Evolving S-boxes Based on Cellular Automata with Genetic Programming

Stjepan Picek[1], Luca Mariot[2], Domagoj Jakobovic[3], and Alberto Leporati[1]

[1]KU Leuven, imec-COSIC, Kasteelpark Arenberg 10 3001 Leuven, Belgium ,
stjepan@computer.org
[2]Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano-Bicocca, Viale Sarca 336, 20126 Milano, Italy ,
{luca.mariot, leporati}@disco.unimib.it
[3]University of Zagreb, Unska 3, 10000 Zagreb, Croatia ,
domagoj.jakobovic@fer.hr

Abstract

The design of cryptographically strong Substitution Boxes (S-boxes) is an interesting problem from both a cryptographic perspective as well as the combinatorial optimization one. A number of papers appeared throughout the years that utilized various forms of heuristic search techniques in order to find S-boxes with good cryptographic values. The permutation encoding showed to be the most successful one but even there the obtained results were bad for sizes larger than 4×4. Consequently, for a number of years the general perception was that heuristics cannot compete with deterministic techniques when designing good S-boxes. Here, we introduce the concept of evolving cellular automata rules that can be then translated into S-boxes. With it, we are able to find optimal S-boxes for sizes from 4×4 up to 7×7. As far as we are aware, this is the first time a heuristic approach is able to find optimal S-boxes for sizes larger than 4. Besides that, we also present a Genetic Programming implementation that is able to "reverse engineer" S-boxes, i.e., to find a cellular automata rule from an S-box corresponding to it.

Keywords Substitution boxes, Genetic Programming, Cellular automata, Cryptography

1 Introduction

In the process of designing block ciphers, one may follow several options. One well explored direction is to build the so-called Substitution-Permutation

Network (SPN) cipher. Such ciphers usually consist of an XOR operation with the key/subkeys, a linear layer, and a substitution layer [18]. A standard way to build the substitution layer is to use one or more Substitution Boxes (S-boxes, vectorial Boolean functions) where a number of properties need to be satisfied for an S-box to be useful in practice. An S-box, or (n, m) function is a mapping from n inputs into m outputs.

Indeed, already Shannon in his seminal work on the design of block ciphers introduced the concept of *confusion* that an S-box needs to have [26]. Here, confusion can be defined as the property that the ciphertext statistics should depend on the plaintext statistics in a manner which is too complicated to be exploited by an attacker. This concept is most easily connected with the cryptographic property of *nonlinearity*. However, to find an S-box that is resilient against various attacks is not easy and the problem becomes even more complicated if we consider various sizes of S-boxes that are of practical relevance. For instance, some occurring S-box sizes are 3×3 (3Way [15]), 4×4 (PRESENT [5]), 5×5 (Keccak [3]), and 8×8 (AES [16]). Note that the three most used sizes of S-boxes are 4×4, 5×5, and 8×8.

From the cryptographic properties perspective, the bare minimum one would need to consider if designing S-boxes with the same number of inputs and outputs (as we consider in this paper) is for them to be bijective, with high nonlinearity, and low differential uniformity. When utilizing heuristics in the design of S-boxes, a common approach is to use the permutation encoding since it ensures the bijectivity property. However, already for the size 5×5 heuristics are not able to reach the optimal values of nonlinearity and differential uniformity and therefore algebraic constructions are the common method of choice [21]. Here, by algebraic constructions we consider deterministic methods relying on results from field theory [9].

However, there are several ciphers that use S-boxes not directly obtained by algebraic constructions or heuristics, but where the S-boxes are actually cellular automata (CA). The best known example is the Keccak sponge construction [3] that is now a part of the SHA-3 standard. There, the authors use a cellular automaton rule affecting only two neighborhood positions for each bit, which results in an extremely lightweight definition of the S-box and a small implementation cost. However, that S-box has suboptimal cryptographic properties, which results in a construction that requires more rounds than would be the case with optimal S-boxes. As far as the authors know, all the other ciphers using CA rules for the S-box definition actually use that same rule. This is the case of Panama [13], RadioGatún [2], Subterranean [11], and 3Way [15] ciphers. Besides those S-boxes, there are also designs that use an S-box that is an affine transformation of the Keccak S-box like Ascon [10].

In this paper, we focus on the investigation of cellular automata rules that are able to produce S-boxes with optimal cryptographic properties. In order to achieve that, we use Genetic Programming (GP) to evolve CA rules in

the form of Boolean functions, where those rules are actually the description of S-boxes. With our approach we are able to produce large quantities of S-boxes with optimal cryptographic properties, dependent on the chosen optimization criteria. Furthermore, due to the GP solution encoding in the form of trees, we can also easily impose the length of the solution which can be then mapped to the latency of such an S-box, but also to the area when implemented in hardware. We emphasize that not every S-box that is a permutation can be represented with a CA rule, and therefore the number of S-boxes expressible with CA rules is smaller than the total number of S-boxes of a certain size. Indeed, if we consider the AES cipher and the S-box of size 8×8 used there, it is possible to see that this S-box cannot be obtained with a single CA rule, but this does not mean there are no S-boxes of size 8×8 with the same properties that cannot be designed with a single CA rule. Moreover, there are infinitely many ways how one can represent an S-box with CA rules: for example, adopting a tree representation for the rule, it suffices to consider the trivial approach where one adds subexpressions that cancel themselves out. Therefore, the number of CA rules representations is much larger than the number of S-boxes and it is impossible to exhaustively visit them even for smaller sizes.

Finally, we present here a"reverse-engineering" procedure that is able to find underlying CA rules that result in specific S-boxes. Indeed, if we start with an S-box that can be represented with a single CA rule, the question is whether there exists a shorter rule resulting in the same S-box. Our reverse engineering can help in obtaining such shorter rules.

2 Cryptographic Properties of S-boxes

Let n, m be positive integers, i.e., $n, m \in \mathbb{N}^+$. The set of all n-tuples of elements in the field $\mathbb{F}_2$ is denoted as $\mathbb{F}_2^n$, where $\mathbb{F}_2$ is the Galois field with two elements. The inner product of two vectors a and b equals $a \cdot b = \bigoplus_{i=1}^{n} a_i b_i$ and "$\bigoplus$" represents addition modulo two. An (n, m)-function is any mapping F from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. A (n, m)-function F can be defined as a vector $F = (f_1, \cdots, f_m)$, where the Boolean functions $f_i : \mathbb{F}_2^n \to \mathbb{F}_2$ for $i \in \{1, \cdots, m\}$ are called the *coordinate functions* of F. The *component functions* of an (n, m)-function F are all the linear combinations of the coordinate functions with non all-zero coefficients. To graphically depict the difference between the component and coordinate functions, we show an example of an S-box of size 3×3 in Figure 1.

A Boolean function f on $\mathbb{F}_2^n$ is represented by a truth table (TT), which is a vector $(f(0), ..., f(1))$ that contains the function values of f in lexicographical order with respect to the input entries, i.e., for $a, b \in \mathbb{F}_2^n$, it holds $a \leq b$ if and

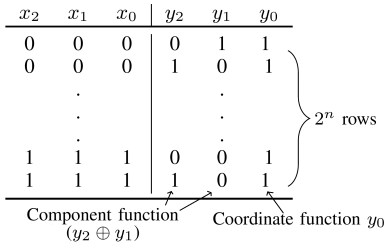

Figure 1: S-box of size (3×3).

only if $a_i \leq b_i$ for all $i \in \{1, \cdots, n\}$ [8]. An (n, m) S-box can be represented in the truth table form as a matrix of dimension $2^n \times m$ where each of the m columns represents a Boolean function.

The Walsh-Hadamard transform of an (n, m)-function F is defined as [9]:

$$W_F(a, v) = \sum_{x \in \mathbb{F}_2^n} (-1)^{v \cdot F(x) \oplus a \cdot x}, \ a, v \in \mathbb{F}_2^m. \tag{1}$$

An (n, m)-function F is *balanced* (BAL) if it takes every value of $\mathbb{F}_2^m$ the same number 2^{n-m} of times.

The *nonlinearity* N_F of an (n, m)-function F equals the minimum nonlinearity of all its component functions $v \cdot F$, where $v \in \mathbb{F}_2^{m*}$ [20, 9]:

$$N_F = 2^{n-1} - \frac{1}{2} \max_{\substack{a \in \mathbb{F}_2^n \\ v \in \mathbb{F}_2^{m*}}} |W_F(a, v)|. \tag{2}$$

The maximal nonlinearity for any (n, n)-function F is bounded by the following inequality:

$$N_F \leq 2^{n-1} - 2^{\frac{n-1}{2}}. \tag{3}$$

In the case of equality in Eq. (3), F is called an Almost Bent (AB) function.

Let F be a function from $\mathbb{F}_2^n$ into $\mathbb{F}_2^n$ and $a, b \in \mathbb{F}_2^n$. We denote:

$$D_F(a, b) = |\{x \in \mathbb{F}_2^n : F(x + a) + F(x) = b\}|. \tag{4}$$

The entry at the position (a, b) corresponds to the cardinality of the *delta difference table* $D(a, b)$ and is denoted as $\delta(a, b)$. The *differential uniformity* δ_F is then defined as [4, 19]:

$$\delta_F = \max_{a \neq 0, b} \delta(a, b). \tag{5}$$

When discussing the differential uniformity property, the best possible value is 2 for any odd n and also for $n = 6$. For n even and larger than 6, this is an open question. The best known differential uniformity value for the sizes 4×4 and 8×8 equals 4. Functions that have differential uniformity δ_F equal to 2 are called Almost Perfect Nonlinear (APN) functions [9]. All AB functions are also APN functions, but the converse is not true in general. For further information about S-boxes, we refer the interested reader to [9, 18].

3 Cellular Automata

A Cellular Automaton (CA) is a parallel computational model characterized by a regular grid of cells, such that each cell synchronously updates its state according to a local rule, which depends on a specific number of neighboring cells.

In this paper we consider a CA model where the cells are arranged on a finite one-dimensional array and are described by a binary state, 0 or 1. This can be formalized through the following definition:

Definition 1 *A* cellular automaton *is a quintuple* $A = \langle c, n, \delta, \omega, f\rangle$ *where* c *is a one dimensional array of* n cells*, each of which takes a value in* $\mathbb{F}_2$*,* $\delta \in \mathbb{N}$ *is the* diameter*,* $\omega \in \{-\delta+1, \cdots, 0, \cdots, \delta-1\}$ *is the* offset *and* $f : \mathbb{F}_2^\delta \to \mathbb{F}_2$ *is the* local rule*.*

Since the local rule $f : \mathbb{F}_2^\delta \to \mathbb{F}_2$ is a Boolean function, it can be represented by a truth table of 2^δ bits. In the CA literature a local rule is usually indexed by the decimal representation of its truth table, which is commonly referred to as the *Wolfram code* of the rule.

The next state of a cell c_i is determined by applying rule f to the neighborhood $(c_{i-\omega}, \cdots, c_{i-\omega+\delta-1})$. Clearly, this leads to the problem of updating the cells at the left and right boundaries, where there are no enough left (respectively, right) neighbors. In this work we adopt *periodic boundary conditions* to address this issue, where the cellular array is viewed as a ring with the last cell preceding the first one.

The *global rule* $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ of a CA $\langle c, n, \delta, \omega, f\rangle$ maps the current state of array c to the next one by applying in parallel rule f to all n cells. In particular, under periodic boundary conditions the global rule F is defined for all values $x \in \mathbb{F}_2^n$ of c as:

$$F(x) = (f(x_{-\omega}, \cdots, x_{\delta-\omega}), \cdots, f(x_{n-1-\omega}, \cdots, x_{n-\omega+\delta-2})) \ , \qquad (6)$$

where all indices are computed modulo n. As a consequence, the global rule of a finite CA is an (n, n)-function, where $\forall i \in \{0, \cdots, n-1\}$ the i-th coordinate function $f_i : \mathbb{F}_2^n \to \mathbb{F}_2$ corresponds to the local rule f applied to the neighborhood of cell c_i.

Since the Keccak cipher is the best known example of a CA rule used in the construction of an S-box, here we give further details about it. The rule used in Keccak on an array of length $n = 5$ can be represented as:

$$c_i(t+1) = c_i(t) XOR((NOT(c_{i+1}(t)))\ AND\ c_{i+2}(t)), \qquad (7)$$

where $0 \le i < 5, t \in \mathbb{N}$. The above rule is applied with periodic boundary conditions to each of the 5 cells composing the current state at step t to produce the next state in step $t+1$. Note that Eq. (7) is in fact a Boolean function which defines the CA local update rule.

4 Related Work

Here, we present a number of relevant works where CA rules are used in the construction of block ciphers/S-boxes and where EC is used in order to evolve S-boxes with improved cryptographic properties.

4.1 CA-Based Block Ciphers and S-Boxes

Most of the block ciphers based on the dynamics of cellular automata focus on the use of *reversible* CA (RCA). Formally, a CA is reversible if its global rule $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is bijective and the inverse $G = F^{-1}$ is again the global rule of a CA. In a CA-based block cipher, the idea is to represent a block of plaintext as the initial configuration of the CA. The global rule is then applied for a certain number of steps to obtain the encrypted block. For decryption, the inverse global rule is applied for the same number of steps starting from the ciphertext block to recover the plaintext.

The first block cipher based on cellular automata was proposed by Gutowitz [17]. In particular, for the substitution phase *block CA* were used to ensure the invertibility of the resulting S-box. In a block CA, the local rule does not determine the next state of a single cell, but rather the state of a block of adjacent cells. The cellular array is partitioned in blocks of equal length, and then a permutation is applied to each block in parallel. In the next step, the partition is shifted one cell to the right with periodic boundary conditions.

A second type of CA which have been used for block ciphers are *second-order CA*, where the state of a cell is determined by XOR-ing its previous state with the result of the local rule. Hence, the configuration at time $t-1$ can be computed by knowing both the configurations at time t and $t+1$. Seredynsky et al. [25] investigated second-order CA as S-boxes, by assessing the *avalanche properties* of several rules with diameter $\delta = 5, 7$ and array lengths $n = 32, 64$.

Another interesting kind of CA for block ciphers are the so-called *complementing landscapes cellular automata* (CLCA), where the state of a cell is flipped if and only if a pattern belonging to a specific landscape occurs in the surrounding cells. Daemen et al. [14] studied CLCA for designing block ciphers, discovering a particular rule (called χ) with a diameter $\delta = 3$ and offset $\omega = 0$, which induces invertible CA if the length n of the cellular array is odd. Additionally, rule χ has a simple description in terms of correlation and propagation characteristics, which makes it interesting for cryptographic applications. Indeed, this CA is the only nonlinear component used in Keccak [3]. Note that if such a rule would be used when the length of the cellular array is even, then the resulting S-box would not be bijective. Furthermore, since the Keccak rule has diameter of size 3, it results in an optimal S-box of size 3×3, while for the size 5×5 the resulting S-box has

suboptimal cryptographic properties. Naturally, as one would extend the size of an S-box, the cryptographic properties would become increasingly worse.

4.2 Evolutionary Design of S-boxes

From the EC perspective, we mention only several characteristic approaches, all of which use the permutation encoding. Clark et al. used the principles from the evolutionary design of Boolean functions to evolve S-boxes with the desired cryptographic properties for sizes up to 8×8 [12].

Burnett et al. used a heuristic method to generate MARS-like S-boxes [7]. With their approach, they were able to generate a number of S-boxes of appropriate sizes that satisfy all the requirements placed on a MARS S-box.

Picek et al. used Cartesian Genetic Programming and Genetic Programming to evolve S-boxes and discussed how to obtain permutation based encoding with those algorithms [22]. Furthermore, they argued that the representation in the form of a truth table of coordinate Boolean functions is an approach that works only for small S-boxes (up to the size 4×4). Finally, Picek et al. present an improved fitness function with which EC is able to find higher nonlinearity values for a number of S-box sizes [21].

5 Experimental Setup and Results

In this section we present our experimental setting as well as the results we obtained.

5.1 Genetic Programming Approach

Genetic Programming (GP) is an evolutionary algorithm in which the data structures that undergo optimization are computer programs [1]. Although GP has a history longer than 50 years, its full acceptance is due to the work of John Koza at the beginning of the 1990s in which he formalized the idea of employing chromosomes on the basis of tree data structures. Since the aim of GP is to automatically generate new programs, each individual of a population represents a computer program [1], where the most common are symbolic expressions representing parse trees. A parse tree (syntax tree) is an ordered, rooted tree that represents the syntactic structure of a string according to some context-free grammar. A tree can represent a mathematical expression, a rule set or a decision tree, for instance. The building elements in a tree-based GP are functions (inner nodes) and terminals (leaves, problem variables) where both functions and terminals are known as primitives.

In our approach, the task of GP is to evolve a Boolean function of n variables, in the form of a tree, which is used as a CA local rule. In this process, we assume the following: the state of a CA is represented with a

periodic one-dimensional binary array of size n. The elements of the binary array are used as GP terminals, where the terminal c_0 denotes the value that is being updated. The terminals $c_1, \ldots, c_{n-1}$ denote the cells to the right of the current cell. In our experiments, the neighborhood of a cell is formed by the cell itself and the $n-1$ cells to its right, so each value in the current state can be used in a local update rule.

A candidate Boolean function obtained with GP is evaluated in the following manner: all the possible 2^n input states are considered, and for each state the same rule is applied in parallel to each of the bits to determine the next state. The obtained global rule represents a candidate S-box that is then evaluated according to the desired cryptographic criteria. The function set consists of several Boolean primitives necessary to represent any Boolean function. Here, we use the following function set: NOT, which inverts its argument, XOR, NAND, NOR, each of which takes two input arguments. Additionally, we use the function IF, which takes three arguments and returns the second one if the first one evaluates to *true*, and the third one otherwise. This function corresponds to the multiplexer gate (MUX).

In the evolution process, GP uses a 3-tournament selection, where the worst of 3 randomly selected individuals is eliminated. A new individual is then created by applying crossover to the remaining two individuals from the tournament. The new individual is then mutated with a probability of 0.5. We note that we use the mutation probability to select whether an individual would be mutated or not, and the mutation operator is executed only once on a given individual; e.g. if the mutation probability is 0.5, then on average 5 out of every 10 new individuals will be mutated and one mutation will be performed on that individual. The variation operators are simple tree crossover, uniform crossover, size fair, one-point, and context preserving crossover [24] (selected at random) and subtree mutation. All our experiments suggest that having a maximum tree depth equal to the size of S-box is sufficient (i.e., tree depth equals n, which is the number of terminals). The initial population is created at random with ramped half-and-half initialization; every experiment is repeated 50 times.

In order to examine the influence of the GP parameters, we conduct a tuning phase for the stopping criterion and the population size. The starting set of parameters was tested on S-boxes of size $n = 6$, with population 500, 1 000, and 2 000, for which 30 runs were executed. Although there were no significant differences, the best results in terms of average fitness value and the number of optimal solutions were obtained with a population size of 2 000, which we use in the subsequent experiments. Furthermore, the stopping criterion is set to 2 000 000 evaluations, since no change of the best solution was detected afterwards.

5.2 Fitness Function

In this paper, we try to find (n, n) S-boxes that have at least the minimal necessary properties to be used in real world ciphers. Therefore, we require that the evolved S-boxes are balanced, with high nonlinearity, and low differential uniformity. We note that those are the standard minimum properties one should consider, although there are other properties that are important, but out of the scope of this work. The balancedness and nonlinearity (and seldom differential uniformity) are the properties of choice when using EC to evolve S-boxes, see e.g., [22, 23].

With the goal of finding balanced S-boxes that have as high as possible nonlinearity and as low as possible differential uniformity, we use a two-stage fitness function. First, the balancedness is verified, and if an S-box is balanced, we give it a value of zero, otherwise the value equals -1; this is denoted with the label BAL. Only if the S-box is balanced, we calculate the nonlinearity and δ-uniformity (which is subtracted from the value 2^n, since we aim to minimize the value of that property) and **maximize** the resulting value:

$$fitness_1 = BAL + \Delta_{BAL,0}(N_F + (2^n - \delta_F)). \tag{8}$$

Here, $\Delta_{BAL,0}$ represents the Kronecker delta function that equals one when the function is balanced (i.e., $BAL = 0$) and zero otherwise. We emphasize that we opted not to use a multi-objective approach since we consider balancedness as a constraint rather than a separate objective, and we are not interested in solutions (no matter how good with the respect to the other properties) if they are not balanced.

5.3 Improving the Granularity of the Fitness Function

In the early phase of experiments, we noted that the nonlinearity property as defined in Eq. (2) offers little gradient information which would guide the evolution towards better solutions. This is also apparent in the related work where nonlinearity is considered as an objective [21]. Rather than observing only the minimum nonlinearity of all component functions, as defined in Eq. (2), we include additional information in terms of the number of component functions which posses the same minimum nonlinearity. This number of occurrences is added as a secondary criterion that should be *minimized*, since we aim to eliminate the components that contribute to the lowest nonlinearity value.

Following the above, we define another fitness function which aims to maximize the nonlinearity while still keeping balancedness as a constraint:

$$fitness_2 = BAL + \Delta_{BAL,0}\left(N_F + \left(1 - \frac{nMinN_F}{2^n}\right)\right), \tag{9}$$

where $nMinN_F$ represents the number of occurrences of the current value of N_F. Since the difference in neighboring levels of nonlinearity is always

greater than 1, the second part of the expression acts as a secondary criterion which is being minimized and assumes values in the range $[0, 1]$.

Finally, we combine the previous and the first fitness function by adding the term that minimizes differential uniformity, as follows:

$$fitness_3 = BAL + \Delta_{BAL,0}\left(N_F + \left(1 - \frac{nMinN_F}{2^n}\right) + (2^n - \delta_F)\right). \quad (10)$$

For all the previous fitness functions the goal is maximization, i.e., a larger fitness value denotes a better solution.

5.4 Experimental Results

In Table 1, we give statistical results for our heuristic search. The results are averaged over the best obtained values for each run. In column *Theory max* we denote the theoretical maximal value of the fitness (or one that would correspond to the assumed theoretical maximal value). These results serve as an indicator on the performance of our search technique.

Note that the complete search space size equals $2^{n \cdot 2^n}$ for an $n \times n$ function and within that search space there are $n!$ permutations. However, only a part of that permutation space can be expressed with a single CA rule. Furthermore, when comparing the difficulty of evolving Boolean functions and S-boxes we note that the complexity rises exponentially with the number of outputs, i.e., we are not interested in only checking each of the output functions but also all the non-zero linear combinations of those functions. For instance, for the 8×8 function, there are 255 combinations of output functions where nonlinearity equals the lowest nonlinearity of all combinations.

As can be seen in the table, for dimensions 4×4 and 5×5 the problem is easy. Indeed, for the former, all the experiments finished with the optimal value, and for the latter, most of the runs reached the optimal solution.

For the size 6×6, we observe a larger standard deviation, but the average value is still high. However, for the size 7×7, we see that the problem becomes difficult, although there are a few runs reaching the optimal value. Finally, for the size 8×8, we see that our approach does not even come close to the optimal value. Observe that for sizes 4×4, 5×5, and 7×7 we obtained S-boxes with the best possible values of nonlinearity and differential uniformity. For 6×6 size our best solution is close to the optimal one. Actually, the difference between our solution and the optimal one is in the differential uniformity property where our solution has value 4 and the optimal value equals 2 [6]. However, we investigated that S-box and it is not possible to obtain it with a single CA rule (see Sec. 5.5), which means that our approach results in the best possible obtainable values for the 6×6 size.

Next, in Table 2 we give the best obtained values for the cryptographic properties for each S-box size. In the column *CA Rule*, we give the rule (the shortest we found to be easier to present in a table) that defines the specific

Table 1: Statistical results, $fitness_1$, Eq. (8)

S-box size	Theory max	GP		
		Max	Avg	Std dev
4×4	16	**16**	16	0
5×5	42	**42**	41.73	1.01
6×6	86	**84**	80.47	4.72
7×7	182	**182**	155.07	8.86
8×8	364	**318**	281.87	13.86

Table 2: Examples of S-boxes and their properties.

S-box size	N_F	δ_F	CA Rule
4×4	4	4	IF(((v3 NOR v1) XOR v0), v2, v1)
5×5	8	8	((v2 NOR NOT(v4)) XOR v1)
5×5	8	4	((v4 NAND (v2 XOR v0)) XOR v1)
5×5	12	2	(IF(v1, v2, v4) XOR (v0 NAND NOT(v3)))
6×6	24	4	(v2 XOR ((v0 NAND v1) XOR IF(v4, v3, v0)))
7×7	56	2	–
8×8	82	20	–

S-box. Note that for sizes 7×7 and 8×8 we do not present the actual CA rules since they are complex and consist of large number of gates (e.g., for the best 7×7 solution there are 49 primitives). In Figure 2 we display the smallest rule for 5×5 S-box size that results in an optimal S-box.

In Table 3 we display statistics for tree sizes for every S-box dimension. Note that the values are averaged over all runs and not only over those that resulted in S-boxes with optimal values. The two most interesting sizes seem to be 4×4 and 8×8. In the former case it appears to be easy to obtain optimal values (note that in all runs we obtained optimal solutions as given in Table 1) and therefore GP relatively easily found even longer rules that result in optimal S-boxes. In the latter case, we see that even finding solutions that result in bijective S-boxes requires on average long trees. We consider this to be one of the reasons for relative lack of success for the 8×8 size, i.e., to obtain better solutions one would need much larger trees and consequently much longer evolution.

Finally, in Figure 3 we show the distribution of solutions for all S-box

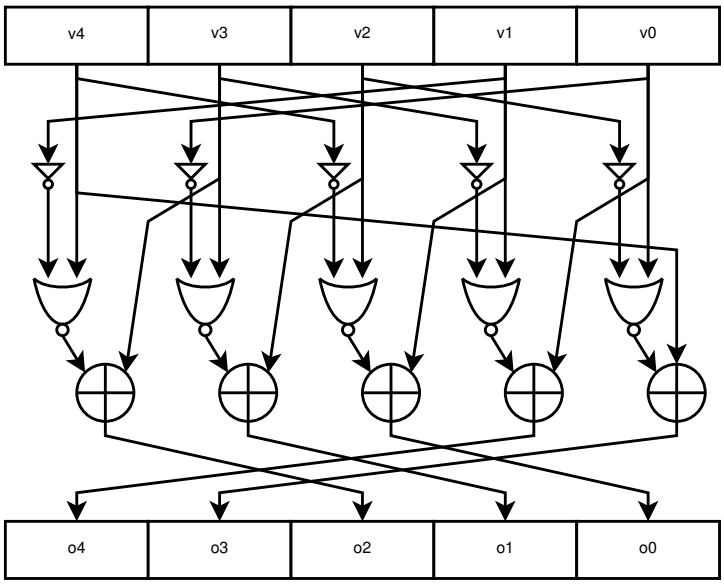

Figure 2: 5×5 rule – ((v2 NOR NOT(v4)) XOR v1).

Table 3: Tree sizes, $fitness_1$, Eq. (8)

S-box size	Min	Max	Avg	Std dev
4×4	**8**	103	48.77	25.03
5×5	**6**	67	26.27	11.93
6×6	**9**	82	35.13	18.71
7×7	**6**	64	30.63	14.77
8×8	**15**	119	68.7	28.93

sizes and fitness function 1 where we see that the larger S-box sizes also result in a wider distribution of fitness values.

On the basis of the results from Table 1 we see that the problem for sizes 4×4 and 5×5 can be considered easy to evolve. Moreover, for sizes 6×6 and 7×7 the results obtained here surpass any other results obtained with heuristics. However, the results for 8×8 are far from optimal where even the best obtained solution is worse than one can expect to obtain with the random search and permutation encoding [21]. We emphasize that the random search results for 8×8 size when evolving CA rules reach on average nonlinearity equal to 64. Therefore, for the 8×8 size we can assume it is significantly more difficult to work with CA rules encoding than with the permutation encoding.

In what follows, we discuss how to improve the behavior of GP for 8×8 size. As the first step, we consider only the balancedness and the

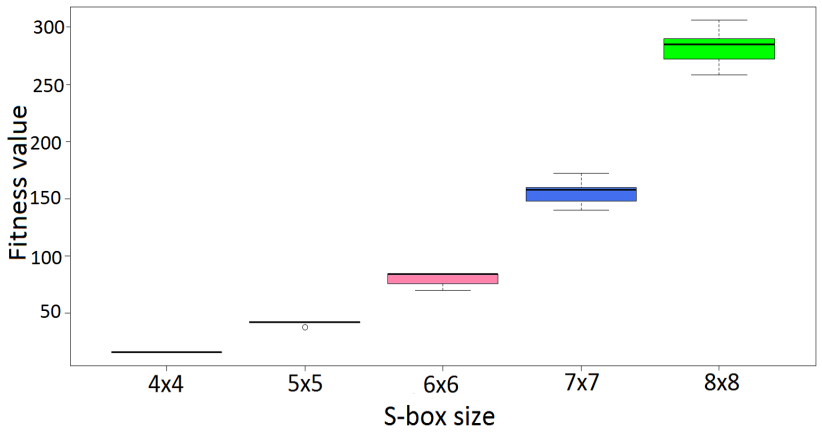

Figure 3: Fitness distributions, fitness function 1.

Table 4: Statistical results, $fitness_2$, Eq. (9)

S-box size	Max	Min	Avg	Std dev
8×8	**94.02**	56.125	69.85	8.55

nonlinearity property where we do not consider only the extreme value of the Walsh-Hadamard transform, but rather the whole Walsh-Hadamard spectrum. Therefore, we use fitness function as in Eq. (9) where the decimal part of the Max value stems from the fact that we add the number of occurrences of the maximal Walsh-Hadamard value to the nonlinearity value.

The results for $fitness_2$ are averaged over the best obtained values for each run. As can be seen in Table 4, we see that our new fitness function brings improvement where the best nonlinearity value significantly increases when compared with the best result from Table 2. Naturally, this value is still far from the optimal nonlinearity value and therefore represents only a relative improvement. Next, in Table 5 we give results for fitness defined in Eq. (10) where we also add into account differential uniformity. We observe that the nonlinearity and differential uniformity improve over results from Table 1, but only marginally ($N_F = 84, \delta_F = 18$). Interestingly, we see that now the best obtained nonlinearity is lower than with fitness function 2 where it equals 94. This points us to the conclusion that for larger S-box sizes using a multi-objective approach could be beneficial. When analyzing the tree size, with the fitness function 3 we obtain on average somewhat shorter trees with the average size of 60.37 primitives. However, the largest tree is now much longer and consists of 177 primitives.

Finally, in Table 6 we compare the best values obtained here with the state-of-the-art results obtained with EC where we give better values in bold style. For the 4×4 size both our technique and related work can

Table 5: Statistical results, $fitness_3$, Eq. (10)

S-box size	Theory max	Max	Min	Avg	Std dev
8×8	384	**322.98**256.88		288.99	15.74

Table 6: Comparison with the state-of-the-art from EC.

	Ours		Related	
S-box size	N_F	δ_F	N_F	δ_F
4×4	4	4	4	4
5×5	**12**	**2**	10	4
6×6	**24**	**4**	22	6
7×7	**56**	**2**	48	6
8×8	94	20	**104**	**8**

easily reach optimal values. However, for sizes 5×5 till 7×7 our approach yielded significantly improved results. On the other hand, for the 8×8 size, related work managed to obtain better nonlinearity and differential uniformity. Still, we emphasize that even those results are far from the best one where nonlinearity equals 112 and δ-uniformity equals 4 (note that it is only assumed but not proven that the maximal nonlinearity equals 112).

5.5 Reverse Engineering of CA-based S-boxes

Up to now we considered a problem where for a given S-box size we want to find a CA rule that maps to an S-box that is as good as possible with regards to the cryptographic properties. In this section, we change our objective and assume that we already have an S-box and we want to obtain its CA rule representation. There are two obvious reasons why one would want to do this. The first reason is to check whether some S-box is expressible with a CA rule. The second reason is to obtain a combinatorial circuit representation of an S-box (in the case that the S-box can be represented with a CA rule). The first objective can be reached with another technique that we briefly explain.

Given the truth table description of an S-box, the task of determining the local rule of the CA can be determined using the De Bruijn graph representation [27]. The *De Bruijn graph* associated to a CA $\langle c, n, \delta, \omega, f \rangle$ is a directed graph $G = (V, E)$ where $|V| = 2^{\delta-1}$. In particular, each vertex in G is labeled with a binary vector of length $\delta - 1$. An edge from vertex $a \in V$ to $b \in V$ exists if and only if a and b *overlap* respectively on the last and the first $\delta - 2$ coordinates. For example, for $\delta = 3$ the De Bruijn graph has an edge from $a = 01$ to $b = 10$ since a and b have a 1 respectively in the last and in the first position. A CA local rule is represented over the

De Bruijn graph as a labeling of the edges, i.e., a function $l : E \to \{0, 1\}$. Hence, in the example above the labeling of $(01, 10)$ would be the result of the local rule applied to the input 010. To check if a given S-box of length n can be expressed using a CA rule with diameter $\delta < n$ and offset ω, one could start from a De Bruijn graph with $2^{\delta-1}$ vertices and iteratively label the edges by reading the entries in the truth table of the S-box. As soon as an inconsistency is found (i.e., an edge gets more than one label), one knows that the S-box is not representable with a CA of diameter δ. On the other hand, if after reading the whole S-box each edge has a unique label, then the De Bruijn graph of a CA rule implementing that S-box is obtained.

As an example, we consider the S-box of size 6×6 with δ-uniformity equal to 2 [6]. Considering the last occurring value that equals 22, we see that this S-box cannot be generated with a CA rule. This is due to the fact that for the input 63 (111111 in binary) the output equals 22 (010110), which means that the local rule is not consistent because it assigns different values to the cells 1, 3, and 6 (value 0) and to the cells 2, 4, and 5 (value 1).

However, we note that the above procedure cannot help us to find a combinatorial representation of an S-box. Moreover, this problem is much more difficult since there exist many circuits mapping to the same truth table and there is no easy way to determine the smallest circuit. Therefore, we can use a regression process in order to find combinatorial circuits for a given S-box. In this process, we apply the same evolutionary algorithm (GP) with the same functions and terminals as before. However, the evolution is now guided with the fitness function that describes the difference between the S-box obtained by a CA rule, and the one given as an input parameter. The design of the objective function is such that the truth table output of the current CA rule is compared with the truth table of the given S-box.

Rather than simply counting all the bits in which the two differ, we employ a two-stage fitness as in the previous sections: in the first stage only a single output function (the first component) of an S-box is compared with the desired output. Only if all the bits in this component match the ones in the given S-box, the other part of the truth table is included in the comparison. Note that this is only an implementation and possibly a convergence issue, since in this way we hope to make it easier for the GP to find an equivalent rule.

Additionally, we add a term devoted to minimizing the size of the resulting CA rule (enforcing parsimony); this is only included if the error described above reaches zero. If that is the case, we add a term that is inversely proportional to the size of the GP individual, in this case corresponding to a CA rule. The final fitness function in this case equals:

$$fitness_4 = nErrors + \Delta_{nErrors,0} \left(\frac{treeSize}{maxTreeSize} \right), \tag{11}$$

where $nErrors$ denotes the number of differing bits, $treeSize$ is the actual

Table 7: Reverse engineering, $fitness_4$, Eq. (11)

S-box size	Original size	New size			
		Max	Min	Avg	Std dev
4×4	77	26	**11**	13.96	3.36
5×5	27	30	**9**	15.32	6.13
6×6	26	31	**13**	20.11	5.34
7×7	23	42	**13**	22.19	8.99

tree size, and $maxTreeSize$ is the maximum size that the tree may assume given a maximum tree depth and the number of arguments of the GP functions.

In our experiments, we use as input S-boxes previously evolved solutions with the best obtained properties. That way, we can be sure that the S-box can be represented with a CA rule, while trying to find an implementation with a smaller complexity. In Table 7 we give results for each S-box size. Column *Original size* gives the size of the S-box used in the regression process, and the other columns give statistics for the obtained results (here, column Min represents the best obtained solution). Note that we selected S-boxes randomly among those with the best obtained properties.

We see that for all presented sizes our procedure is able to find much shorter CA rules than used in the original case. Therefore, this makes our methodology a viable option when the goal is to implement the S-box obtained via a CA rule in hardware since a shorter rule will mean smaller gate count and consequently a smaller area. As an interesting fact, we note that we also tried this approach with the Keccak S-box. Among obtained solutions there were several occurrences of the exact same CA rule as used in Keccak. When working with the 8×8 S-box size, our regression technique was unable to find any correct rule corresponding to the given S-box. However, we note that we experimented with an S-box originally obtained with a CA rule consisting of 177 primitives, which is a much longer rule when compared with the sizes where our approach found correct rules.

6 Discussion and Future Work

On the basis of the presented results one can observe that for dimensions from 4×4 up to 7×7 it is possible to find CA rules that result in S-boxes with very good cryptographic properties. Here, only 6×6 size remains slightly suboptimal but we show in the previous section that the optimal solution (from the cryptographic perspective) is not even attainable with a single CA rule. Naturally, there is also a clear distinction in the difficulty of

this problem between size 4×4 and 7×7.

On the other hand, 8×8 size results in CA rules that are far from optimal and even far from the results obtainable with heuristics using permutation encoding. We see that the average sizes of 8×8 rules are significantly larger when compared with the other investigated dimensions, which certainly results in a more complicated evolution process. Besides the goal of evolving CA rules resulting in S-boxes with good cryptographic properties, we also changed the paradigm and looked for CA rules that map to a specific S-box. Again, results for sizes 4×4 up to 7×7 are very good, where up to the 7×7 size we have 100% success rate in obtaining correct rules. For 7×7 size, that percentage equals 96.7%, which is still an excellent result. As with the previous goal, size 8×8 presents an unsurmountable obstacle where our approach did not succeed in obtaining any equivalent CA rule.

As this work opens a new direction in the optimization of S-boxes with good cryptographic properties, there are many possible future research directions. The most obvious one is concentrating on the 8×8 size and trying to improve the values of cryptographic properties. Naturally, in this paper we concentrated only on a small set of cryptographic properties and one could include in the fitness function other relevant properties like algebraic degree or branch number. Another interesting direction would be to concentrate on the implementation perspective where one would try to minimize the size of CA rules resulting in S-boxes with optimal implementation properties like area or latency. This could be done in a naive way by just counting the number of primitives or even by accounting for the relative size of each terminal (e.g., IF function, which corresponds to the MUX gate requires more gates than the NOT function).

7 Conclusions

In this paper, we approach the problem of evolving S-boxes with good cryptographic properties from a completely new perspective. Instead of evolving S-boxes directly (regardless of the representation) we evolve cellular automata rules that are then mapped to S-boxes with good cryptographic properties. Our approach shows great potential where this is the first time evolutionary computation (or more generally, heuristic) techniques are able to find optimal S-boxes for sizes larger than 4×4. Moreover, our approach transforms a problem that has been up to now of extreme difficulty into a simpler problem for certain S-box sizes. Naturally, since not all S-boxes can be represented with a cellular automaton rule, our technique cannot be used to design all optimal S-boxes of the corresponding size. However, we believe that the corpus of obtainable functions is still large enough to give a sufficient diversity in the future block cipher designs. Furthermore, we note that the GP approach also offers easy handling of the resulting S-box latency

and area when implemented in hardware, which makes our methodology even more usable.

Finally, we show how one can use GP in order to "reverse-engineer" an S-box. There, we use the regression approach to find the shortest CA rule resulting in a specific S-box. This approach has interesting ramifications from two aspects: fast checking whether an S-box is expressible through CA rules and obtaining different rules (and consequently their sizes) resulting in a specific S-box.

8 Acknowledgments

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

References

[1] T. Bäck, D. Fogel, and Z. Michalewicz, editors. *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing, Bristol, 2000.

[2] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Radiogatún, a belt-and-mill hash function. *IACR Cryptology ePrint Archive*, 2006:369, 2006.

[3] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. The Keccak reference, January 2011. http://keccak.noekeon.org/.

[4] E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In *Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '90, pages 2–21, London, UK, UK, 1991. Springer-Verlag.

[5] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '07, pages 450–466, Berlin, Heidelberg, 2007. Springer-Verlag.

[6] K. A. Browning, J. F. Dillon, M. T. McQuistan, and A. J. Wolfe. An APN permutation in dimension six. *Finite Fields: theory and applications*, pages 33–42, 2010.

[7] L. Burnett, G. Carter, E. Dawson, and W. Millan. Efficient Methods for Generating MARS-Like S-Boxes. In *Proceedings of the 7th International Workshop on Fast Software Encryption*, FSE '00, pages 300–314, London, UK, UK, 2001. Springer-Verlag.

[8] 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.

[9] 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, USA, 1st edition, 2010.

[10] F. M. Christoph Dobraunig, Maria Eichlseder and M. Schläffer. Ascon, 2014. CAESAR submission, http://ascon.iaik.tugraz.at/.

[11] L. Claesen, J. Daemen, M. Genoe, and G. Peeters. Subterranean: A 600 Mbit/sec cryptographic VLSI chip. In *Computer Design: VLSI in Computers and Processors, 1993. ICCD '93. Proceedings., 1993 IEEE International Conference on*, pages 610–613, Oct 1993.

[12] J. A. Clark, J. L. Jacob, and S. Stepney. The design of S-boxes by simulated annealing. *New Generation Computing*, 23(3):219–231, Sept. 2005.

[13] J. Daemen and C. S. K. Clapp. Fast Hashing and Stream Encryption with PANAMA. In *Fast Software Encryption, 5th International Workshop, FSE '98, Paris, France, March 23-25, 1998, Proceedings*, pages 60–74, 1998.

[14] J. Daemen, R. Govaerts, and J. Vandewalle. Invertible shift-invariant transformations on binary arrays. *Applied Mathematics and Computation*, 62(2):259 – 277, 1994.

[15] J. Daemen, R. Govaerts, and J. Vandewalle. A new approach to block cipher design. In R. Anderson, editor, *Fast Software Encryption: Cambridge Security Workshop Cambridge, U. K.,1993 Proceedings*, pages 18–32, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[16] J. Daemen and V. Rijmen. *The Design of Rijndael.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[17] H. Gutowitz. Cryptography with dynamical systems. In *Cellular Automata and Cooperative Systems*, pages 237–274. Springer, 1993.

[18] L. R. Knudsen and M. Robshaw. *The Block Cipher Companion.* Information Security and Cryptography. Springer, 2011.

[19] K. Nyberg. Perfect Nonlinear S-Boxes. In *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of*

Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings, volume 547 of *Lecture Notes in Computer Science*, pages 378–386. Springer, 1991.

[20] K. Nyberg. On the construction of highly nonlinear permutations. In R. Rueppel, editor, *Advances in Cryptology - EUROCRYPT' 92*, volume 658 of *Lecture Notes in Computer Science*, pages 92–98. Springer Berlin Heidelberg, 1993.

[21] S. Picek, M. Cupic, and L. Rotim. A New Cost Function for Evolution of S-boxes. *Evolutionary Computation*, 2016.

[22] S. Picek, J. F. Miller, D. Jakobovic, and L. Batina. Cartesian Genetic Programming Approach for Generating Substitution Boxes of Different Sizes. In *GECCO Companion '15*, pages 1457–1458, New York, NY, USA, 2015. ACM.

[23] S. Picek, K. Papagiannopoulos, B. Ege, L. Batina, and D. Jakobovic. Confused by Confusion: Systematic Evaluation of DPA Resistance of Various S-boxes. In *Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India, December 14-17, 2014, Proceedings*, pages 374–390, 2014.

[24] 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).

[25] M. Seredynski and P. Bouvry. Block encryption using reversible cellular automata. In *Cellular Automata, 6th International Conference on Cellular Automata for Research and Industry, ACRI 2004, Amsterdam, The Netherlands, October 25-28, 2004, Proceedings*, pages 785–792, 2004.

[26] C. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):656–715, 1949.

[27] K. Sutner. De bruijn graphs and linear cellular automata. *Complex Systems*, 5(1):19–30, 1991.