, 2^{n-1} - 2^{r-1})$, where $r \geq \frac{n}{2}$, from which it follows that $W_{max}(f) = 2^r$. Notice that if n is even and $r = \frac{n}{2}$, then a plateaued function is bent. In what follows, we will apply our GA for evolving plateaued boolean functions.

3 Our Genetic Algorithm

3.1 Chromosomes Encoding

The main idea underlying the chromosome encoding of our GA is to represent a candidate solution as a *permutation* of a Walsh spectrum $\mathcal{S} \in \mathbb{R}^{2^n}$. This *spectral inversion* approach to heuristic design of cryptographic boolean functions was originally introduced by Clark, Jacob, Maitra and Stanica in [3].

As a first observation, notice that representing the chromosome as a permutation of the spectrum *positions* would allow us to employ classic permutation-based GA, such as those designed for the Traveling Salesman Problem [4]. However, the Walsh spectrum is generally composed of *repeated* values. This means that a position-based encoding would make the GA search into a space which is bigger than what is actually needed, since several swaps performed by permutation-based genetic operators would map to the same values in the Walsh spectrum. Hence, we represent our candidate solution directly by its Walsh spectrum *values*. From the combinatorial point of view, this representation is equivalent to performing permutations over a *multiset* $\mathcal{M}$.

Recall from the previous section that, by Parseval's identity, the sum of the squared Walsh coefficients of any n-variable boolean function equals 2^{2n}. Moreover, the values summed in the Walsh transform defined in Equation (2) are all integers, hence we can start to model a candidate solution as a vector of 2^n integers which sum to 2^{2n}. Additionally, we are interested only in plateaued boolean functions, so that each Walsh coefficient can only take its value in the set $V = \{-2^r, 0, +2^r\}$. We thus need to determine the *multiplicities* of the elements of V in order to characterise the multiset $\mathcal{M}$ required to build the spectrum. Using the approach sketched in [3], these multiplicities can be derived from the following observations:

(1) Since a plateaued boolean function is m-resilient with $m = r-2$, all positions which correspond to input vectors having at most m nonzero coordinates must be set to zero. Therefore, in order to meet the resiliency constraint there must be *at least* $\#0_{res} = \sum_{i=0}^{m} \binom{n}{i}$ zero-valued positions in the spectrum.
(2) Each *nonzero position* in the spectrum contributes by a term of $(\pm 2^r)^2 = 2^{2r}$ in Parseval's identity. Thus, the total number of nonzero positions in the spectrum is given by $\#\pm 2^r = \frac{2^{2n}}{2^{2r}}$.
(3) From (1) and (2) we deduce that there are $\#0_{add} = 2^n - ((\#\pm 2^r) + (\#0_{res}))$ additional positions set to zero other than the ones used to satisfy m-resiliency.
(4) By setting $\hat{f}(\underline{0}) = 1$, it follows that $\sum_{\omega \in \mathbb{F}_2^n} \hat{F}(\omega) = 2^n$. Notice that this is an arbitrary assumption, since we are considering only those functions mapping the null vector to 0. However, this does not bias the final search space, since by setting $\hat{f}(\underline{0}) = -1$ we would always get plateaued functions having the same profile.
(5) By combining observations (2) and (4), we finally obtain the number of positions to be set to -2^r and $+2^r$ by solving the following system:

$$\begin{cases} (\#+2^r) + (\#-2^r) & = \frac{2^{2n}}{2^{2r}} \\ (\#+2^r) - (\#-2^r) & = 2^n \end{cases}$$

which gives

$$\begin{cases} \#+2^r & = 2^{n-1}(2^{n-2r}+1) \\ \#-2^r & = 2^{n-1}(2^{n-2r}-1) \end{cases}$$

In what follows, we denote by $x[i]$ the element at position i of vector x. Since there are $\#0_{res}$ positions in the spectrum which are set to zero for the resiliency constraint, we can restrict our representation only to those positions whose binary expansions have more than m nonzero coordinates. Let us thus consider the *restricted ordered spectrum*

$\mathcal{S}_{ro} = (w_1, \cdots, w_l)$ having length $l = 2^n - \#0_{res}$ and whose first $\#0_{add}$ positions are set to zero, the next $\# - 2^r$ are set to -2^r and the final $\# + 2^r$ are set to $+2^r$. Additionally, let us denote by $P_r = (j_1, \cdots, j_l)$ the vector of positions j_i such that $hwt(bin(j_i)) > m$, where $hwt(\cdot)$ denotes the Hamming weight of the binary string passed as argument. Clearly, by permuting the components in $\mathcal{S}_{ro}$ the resulting spectrum maintains the desired cryptographic properties, since the multiplicities $\#0_{add}$, $\# - 2^r$ and $\# + 2^r$ are permutation invariant. However, we are interested only in those permutations which swap different values in the restricted spectrum. To address this problem, we employ the following equivalence relation $\sim_p$ on the symmetric group S_l: given two permutations $\pi_1, \pi_2 \in S_l$, define $\pi_1 \sim_p \pi_2$ if and only if $w_{\pi_1(i)} = w_{\pi_2(i)}$ for all $i \in \{1, \cdots, l\}$, where $w_{\pi_1(i)}, w_{\pi_2(i)}$ are components of $\mathcal{S}_{ro}$. We can thus characterize the permutations which map different values in the restricted spectrum as the representatives of the equivalence classes in the quotient set $S_l/\sim_p$. With a little abuse of notation, in what follows we write $\pi \in S_l/\sim_p$ to directly denote the representative permutation π instead of the equivalence class $[\pi]_{\sim_p}$.

The *chromosome* which encodes a candidate solution evolved by our GA is a permutation $c = (w_{\pi(1)}, \cdots, w_{\pi(l)})$ of the restricted ordered spectrum $\mathcal{S}_{ro}$, where $\pi \in S_l/\sim_p$. The *decoding* of chromosome c which yields the corresponding pseudoboolean function $f : \mathbb{F}_2^n \to \mathbb{R}$, denoted by $dec(c)$, is carried out using the following procedure:

1. Initialize the Walsh spectrum $\mathcal{S}_f$ to the null vector $(0, \cdots, 0) \in \mathbb{R}^{2^n}$.
2. For all $i \in \{1, \cdots, l\}$ set $\mathcal{S}_f[j_i] = c[i]$, where $j_i = P_r[i]$.
3. Perform *spectral inversion*: apply to $\mathcal{S}_f$ the inverse Walsh transform defined in Equation (4) in order to obtain the polar form $\hat{f}$ of function f.

3.2 Objective and Fitness Functions

In order to measure how good a pseudoboolean function is, the authors of [3] proposed an objective function based on the distance from the *nearest boolean function*. Formally, given the polar form $\hat{f}$ of $f : \mathbb{F}_2^n \to \mathbb{R}$, the polar truth table of the nearest boolean function $\hat{b} : \mathbb{F}_2^n \to \{-1, +1\}$ is obtained for all $x \in \mathbb{F}_2^n$ as follows:

$$\hat{b}(x) = \begin{cases} +1 & , \text{ if } \hat{f}(x) > 0 \\ -1 & , \text{ if } \hat{f}(x) < 0 \\ +1 \text{ or } -1 \text{ (chosen randomly)} & , \text{ if } \hat{f}(x) = 0 \end{cases} \tag{5}$$

Given a chromosome c and the corresponding pseudoboolean function $f = dec(c)$, the *objective function* to be minimized proposed in [3] is defined as:

$$obj(f) = \sum_{x \in \mathbb{F}_2^n} (\hat{f}(x) - \hat{b}(x))^2 \ . \tag{6}$$

This objective function measures the *deviation* of f from being a true boolean function. Hence, an optimal solution to our problem is encoded by a chromosome c such that $obj(dec(c)) = 0$. Given how we designed the Walsh spectrum, such a solution corresponds to a plateaued boolean function.

The *fitness function* $fit(\cdot)$ maximised by our GA is simply defined as the opposite of the objective function (6), that is, $fit(f) = -obj(f)$.

3.3 Genetic Operators

Considering the chromosome encoding adopted for the candidate solutions, an appropriate crossover operator for our GA has to preserve the multiplicities $\#0_{add}$, $\#-2^r$ and $\#+2^r$ of the restricted spectrum, so that Parseval's identity and the other properties of plateaued functions are maintained. To this end, we designed a crossover operator loosely inspired by the one proposed in [7].

The main idea is to work at the loci level, and to use *counters* in order to keep track of the multiplicities of the three values 0, -2^r and $+2^r$ inserted in the offspring during the crossover phase. More precisely, given two parent chromosomes c_1 and c_2, our crossover operator builds an offspring chromosome o as follows:

1. Initialize to zero the three counters *cnt_z*, *cnt_n* and *cnt_p* respectively associated to the spectral values 0, -2^r and $+2^r$.
2. For all $i \in \{1, \cdots, l\}$ such that $c_1[i] = c_2[i]$, copy either $c_1[i]$ or $c_2[i]$ in $o[i]$. Depending on the copied value, update the relevant counter.
3. For all $i \in \{1, \cdots, l\}$ such that $c_1[i] \neq c_2[i]$, determine the value to be copied in $o[i]$ as follows:
   (a) If all three counters are below their maximum values (that is, $cnt\_z < \#0_{add}$, $cnt\_n < \#-2^r$ and $cnt\_p < \#+2^r$), randomly select $c_1[i]$ or $c_2[i]$ with probability $1/2$, and copy it in $o[i]$. Depending on the copied value, update the relevant counter.
   (b) If one of the three counters reached its maximum value, check if either $c_1[i]$ or $c_2[i]$ is equal to the value associated to that counter. If so, copy the gene of the other parent in $o[i]$. Otherwise, randomly select $c_1[i]$ or $c_2[i]$ with probability $1/2$, and copy it in $o[i]$. In both cases, depending on the copied value, update the relevant counter.
   (c) If two out of three counters reached their respective maximum values, copy the value associated to the remaining counter in $o[i]$.
4. Return the offspring chromosome o.

Concerning the mutation operator, we adopted a simple swap procedure which checks that the swapped values are different. In particular, let us assume that c is a chromosome of length l and that pos_0, pos_{-2^r} and pos_{+2^r} are the vectors specifying the positions of the 0s, -2^rs and $+2^r$s in c, respectively. Then, our mutation operator is applied to each locus $i \in \{1, \cdots, l\}$ of c with probability $p_\mu \in [0, 1]$, and it performs the following steps:

1. Setting $v = c[i]$, randomly select with probability 1/2 one of the two positions vectors pos_t or pos_u, where $t \neq v$ and $u \neq v$.
2. Denoting by pos_s the selected positions vector, randomly draw with uniform probability an index j of pos_s.
3. Swap the values $c[i]$ and $c[pos_s[j]]$.
4. Swap the occurrence of i in pos_v with $pos_s[j]$.

Finally, for the selection operators we tested both *roulette wheel selection* and *deterministic tournament selection*.

3.4 Overall GA Procedure

We can now summarise the overall procedure of our GA. The input parameters for the algorithm are the number of variables n and the index $r \geq \frac{n}{2}$ of the target plateaued functions, the size of the population N (where N is even), the number of generations G to be performed, the crossover and mutation probabilities p_χ and p_μ, and the selection operator S.

1. *Initialization*: Compute the profile $(n, r-2, n-r-3, 2^{n-1}-2^{r-1})$ of the target functions and the multiplicities $\#0_{res}$, $\#0_{add}$, $\#-2^r$ and $\#+2^r$ of the Walsh spectrum.
2. *Create Population*: For $i \in \{1, \cdots, N\}$, create a chromosome $c = (w_{\pi(1)}, \cdots, w_{\pi(l)})$ of length $l = 2^n - \#0_{res}$, where π is a random permutation of $S_l/\sim_p$, and add it to the current population $\mathcal{P}$.
3. *Initial Fitness Evaluation*: For each chromosome $c \in \mathcal{P}$, decode its respective pseudoboolean function $f = dec(c)$ and compute the fitness value $fit(f) = -obj(f)$, where $obj(\cdot)$ is defined as in Equation (6). Set the best solution B as the individual scoring the highest fitness value.
4. *Selection Phase*: Apply N times the selection operator S on the current population $\mathcal{P}$, thus creating a candidate population $\mathcal{C}$ of (eventually repeated) N chromosomes which will produce the next generation.
5. *Crossover Phase*: For all $i \in \{1, 3, \cdots, N-1\}$, sample a random number $r \in [0, 1]$. If $r < p_\chi$, apply the crossover operator *twice* to the pair $c_i, c_{i+1} \in \mathcal{C}$, and copy the two offspring chromosomes (o_i, o_{i+1}) in the new population $\mathcal{N}$. Otherwise, set $o_i = c_i$ and $o_{i+1} = c_{i+1}$, and copy them in $\mathcal{N}$.
6. *Mutation Phase*: For each chromosome $o \in \mathcal{N}$ and for all $j \in \{1, \cdots, l\}$, sample a random number $r \in [0, 1]$. If $r < p_\mu$, apply the mutation operator to $o[j]$.
7. *Fitness Evaluation*: For each chromosome $o \in \mathcal{N}$, compute the fitness value of $f = dec(o)$, and find the current best individual B_c having the highest fitness value in $\mathcal{N}$.
8. *Elitism*: If $fit(B_c) \leq fit(B)$, replace a random individual in $\mathcal{N}$ with B. Otherwise, update the best solution found so far by setting $B = B_c$.
9. *Population Update*: Set the current population $\mathcal{P}$ equal to $\mathcal{N}$.
10. *Termination Condition*: If the best solution found is optimal ($obj(B) = 0$) or the maximum number of generations G has been reached, output the best solution B found by the GA. Otherwise, return to Step 4.

4 Experiments and Results

We tested our GA on the spaces of pseudoboolean functions of $n = 6$ and $n = 7$ variables, adopting in both cases index $r = 4$. This is the smallest integer value, yielding maximum nonlinearity, such that the resulting functions are not bent for $n = 6$. Table 1 reports the profiles and the multiplicities of the spectrum values for the corresponding plateaued boolean functions. We limited our experimentation to these two problem instances in order to compare our GA with Simulated Annealing. As a matter of fact, the basic SA algorithm described in [3] was able to find only 5 plateaued functions with profile $(7, 2, 4, 56)$ out of 500 optimization runs, and a *change of basis* procedure [2] had to be

Table 1. Cryptographic profiles and spectral multiplicities for plateaued functions of $n = 6$ and $n = 7$ variables

(n,m,d,nl)	$\#0_{res}$	$\#0_{add}$	$\#-2^r$	$\#+2^r$
$(6,2,3,24)$	22	26	6	10
$(7,2,4,56)$	29	35	28	36

applied in order to convert some generated sub-optimal solutions into actual boolean functions. Further, for $n = 6$ only bent functions were considered, but not generic plateaued functions. On the other hand, for higher number of variables the basic version of SA always failed to generate boolean functions, hence the authors of [3] restricted their search space to the family of *rotation symmetric boolean functions*, which we did not consider in this work.

For each value of n and selection operator considered, we performed $R = 500$ independent runs of our GA, using a population of $N = 30$ chromosomes evolved for $G = 500000$ generations. Thus, each GA run consisted of $F = 1.5 \cdot 10^7$ fitness evaluations. The crossover and mutation probabilities were respectively set to $p_\chi = 0.95$ and $p_\mu = 0.05$, while in the case where tournament selection was used we adopted a tournament size of $k = 3$.

Concerning the comparison with Simulated Annealing, we implemented the SA algorithm described in [3] and we tested it for $n = 6$ and $n = 7$ by setting the number of inner loops $MaxIL$ and moves within an inner loop MIL respectively to $MaxIL = 5000$ and $MIL = 3000$, thus yielding the same number $F = 1.5 \cdot 10^7$ of fitness evaluations performed by our GA. Since the authors of [3] did not mention the initial temperature which they adopted for their experiments, we tested the values $T_1 = 100$ and $T_2 = 1000$ with cooling parameter respectively set to $\alpha_1 = 0.95$ and $\alpha_2 = 0.99$. As for our GA, for each combination of parameters (n, T_0, α) we performed 500 runs of the SA algorithm.

We performed all our experiments on a 64-bit Linux machine with a Core i5 architecture and a CPU running at 2.8 GHz. For $n = 6$, a set of 500 runs of GA or SA took approximately 11.5 hours to complete, while for $n = 7$ it took about 28.3 hours and 25 hours for GA and SA, respectively. Table 2 reports the results of the experiments. By $GA(RWS)$ and $GA(DTS)$ we denote our GA respectively with roulette wheel selection and deterministic tournament selection, while $SA(T_i, \alpha_i)$ stands for the SA algorithm run with initial temperature T_i and cooling parameter α_i, for $i \in \{1,2\}$. For each parameters combination, Table 2 reports the average (avg_o), minimum (min_o), maximum (max_o) and standard deviation (std_o) values of the objective function $obj(\cdot)$ computed on the best solutions found, along with the numbers of optimal solutions generated ($\#opt$) and the average time per run in seconds (avg_t).

For $n = 6$ it can be observed that both versions of our GA outperformed SA with respect to the ratio of generated $(6,2,3,24)$ functions versus the total number of optimization runs. In particular, the adoption of tournament selection produced better results than roulette wheel selection, with 93 plateaued functions achieved using the former operator against the 60 obtained using the latter one. On the other hand, changing the initial temperature and the cooling parameter α did not seem to influence the SA performances, with only 11 plateaued functions generated by $SA(T_1, \alpha_1)$ and 10 functions

Table 2. Statistics of the best solutions found by our GA and SA over $R = 500$ runs

n	Stat	$GA(RWS)$	$GA(DTS)$	$SA(T_1,\alpha_1)$	$SA(T_2,\alpha_2)$
6	avg_o	14.08	13.02	19.01	19.03
	min_o	0	0	0	0
	max_o	16	16	28	28
	std_o	5.21	6.23	4.89	4.81
	$\#opt$	60	93	11	10
	avg_t	83.3	79.2	79.1	79.4
7	avg_o	53.44	52.6	45.09	44.85
	min_o	47	44	32	27
	max_o	58	59	63	57
	std_o	2.40	2.77	4.39	4.18
	$\#opt$	0	0	0	0
	avg_t	204.2	204.5	180.3	180.2

generated by $SA(T_2,\alpha_2)$. Notice also that the computational overhead introduced by our GA is not very high: for example, using roulette wheel selection the average time per run of our GA was 83.3 seconds, while with tournament selection a single run took on average 79.2 seconds, which is in the same range as that employed by *SA*.

In the case of $n = 7$ variables, no version of our GA nor SA was able to generate a plateaued boolean function of profile $(7,2,4,56)$. However, it can be seen that SA outperformed both versions of our GA. In particular, the GA obtained slightly better results using tournament selection than roulette wheel selection, but SA scored lower average objective function values than GA. The same difference can also be observed by comparing the minimum objective function values.

5 Conclusions and Directions for Further Research

In this paper, we proposed a genetic algorithm to evolve plateaued boolean functions which satisfy good cryptographic properties. Instead of searching the space of boolean functions (as it is usually done in the existing literature), we adopted the *spectral inversion* approach set forth by Clark, Jacob, Maitra and Stanica in [3], which represents a candidate solution as a Walsh spectrum already satisfying the desired cryptographic properties. The search space thus becomes the set of all plateaued pseudoboolean functions, and the objective function to be minimized is the distance of the candidate solution from the nearest boolean function. The representation adopted for the chromosomes of our GA consists in a permutation of a restricted Walsh spectrum, in which the positions related to m-resiliency are not considered, being constantly set to zero. Since the coefficients in the spectrum of a plateaued boolean function can take only three values, the chromosome actually encodes a permutation on a multiset. The decoding process first maps the loci of the chromosome to the positions in the Walsh spectrum having Hamming weight higher than m, and then the inverse Walsh transform is applied to obtain the associated pseudoboolean function. We designed a specialized crossover operator which employs

counters in order to preserve the multiplicities of the three values characterizing the spectrum of plateaued functions, while for mutation we adopted a simple swap-based operator which exchanges only those positions in the chromosome corresponding to different spectral values.

The performed experiments show that in the case of $n = 6$ variables our GA achieved better results than the Simulated Annealing algorithm proposed in [3] with respect to the ratio of generated $(6,2,3,24)$ boolean functions per number of optimization runs. In particular, our GA performed better when adopting deterministic tournament selection instead of basic roulette wheel selection, while modifying the initial temperature and the cooling parameter did not significantly change the SA performances. On the other hand, for $n = 7$ no heuristic technique was able to generate a $(7,2,4,56)$ plateaued boolean function, but SA scored on average lower objective function values than GA.

Extending the comparison to other *direct* heuristic methods (that is, heuristics which directly explore the space of boolean functions) is not a straightforward task. The reason for this difficulty is twofold. First, there are no obvious ways to compare the sub-optimal solutions found, due to the different representations adopted. In particular, in our GA a sub-optimal solution is a pseudoboolean function which already satisfies the desired cryptographic properties, while in direct methods it is a boolean function which do not satisfy these criteria. Second, to our knowledge only two direct heuristic methods have been reported in the literature to generate $(6,2,3,24)$ plateaued functions [2,**?**], but no information on the ratio of optimal solutions found per number of optimization runs are available. Nonetheless, these methods were also able to locate $(7,2,4,56)$ functions.

The results presented in this paper, together with the above considerations on direct heuristic methods, suggest that our GA does not scale well for $n \geq 7$, the likely reason being that it gets stuck in local optima. A possible way to overcome this drawback is to combine the global search capabilities of GA with a local search technique. A straightforward method to investigate this idea could be the integration of our GA inside the SA algorithm of [3], using for example the *Genetic Annealing* framework [12]. The obvious downside to this solution, however, would be the significantly higher amount of computational resources required to carry out a single optimization run.

An alternative solution could be to add a *Hill Climbing* optimization step in our GA, similarly to the strategy adopted by Millan, Clark and Dawson in [7]. In the context of our GA, a Hill Climbing optimization step would require characterising the pairs of Walsh coefficients which, if swapped, would decrease the deviation of the resulting pseudoboolean function. Further, one could also consider substituting the classic GA by more refined evolutionary heuristics, such as the *Bacterial Evolutionary Algorithm* (BEA) [**?**] which could allow achieving better convergence.

An additional direction for further research would be to consider different cryptographic properties other than nonlinearity, algebraic degree and resiliency. The *propagation criterion* $PC(l)$, for instance, can be characterized by the zeros of the *autocorrelation function*, which is related to the Walsh transform by the *Wiener-Khintchine theorem* [1]. Heuristic search of boolean functions satisfying only $PC(l)$ could be done using the same basic spectral inversion method of [3]: in this case, it would suffice to evolve through our GA or SA autocorrelation spectra instead of Walsh spectra. However, finding boolean functions satisfying both Walsh-related and autocorrelation-related properties by spectral

inversion would require modifying the representation of the candidate solutions, since a valid swap on the Walsh spectrum could induce an invalid swap on the autocorrelation function (and vice versa), due to the aforementioned Wiener-Khintchine theorem.

Appendix: Source Code and Experiments Data

The Java source code for the GA described in this paper and the SA algorithm proposed in [3] can be found at `http://openit.disco.unimib.it/~mariot/ga_platbf`, together with the data of the experiments discussed in Section 4.

References

1. Carlet, C.: Boolean Functions for Cryptography and Error-Correcting Codes. In: Crama, Y., Hammer, P.L. (eds.) Boolean Models and Methods in Mathematics, Computer Science, and Engineering. Cambridge University Press, New York (2010)
2. Clark, J.A., Jacob, J., Stepney, S., Maitra, S., Millan, W.: Evolving Boolean Functions Satisfying Multiple Criteria. In: Menezes, A., Sarkar, P. (eds.) INDOCRYPT 2002. LNCS, vol. 2551, pp. 246-259. Springer, Heidelberg (2002)
3. Clark, J.A., Jacob, J., Maitra, S., Stanica, P.: Almost Boolean Functions: The Design of Boolean Functions by Spectral Inversion. Comput. Intell. 20(3):450-462 (2004)
4. Goldberg, D.E., Lingle, R.: Alleles, loci and the Travelling Salesman Problem. In: Grefenstette, J.J. (ed.) ICGA 1985. pp. 154-159. Lawrence Erlbaum Associates (1985)
5. Mariot, L., Leporati, A.: Heuristic Search by Particle Swarm Optimization of Boolean Functions for Cryptographic Applications. In: Laredo, J.L.J., Silva, S., Esparcia-Alcázar, A.I. (eds.) GECCO'15 (Companion), pp. 1425-1426. ACM (2015)
6. Millan, W., Clark, A., Dawson, E.: Smart Hill Climbing Finds Better Boolean Functions. In: Adams, C., Just, M. (eds.) Workshop on Selected Areas in Cryptology 1997, pp. 50-63. Workshop Record, Ottawa (1997)
7. Millan, W., Clark, A., Dawson, E.: Heuristic Design of Cryptographically Strong Balanced Boolean Functions. In: Nyberg, K. (ed.) EUROCRYPT '98. LNCS, vol. 1403, pp. 489-499. Springer, Heidelberg (1998)
8. Picek, S., Jakobovic, D., Golub, M.: Evolving Cryptographically Sound Boolean Functions. In: Blum, C., Alba, E. (eds.): GECCO'13 (Companion), pp. 191-192. ACM (2013)
9. Picek, S., Jakobovic, D., Miller, J., Batina, L.: Evolutionary Methods for the Construction of Cryptographic Boolean Functions. In: Machado, P., Heywood, M.I., McDermott, J., Castelli, M., García-Sánchez, P., Burelli, P., Risi, S., Sim, K. (eds.) EuroGP 2015. LNCS, vol. 9025, pp. 192-204. Springer (2015)
10. Siegenthaler, T.: Correlation-Immunity of Nonlinear Combining Functions for Cryptographic Applications. IEEE Trans. Inf. Theory 30(5), 776–780 (1984)
11. Tarannikov, Y.V.: On Resilient Boolean Functions with Maximum Possible Nonlinearity. In: Roy, B.K., Okamoto, E. (eds.) INDOCRYPT 2000. LNCS, vol. 1977, pp. 19-30. Springer, Heidelberg (2000)
12. Yao, X.: Optimization by Genetic Annealing. In: Jabri, M. (ed.) ACNN '91, pp. 94-97. Sydney Univ. Electr. Eng. (1991)
13. Zheng, Y., Zhang, X.-M.: Plateaued Functions. In: Varadharajan, V., Mu, Y. (eds.) ICICS '99. LNCS, vol. 1726, pp. 284-300. Springer, Heidelberg (1999)