A New Cost Function for Evolution of S-Boxes

Stjepan Picek stjepan@computer.org
KU Leuven, ESAT/COSIC and iMinds, Kasteelpark Arenberg 10, bus 2452,
B-3001 Leuven-Heverlee, Belgium and LAGA, UMR 7539, CNRS,
University of Paris 8, France

Marko Cupic marko.cupic@fer.hr
Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia

Leon Rotim leon.rotim@fer.hr
Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia

doi:10.1162/EVCO_a_00191

Abstract

Substitution Boxes (S-Boxes) play an important role in many modern-day cryptographic algorithms, more commonly known as ciphers. Without carefully chosen S-Boxes, such ciphers would be easier to break. Therefore, it is not surprising that the design of suitable S-Boxes attracts a lot of attention in the cryptography community. The evolutionary computation (EC) community also had several attempts using evolutionary paradigms to evolve S-Boxes with good cryptographic properties. This article focuses on a fitness function one should use when evolving highly nonlinear S-Boxes. After an extensive experimental analysis of the current state-of-the-art fitness functions, we present a new one that offers higher speed and better results when compared with the aforementioned fitness functions.

Keywords

Evolutionary algorithms, S-Boxes, cryptography, fitness function, solution representation.

1 Introduction

Today, modern communications would not be possible without cryptography. One can consider online shopping as an example; there, although the end users do not notice it (when all is going well), they use cryptography extensively. Credit card details are transferred through the Internet in an encrypted form, which means no one except the seller can read the data. Naturally, there are a multitude of ciphers that can be used. However, for transmitting protected data most often symmetric-key cryptography, and more precisely block ciphers, are used (Schneier, 1995).

All block ciphers have in common that they need some nonlinear operation/element to be secure. The choice of the nonlinear element depends on the design strategy one follows. A well-known and explored element of this kind is the Substitution Box (S-Box). When explaining S-Boxes, the easiest way is to consider them as a set of Boolean functions; hence S-Boxes are also known as the vectorial Boolean functions.

The input size n and output size m of an S-Box or (n, m)-function does not need to be the same. For instance, several S-Boxes are used in the DES cipher which has 8 S-Boxes of size 6×4 (FIPS, 1999). However, there are many ciphers that have the same input and output sizes and today two of the most popular sizes for S-Boxes are 4×4 and

Manuscript received: September 1, 2015; revised: April 30, 2016, May 13, 2016, and June 29, 2016; accepted: July 5, 2016.

8×8. The first S-Box size is usually used in lightweight cryptography that is primarily intended for the constrained environments; an example of such a lightweight cipher is the PRESENT cipher (Bogdanov et al., 2007). On the other side, 8×8 size S-Boxes are used when the security of a cipher is of primary importance; as an example we mention the AES cipher (Daemen and Rijmen, 2002).

As already said, ciphers that utilize S-Boxes could be easier to break if S-Boxes are not carefully chosen. Therefore, the design of S-Boxes with good cryptographic properties is a very active research area. However, it is impossible to design an S-Box that has all the cryptographic properties optimal and therefore a certain trade-off is necessary (Carlet, 2005).

In the process of the design of S-Boxes (similarly as in the design of Boolean functions), one can roughly follow three directions, namely, algebraic constructions, random search, and heuristics (Picek, Marchiori et al., 2014). Naturally, it is also possible to employ design strategies that represent various combinations of the aforementioned techniques. For the S-Boxes, algebraic constructions (for example, the finite field inversion method (Nyberg, 1991)) give unsurpassed results with regards to the most of the cryptographic properties (Carlet, 2010b).

The random search method has an advantage that it is relatively easy to produce a large number of S-Boxes with it. However, when discussing the quality of such solutions, they are far from those created by algebraic constructions.

The third direction utilizes various heuristics to find S-Boxes with good properties. This direction resulted in a large body of research over the years. If we consider the quality of solutions produced by such methods, they could fit somewhere between the random search and algebraic constructions. Since we stated that the quality of solutions obtained with heuristics is most often inferior to those obtained with algebraic constructions, the question one could ask is why such methods are of interest. Indeed, if we consider the finite field inversion method and 8×8 S-Box size, the properties of S-Boxes obtained in such a way are superior to any other known method (Carlet, 2010b). However, there are scenarios where new S-Boxes would be beneficial. The first example is when the primary goal is to evolve S-Boxes with properties having values that cannot be obtained with algebraic constructions (for example, when optimizing properties which are opposing). The second example is when the goal is to evolve S-Boxes that have values comparable to those of S-Boxes created with algebraic constructions (i.e., to create proprietary S-Boxes). The third scenario does not reflect so much the quality of properties (although that is still of high importance), but rather it deals with solutions that are desirable from the implementation perspective. One example is when the designer requires a solution which, when implemented in hardware, offers the best performance (for instance, with regards to the power, area, or latency).

Improving the quality of heuristics (and consequently the results obtained by it) represents a valuable goal since it would help raise the confidence in such methods. In accordance with that, devising heuristics that could reach the solutions obtained with algebraic constructions would have a significant impact for both cryptography and EC community. Therefore, *the central question* of this research is how to improve the quality of results obtained by heuristics. One quite common way is to explore different heuristics in a hope to find the most appropriate one. However, such a research avenue cannot help to strengthen the foundations of one's knowledge nor would it allow more general answers about the problem difficulty. Before going further into details, we briefly discuss the number of possible S-Boxes for different input/output sizes to see how difficult this problem really is. The number of (n, m) functions equals $(2^m)^{2^n}$ and

Table 1: Search space size for (n, n) functions.

n	4	5	6	7	8
#	2^{64}	2^{160}	2^{384}	2^{896}	$2^{2\,048}$

for an $n = m$ case, we show several search space sizes in Table 1. Seeing the exponential growth of the search space with the number of variables, it is obvious that the exhaustive search does not present a viable option for sizes larger than four.

In this article, we focus on the questions of representation and fitness function instead of the choice of heuristics. Indeed, our research indicates that the varying success of different evolutionary methods stems from the choice of fitness function and not so much from the choice of the algorithm itself. To this end, we work with several evolutionary algorithms (EAs) where we investigate the choices behind the representation perspective and fitness functions. We consider the choice of the fitness function as of the utmost importance and therefore we examine it thoroughly in order to find fitness functions that have sound bases and good performance regardless of the size of S-Boxes.

The remainder of this article is organized as follows. In Section 2, we present the necessary background information on S-Boxes and their properties, as well as some bounds on the values that are possible to obtain. Next, in Section 3, we give a survey of related work. In Section 4, we discuss the motivation behind this research as well as our contributions. Section 5 gives details about the algorithms we use and details about the fitness functions from related work. In Section 6, we give the results for five different sizes of bijective S-Boxes when using state-of-the-art fitness functions. Furthermore, we present the details about our new cost function that offers greater speed and higher nonlinearity values and we support that with detailed experimental results and a short discussion. Finally, in Section 7, we give a short conclusion and offer several potential research directions.

2 Preliminaries

Let n, m be positive integers, that is, $n, m \in \mathbb{N}^+$. The set of all n-tuples of the 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 $\vec{a}$ and $\vec{b}$ is denoted as $\vec{a} \cdot \vec{b}$ and equals $\vec{a} \cdot \vec{b} = \oplus_{i=1}^{n} a_i b_i$. Here, "$\oplus$" represents addition modulo two (bitwise XOR). The Hamming weight (HW) of a vector $\vec{a}$, where $\vec{a} \in \mathbb{F}_2^n$, is the number of non-zero positions in the vector.

An (n, m)-function is any mapping F from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. If m equals 1, then the function f is called a Boolean function, and when $m > 1$ the function F is called an S-Box or a vectorial Boolean function. An (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 coordinate functions with non-all-zero coefficients.

2.1 Representations

In this section, we present several unique S-Box representations that are of relevance for this work. We note that these representations are not the only unique ones. An (n, m)-function F can be represented as a list of values (lookup table, LUT), with each value ranging from 0 to $2^m - 1$. Alternatively, an (n, m)-function can be implemented

as a lookup table with 2^n words of m bits each. When $n = m$ it is usual that the S-Box is bijective, that is, that each value in the output appears exactly once.

A Boolean function f on $\mathbb{F}_2^n$ is represented by a truth table (TT), which is a vector $(f(\vec{0}), \ldots, f(\vec{1}))$ that contains the function values of f, ordered lexicographically, i.e., $\vec{a} \leq \vec{b}$, where $\vec{a}$ and $\vec{b}$ are two input entries for the truth table (Carlet, 2010a). An S-Box can be represented in the truth table form as a matrix of dimension $2^n \times m$ where each column m represents one Boolean function (i.e., one coordinate function).

The Walsh–Hadamard transform of an (n, m)-function F equals (Carlet, 2010b):

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

where $\vec{a} \in \mathbb{F}_2^n$ and $\vec{v} \in \mathbb{F}_2^{m*}$.

2.2 Properties

Since we are interested in evolving S-Boxes with good cryptographic properties, it is necessary first to discuss which S-Boxes are suitable in practice. When considering 4×4 S-Boxes, there exist in total 16! bijective S-Boxes, which is approximately 2^{44} options to search from. Leander and Poschmann (2007) define 4-bit *optimal* S-Boxes as those that are bijective (balanced), have linearity equal to 8, and δ-uniformity equal to 4. The linearity property that equals 8 is the same as the nonlinearity property that equals 4 and we continue using the nonlinearity property value as an indicator. The optimal S-Boxes possess the best possible values of the aforementioned properties. For the 4×4 S-Box size, we concentrate only on optimal S-Boxes as these are of practical interest. Indeed, as far as the authors know, all ciphers that have 4×4 S-Boxes use optimal S-Boxes (Bogdanov et al., 2007; Borghoff et al., 2012; Daemen et al., 2000). For all other S-Box sizes that we consider in this work, we restrict our attention to the same properties as for the 4×4 size.

An (n, m)-function F is called *balanced* if it takes every value of $\mathbb{F}_2^m$ the same number 2^{n-m} of times. Balanced (n, n)-functions are permutations on $\mathbb{F}_2^n$ (Carlet, 2010b).

The *nonlinearity* N_F of an (n, m)-function F equals the minimum nonlinearity of all nonzero linear combinations $\vec{b} \cdot F$ of its coordinate functions f_i, where $\vec{b} \in \mathbb{F}_2^{m*}$ (Carlet, 2010b; Nyberg, 1993):

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

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:

$$D(a, b) = |\left\{x \in \mathbb{F}_2^n : F(x + a) + F(x) = b\right\}|. \tag{3}$$

The entry at the position (a, b) corresponds to the cardinality of $D(a, b)$ and is denoted as $\delta(a, b)$. The δ-uniformity δ_F is then defined as (Biham and Shamir, 1991; Nyberg, 1991):

$$\delta_F = \max_{a \neq 0, b} \delta(a, b). \tag{4}$$

Here, we give a small S-Box example in Tables 2 and 3. First, in Table 2, we give the truth table and lookup table representations. In Table 3, we present all linear nonzero combinations (component functions) and the Walsh–Hadamard spectrum. By simply setting the Walsh–Hadamard values from Table 3 to Equation (2), we see that the nonlinearity of this S-Box equals 0.

Table 2: 2×2 S-Box example, part 1.

Truth table input		Truth table output		LUT input	LUT output
x_1	x_0	y_1	y_0	x	y
0	0	0	1	0	1
0	1	1	0	1	2
1	0	1	1	2	3
1	1	0	0	3	0

Table 3: 2×2 S-Box example, part 2.

Truth table input		Linear combinations			Walsh–Hadamard spectrum		
x_1	x_0	y_1	y_0	$y_1 \oplus y_0$	y_1	y_0	$y_1 \oplus y_0$
0	0	0	1	1	0	0	0
0	1	1	0	1	0	−4	0
1	0	1	1	0	0	0	−4
1	1	0	0	0	4	0	0

2.3 Bounds

The maximal nonlinearity of any (n, n) S-Box is upper bounded by the following inequality:

$$N_F \leq 2^{n-1} - 2^{\frac{n-1}{2}}. \tag{5}$$

In the case of equality, we talk about Almost Bent (AB) functions, where examples are power functions like Gold and Kasami. Note that AB functions exist only for odd dimensions (Carlet, 2010b). When n is even, the best known nonlinearity is obtained for the inverse function and it equals (Budaghyan et al., 2006):

$$N_F = 2^{n-1} - 2^{\frac{n}{2}}. \tag{6}$$

For the δ-uniformity property the best possible value is 2 (the smaller the better), which is a value obtained for Almost Perfect Nonlinear (APN) functions. When talking about bijective functions, this bound is achieved for any odd n and also for $n = 6$. For n even and larger than 6, it is an open question whether such functions exist. The best known δ-uniformity value for the 8×8 bijective S-Box equals 4 and is obtained with the inverse function (Carlet, 2010b). Note that every AB function is also APN function, but the converse is not true. For further information about S-Boxes, their properties, and role in cryptography, we refer interested readers to Carlet (2010b).

3 Related Work

There are several successful applications of Monte Carlo algorithms when creating S-Boxes and we mention here only a representative subset. Clark et al. (2005) use the principles from the evolutionary design of Boolean functions to evolve S-Boxes with

desired cryptographic properties. They use simulated annealing (SA) coupled with the hill climbing algorithm to evolve bijective S-Boxes of sizes up to 8×8 that have high nonlinearity values.

Millan et al. (1999) work with genetic algorithms (GAs) to evolve S-Boxes with high nonlinearity and low autocorrelation. Furthermore, the authors discuss the selection of the appropriate genetic algorithm (GA) parameters.

On the other hand, Burnett et al. (2001) use a heuristic method to generate MARS-like S-Boxes. They are able to generate a number of S-Boxes of appropriate size that satisfy all the requirements placed on the MARS S-Box (Burwick et al., 1999). With a combination of several techniques, they even manage to find S-Boxes with improved nonlinearity values.

Fuller et al. (2004) use a heuristic method to evolve bijective S-Boxes from the power mappings. They use only iterated mutation operators and report to generate S-Boxes with the best known trade-offs among the criteria they consider.

Burnett (2005) in her thesis experiments with a GA, hill climbing or with a combination of those two methods in order to evolve S-Boxes where $M \neq N$.

Tesař (2010) uses a combination of a special GA with a total tree searching to generate 8×8 S-Boxes with nonlinearity equal up to 104.

Kazymyrov et al. (2013) use an improved gradient descent method to find 8×8 S-Boxes that have high nonlinearity values.

Ivanov et al. (2016) experiment with a GA that works in a reverse way in order to generate bijective S-Boxes of dimensions from 8×8 to 16×16. They seed the initial S-Boxes population with solutions based on the finite field inversion method and then evolve them to find new solutions. The same authors use a modified immune algorithm to generate 8×8 S-Boxes that are balanced with high nonlinearity and low δ-uniformity (Ivanov et al., 2015).

Picek, Ege, Batina et al. (2014) explore how to generate S-Boxes of size 8×8 with a better resistance against side-channel attacks (SCA) as measured with the transparency order property. Next, Picek, Ege, Papagiannopoulos et al. (2014) and Picek, Papagiannopoulos, Ege et al. (2014) investigate side-channel resilience of 4×4 S-Boxes as well as when considering the confusion coefficient property. Ege et al. (2015) use genetic algorithms to evolve S-Boxes with better SCA resilience and they implement such S-Boxes in both software and hardware settings in order to properly evaluate them. Finally, Picek, Mazumdar et al. (2015) experiment with the modified transparency order property in order to achieve S-Boxes with better SCA resistivity.

Picek, Miller et al. (2015) use Cartesian genetic programming (CGP) and genetic programming (GP) to evolve S-Boxes where they discuss how to obtain permutation-based encoding with those algorithms.

4 Motivation and Contributions

As already mentioned, the quality of solutions achieved with heuristics lies somewhere between the random search and algebraic constructions. To clarify this, we consider here only the two (since we assume S-Boxes are always bijective) most important properties (nonlinearity and δ-uniformity) and observe the following. When using random search for generating S-Boxes of size 8×8, the nonlinearity goes approximately up to 98 (the higher the better) and the δ-uniformity is in the range of 10 to 18 (the smaller the better). More generally, S-Boxes created with random search usually have good nonlinearity values, but poor δ-uniformity values (Dib, 2014). However, when using heuristics, nonlinearity is usually in range between 98 and 104. The δ-uniformity property is

somewhat less considered, but again we can observe a slight improvement with values around 10 when compared with the random search. Lastly, when using the finite field inversion construction, the nonlinearity equals 112 and the δ-uniformity is equal to 4 for 8×8 size.

Next, we discuss various options regarding the representations of Boolean functions and the influence of a fitness function and then we enumerate our contributions.

4.1 On the Representation Options

We start from the truth table representation that consists of 2^n rows and 2^m columns. The rows represent all possible combinations of the input bits while the columns represent all possible combinations of the output bits. We first investigate a single Boolean function scenario (i.e., when $m = 1$) and then extend it to m Boolean functions.

For a single Boolean function, the most natural way of representation is the truth table form. There, to obtain a balanced solution, it is enough to set exactly half of the zeros and half of the ones in the output vector. However, when talking about an S-Box, it is not enough that each column (a Boolean function) is balanced, but also all linear combinations need to be balanced. If we work with the bitstring representation, this immediately starts to present a problem. For instance, if we use a GA, then the resulting chromosome has m dimensions, or when using GP, m independent trees. The situation is slightly less complicated with CGP where we can have a graph with m outputs.

For smaller dimensions (up to 4), our experiments show it is possible to obtain balanced solutions with the truth table representation. However, even that represents a moderately difficult problem. Furthermore, if we add just one more property, for example the nonlinearity, then we again need to check not only the coordinate functions (columns), but also all linear combinations of columns. Therefore, although a valid representation, it presents difficulties that are hard for EC to circumvent.

Next, we concentrate on the Walsh–Hadamard representation and a single Boolean function scenario. In this representation, each coefficient is a number in $\mathbb{Z}$. However, there is no straightforward way to set those values. Only if we want to create a bent (where bent functions exist only for n even) Boolean function, we know that the Walsh–Hadamard spectrum is flat, that is, $W_F(\vec{v}) = 2^{\frac{n}{2}}$. However, bent functions are not balanced and therefore not appropriate for most usages in cryptography (Carlet, 2010a). Besides that information, there is also the Parseval theorem:

$$\sum_{\vec{v} \in \mathbb{F}_2^n} W_f(\vec{v})^2 = 2^{2n}. \tag{7}$$

Unfortunately, this theorem only gives a necessary condition on a Boolean function and not a sufficient one. Therefore, to evolve a Boolean function represented with the Walsh–Hadamard spectrum, it would be necessary to constantly check whether the solution is indeed a Boolean function and to implement an operator that would fix those solutions that are not valid (alternatively, such solutions would need to be discarded). Going to the S-Box case would be even harder since there are a number of Boolean functions we need to check.

Finally, we are left with the lookup table representation where in the case that $n = m$, we can represent an S-Box as a permutation (if the resulting S-Box is bijective, which is the usual design choice). In such a case, an S-Box is always balanced and we do not need to verify that property. Therefore, it is easy to conclude that the permutation encoding represents the most natural one for S-Boxes and is consequently the encoding we use in the rest of this article. As already stated, there are other unique representations of

Boolean functions (and consequently S-Boxes), but they provide even less information in the evolution process, so we do not consider them here.

4.2 On the Fitness Functions

Since we decide to work with the permutation encoding, we do not need to consider the balancedness property because it is satisfied automatically. Therefore, we consider now only the nonlinearity and the δ-uniformity properties. We start with the analysis of the nonlinearity property first. By checking related work we can observe that a significant part of it uses only the nonlinearity value itself (i.e., the extreme value of the Walsh–Hadamard spectrum) as a part of the fitness function. Therefore, the fitness function is of the form:

$$fitness = N_F, \tag{8}$$

where the goal is maximization. Often the fitness function has other parts in accordance with the properties one wants to obtain. In common for all works where the fitness function is of the previous form is that the final nonlinearity value usually does not reach over 100 for the 8×8 size. A possible explanation is that such a fitness function actually loses a significant part of the information since it works with only one, extreme value and not with all the values in the Walsh–Hadamard spectrum. On the basis of Equation (2), it is evident that by lowering the values of the whole Walsh–Hadamard spectrum (i.e., making the spectrum more flat), the final nonlinearity value will be higher. This also stems from the fact that the maximal nonlinearity is obtained for bent functions where the Walsh–Hadamard spectrum is flat and equals $W_f(\vec{a}) = 2^{n/2}$. In accordance with that, Clark et al. (2004) introduced a cost function that considers the whole Walsh–Hadamard spectrum for a Boolean function and equals:

$$cost(f) = \sum_{\vec{a}} ||W_f(\vec{a})| - X|^R, \tag{9}$$

where X and R are real-valued parameters. One can immediately observe a significant downside of this cost function: it uses additional parameters where the only way to adjust them is to conduct the parameter tuning phase. Furthermore, such a procedure needs to be done for each Boolean function size where the authors experiment with different parameter values for n up to 9. To elaborate more on the downside of such fitness function, we cite Clark et al. (2004, page 2): "The parameters X and R provide freedom to experiment. It is difficult to predict what the best parameter values should be: it is far from clear what is the effect of imposing a balance requirement, and what is the effect of an odd n."

The fitness from Equation (9) can be naturally extended to a multioutput case where it equals (Clark et al., 2005):

$$cost(F) = ||W_F(\vec{a}, \vec{v})| - X|^R. \tag{10}$$

Here, R was set to 3.0 and X was in set $[4, 3, 2, 1, 0, 1, 2, 3, 4]$. The best nonlinearity obtained with this fitness function and 8×8 S-Box size equals 102. The authors also try to give at least a notion of the reasoning behind the choice of parameter values, but cannot give definitive guidelines. Note that the multioutput case cost function suffers from the same downside as for the single output case, namely, the need for tuning X and R parameters.

Tesař (2010) conducted an extensive parameter tuning where both parameters are integers and R goes in range $[4, 24]$ while X goes in range $[-24, 24]$. The author found that the best set of parameters has values $R = 7$ and $X = 21$, but offers no reasoning

why this set of values would perform the best. Furthermore, with his algorithm that consists of a combination of the genetic algorithm and hill climbing the author states to regularly find S-Boxes of nonlinearity 104 for the 8×8 size.

Besides the aforesaid results, there exist a small number of research papers where authors use heuristics and obtain nonlinearities higher than 104. However, we note that the comparison between them is not entirely fair since the authors do not start from a random initial population, but from a population (or a single individual) that is highly competitive. Ivanov et al. (2016) start from the population of AES affine equivalent S-Boxes and then use the "reverse genetic algorithm" to evolve solutions up to a certain level (i.e, a nonlinearity value). The best S-Box the authors report has the nonlinearity of 112, which is the same as in the case of the AES S-Box. Next, Fuller and Millan (2003) explore the linear redundancy in S-Boxes and experiment with the AES S-Box where they conduct swaps of 8 value pairs. In such a way, the best obtained S-Box has the nonlinearity of 106. Finally, Kazymyrov et al. (2013) use a gradient descent method where they start with an S-Box that has good δ-uniformity value and they conduct a number of steps until they find an S-Box with good nonlinearity, which in their work reaches 104 for the 8×8 size. To conclude, if disregarding the last three mentioned papers, since they use extra knowledge in the form of special initial population, the best nonlinearity equals 104 where it is obtained by a combination or customization of algorithms. However, a single algorithm that uses Equation (9) goes up to only 102 where the best choice of parameters is unclear (considering only a single S-Box size, let alone a number of sizes).

The situation with the δ-uniformity is much simpler since many of the related works do not consider that property. Usually when this property is of interest then it is evaluated only a posteriori. The exceptions are works by Picek, Papagiannopoulos et al. (2015) and Picek, Mazumdar, Ege et al. (2014) where the authors consider the δ-uniformity value in the fitness function. However, this is somewhat simplistic in a sense that only the smallest (extreme) value is considered. On the other hand, Ivanov et al. (2015) evolve bijective S-Boxes where they use a more complex expression in the fitness function to calculate the δ-uniformity property.

4.3 Our Contributions

The main contribution of this article is the definition of a *new fitness function* that is able to produce highly competitive results with greater speed. Furthermore, the nonlinearity property easily reaches the value of 104 without using combinations of algorithms as used in related work. Finally, our new fitness function does not use parameters that are necessary to tune or difficult to justify theoretically. Rather, it uses only a single parameter that enables the researcher to select how fine-grained a procedure is wanted.

Besides that contribution, we give a number of smaller ones in forms of the analysis of the existing fitness functions as well as the extensive analysis of the performance of several evolutionary algorithms and fitness functions on five sizes of bijective S-Boxes.

5 Experimental Setup

In our experiments, we use three different algorithms, namely, Genetic Algorithm (GA) (Eiben and Smith, 2003), Genetic and Tree Algorithm (GaT) (Tesař, 2010), and our Local Search Algorithm (LSA) with two different cost functions. The experiments are first executed using the Clark's cost function (Clark, 1998) with two sets of parameters and again with our newly developed cost function that finds cryptographically

Table 4: Common parameters. NoE—maximal number of solution evaluations, NL—target nonlinearity, NLH—nonlinearity highest known value.

S-Box size	NoE	NL	NLH
4×4	3 000 000	4	4
5×5	5 000 000	10	12
6×6	5 000 000	22	24
7×7	8 000 000	48	56
8×8	9 000 000	104	112

strong S-Boxes with greater speed. The details about our new cost function are presented in Section 6.

A number of experiments are performed using the Evolution Strategy (ES) algorithm, the Artificial Immune System (AIS) algorithm, and the Particle Swarm Optimization (PSO) algorithm. However, the obtained results show that they do not outperform those reached with the GA. In all our experiments, we use a solution representation as explained in Sections 2.1 and 4.1. All experiments are conducted on bijective S-Boxes of dimensions 4×4, 5×5, 6×6, 7×7, and 8×8.

5.1 Common Parameters

The number of independent runs for each experiment is 50. The stopping conditions for every algorithm we use are the number of evaluations or the target nonlinearity value as given in Table 4. The values presented in the table are determined after performing a small set of tuning experiments. For an S-Box of dimensions 4×4, the highest known nonlinearity value was found, but for larger sizes the experiments showed us that, by using any of the presented algorithms, S-Boxes of higher nonlinearity values cannot be found in a reasonable amount of time.

5.2 Genetic Algorithm

The GA starts with an initial population created by randomly setting each value from 0 to $2^n - 1$ as outputs of a lookup table, where n is the dimension of an S-Box ($n \times n$).

After the initial population is generated, the genetic algorithm starts with the evolution process. In each iteration it randomly chooses 3 possible solutions (the tournament of size k equal to 3) and selects the worst solution among those (this selection method also ensures elitism, i.e., the best solutions are always propagated to the next iteration). The remaining two solutions are used as parents, which create one offspring via variation operators and the offspring is then evaluated. If it is better than the worst solution in the tournament and its genome is different from all other solutions in the population, then it replaces the worst solution; otherwise, the process is repeated.

For variation operators, we use three mutation operators and three crossover operators where we chose among the most common ones in use today. The mutation operators used are insert mutation (Eiben and Smith, 2003), inversion mutation (Eiben and Smith, 2003), and swap mutation (Ryan et al., 2004). As for the crossover, we used partially mapped crossover (PMX) (Goldberg and Lingle, 1985), position-based crossover (PBX) (Syswerda, 1985), and order crossover (OX) (Davis, 1985). For each offspring, an operator is selected uniformly at random between all operators within a class (mutation and crossover).

Table 5: Parameters for GaT algorithm: NT—Nonlinearity value to enter the tree part of the algorithm.

S-Box size	4×4	5×5	6×6	7×7	8×8
NT	2	8	20	46	102

The GA parameters are the following ones: the size of the population is 100. The mutation probability per individual is set to 0.13 for the insert and inversion mutation and 0.25 for the swap mutation. These parameters were chosen on a basis of a small set of tuning experiments in which they showed the best results on average.

5.3 Genetic and Tree Algorithm

As mentioned before, this algorithm consists of two parts: a special case of the GA and the total tree search. This algorithm's parameters are set in accordance with the previous results (Tesař, 2010). We emphasize the two most important parameters of the algorithm: the criterion to enter the tree part of the algorithm and the stopping criterion. Here, the criterion to enter the tree part is the S-Box nonlinearity value as given in Table 5.

The first part of this algorithm is a population algorithm where its initial population is generated in the same way as it is done for the GA. The algorithm runs the GA part until it finds the solution that satisfies the criterion to enter the tree part. With that solution the total tree search algorithm starts and runs until any of the stopping criteria are met. For more details on this algorithm, we refer interested readers to Tesař (2010).

5.4 Local Search Algorithm

Here, we illustrate how the local search algorithm works where we use it for a comparison with the other evolutionary algorithms we investigate in this article. The algorithm starts with a single randomly generated solution which is set as algorithm's current solution. In each iteration, the algorithm produces N new solutions generated with given mutation operators. The mutation operator randomly chooses k different positions in the solution representation and then randomly permutes elements on chosen positions. To achieve the best results, we provide mutation operators with $k \in \{2, 3, 4, 5, 6, 7\}$. By using a small set of tuning experiments, we determine that larger values of k are destructive, that is, they are more likely to harm than to improve the solution they are applied on. Due to that reason, we opted to use mutation operators with small values of k.

As an input to the LS algorithm, a set of mutation operations is provided. Each mutation operator is defined with two parameters, k and l, where k is a number of positions whose elements are to be permuted and l defines how many times that mutation operator will be applied on the current solution in each iteration. The mutation operators used in our experiments are presented in Table 6. Here, the total number of solutions generated in each iteration is 97, where that value is easily obtained by summation of all values in a single row l of Table 6. In each iteration, N new solutions are generated by applying given mutation operators. From N produced solutions, the best one is selected and set as the current solution.

Table 6: Mutation operators used in LS algorithm.

k	2	3	4	5	6	7
l	50	25	12	6	3	1

5.5 Cost Functions Parameters

We use two different cost functions in our experiments: the Clark's cost function as given in Equation (9) and our newly developed cost function as given in Equation (12). We investigate the Clark's fitness function with parameters $R = 3$ and $X = 4$ on bijective S-Boxes of dimensions $n = \{4, 5, 6, 7, 8\}$ (Clark et al., 2005). Following that, we set parameters X and R to 21 and 7, respectively. Those values are presented as the best performing in the experiments for S-Boxes of dimension 8×8 (Tesař, 2010).

Contrasting that, the cost function we develop uses only a single parameter which defines how many values (N) of the Walsh–Hadamard spectrum will be used in the solution evaluation. The results presented are obtained with the parameter values $N = 1$ and $N = 10$. With $N = 1$ the cost function reduces to a base case, which is the initial form of our cost function. Later, we generalize it to use an arbitrary number of N values. The value $N = 10$ is chosen based on a small set of tuning experiments in which it showed the best results on average for all S-Box sizes we investigated.

In all algorithms used, the fitness function is represented with a two-component vector (N_F, c), where N_F represents the solutions' nonlinearity and c is a value obtained by the Clark's or our cost function. The Clark's cost function is presented in Section 4.2 and ours is described in detail in Section 6.2. A solution S_i with a fitness (N_{F_i}, c_i) is better than the solution S_j with a fitness (N_{F_j}, c_j) if $N_{F_i} > N_{F_j}$, or $N_{F_i} = N_{F_j}$ but $c_i < c_j$ (which, in effect, represents a hierarchical vector comparison).

6 Results and Discussion

In this section, we present the results obtained in our experiments. First, we give the results acquired when using the Clark's cost function and continue with the development of our cost function and consequently the results obtained with it. Note that if not stated otherwise, we evaluate the δ-uniformity property a posteriori.

6.1 Results with the Clark's Cost Function

Recall from Section 4.2 that the Clark's cost function is calculated on the basis of values in the Walsh–Hadamard spectrum and its value depends on two parameters, namely, X and R. Clark et al. (2005) run the experiments with the parameter values $X = \{-4, -3, -2, -1, 0, 1, 2, 3, 4\}$ and $R = 3$. However, values $X = 21$ and $R = 7$ were determined as the best choice for an S-Box of 8×8 size after performing the exhaustive grid search (Tesař, 2010). Furthermore, there is no intuition provided why those exact values would give the best results. For S-Boxes of smaller dimensions no such exhaustive search was performed and based on the values of their Walsh–Hadamard spectra and the Clark's cost function definition one can expect those parameters not to work well. Therefore, we perform two sets of experiments with the Clark's cost function: one with the parameters $X = 4$ and $R = 3$ and the other with $X = 21$ and $R = 7$.

The results obtained using the Clark's cost function and different X and R parameter values are presented in Tables 7 and 8 in which we give the best and the average results obtained by using all three algorithms.

Table 7: Results obtained using the Clark's cost function, $X = 4$ and $R = 3$. Presented values are best and average solutions found in format nonlinearity/δ-uniformity.

	GA		GaT		LSA	
S-Box size/Algorithm	Best	Avg	Best	Avg	Best	Avg
4×4	4/4	4/4.72	4/4	4/5.36	4/4	4/5
5×5	10/4	9.56/5.78	10/4	10/6.04	10/4	10/5.6
6×6	22/6	21.76/6.56	22/6	22/7.72	22/6	22/6.5
7×7	48/6	48/8	48/6	48/7.3	48/6	48/7.6
8×8	102/8	102/8.56	102/8	102/9.50	102/8	102/8.25

Table 8: Results obtained using the Clark's cost function, $X = 21$ and $R = 7$. Presented values are best and average solutions found in format nonlinearity/δ-uniformity.

	GA		GaT		LSA	
S-Box size/Algorithm	Best	Avg	Best	Avg	Best	Avg
4×4	4/4	4/5	4/4	4/5.14	4/4	4/4.72
5×5	10/4	10/5.84	10/4	10/6.04	10/4	10/5.60
6×6	22/6	21.56/6.56	22/6	21.72/6.76	20/6	20/7.40
7×7	46/8	46/8.56	46/8	46/9.24	46/8	46/8.44
8×8	104/8	102.04/10.08	104/8	103.32/9.52	102/8	102/9.57

Table 9: Average number of evaluations needed with the Clark's cost function, $X = 4$ and $R = 3$, S-Box size 8×8.

Algorithm/Nonlinearity	98	100	102	104
GA	457	9194	119229	NPR
GaT	677	3693	90059	NPR
LSA	623	2814	69420	NPR

Next, in Tables 9 and 10, we give the average number of solution evaluations before a certain nonlinearity value was found while optimizing S-Boxes of dimension 8×8. In the case that an algorithm is not able to obtain a certain solution repeatedly, we note that with the acronym NPR (Not able to Produce Repeatedly). From the presented results, we can conclude that the performance of the Clark's cost function is highly dependent on the parameters X and R and without exhaustive search it is difficult to say which ones would be the optimal for an arbitrary S-Box size.

6.2 Results with the New Cost Function

In this section, we present the results obtained with our cost function. The motivation for our function comes from the promising results of the Clark's cost function which is based on the Walsh–Hadamard spectrum. More precisely, from Equation (2) it can be

Table 10: Average number of evaluations needed with the Clark's cost function, $X = 21$ and $R = 7$, S-Box size 8×8.

Algorithm/Nonlinearity	98	100	102	104
GA	416	6492	28200	NPR
GaT	171	1055	9024	3 849 881
LSA	371	1095	6701	NPR

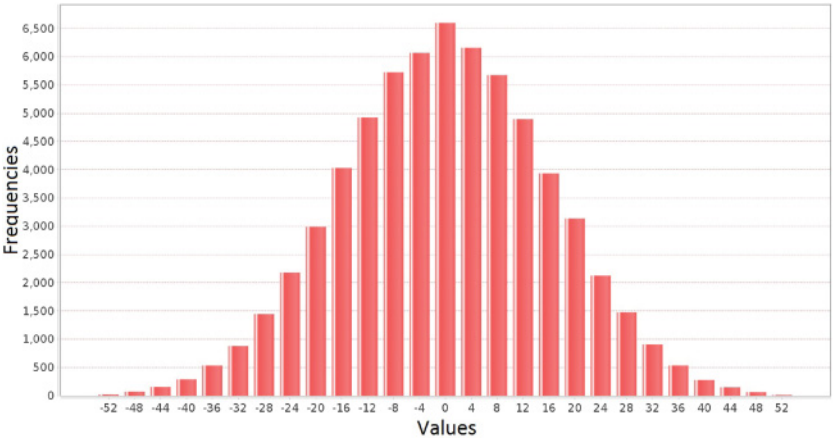

Figure 1: The Walsh–Hadamard spectrum of an S-Box S_1, size 8×8, $N_F = 102$.

seen that the nonlinearity is directly defined with the maximum absolute value of the Walsh–Hadamard coefficient in the Walsh–Hadamard spectrum. Note that there can be a variety of different coefficient values with different multiplicities in the spectrum, in accordance with Equation (1).

With that in mind, the definition of our cost function stems quite naturally. We want to *decrease* the number of coefficients that have the maximum absolute value. Once the number of the maximum absolute value coefficients is reduced to zero, the nonlinearity of an S-Box is increased. As stated in Section 5.5, S-Boxes are first compared on their nonlinearity value (the larger the better) and if the nonlinearity value is equal, then we see that the better S-Box is the one with a smaller value of the cost function. The value of the initial version of our cost function $CF_i(S)$ when applied on an S-Box S is evaluated as a number of the Walsh–Hadamard coefficients of the maximum absolute value (denoted here as $H(S)_l$):

$$CF_i(S) = H(S)_l. \tag{11}$$

In Figure 1, we display the Walsh–Hadamard spectrum of an 8×8 S-Box with the nonlinearity value of 102. Its maximum absolute coefficient value equals 52. Only when the number of coefficients with the value ± 52 is reduced to 0, the nonlinearity of that S-Box would increase to 104, with coefficients ± 48 remaining the new largest ones. The Walsh–Hadamard spectrum of an 8×8 S-Box and the nonlinearity 104 is given in Figure 3, where we can see that there are no coefficients with the absolute value 52. To illustrate it better, consider a case where there are two 8×8 S-Boxes with the

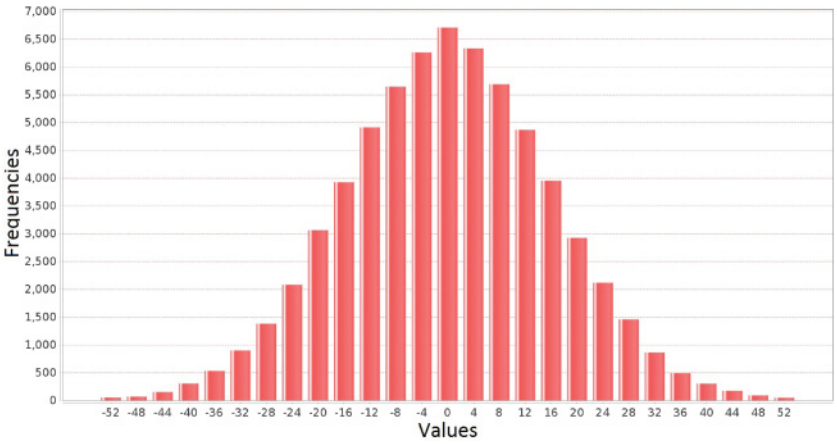

Figure 2: The Walsh–Hadamard spectrum of an S-Box S_2, size 8×8, $N_F = 102$.

Table 11: Results obtained using the initial version of our cost function. Presented values are best and average solutions found in format nonlinearity/δ-uniformity.

	GA		GaT		LSA	
S-Box size/Algorithm	Best	Avg	Best	Avg	Best	Avg
4×4	4/4	4/4.96	4/4	4/5.36	4/4	4/4.92
5×5	10/4	10/5.88	10/4	10/6.04	10/4	10/6.04
6×6	22/6	21.92/8.16	22/6	22/8.08	22/6	22/8.04
7×7	48/8	48/9.5	48/8	48/8.7	48/8	48/9.3
8×8	104/8	102.04/10.34	104/8	102.16/10.36	102/10	102/10.15

Table 12: Average number of evaluations needed with the initial version of our cost function, S-Box size 8×8.

Algorithm/Nonlinearity	98	100	102	104
GA	416	7 153	38 472	NPR
GaT	453	1 091	12 822	NPR
LSA	468	1 190	17 881	NPR

nonlinearity 102: S-Box S_1 and S-Box S_2 with their respective Walsh–Hadamard spectra given in Figures 1 and 2. When using our cost function, S_1 is considered better since it has fewer coefficients with the absolute value of 52.

The results obtained with our initial version of cost function are presented in Tables 11 and 12. In the latter table, we show the speed of convergence for our cost function for different nonlinearity values and 8×8 S-Box size.

The initial version of our cost function gives promising results, but not as good as those acquired with the Clark's cost function. To improve our cost function, we consider

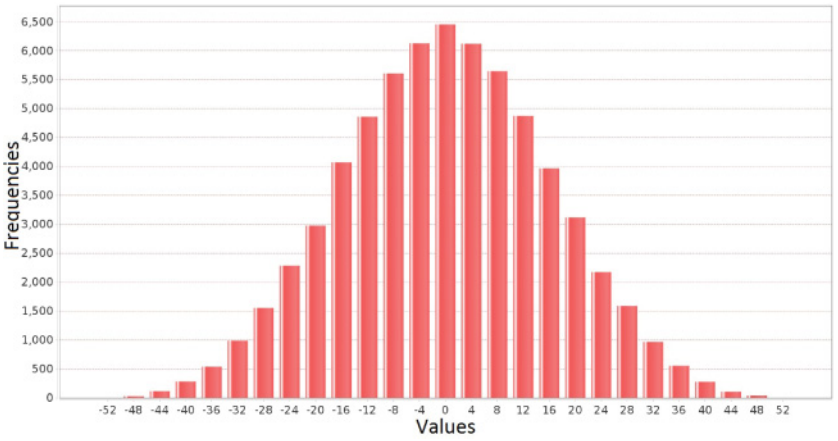

Figure 3: The Walsh–Hadamard spectrum of an S-Box, size 8×8, $N_F = 104$.

now up to N coefficients in the Walsh–Hadamard spectrum. By doing so, we aim to decrease the number of any of those N absolute coefficients.

The "importance" of reducing the number of the absolute value coefficients is proportional with the coefficients' absolute values. To state it differently, the most important thing is to reduce the number of maximum absolute value coefficients and then in the decreasing order the number of all absolute value coefficients. Therefore, one can expect good results if one reduces only the biggest absolute coefficient values. However, to improve the obtained results, one should also consider smaller values of coefficients.

Let $\vec{H(S)}$ be a histogram of absolute values of the Walsh–Hadamard coefficients for an S-Box S. It is a vector having on position i the number of coefficients equal to $|4i|$ in the Walsh–Hadamard spectrum of an S-Box S. Let l denote the maximal (last) position in this vector with the nonzero value. Then the maximum absolute value of the Walsh–Hadamard coefficients for an S-Box S is $4l$ and this coefficient determines the S-Box nonlinearity. Now, our cost function is given as:

$$CF(S) = \sum_{i=0}^{N-1} 2^{-i} H(S)_{l-i}, \tag{12}$$

where $CF(S)$ is a cost function applied to an S-Box S, $H(S)_k$ is the k-th component of a zero-indexed vector $\vec{H(S)}$, and we set $H(S)_k = 0, \forall k < 0$. Note when $N = 1$, this expression reduces to our initial cost function.

In Equation (12), by using the terms 2^{-i} we set the aforementioned "importance" of reducing the number of absolute value coefficients in the vector $\vec{H(S)}$. We see from the expression that the number of coefficients with the maximum absolute value will be multiplied with the largest value of the term 2^{-i} (which is $2^{-0} = 1$) making it the most influential while computing the value of the cost function.

As stated in Section 5.5, in our experiments we set N to the value of 10. Note that if an S-Box does not have 10 coefficient levels (i.e., $N < 10$), we then take all the possible coefficients into account when calculating the value of the cost function. For a concrete example we take 8×8 S-Box with the nonlinearity 104 with its Walsh–Hadamard spectrum shown in Figure 3. Here, we take the frequency of coefficients with the value ± 48, then we add the number of coefficients with the value ± 44 divided by 2^1 and continue, as defined in Equation (12), all the way until the number of coefficients

Table 13: Results obtained using the final version of our cost function. Presented values are best and average solutions found in format nonlinearity/δ-uniformity.

	GA		GaT		LSA	
S-Box size/Algorithm	Best	Avg	Best	Avg	Best	Avg
4×4	4/4	4/7.72	4/4	4/6	4/4	4/5.6
5×5	10/4	10/6	10/4	10/5.4	10/4	10/5.76
6×6	22/6	22/6.56	22/6	22/6.72	22/6	22/6.64
7×7	48/6	48/8.16	48/6	48/8.08	48/6	48/8.04
8×8	104/8	104/9.12	104/8	104/8.88	104/8	104/8.68

Table 14: The average number of evaluations needed with the final version of our cost function, S-Box size 8×8.

Algorithm/Nonlinearity	98	100	102	104
GA	473	6184	24 963	741 371
GaT	172	751	4519	167 451
LSA	387	1003	4362	172 280

Table 15: The maximum nonlinearity value found for each cost function.

S-Box dimensions/	Clark's cost function		Our cost function	
Cost function	$X = 21, R = 7$	$X = 4, R = 3$	$N = 1$	$N = 10$
4×4	4	4	4	4
5×5	10	10	10	10
6×6	22	22	22	22
7×7	46	48	48	48
8×8	104	102	104	104

with the value ± 12 divided by 2^9 is added. The results obtained with the final version of our cost function are presented in Tables 13 and 14.

6.3 Result Comparison

Here, we provide a comparison of results acquired with the Clark's and our cost functions. In Table 15 are given the maximum nonlinearity values acquired with the aforesaid cost functions. Naturally, since the problem of nonlinearity is a discrete one, where we have only a small number of possible values, it is hard to conclude which of the cost functions performs the best.

Next, we present algorithms' convergence for an S-Box of 8×8 size. In Figure 4, we compare convergence of the Clark's function with the parameters $X = 4$, $R = 3$ (Clark et al., 2005) and $X = 21$, $R = 7$ (Tesař, 2010). Based on the results presented, we conclude that the parameter pair $X = 21$, $R = 7$ is indeed better than $X = 4$, $R = 3$. The

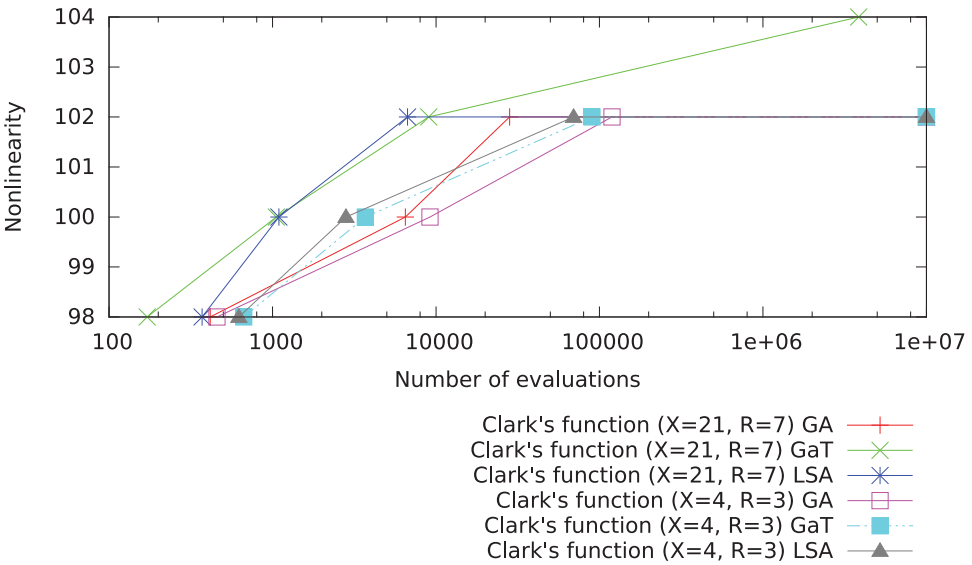

Figure 4: Convergence of algorithms GA, GaT, and LSA using the Clark's cost function with two sets of parameter pairs: $X = 21$, $R = 7$ and $X = 4$, $R = 3$, S-Box size 8×8.

convergence speed of the pair $X = 21$, $R = 7$ is much better and with the GaT algorithm it repeatedly manages to find S-Boxes with the nonlinearity value 104. On the other hand, with the parameters $X = 4$, $R = 3$ none of the three algorithms used could repeatedly produce S-Boxes with the nonlinearity 104. However, as mentioned in Section 6.1, for smaller S-Box sizes, parameter pair $X = 21$, $R = 7$ gives worse results than the parameter pair $X = 4$, $R = 3$. In order to achieve the best results using the Clark's function we would have to perform a more exhaustive parameter tuning for every S-Box dimension, which is of course possible, but is a time-consuming process.

In Figure 5, we compare the performance of two versions of our cost function. As expected, the final version of our cost function outperforms the initial version. With the initial version we considered only the frequency of the maximum absolute value of the Walsh–Hadamard spectrum which seems to cause algorithms to get "stuck" in locally optimal solutions as soon as they reach solutions with the nonlinearity value 102. With the final version of our cost function we consider more frequencies of the Walsh–Hadamard spectrum (as defined in Equation (12)) which allowed algorithms to repeatedly find solutions with the nonlinearity values of 104.

Finally, in Figure 6, we provide a graph of convergence using the final version of our cost function and the Clark's cost function ($R = 7$, $X = 21$) with all three algorithms while optimizing 8×8 S-Box. Using our cost function an S-Box with the nonlinearity 104 is found with GaT and LSA algorithms in less than 200 000 evaluations, which is by an order of magnitude faster than the result obtained with GaT algorithm using the Clark's cost function which took over 3 million evaluations. The algorithms GA and LSA were not able to find an S-Box with the nonlinearity 104 using the Clark's cost function before 9 million evaluations were performed. However, when using our cost function they both repeatedly generated S-Boxes with the nonlinearity 104.

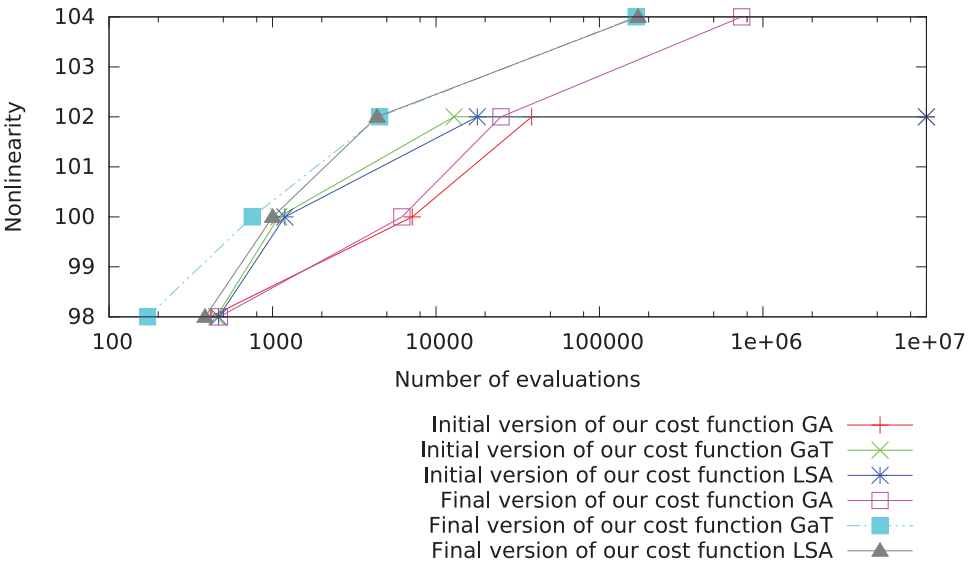

Figure 5: Convergence of algorithms GA, GaT, and LSA using the initial and final version of our cost function, S-Box size 8×8.

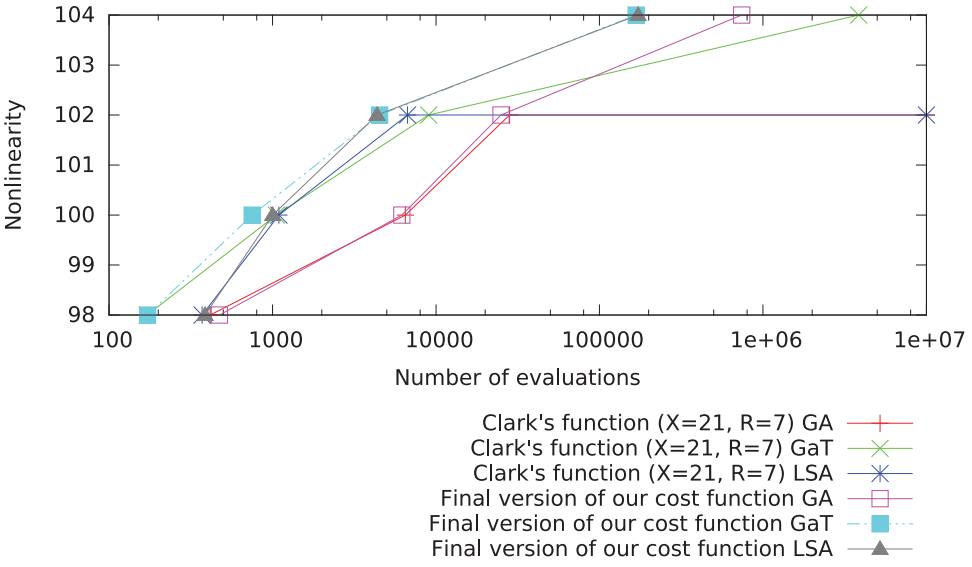

Figure 6: Convergence of algorithms GA, GaT, and LSA using both Clark's cost function with parameters $X = 21$ and $R = 7$ and the final version of our cost function, S-Box size 8×8.

6.4 Multiobjective Optimization

In experiments presented up to now, we focused only on the nonlinearity property as our objective in the optimization process. As mentioned in Section 2, δ-uniformity is another

Table 16: Results obtained using NSGA-II algorithm with goal objectives o_1 and o_2 defined in Equation (13) and Equation (14), respectively.

			Average	
S-Box dimensions	Max N_F	Min δ_F	N_F	δ_F
4×4	4	4	4	4
5×5	10	4	10	4.2
6×6	22	6	22	6
7×7	48	6	48	6.76
8×8	104	8	104	8.24

important cryptographic property, which we calculated a posteriori in the previous experiments. Here, we include it in our objective function to investigate whether we can improve on its value while maintaining high values of the nonlinearity property.

One way to include δ-uniformity in the optimization process would be to merge it with Equation (12). However, that would necessitate a new set of parameters used to blend the original equation and the δ-uniformity; these new parameters would then have to be tuned, which is undesirable.

To alleviate these problems, we follow a different approach. We decide to apply multiobjective optimization with NSGA-II algorithm (Deb et al., 2002). Our effective cost function used in the previous algorithms is already composite, having the nonlinearity as the first component and the cost function defined in Equation (12) as the second component. When comparing two solutions we perform a hierarchical comparison. Here, we decide to blend the two components into a single objective $o_1(S)$ as defined by the expression:

$$o_1(S) = 2^{2n} N_F - CF(S). \tag{13}$$

where n is the dimension of an S-Box ($n \times n$) and N_F is the nonlinearity of the S-Box.

The second objective is defined as the negative value of the δ-uniformity itself:

$$o_2(S) = -\delta_F. \tag{14}$$

The reason for defining the o_1 as a single objective instead of applying multiobjective optimization with three criteria (nonlinearity, $CF(S)$, δ-uniformity) is that solutions having low $CF(S)$ could have low nonlinearity (since $CF(S)$ is computed considering only a part of the Walsh–Hadamard spectrum) and these solutions are not of interest. Therefore, the objective o_1 reflects the nonlinearity and the quality of spectral components combined, and tries to emulate the hierarchical comparison we used previously. To justify multiplying the N_F parameter with 2^{2n}, observe that by the definition, for an S-Box of size $n \times n$, the total number of the Walsh–Hadamard coefficients is not larger than 2^{2n}. Since in our definition of $CF(S)$ we sum over a part of the spectrum, our $CF(S)$ is also upper bounded by 2^{2n}. This then ensures that the objective o_1 is monotone with respect to N_F.

Then, the multiobjective optimization was performed by maximizing both objectives. The results obtained are presented in Tables 16 and 17. The results show that using δ-uniformity as another objective did not have any impact with respect to the nonlinearity, and it actually slowed down the algorithm convergence when compared to the previous experiments. However, it did slightly help to improve the average value of the δ-uniformity when compared to the results shown in Table 13. One of the reasons that

Table 17: Convergence rate of NSGA-II algorithm with goal objectives o_1 and o_2 defined in Equation (13) and Equation (14), respectively.

Nonlinearity	98	100	102	104
Evaluations	498	26 906	53 431	878 383

the algorithm could not find much better S-Boxes with respect to the δ-uniformity could be the fact that δ-uniformity is equally noninformative for the evolutionary search process as is the nonlinearity property. Possibly better results could be obtained by applying the same principle of creating the histogram of values in the difference distribution table and creating a weighted sum as done in Equation (12) for the Walsh–Hadamard coefficients when considering the nonlinearity property.

6.5 Discussion

When considering the performance of the GaT algorithm (Tesař, 2010), we observe that its special case of GA performed quite well, but its efficiency lies in the tree part of the algorithm. The tree part focuses on discreteness of this problem and provides promising results. Because of the total tree search, this algorithm was the only one able to repeatedly provide S-Box solutions with the nonlinearity 104 when using the Clark's cost function, but we note its considerably slow convergence rate.

Results acquired with GA were commensurate with the ones provided by the other two algorithms, but its convergence rate was considerably slower as can be seen in Figures 4, 5, and 6. There is also a question whether the crossover operator is overly destructive. To answer that, the Evolution Strategy algorithm was executed (with the same mutation operators), but the acquired results did not differ much from the ones obtained with the GA.

When discussing the LSA, we note that it is a simple, greedy local search algorithm that uses random permutations of the current solution in an attempt to find better solutions. LSA was used to compare the results of a simple algorithm to more complex algorithms that were investigated. With the results obtained we would like to emphasize the difficulty of this problem, as other well-known sophisticated evolutionary algorithms like Genetic Algorithm, Particle Swarm Optimization Algorithm, and Evolution Strategy can hardly outperform results obtained using an algorithm as simple as LSA.

Next, when considering the δ-uniformity property, we see that including it in the fitness function does not lead to better results, and makes the algorithm's convergence slightly slower. This points us to the conclusion that when obtaining high nonlinearity values, δ-uniformity will also have good values and it remains to be seen whether there is actual advantage in including that property to the fitness function. An interesting research question would be whether it is enough to evolve S-Boxes only with regards to the δ-uniformity property and would the nonlinearity then reach high values on the average. We note that all the presented results are obtained when using random initial populations. In the case that we use initial populations with good S-Boxes as done by Ivanov et al. (2016), it seems the problem becomes easier, and we can reach higher nonlinearity values (up to 112 for the 8×8 size). However, it remains to be seen how such a seeding technique would behave if we seed our initial population with good, but suboptimal S-Boxes, since the authors use as the initial population S-Boxes with optimal properties.

7 Conclusion

In this article, we examine the problem of evolving highly nonlinear S-Boxes. We present a cost function that is faster than those in the related work and yet is able to repeatedly find S-Boxes with good nonlinearity. This cost function is defined as a two-component vector with the nonlinearity as the first component and a cost associated with a part of the Walsh–Hadamard spectrum as the second component; when using in a single criterion optimizer, to determine better solutions, a hierarchical comparison should be performed. To prove that our cost function is indeed competitive, we conducted extensive experiments on five different sizes of bijective S-Boxes. All our results confirm that this new cost function should be a function of choice when evolving highly nonlinear bijective S-Boxes. Contrasting other cost functions, ours can be understood intuitively and has no additional parameters (N could be regarded as a parameter, but with semantics). In our future work, we plan to concentrate on S-Boxes that have different sizes of inputs and outputs. Finally, since better nonlinearity values still elude our attempts, especially for larger S-Box sizes, we plan to continue investigating how to achieve results comparable with algebraic constructions.

Acknowledgments

This work has been supported in part by the Croatian Science Foundation under the project IP-2014-09-4882. In addition, this work was supported in part by the Research Council KU Leuven (C16/15/058), (CREA/14/005), and IOF project EDA-DSE (HB/13/020).

References

Biham, E., and Shamir, A. (1991). Differential cryptanalysis of DES-like cryptosystems. In *Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*, pp. 2–21.

Bogdanov, A., Knudsen, L. R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J., Seurin, Y., and Vikkelsoe, C. (2007). PRESENT: An ultra-lightweight block cipher. In *Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 450–466.

Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E., Knezevic, M., Knudsen, L., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S., and Yalçin, T. (2012). PRINCE: A low-latency block cipher for pervasive computing applications. In *Advances in Cryptology*, pp. 208–225. Lecture Notes in Computer Science, vol. 7658.

Budaghyan, L., Carlet, C., and Pott, A. (2006). New classes of almost bent and almost perfect nonlinear polynomials. *IEEE Transitions in Information Theory*, 52(3):1141–1152.

Burnett, L., Carter, G., Dawson, E., and Millan, W. (2001). Efficient methods for generating MARS-like S-Boxes. In *Proceedings of the 7th International Workshop on Fast Software Encryption*, pp. 300–314.

Burnett, L. D. (2005). *Heuristic optimization of Boolean functions and substitution boxes for cryptography*. PhD thesis, Queensland University of Technology.

Burwick, C., Coppersmith, D., D'Avignon, E., Gennaro, R., Halevi, S., Jutla, C., Matyas, S. M., O'Connor, L., Peyravian, M., Safford, D., and Zunic, N. (1999). The MARS Encryption Algorithm.

Carlet, C. (2005). On highly nonlinear s-boxes and their inability to thwart DPA attacks. In *Proceedings of the 6th International Conference on Cryptology in India*, pp. 49–62.

Carlet, C. (2010a). Boolean functions for cryptography and error correcting codes. In Y. Crama and P. L. Hammer (Eds.), *Boolean models and methods in mathematics, computer science, and engineering*, pp. 257–397. New York: Cambridge University Press.

Carlet, C. (2010b). Vectorial Boolean functions for cryptography. In Y. Crama and P. L. Hammer (Eds.), *Boolean models and methods in mathematics, computer science, and engineering*, pp. 398–469. New York: Cambridge University Press.

Clark, A. J. (1998). *Optimisation heuristics for cryptology*. PhD thesis, Queensland University of Technology.

Clark, J. A., Jacob, J., and Stepney, S. (2004). Searching for cost functions. In *Congress on Evolutionary Computation*, vol. 2, pp. 1517–1524.

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

Daemen, J., Peeters, M., Assche, G. V., and Rijmen, V. (2000). Nessie proposal: The block cipher Noekeon. Nessie submission.

Daemen, J., and Rijmen, V. (2002). *The design of Rijndael*. New York: Springer-Verlag.

Davis, L. (1985). Applying adaptive algorithms to epistatic domains. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pp. 162–164.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Dib, S. (2014). Asymptotic nonlinearity of vectorial Boolean functions. *Cryptography and Communications*, 6(2):103–115.

Ege, B., Papagiannopoulos, K., Batina, L., and Picek, S. (2015). Improving DPA resistance of s-boxes: How far can we go? In *2015 IEEE International Symposium on Circuits and Systems*, pp. 2013–2016.

Eiben, A. E., and Smith, J. E. (2003). *Introduction to evolutionary computing*. Berlin/Heidelberg/New York: Springer-Verlag.

FIPS (1999). FIPS 46-3, Data Encryption Standard (DES). National Institute for Standards and Technology (NIST), Gaithersburg, MD, USA.

Fuller, J., and Millan, W. (2003). Linear redundancy in s-boxes. In *Fast Software Encryption*, pp. 74–86. Lecture Notes in Computer Science, vol. 2887.

Fuller, J., Millan, W., and Dawson, E. (2004). Multi-objective optimisation of bijective s-boxes. In *Congress on Evolutionary Computation*. pp. 1525–1532.

Goldberg, D. E., and Lingle, R. (1985). Alleles, loci, and the traveling salesman problem. In *Proceedings of the International Conference on Genetic Algorithms and Their Applications*, pp. 154–159.

Ivanov, G., Nikolov, N., and Nikova, S. (2015). Cryptographically strong s-boxes generated by Modified Immune Algorithm. In *Cryptography and Information Security in the Balkans—Second International Conference, Revised Selected Papers*, pp. 31–42.

Ivanov, G., Nikolov, N., and Nikova, S. (2016). Reversed genetic algorithms for generation of bijective s-boxes with good cryptographic properties. *Cryptography and Communications*, 8(2):247–276.

Kazymyrov, O., Kazymyrova, V., and Oliynykov, R. (2013). A method for generation of high-nonlinear s-boxes based on gradient descent. Cryptology ePrint Archive, Report 2013/578.

Leander, G., and Poschmann, A. (2007). On the classification of 4 bit s-boxes. In *Arithmetic of Finite Fields*, pp. 159–176. Lecture Notes in Computer Science, vol. 4547.

Millan, W., Burnett, L., Carter, G., Clark, A., and Dawson, E. (1999). Evolutionary heuristics for finding cryptographically strong s-boxes. In *Information and Communication Security*, pp. 263–274. Lecture Notes in Computer Science, vol. 1726.

Nyberg, K. (1991). Perfect nonlinear s-boxes. In *Proceedings of Advances in Cryptology—EUROCRYPT '91, Workshop on the Theory and Application of Cryptographic Techniques*, pp. 378–386. Lecture Notes in Computer Science, vol. 547.

Nyberg, K. (1993). On the construction of highly nonlinear permutations. In *Advances in Cryptology*, pp. 92–98. Lecture Notes in Computer Science, vol. 658.

Picek, S., Ege, B., Batina, L., Jakobovic, D., Chmielewski, L., and Golub, M. (2014). On using genetic algorithms for intrinsic side-channel resistance: The case of AES s-box. In *Proceedings of the First Workshop on Cryptography and Security in Computing Systems*, pp. 13–18.

Picek, S., Ege, B., Papagiannopoulos, K., Batina, L., and Jakobovic, D. (2014). Optimality and beyond: The case of 4x4 s-boxes. In *2014 IEEE International Symposium on Hardware-Oriented Security and Trust*, pp. 80–83.

Picek, S., Marchiori, E., Batina, L., and Jakobovic, D. (2014). Combining evolutionary computation and algebraic constructions to find cryptography-relevant Boolean functions. In *Proceedings of Parallel Problem Solving from Nature*, pp. 822–831.

Picek, S., Mazumdar, B., Mukhopadhyay, D., and Batina, L. (2015). Modified transparency order property: Solution or just another attempt? In *Proceedings Security, Privacy, and Applied Cryptography Engineering*, pp. 210–227.

Picek, S., Miller, J. F., Jakobovic, D., and Batina, L. (2015). Cartesian genetic programming approach for generating substitution boxes of different sizes. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation, GECCO*, pp. 1457–1458.

Picek, S., Papagiannopoulos, K., Ege, B., Batina, L., and Jakobovic, D. (2014). Confused by confusion: Systematic evaluation of DPA resistance of various s-boxes. In *Proceedings of Progress in Cryptology*, pp. 374–390.

Ryan, E., Azad, R., and Ryan, C. (2004). On the performance of genetic operators and the random key representation. In *Genetic Programming*, pp. 162–173. Lecture Notes in Computer Science, vol. 3003.

Schneier, B. (1995). *Applied cryptography (2nd ed.): Protocols, algorithms, and source code in C*. New York: Wiley and Sons.

Syswerda, G. (1985). Schedule optimization using genetic algorithms. In L. Davis (Ed.), *Handbook of genetic algorithms*, pp. 332–349. New York: Van Nostrand Reinhold.

Tesař, P. (2010). A new method for generating high non-linearity s-boxes. *Radioengineering*, 19(1):23–26.