http://dx.doi.org/10.1145/3071178.3071284

Evolutionary Algorithms for the Design of Orthogonal Latin Squares based on Cellular Automata

Luca Mariot[1], Stjepan Picek[2], Domagoj Jakobovic[3], 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
[2]Cyber Security Research Group, Delft University of Technology, Mewelweg 2, Delft, The Netherlands ,
s.picek@tudelft.nl
[3]University of Zagreb, Unska 3, 10000 Zagreb, Croatia ,
domagoj.jakobovic@fer.hr

Abstract

We investigate the design of Orthogonal Latin Squares (OLS) by means of Genetic Algorithms (GA) and Genetic Programming (GP). Since we focus on Latin squares generated by Cellular Automata (CA), the problem can be reduced to the search of pairs of Boolean functions that give rise to OLS when used as CA local rules. As it is already known how to design CA-based OLS with linear Boolean functions, we adopt the evolutionary approach to address the nonlinear case, experimenting with different encodings for the candidate solutions. In particular, for GA we consider single bitstring, double bitstring and quaternary string encodings, while for GP we adopt a double tree representation. We test the two metaheuristics on the spaces of local rules pairs with $n = 7$ and $n = 8$ variables, using two fitness functions. The results show that GP is always able to generate OLS, even if the optimal solutions found with the first fitness function are mostly linear. On the other hand, GA achieves a remarkably lower success rate than GP in evolving OLS, but the corresponding Boolean functions are always nonlinear.

Keywords Orthogonal Latin Squares, Cellular Automata, Genetic Algorithms, Genetic Programming, Boolean Functions, Pairwise Balancedness, Quaternary Strings, Nonlinearity

1 Introduction

Orthogonal Latin Squares (OLS) are one the most studied research topics in the field of combinatorial design theory, and have also several applications in cryptography, coding theory and statistics. The description of OLS is deceptively simple: namely, a *Latin square* of order N is a square matrix of size $N \times N$ where each row and each column is a permutation of the first N positive integers, while two Latin squares are *orthogonal* if by superimposing them one gets all the ordered pairs of the first N positive integers. Yet, this simple definition contrasts with the complexity of constructing OLS for any given order, or even to count them. As a matter of fact there exists a rich body of literature concerning algebraic constructions of OLS, based on Abelian groups, finite fields and other types of combinatorial designs [5, 8, 20].

As far as the authors are aware, there have been no attempts in the literature to apply evolutionary algorithms (EA) for the design of OLS. The closest example one can find is a work by Safadi et al. [18] where genetic algorithms were used to evolve *orthogonal arrays*, a kind of combinatorial designs related to OLS. Ashlock [1] also used genetic algorithms to generate other kinds of combinatorial designs, but not OLS. The reason for this gap in the literature could lie in the difficulty of designing a suitable encoding for the feasible solutions handled by EA. Indeed, it is not simple to optimize the orthogonality of two Latin squares while simultaneously preserving their row-column permutation property using stochastic operators like crossover and mutation.

The goal of this paper is to investigate the use of Genetic Algorithms (GA) and Genetic Programming (GP) to evolve orthogonal Latin squares engendered by Cellular Automata (CA). As shown in [10], every CA with a *bipermutive* local rule induces a Latin square. Hence, we cast our optimization problem as the search of suitable pairs of bipermutive Boolean functions that generate OLS when used as local rules of CA. Moreover, since the authors of [10] already settled the construction problem when the CA rules are *linear*, we consider the general case of OLS based on *nonlinear* bipermutive rules. The motivation for this approach is twofold. The first is to search for regularities and patterns in the solutions found by GA and GP in order to inform the theoretical investigation of OLS generated by nonlinear CA. The second is of cryptographic interest, since orthogonal Latin squares arising from nonlinear constructions can be used to define *cheater immune* secret sharing schemes [21].

To tackle the optimization problem of evolving CA-based OLS, we leverage on the fact that bipermutive rules of n variables are defined by *generating functions* of $n-2$ variables. Hence, the genotype of each individual in the population represents a pair of generating functions, thus ensuring that the corresponding phenotype of the candidate solution is a pair of bipermutive CA producing two Latin squares. Consequently, we can focus the optimization

effort of GA and GP on the orthogonality and nonlinearity properties of the solutions, without checking the row-column permutation constraint.

In the case of GA, we adopt three different encodings for the chromosomes of the candidate solutions. The first one concatenates the truth tables of the two generating functions in a single bitstring, upon which the standard crossover and mutation operators are applied. In the second encoding we consider the two generating functions separately, and apply the genetic operators independently on both of them. Finally, we exploit two facts that we empirically observed in our exhaustive search experiments. First, bipermutive rules of CA producing OLS must be *pairwise balanced*, meaning that each of the four pairs of bits must occur an equal number of times in the superposition of their truth tables. Second, if two generating functions are pairwise balanced then the corresponding bipermutive rules are pairwise balanced as well. We use these observations for the third encoding, where the genotype is represented as a *balanced quaternary string*. For this reason, we also design specific crossover and mutation operators for GA that preserve the balancedness property in quaternary strings. On the other hand, with GP we only use an encoding analogous to the GA double bitstring, where the generating functions are represented by two independent Boolean trees.

We test our GA and GP on the smallest problem instances that are not amenable to exhaustive search, namely the spaces of generating functions pairs of 5 and 6 variables. These instances correspond, respectively, to the sets of bipermutive CA pairs with local rules of $n = 7$ and $n = 8$ variables, or equivalently to the sets of CA-based Latin squares pairs of size 64×64 and 128×128. In the experiments, we adopt two fitness functions, both of which are minimized: the first only counts the number of repeated pairs in the superposition of two Latin squares, while the second also adds a penalty if either or both generating functions are linear.

The results show that GP outperforms GA under all combinations of experiments. As a matter of fact, we observe that GP always converges to an optimal solution, but most of the solutions found under the first fitness function are linear. On the other hand, GA manages to evolve OLS with a much lower success rate, but in this case the solutions found are always nonlinear.

The rest of this paper is organized as follows. Section 2 presents the basic definitions pertaining to Latin squares, Boolean functions and cellular automata, and discusses the construction of Latin squares through bipermutive CA and their exhaustive enumeration for local rules with up to $n = 6$ variables. Section 3 presents the four encodings for the genotype of the candidate solutions used in GA and GP, and describes the ad-hoc genetic operators for balanced quaternary strings. Section 4 describes the experimental setting adopted and discusses the obtained results. Section 5 concludes the paper and points out possible directions for future research.

1	3	4	2
4	2	1	3
2	4	3	1
3	1	2	4

1	4	2	3
3	2	4	1
4	1	3	2
2	3	4	1

1,1	3,4	4,2	2,3
4,3	2,2	1,4	3,1
2,4	4,1	3,3	1,2
3,2	1,3	2,1	4,4

Figure 1: Orthogonal Latin squares of order $N = 4$, and their superposition.

2 Preliminaries

In this section we cover the basic definitions about Latin squares, Boolean functions and cellular automata used in the paper. We also discuss the construction of Latin squares through bipermutive CA, and perform an exhaustive search of orthogonal Latin squares generated by CA with bipermutive local rules of at most $n = 6$ variables.

2.1 Basic Definitions

We start by defining the basic objects of our interest, namely orthogonal Latin squares, referring the reader to [8] for a complete coverage of the subject. In what follows, let $[N]$ denote the set $\{1, \cdots, N\}$ for all $N \in \mathbb{N}$.

Definition 1 *Let $N \in \mathbb{N}$. A* Latin square *of order N is a $N \times N$ matrix L with entries from $[N]$ such that every row and every column are permutations of $[N]$. Two Latin squares L_1 and L_2 of order N are called* orthogonal *if $(L_1(i_1, j_1), L_2(i_1, j_1)) \neq (L_1(i_2, j_2), L_2(i_2, j_2))$ for all $(i_1, j_1), (i_2, j_2) \in [N] \times [N]$, such that $(i_1, j_1) \neq (i_2, j_2)$.*

In other words, two Latin squares are orthogonal if by superposing them one obtains all the pairs of the Cartesian product $[N] \times [N]$. Figure 1 depicts an example of OLS of order $N = 4$.

In this work, we consider cellular automata (CA) as a particular kind of *vectorial Boolean functions*. We thus introduce the basic concepts related to the theory of cryptographic Boolean functions, about which the reader can find further information in Carlet [2, 3].

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 with two elements. Once an ordering of the input variables has been fixed, the basic representation of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is its *truth table*, which is the 2^n-bit vector $\Omega(f)$ that specifies the output value of f for each of the possible 2^n values of the input variables $x_1, \cdots, x_n$. As a consequence, the size of the space of Boolean functions of n variables is 2^{2^n}. An *affine* Boolean function is defined as $f(x_1, \cdots, x_n) = a \oplus a_1 \cdot x_1 \oplus \cdots \oplus a_n \cdot x_n$, where $a, a_1, \cdots, a_n \in \mathbb{F}_2$ and with $\oplus$ and $\cdot$ respectively denoting the XOR and AND operations. If $a = 0$, then the function is called *linear*.

The *nonlinearity* of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is defined as the minimum Hamming distance of its truth table $\Omega(f)$ from the set of all affine functions of n variables. This property can be expressed using the *Walsh transform* of f, defined as

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

for all $\omega \in \mathbb{F}_2^n$, where $\omega \cdot x = \omega_1 x_1 \oplus \cdots \omega_n x_n$ is the *scalar product* between ω and x. Then, the nonlinearity of f is defined as

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

In what follows, we will focus on *bipermutive* Boolean functions. Formally, function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is called *bipermutive* if there exists a *generating function* $\varphi : \mathbb{F}_2^{n-2} \to \mathbb{F}_2$ of $n-2$ variables such that

$$f(x_1, x_2, \cdots, x_{n-1}, x_n) = x_1 \oplus \varphi(x_2, \cdots, x_{n-1}) \oplus x_n \tag{3}$$

for all $x = (x_1, x_2, \cdots, x_{n-1}, x_n) \in \mathbb{F}_2^n$. In [9] it has been shown that the nonlinearity of a bipermutive Boolean function is four times the nonlinearity of its generating function, i.e. $Nl(f) = 4 \cdot Nl(\varphi)$.

Vectorial Boolean functions generalize the concept of Boolean functions to multiple outputs. Given $n, m \in \mathbb{N}$, a *vectorial Boolean function* (or (n,m)-*function*) is a mapping $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$. For all $i \in [m]$, the i-th *coordinate function* of F is the Boolean function $f_i : \mathbb{F}_2^n \to \mathbb{F}_2$ that specifies the i-th output bit of F, i.e. $f_i(x) = F(x)_i$ for all $x \in \mathbb{F}_2^n$.

Using the notions above on Boolean functions, we can now give a formal definition of cellular automaton.

Definition 2 *Let $m, n \in \mathbb{N}$ with $m \geq n$, and let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function. A* cellular automaton *(CA) of length m with* local rule *f is a vectorial Boolean function $F : \mathbb{F}_2^m \to \mathbb{F}_2^{m-n+1}$ defined as*

$$F(x_1, \cdots, x_m) = (f(x_1, \cdots, x_n), \cdots, f(x_{m-n+1}, \cdots, x_m)) \tag{4}$$

for all $x = (x_1, \cdots, x_m) \in \mathbb{F}_2^m$.

Hence, a CA can be viewed as a vectorial Boolean function where each coordinate function f_i is the local rule f applied to the neighborhood of size n of the input variable x_i. Using a description more common in the CA literature, one can consider the first $m-n+1$ input variables $x_1, x_2, \cdots, x_{m-n+1}$ as *cells* which simultaneously update their binary states by evaluating rule f on the neighborhood formed by themselves and the $n-1$ cells to their right. Since we do not update the rightmost $n-1$ input cells, we do not have to put any boundary condition on the CA, as it is done in most of the CA

literature where the number of input cells equals the number of output cells (see e.g. [7]).

In the remainder of this paper, we will consider mainly *bipermutive* CA, i.e. CA whose local rules are bipermutive Boolean functions.

2.2 CA-based Orthogonal Latin Squares

Consider now a CA F of length $m = 2(n-1)$ equipped with a local rule f of n variables. Then, F maps binary vectors of length $2(n-1)$ to binary vectors of length $m - n + 1 = n - 1$. We can use this particular CA to build a corresponding matrix L_F as follows:

- For all $x \in \mathbb{F}_2^m$, the decimal encoding of the *leftmost* $n-1$ bits of x is used to index the *row* i of L_F.
- Analogously, the decimal encoding of the *rightmost* $n-1$ bits of x is used to index the *column* j of L_F.
- Finally, the decimal encoding of the output $F(x)$ is the entry of L_F at coordinates (i, j).

In this way, one obtains a square matrix of size $2^{n-1} \times 2^{n-1}$ whose entries range in the set $\{1, \cdots, 2^{n-1}\}$.

In [10], the following result has been proved:

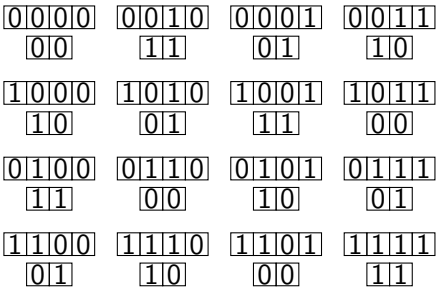
Lemma 1 *Let $F : \mathbb{F}_2^{2(n-1)} \to \mathbb{F}_2^{n-1}$ be a CA defined by a bipermutive local rule $f : \mathbb{F}_2^n \to \mathbb{F}_2$. Then, the square matrix L_F induced by F is a Latin square of order $N = 2^{n-1}$.*

As an example, Figure 2 reports the Latin square L_F induced by the CA $F : \mathbb{F}_2^4 \to \mathbb{F}_2^2$ where the bipermutive local rule is defined as $f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$. The decimal representation of each 2-bit string is summed with 1 so that the result ranges in the set $\{1, \cdots, 2^{n-1}\}$ instead of $\{0, \cdots, 2^{n-1} - 1\}$. Using least significant bit notation, we thus have $00 \to 1$, $10 \to 2$, $01 \to 3$ and $11 \to 4$.

A natural problem to investigate is determining when two bipermutive Boolean functions give rise to orthogonal Latin squares when used as local rules of two CA, using the construction described above. The authors of [10] settled the question for *linear* bipermutive functions, i.e. bipermutive rules defined by linear generating functions. In particular, let $f, g : \mathbb{F}_2^n \to \mathbb{F}_2$ be bipermutive rules defined by linear generating functions $\varphi, \gamma : \mathbb{F}_2^{n-2} \to \mathbb{F}_2$:

$$\varphi(x_2, \cdots, x_{n-1}) = a_2 x_2 \oplus \cdots \oplus a_{n-1} x_{n-1} \tag{5}$$

$$\gamma(x_2, \cdots, x_{n-1}) = b_2 x_2 \oplus \cdots \oplus b_{n-1} x_{n-1} \ , \tag{6}$$



1	4	3	2
2	3	4	1
4	1	2	3
3	2	1	4

Figure 2: Example of Latin square of order $N = 4$ induced by the CA $F : \mathbb{F}_2^4 \to \mathbb{F}_2^2$ with bipermutive rule $f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$.

where $a_i, b_i \in \mathbb{F}_2$ for $i \in \{2, \cdots, n-1\}$. In this case, we can associate to f and g two polynomials $p_f(X), p_g(X) \in \mathbb{F}_2[X]$ of degree $n-1$ using the coefficients of their generating functions as follows:

$$p_f(X) = 1 + a_2 X \oplus \cdots \oplus a_{n-1} X^{n-2} + X^{n-1} \tag{7}$$

$$p_g(X) = 1 + b_2 X \oplus \cdots \oplus b_{n-1} X^{n-2} + X^{n-1} \ . \tag{8}$$

Then, in [10] it is shown that the Latin squares generated by the bipermutive CA with local rules f and g are orthogonal if and only if their associated polynomials $p_f(X)$ and $p_g(X)$ are *relatively prime*.

2.3 Problem Statement and Search Space

Since the problem of designing orthogonal Latin squares has already been solved in [10] for linear bipermutive CA, a natural question that arises is whether there exist also pairs of *nonlinear* bipermutive CA generating OLS, which we formalize as the following combinatorial optimization problem:

Problem 1 *Let $n \in \mathbb{N}$. Maximize the nonlinearity $Nl(f)$ and $Nl(g)$ of two bipermutive Boolean functions $f, g : \mathbb{F}_2^n \to \mathbb{F}_2$ such that the Latin squares L_F and L_G of order 2^{n-1} generated by the CA $F, G : \mathbb{F}_2^{2(n-1)} \to \mathbb{F}_2^{n-1}$ with local rules respectively f and g are orthogonal.*

We remark that Problem 1 is relevant both in combinatorial design theory and in cryptography. In particular, most of the constructions known in literature of combinatorial designs leverage on algebraic methods, based for instance on group theory or finite fields [8]. Hence, finding examples of OLS deriving from other methods like nonlinear CA could lead to develop new algebraic constructions. On the other hand, from a cryptographic perspective OLS can be used to define *threshold secret sharing schemes* (SSS) [19]. However, if one employs OLS coming from linear constructions as in the case of linear bipermutive CA, then the resulting SSS is vulnerable to the *Tompa-Woll attack*, where the players can cheat by submitting fake shares [21]. Therefore,

the search of OLS generated with nonlinear methods is also motivated by the design of *cheater-immune* SSS.

In the rest of this section, we investigate the dimension of the search space containing OLS generated by bipermutive CA with respect to the size of the local rule. Next, we establish up to which value of n the problem is amenable to exhaustive search, and report the numbers of linear/nonlinear pairs of bipermutive CA generating OLS up to that size. Finally, we remark a balancedness property on the pairs of generating functions that produce OLS, which we will exploit in the next section to define an encoding for GA.

Let $\mathcal{B}_n$ be the set of all Boolean functions of n variables. As mentioned in Section 2.1, the size of $\mathcal{B}_n$ is 2^{2^n}. However, we are interested only in pairs of bipermutive Boolean functions, which are completely defined by the respective generating functions lacking the leftmost and rightmost input variables. Hence, for all $n \geq 2$ we can reduce the search space to the set of generating functions of $n-2$ variables, defined as $\mathcal{G}_n = \{(\varphi, \gamma) \in \mathcal{B}_{n-2} \times \mathcal{B}_{n-2}\}$ for $n \geq 2$. It follows that the size of $\mathcal{G}_n$ is $2^{2^{n-2}} \cdot 2^{2^{n-2}} = 2^{2^{n-1}}$, meaning that $\mathcal{G}_n$ is isomorphic to $\mathcal{B}_{n-1}$, i.e. the set of Boolean functions of $n-1$ variables. As a consequence, one can exhaustively search all OLS generated by pairs of bipermutive CA with local rules of up to $n = 6$ variables, since in that case the resulting search space is composed of $2^{2^{6-1}} = 4294967296$ possible pairs. On the other hand, for $n > 6$ the corresponding search space is too huge to be explored in an exhaustive way, motivating the use of heuristic techniques such as GA and GP.

Table 1 reports the distribution of linear and nonlinear pairs of OLS generated by bipermutive CA with local rules of up to $n = 6$ variables that we obtained by exhaustively enumerating the corresponding search spaces. For each value of n, the corresponding size of the Latin squares is reported, along with the number of linear and nonlinear pairs of bipermutive functions generating OLS. Notice that we did not include the trivial case $n = 2$, since there exists only one linear bipermutive function of 2 variables. One can see from Table 1 that the number of nonlinear pairs quickly grows as n increases, overcoming the number of linear pairs. This can be explained by considering that the number of linear functions of n variables is just 2^n, which is a negligible fraction of $\mathcal{B}_n$.

Table 1: Distribution of bipermutive CA-based Orthogonal Latin Squares up to $n = 6$ variables.

n	LS_size	#total	#linear	*#nonlinear*
3	4×4	8	8	0
4	8×8	72	40	32
5	16×16	1704	336	1368
6	32×32	533480	680	532800

From the exhaustive enumeration experiments, we empirically observed the following property: by superimposing the truth tables of two bipermutive rules which give rise to orthogonal Latin squares, the four pairs $(0,0)$, $(1,0)$, $(0,1)$ and $(1,1)$ occur an equal number of times. We call this particular property *pairwise balancedness*. Formally, this can be stated as follows: let $f, g : \mathbb{F}_2^n \to \mathbb{F}_2$ be two bipermutive rules of n variables. Moreover, let $(f,g) : \mathbb{F}_2^n \to \mathbb{F}_2^2$ be the vectorial Boolean function with n inputs and 2 outputs defined as $(f,g)(x) = (f(x), g(x))$ for all $x \in \mathbb{F}_2^n$. Then, rules f and g are called *pairwise balanced* if for all $(y_1, y_2) \in \mathbb{F}_2^2$ it holds $|(f,g)^{-1}(y_1,y_2)| = 2^{n-2}$, that is, the number of preimages mapping to (y_1, y_2) under (φ, γ) is 2^{n-2}. As an example, consider the case where the two bipermutive functions $f, g : \mathbb{F}_2^3 \to \mathbb{F}_2$ are respectively defined by the following truth tables:

$$\Omega(f) = 01011010 \ , \tag{9}$$

$$\Omega(f) = 01101001 \ . \tag{10}$$

It can be seen that each of the four pairs $(0,0)$, $(1,0)$ $(0,1)$ and $(1,1)$ occurs exactly $2^{3-2} = 2$ times as a column in the superposition of Equations (9) and (10). Hence, the bipermutive rules f and g are pairwise balanced.

A second property which we discovered from our experimental observation is that if two generating functions $\varphi, \gamma : \mathbb{F}_2^{n-2} \to \mathbb{F}_2$ are pairwise balanced, then the corresponding bipermutive rules $f, g : \mathbb{F}_2^n \to \mathbb{F}_2$ are pairwise balanced as well, but the converse does not hold. In other words, pairwise balancedness of the generating functions is a sufficient but not necessary condition for two bipermutive rules to be pairwise balanced.

More in general, from our exhaustive search experiments it seems that pairwise balancedness is a necessary but not sufficient condition for two bipermutive rules to induce a pair of OLS. That is, not all pairwise balanced generating functions correspond to pairs of bipermutive CA that generate OLS. Nonetheless, these properties can be employed to further reduce the search space of feasible solutions. In particular, we will use these facts in Section 3.4 to devise an encoding for the chromosomes of GA based on balanced quaternary strings.

3 GA And GP Details

It has been remarked in Section 2.3 that Problem 1 can be reduced to the search of pairs of nonlinear generating functions φ, γ that induce OLS. In fact, by Lemma 1 we already know that the bipermutive functions f, g corresponding to φ, γ give rise to a pair of Latin squares when used as local rules of two CA. Therefore, by representing the genotype of the candidate solutions as pairs of generating functions, we can focus the optimization efforts of GA and GP only on the nonlinearity and orthogonality properties,

without having to consider the row-column permutation constraints of the squares generated by the CA.

In this section, we first describe the two fitness functions that we used to evaluate the phenotype corresponding to a pair of generating functions (φ, γ), i.e. the Latin squares generated by the CA F, G with bipermutive local rules f, g defined by (φ, γ). Then, we proceed by describing the GA and GP encodings that we adopted in our experiments.

3.1 Fitness Functions

Since we are interested in obtaining pairs of OLS, the fitness functions used by GA and GP must in the first place measure the deviation of two Latin squares generated by a pair of generating functions from being orthogonal. The most natural approach is to count the number of *repeated ordered pairs* in the superposition of two Latin squares. The optimization task is thus to minimize such quantity, since having zero repeated pairs in the superposition means that each pair of symbols occurs exactly once (i.e. the two Latin squares are orthogonal). As the exhaustive search results presented in Section 2.3 showed that most of the OLS are generated by pairs of nonlinear CA for $n > 4$, we decided not to check the nonlinearity property in our first fitness function, which is formally defined below.

Let $\varphi, \gamma : \mathbb{F}_2^{n-2} \to \mathbb{F}_2$ be a pair of generating functions of $n-2$ variables, and let $f, g : \mathbb{F}_2^n \to \mathbb{F}_2$ be the corresponding bipermutive functions of n variables. Denote by L_F and L_G the Latin squares of order $[N] = 2^{n-1}$ induced by the CA $F, G : \mathbb{F}_2^{2(n-1)} \to \mathbb{F}_2^{n-1}$ with local rules f and g. Let $Sup_{L_F, L_G} : [N] \times [N] \to [N] \times [N]$ be the *superposition* function of L_F and L_G defined as

$$Sup_{L_F, L_G}(i, j) = (L_F(i, j), L_G(i, j)) \tag{11}$$

for all $(i, j) \in [N] \times [N]$. Then, the value of the fitness function fit_1 evaluated on the individual (φ, γ) is given by:

$$fit_1(\varphi, \gamma) = |rep(L_F, L_G)| \ , \tag{12}$$

where $rep(L_F, L_G)$ is the set defined as:

$$rep(L_F, L_G) = \left\{ (x, y) \in [N] \times [N] : \left| Sup^{-1}_{L_F, L_G}(x, y) \right| > 1 \right\} \ . \tag{13}$$

Remark that the range of fit_1 is $\{0, \cdots, 2^{2(n-1)}\}$, since the two Latin squares L_F and L_G have order 2^{n-1}. In particular, an optimal solution has fitness value 0, hence the optimization objective is to minimize fit_1.

In addition to fit_1, we tested a second fitness function that also takes into account the nonlinearity of the generating functions, to ensure that an optimal solution corresponds to a pair of OLS which is generated by

nonlinear bipermutive CA. In particular, given $\varphi, \gamma : \mathbb{F}_2^{n-2} \to \mathbb{F}_2$ the value of the second fitness function fit_2 computed over (φ, γ) is:

$$fit_2(\varphi, \gamma) = fit_1(\varphi, \gamma) + NlPen(\varphi, \gamma) \cdot N^2 \tag{14}$$

where $N = 2^{n-1}$ and $NlPen(\cdot, \cdot)$ is a penalty factor defined as follows:

$$NlPen(\varphi, \gamma) = \begin{cases} 0 \ , & \text{if } Nl(\varphi) > 0 \ \ AND \ \ Nl(\gamma) > 0 \\ 1 \ , & \text{if } Nl(\varphi) = 0 \ \ XOR \ \ Nl(\gamma) = 0 \\ 2 \ , & \text{if } Nl(\varphi) = 0 \ \ AND \ \ Nl(\gamma) = 0 \end{cases} \tag{15}$$

In other words, the penalty factor is 0 if both generating functions are nonlinear, 1 if only one of them is linear, and 2 if both functions are linear. Considering what we said about the range of fit_1, from (14) and (15) it follows that the range of fit_2 is $\{0, \cdots, 3 \cdot 2^{2(n-1)}\}$. As for fit_1, the optimization objective in this case is to minimize fit_2.

3.2 Single Bitstring Encoding

The first representation for the genotype of GA solutions encodes a pair of generating functions as a single bitstring. Given two generating functions $\varphi, \gamma : \mathbb{F}_2^{n-2} \to \mathbb{F}_2$ respectively with truth tables $\Omega(f), \Omega(g) \in \mathbb{F}_2^{2^{n-2}}$, the chromosome which represents the pair (φ, γ) is defined as $c(\varphi, \gamma) = \Omega(f)||\Omega(g)$, where $||$ denotes the concatenation of two strings. Hence, the chromosome is a bitstring of length 2^{n-1} whose first half corresponds to the truth table of φ, while the second half is the truth table of γ.

Under this encoding, we apply the standard variation operators used in GA, namely one-point crossover and bit-flip mutation. Remark that one-point crossover always produces offspring whose left or right half is inherited from one of the two parents, except when the crossover point happens to be exactly in the middle of the two parents chromosomes (in which case the first child inherits the left half from the first parent and the second half from the second parent, and vice versa for the second child). From the point of view of the phenotype, this means that both children inherit one of the four Latin squares from their parents, or two if the crossover point is in the middle of the parents.

3.3 Double Bitstring and Double Tree Encodings

In the second encoding that we used in our experiments, we considered each individual as composed of two independent parts that represent a pair of generating functions. In particular, in the case of GA each chromosome consists of the two bitstrings representing the truth tables of length 2^{n-2} of the generating functions. Formally, given $\varphi, \gamma : \mathbb{F}_2^{n-2} \to \mathbb{F}_2$, the associated GA chromosome is defined as $c(\varphi, \gamma) = (\Omega(\varphi), \Omega(\gamma)) \in \mathbb{F}_2^{2^{n-2}} \times \mathbb{F}_2^{2^{n-2}}$. Then,

Table 2: Example of balanced quaternary string encoding.

x	000	100	010	110	001	101	011	111
$\varphi(x)$	0	1	0	1	1	0	1	0
$\gamma(x)$	0	1	1	0	1	0	0	1
$c(\varphi,\gamma)$	1	4	3	2	4	1	2	3

one-point crossover and bit-flip mutation are applied independently on the two components of the chromosomes. Notice that in the case of crossover, contrary to the single bitstring encoding, the Latin squares of the offspring always differ from those of the parents, since the two generating functions are recombined independently.

We used a similar double encoding for the candidate solutions evolved by GP. In particular, each chromosome in this case is represented by two Boolean trees which encode the algebraic expressions of the generating functions. Analogously to the GA case, under this encoding we apply the standard tree crossover and mutation operators of GP independently on the two Boolean trees of the generating functions.

3.4 Balanced Quaternary String Encoding

We exploited the pairwise balancedness property mentioned in Section 2.3 to devise a third encoding for the candidate solutions of GA. In particular, given two generating functions $\varphi, \gamma : \mathbb{F}_2^{n-2} \to \mathbb{F}_2$ of $n-2$ variables, in this encoding the chromosome represents the superposition of the truth tables of φ and γ as a *quaternary string* of length 2^{n-2} over the set $Q = \{1, 2, 3, 4\}$, by associating the four pairs of $\mathbb{F}_2^2$ to the elements of Q as follows:

$$(0,0) \to 1; \;\; (1,0) \to 2; \;\; (0,1) \to 3; \;\; (1,1) \to 4 \;\; .$$

Under this encoding, the pairwise balancedness constraint is equivalent to require that each number from 1 to 4 occurs 2^{n-4} times in the string. Consider again the example described in Section 2.3 of the two generating functions $\varphi, \gamma : \mathbb{F}_2^3 \to \mathbb{F}_2$ respectively defined by the truth tables $\Omega(\varphi) = 01011010$ and $\Omega(\gamma) = 01101001$. Table 2 reports the superimposed truth tables of φ and γ along with the corresponding quaternary chromosome $c(\varphi, \gamma)$. From the example, one can observe that each number from 1 to 4 in the chromosome column appears $2^{3-1} = 2$ times.

Clearly, applying classic one-point crossover and mutation operators to balanced quaternary chromosomes does not guarantee that the produced offspring will be balanced as well. Therefore, we designed ad-hoc operators in order to preserve the pairwise balancedness property so that GA searches only the constrained space instead of the whole set of pairs of generating functions.

Our crossover operator is loosely inspired from the operators proposed in [13] for balanced Boolean functions and in [11] for the Walsh spectra of plateaued functions. More precisely, our operator employs four counters to keep track of the multiplicities of the four values in the child chromosome. Given two quaternary chromosomes p_1, p_2 of length N, a child chromosome c is generated as follows:

1. Set the four counters cnt_1, cnt_2, cnt_3 and cnt_4 to 0.
2. If the number of positions where p_1 and p_2 have equal values is greater than $N/2$, then randomly choose p_1 or p_2 and apply the following permutation to each of its loci:
$$1 \to 3;\ \ 2 \to 4;\ \ 3 \to 1;\ \ 4 \to 2\ .$$
3. Determine the positions where p_1 and p_2 have equal values and copy them in the child c.
4. Pick a random position $i \in \{1, \cdots, N\}$ among those which have not already been selected and such that $p_1[i]$ and $p_2[i]$ have different values. Then, the value of $c[i]$ is determined by one of the following cases, depending on the values of the counters $cnt_{p_1[i]}$ and $cnt_{p_2[i]}$:
   (a) If the values of $cnt_{p_1[i]}$ and $cnt_{p_2[i]}$ are both below the threshold 2^{n-4}, randomly copy $p_1[i]$ or $p_2[i]$ in $c[i]$, and increase the corresponding counter.
   (b) If only one of the two counters has reached 2^{n-4}, then copy the value corresponding to the other counter in $c[i]$, and increase such counter.
   (c) If both counters reached 2^{n-4}, copy one of the two remaining values v_1, v_2 in $c[i]$ by applying again cases (a) and (b) to cnt_{v_1} and cnt_{v_2}.
5. Return to step 4 until all positions in the child have been filled.

It can easily be seen from the above procedure that if both parents are balanced quaternary strings, then the generated offspring will be balanced as well. Since this crossover operator produces only one child, we apply it twice for each pair of parents. Notice also that step (2) is performed in order to avoid producing offspring which is too similar to the parents (a similar strategy was also adopted in [13]).

On the other hand, for mutation we adopted a simple operator where each value in a locus to be mutated is swapped with the value in another locus, chosen in a random way. Thus, the balancedness property is preserved since swaps do not change the number of occurrences of the symbols in a string.

4 Experimentation

We tested our GA and GP on the sets of bipermutive functions pairs of $n = 7$ and $n = 8$ variables, which are the smallest instances of Problem 1 not amenable to exhaustive search. In fact, the corresponding search spaces of generating functions pairs of $n - 2 = 5$ and $n - 2 = 6$ variables have sizes 2^{64} and 2^{128}, respectively. From the point of view of the phenotype, $n = 7$ corresponds to Latin squares of size 64×64, while $n = 8$ to Latin squares of size 128×128.

In the remainder of this section, we describe the experimental settings adopted for GA and GP, and we discuss the obtained results.

4.1 Experimental Settings

As mentioned in Section 3.3, the GP encoding uses elementary Boolean functions to build a tree representing each of the two generating functions, whereas the corresponding Boolean variables are used as terminals. The function set in our experiments comprise functions AND, OR, XOR, XNOR, which all take two arguments, and function NOT which takes a single argument. Additionally, we included the function IF, which takes three arguments and returns the second one if the first one evaluates to true, and the third one otherwise. Finally, we set the maximum tree depth to 5. A lower bound on the size of the tree space with such parameters can be estimated using the method described in [6]. In particular, considering only the four binary operators, a tree can be composed of at most 15 internal nodes and 16 leaves. Since each internal node can take 4 different values while the terminal on the leaves are Boolean values, Table 1 of [6] reports a total of $1.82 \cdot 10^{14}$ possible trees, a quantity which is not amenable to exhaustive search. Moreover, as mentioned above, this is a lower bound since we are ignoring the ternary IF operator.

Regarding the population size, in the case of GP we set it to to 500. On the other hand, for GA we set the population size to 30 individuals. In fact, the preliminary experiments that we performed for parameter tuning showed that bigger populations do not produce better results with GA.

For the selection process, we employ a steady-state selection with a 3-tournament operator for both GA and GP, that in each iteration randomly selects three individuals for the tournament and eliminates the worst one. A new individual is created immediately by crossing over the remaining two from the tournament, which then in GP undergoes mutation with probability of 0.5. The variation operators used for GP are simple tree crossover, uniform crossover, size fair, one-point, and context preserving crossover [17] (selected at random) and subtree mutation. For GA, the variation operators are one-point crossover and bit-flip mutation in the case of single and double bitstring encodings, while we adopt the balanced crossover and swap mutation

operators described in Section 3.4 for the quaternary string encoding. In the GA experiments, after an initial phase of parameter tuning we observed that setting the crossover and mutation probabilities respectively to 0.95 and 0.2 yielded the best results.

Common parameters for all the experiments include the termination condition of 300 000 fitness evaluations. We chose this particular bound because our preliminary tests showed that optimal solutions are mostly found before reaching this amount of evaluations, both for GA and GP. Finally, each experiment is repeated 50 times.

4.2 Results

For GA, we performed a total of 6 experiments, given by the combinations of 3 encodings and 2 problem instances. In particular, with GA we used only the first fitness function fit_1 which counts the number of repeated pairs, since we observed that adding the nonlinearity constraint did not modify the performances in a significant way. On the other hand, with GP we performed a total of 4 experiments, given by the combinations of 2 fitness functions and 2 problem instances. In what follows, we compactly denote a GA experiment as (GA, n, enc_i), where n is the number of variables of the bipermutive functions (which thus can be either 7 or 8), while enc_i represents the encoding adopted. In particular, enc_1 stands for single bitstring, enc_2 for double bitstring and enc_3 for balanced quaternary strings. Likewise, we denote a GP experiment as (GP, n, fit_i), where n still stands for the number of variables, while fit_i denotes the fitness function.

Table 3 reports the results obtained in each experiment. In particular, for each combination we show the average fitness and the standard deviation computed on the best solutions generated over all 50 runs, along with the number of optimal solutions found and their distribution as linear/nonlinear pairs. In general, one can observe that GP has a clear advantage over GA, since it always converges to an optimal solution in each experimental run for both $n = 7$ and $n = 8$ variables. On the other hand, GA only manages to generate OLS for $n = 7$ variables (except for a single optimal solution of $n = 8$ variables found under the single bitstring encoding). Moreover, even in the case of $n = 7$ variables it can be seen that the success rate of GA in generating OLS is remarkably lower than that achieved by GP. One can additionally remark that the balanced quaternary encoding gives a slight advantage to GA over single and double bitstrings, with respect to the number of optimal solutions found.

Interestingly, one can notice that the optimal solutions produced by GP under fitness function fit_1 are mostly given by linear pairs. More precisely, under fit_1 GP managed to find only 3 nonlinear pairs of $n = 8$ variables which generated OLS. The addition of the nonlinearity penalty factor with fit_2 however solved the issue, since in this case all optimal solutions found

Table 3: Best solutions found by GA and GP.

Exp.	avg	std	#opt	#lin	#nlin
$(GA, 7, enc_1)$	520.32	360.16	12/50	0	12
$(GA, 7, enc_2)$	565.44	389.03	15/50	0	15
$(GA, 7, enc_3)$	392.64	328.47	18/50	0	18
$(GA, 8, enc_1)$	4165.44	604	1/50	0	1
$(GA, 8, enc_2)$	4222.16	125.03	0/50	0	0
$(GA, 8, enc_3)$	4696.48	135.51	0/50	0	0
$(GP, 7, fit_1)$	0	0	50/50	50	0
$(GP, 7, fit_2)$	0	0	50/50	0	50
$(GP, 8, fit_1)$	0	0	50/50	47	3
$(GP, 8, fit_2)$	0	0	50/50	0	50

are given by pairs of nonlinear generating functions. On the other hand, it can be observed that all optimal solutions found by GA with fitness functions fit_1 are nonlinear. This difference could be explained by the fact that the set of operators used for GP trees include also the XOR, which is a linear operator. Hence, when the optimization criterion is just the minimization of the repeated pairs, it could be easier for GP to find pairs of generating functions whose trees are composed only of XOR, which correspond to linear solutions. Since the number of linear functions is much smaller than the total number of Boolean functions, it could be that GP finds very quickly an orthogonal solution by sticking to linear pairs. On the other hand, there is no clear relationship between the truth-table based encodings used by GA and the nonlinearity of the generating functions, which could explain why GA always find nonlinear optimal solutions, even if with much more difficulty.

Considering the nonlinear optimal solutions found by GA and GP, we can additionally remark an interesting fact. First, all the optimal bipermutive rules found using the single and double bitstring representations with GA and the double tree encoding with GP satisfy the pairwise balancedness property introduced in Section 2.3. Since these encodings do not enforce pairwise balancedness as a constraint (like with quaternary strings on the generating functions), this finding seems to support the conjecture that all bipermutive rules pairs inducing OLS must be pairwise balanced, a fact that we experimentally assessed by exhaustive search up to $n = 6$ variables.

5 Conclusion

In this paper, we addressed the problem of designing orthogonal Latin squares generated by nonlinear bipermutive CA using GA and GP. Specifically, since bipermutive Boolean functions of n variables are completely defined by their

generating functions of $n-2$ variables, we formulated the optimization problem as the search of pairs of nonlinear generating functions inducing OLS. We experimented three different encodings for the candidate solutions of GA, namely single bitstring, double bitstring and balanced quaternary string, introducing ad-hoc crossover and mutation operators for the last one. On the other hand, with GP we adopted a double tree representation. We tested the two metaheuristics on the sets of pairs of bipermutive functions of $n = 7$ and $n = 8$ variables, remarking that GP is always able to converge to an optimal solution in both problem instances, while GA manages to generate OLS with a lower success rate only for $n = 7$. On the other hand, we also observed that GP mostly finds linear solutions when the fitness function counts only the number of repeated pairs, while the solutions found by GA are always nonlinear.

We emphasize that, as far as we know, there are no other works in the literature concerning the design of orthogonal Latin squares by means of evolutionary algorithms to compare our results with. Nonetheless, we deem that this problem is interesting from the evolutionary computation perspective, and that our results set a first baseline of comparison for future works on the subject.

There are several research directions along which the present work can be extended. From a theoretical side, an interesting problem arising from our results would be to prove the conjectures about the pairwise balancedness on the bipermutive rules and their generating functions. In particular, pairwise balancedness on the generating functions seems to be a sufficient condition on the respective bipermutive rules to be pairwise balanced as well. Pairwise balancedness on the bipermutive rules, on the other hand, seems to be a necessary condition for two CA to generate orthogonal Latin squares. On the experimental side, it would be interesting to compare the performance of GA and GP with other optimization algorithms. Since the objects we are dealing with in this optimization problem are Boolean functions used as CA rules, one could leverage on the several works which have been published about the evolution of Boolean functions with good cryptographic properties. These include both population-based approaches like discrete PSO [12] and Cartesian GP [15, 16], as well as local search methods such as Simulated Annealing [4]. A different comparison perspective worth exploring would also be to adapt algebraic constructions of Boolean functions evolved through GP [14] in order to generate orthogonal Latin squares.

Another interesting experimental direction to investigate would be to increase the number of variables of the generating functions, to assess up to which dimension of the problem GP is able to produce optimal solutions. Moreover, one could also consider the natural extension of evolving k *Mutually Orthogonal Latin Squares* (MOLS) based on CA. In this case, the encoding is a straightforward extension of the double tree representation, since it suffices to represent a candidate solution with k independent trees. The fitness

function can also be easily modified by summing the number of repeated pairs in each superposition of the k Latin squares.

6 Acknowledgments

This work has been supported in part by Croatian Science Foundation under the project IP-2014-09-4882.

References

[1] D. Ashlock. Finding designs with genetic algorithms. In *Computational and Constructive Design Theory*, pages 49–65. Springer, 1996.

[2] C. Carlet. Boolean Functions for Cryptography and Error Correcting Codes. In Y. Crama and P. L. Hammer, editors, *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, pages 257–397. Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[3] C. Carlet. Vectorial Boolean Functions for Cryptography. In Y. Crama and P. L. Hammer, editors, *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, pages 398–469. Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[4] J. A. Clark, J. L. Jacob, S. Stepney, S. Maitra, and W. Millan. Evolving boolean functions satisfying multiple criteria. In *Progress in Cryptology - INDOCRYPT 2002, Third International Conference on Cryptology in India, Hyderabad, India, December 16-18, 2002*, pages 246–259, 2002.

[5] C. J. Colbourn and J. H. Dinitz. Making the mols table. In *Computational and Constructive Design Theory*, pages 67–134. Springer, 1996.

[6] M. Ebner. On the search space of genetic programming and its relation to nature's search space. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 2, pages 1357–1361, 1999.

[7] J. Kari. Basic concepts of cellular automata. In *Handbook of Natural Computing*, pages 3–24. 2012.

[8] A. D. Keedwell and J. Dénes. *Latin squares and their applications*. Elsevier, 2015.

[9] A. Leporati and L. Mariot. Cryptographic properties of bipermutive cellular automata rules. *J. Cellular Automata*, 9(5-6):437–475, 2014.

[10] L. Mariot, E. Formenti, and A. Leporati. Constructing orthogonal latin squares from linear cellular automata. *CoRR*, abs/1610.00139, 2016.

[11] L. Mariot and 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*, pages 33–45, 2015.

[12] L. Mariot and 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*, pages 1425–1426, 2015.

[13] W. Millan, A. Clark, and E. Dawson. Heuristic design of cryptographically strong balanced Boolean functions. In *Advances in Cryptology - EUROCRYPT '98*, pages 489–499, 1998.

[14] S. Picek and D. Jakobovic. Evolving algebraic constructions for designing bent boolean functions. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*, pages 781–788. ACM, 2016.

[15] S. Picek, D. Jakobovic, and M. Golub. Evolving Cryptographically Sound Boolean Functions. In *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '13 Companion, pages 191–192, New York, NY, USA, 2013. ACM.

[16] S. Picek, D. Jakobovic, J. F. Miller, L. Batina, and M. Cupic. Cryptographic boolean functions: One output, many design criteria. *Appl. Soft Comput.*, 40:635–653, 2016.

[17] R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[18] R. Safadi and R. Wang. The use of genetic algorithms in the construction of mixed multilevel orthogonal arrays. Technical report, DTIC Document, 1992.

[19] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[20] D. R. Stinson. *Combinatorial designs - constructions and analysis*. Springer, 2004.

[21] M. Tompa and H. Woll. How to share a secret with cheaters. *J. Cryptology*, 1(2):133–138, 1988.