Evolutionary Methods for the Construction of Cryptographic Boolean Functions

Stjepan Picek[1,3], Domagoj Jakobovic[1](✉), Julian F. Miller[2], Elena Marchiori[3], and Lejla Batina[3]

[1] Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia
stjepan@computer.org, domagoj.jakobovic@fer.hr
[2] Department of Electronics, University of York, York, UK
[3] Radboud University Nijmegen, Nijmegen, The Netherlands

Abstract. Boolean functions represent an important primitive when constructing many stream ciphers. Since they are often the only nonlinear element of such ciphers, without them the algorithm would be trivial to break. Therefore, it is not surprising there exist a substantial body of work on the methods of constructing Boolean functions. Among those methods, evolutionary computation (EC) techniques play a significant role. Previous works show it is possible to use EC methods to generate high-quality Boolean functions that even surpass those built by algebraic constructions. However, up to now, there was no work investigating the use of Cartesian Genetic Programming (CGP) for producing Boolean functions suitable for cryptography. In this paper we compare Genetic Programming (GP) and CGP algorithms in order to reach the conclusion which algorithm is better suited to evolve Boolean functions suitable for cryptographic usage. Our experiments show that CGP performs much better than the GP when the goal is obtaining as high as possible nonlinearity. Our results indicate that CGP should be further tested with different fitness objectives in order to check the boundaries of its performance.

Keywords: Boolean functions · Genetic programming · Cartesian Genetic Programming · Cryptographic properties · Comparison

1 Introduction

Most cryptographic systems in use today are built as hybrid cryptosystems. In these systems asymmetric-key cryptography is used to exchange the keys and symmetric-key cryptography is used to encrypt and decrypt data. This separation is due to the fact that symmetric-key cryptography is much faster than the asymmetric-key [8]. The name symmetric-key denotes the fact that the same key is used for both data encryption and decryption.

One usual division of symmetric-key cryptography is block and stream ciphers [8]. In block ciphers algorithms encrypt and decrypt data in blocks of

P. Machado et al. (Eds.): EuroGP 2015, LNCS 9025, pp. 192–204, 2015.
DOI: 10.1007/978-3-319-16501-1_16

certain size and in stream ciphers this is done bitwise. In both of these types of cipher often the only nonlinear elements are Boolean functions or vectorial Boolean functions (vectorial Boolean functions are better known as Substitution boxes or S-boxes). Boolean functions are in general used in stream ciphers whereas S-boxes are used in block ciphers. In the rest of this paper we concentrate only on Boolean functions suitable for cryptographic usage in stream ciphers.

There exist three main approaches to generate Boolean functions for cryptographic usage: algebraic construction, random generation and heuristic construction [6]. In algebraic construction one usually uses some mathematical procedure that gives very good results such as the cipher RAKAPOSHI [3]. One of the most famous constructions is a finite field inversion [17]. However, although finite field inversion can be used to generate S-boxes with the highest possible nonlinearity levels, this is not so for Boolean functions. Furthermore, such constructions cannot give optimal results when considering side-channel attack resistance [21].

Random generation of Boolean functions has its strong points, the most prominent being that it is easy and fast, but the resulting Boolean functions usually have suboptimal properties for cryptography [11].

Heuristic methods offer easy and efficient way of producing large number of Boolean functions with very good cryptographic properties [2]. Among other heuristic methods, evolutionary computation (EC) offers highly competitive results when generating Boolean functions [19]. More details about different methods for evolving Boolean functions are given in Sect. 1.1. However, as far as the authors know, Cartesian Genetic Programming (CGP) has never been used for constructing Boolean functions suitable for cryptography. Since CGP is recognized as a suitable option when generating Boolean functions [13,14], its absence in the evolution of cryptography-suitable Boolean function creation is somewhat surprising.

In this paper we concentrate only on Boolean functions with 8 inputs since that represents the size most used in practical scenarios (e.g. cipher RAKAPOSHI [3]). Evolving Boolean functions with 8 inputs is a challenging task since there exist 2^{2^n} possible functions of n inputs (i.e. for 8 inputs this gives 2^{256} candidate solutions). To serve as a benchmark problem when comparing the algorithms, we look for a balanced Boolean function with maximum nonlinearity. However, this problem should not be only be considered as a benchmark, but rather as a difficult problem that has practical implications. It is a well known fact among the cryptography community that the upper bound for the nonlinearity property in the case of an 8-bit balanced Boolean function is 118 [22]. However, no one has been able to find such a function. Indeed, finding it would represent not only a significant result from the cryptographic perspective but also from the EC perspective since it would help profile EC methods as the truly viable option for cryptographic usages.

1.1 Related Work

As previously stated, there have been several applications of heuristic methods to the generation of Boolean functions for cryptography. Here we give only a few representative examples of work related to our research.

Millan et al. use Genetic Algorithms to evolve Boolean functions that have high nonlinearity [10]. Clark et al. experiment with Simulated Annealing when evolving Boolean functions with cryptography-relevant properties [4]. Burnett in her thesis use Genetic Algorithms in a combination with hill climbing to evolve Boolean functions and S-boxes [2]. McLaughlin and Clark on the other hand use Simulate Annealing to evolve Boolean functions that have several cryptographic properties with optimal values [9]. Picek et al. experiment with Genetic Programming and Genetic Algorithms to find Boolean functions that possess several optimal properties [19]. Several evolutionary algorithm methods are used by Picek et al. to evolve Boolean functions that have better DPA-related properties [18]. With the goal of finding maximal nonlinearity values of Boolean functions Picek et al. experiment with a handful of evolutionary algorithms and approaches [20]. Hrbacek and Dvorak use CGP to evolve bent Boolean functions of size up to 16 inputs. However, since bent Boolean functions should not be used in cryptography [5] this work has a limited applicability from the cryptographic perspective.

1.2 Our Contributions

To our best knowledge we are the first to consider CGP when evolving Boolean functions suitable for cryptographic usage. Furthermore, we experiment with different genotype sizes and mutation rates to investigate their influence on the ability of CGP to find good solutions. Since there is no prior experimental work, this should also be regarded as a guideline for future research. When experimenting with GP, we also investigate the influence of tree depth on the quality of the obtained solutions. We compare GP and CGP algorithms on a real-world difficult cryptographic problem to investigate their suitability.

The remainder of this paper is organized as follows: in Sect. 2 we describe relevant cryptographic properties and representations of Boolean functions. In Sect. 3 experimental setup and algorithms are presented, while results and short discussion are given in Sect. 4. Finally, Sect. 5 concludes with some suggestions for future work.

2 Boolean Functions and Their Properties

In this section we give a short overview of relevant cryptographic properties of Boolean functions. For further details we refer interested readers to [1,5].

The inner product of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ is denoted as $\boldsymbol{a} \cdot \boldsymbol{b}$. It is defined as $\oplus_{i=1}^{n} a_i b_i$, where "$\oplus$" represents addition modulo 2 (bitwise XOR).

A Boolean function f on $\mathbb{F}_2^n$ can be uniquely represented by a truth table (TT), which is a vector $(f(\boldsymbol{0}), ..., f(\boldsymbol{1}))$ that contains the function values of f, ordered lexicographically [1].

The Hamming weight $HW(f)$ of a Boolean function f is the number of ones in its binary truth table representation [1].

The second unique representation of Boolean function is the Walsh transform. It measures the similarity between $f(\boldsymbol{x})$ and the linear function $\boldsymbol{a} \cdot \boldsymbol{x}$ [1]. The Walsh transform of a Boolean functions f equals:

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

A Boolean function is **balanced** (denoted "BAL" throughout the paper) if its Hamming weight is equal to 2^{n-1} [1].

The **nonlinearity** NL_f of a Boolean function f can be expressed in terms of the Walsh coefficients as [1]:

$$NL_f = 2^{n-1} - \frac{1}{2} max_{\boldsymbol{a} \in \mathbb{F}_2^n} |W_f(\boldsymbol{a})|. \tag{2}$$

A Boolean function f is **correlation immune** of order t - $CI(t)$ if the output of the function is statistically independent of the combination of any t of its inputs [1]. For the Walsh spectrum it holds that

$$W_f(\boldsymbol{a}) = 0, \text{ for } 0 \leq HW(\boldsymbol{a}) \leq t. \tag{3}$$

A Boolean function f is **t-resilient** if it is balanced and with correlation immunity of degree t [1]. Due to the lack of space, we do not explain the roles of each property in the security application of Boolean function, but we rather refer readers to relevant literature.

2.1 Balanced Boolean Functions and Maximal Nonlinearity

Sarkar and Maitra showed that if a t-resilient Boolean function f has an even number of inputs n and $t + 1 \leq \frac{n}{2} - 1$ then its nonlinearity NL_f is bounded as follows [22]:

$$NL_f \leq 2^{n-1} - 2^{\frac{n}{2}-1} - 2^{t+1}. \tag{4}$$

Since we are looking for a Boolean function that has maximal nonlinearity, we can see that the resilience needs to be 0 which then simplifies the equation to the following one:

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

From the formula it follows that the maximum nonlinearity for $n = 8$ and $t = 0$ equals 118.

3 Algorithms and Experimental Setup

We remind the reader that we focus on the evolution of Boolean functions that are balanced and with as high nonlinearity as possible. Naturally, the end goal is to find such a function that has nonlinearity 118, but even lower values can help us to reach the conclusion about the strength of a certain method. Moreover, such Boolean functions can have also practical applications in the design of stream ciphers. To conclude, the goals of our experiments can be stated through the following questions.

- Is CGP suitable for evolving Boolean functions when the focus is on the cryptographic usage?
- What is the influence of the genotype size on the quality of the solutions obtained?
- How does the performance of CGP compare with GP?
- What is the influence of tree depth in GP when evolving cryptographically suitable Boolean functions?

Additionally, we experiment with Genetic Algorithm (GA) which serves as a basic case scenario to determine a reference performance of the algorithm.

3.1 Genetic Algorithm

Our GA implementation uses the function truth table as chromosome representation, which is an array of bits of length 2^n, where n is the size of a Boolean function (therefore, in this research the chromosome length is 256 bits). For GA we use a steady state tournament selection with tournament size k equal to 3 and population size 100. In steady state tournament selection mechanism the worst of k randomly selected individuals is identified and replaced with a new individual. The new individual is constructed with the crossover of two random surviving parents from the tournament. After crossover, each new individual undergoes a mutation with a given probability.

We experimented with several genetic operators, but the best results were obtained with one-point crossover and simple mutation which inverts a randomly selected bit.

3.2 Tree-Based Genetic Programming

In Genetic Programming a function is represented as a tree of a certain depth. The inner nodes (function set) of a tree are Boolean primitives (such as AND, OR, NOT), while the leaves (terminals) may be a single input Boolean variables (v0..v7). We use the same function set, which is given below, for both GP and CGP. In GP experiments, the mutation probability is set to 0.3 per individual, and the population size is 500. Steady-state tournament selection with tournament size of 3 is used.

A small number of experiments were also conducted to select the appropriate operators, and based on that we used a simple tree crossover with 90 % bias for functional nodes and a subtree mutation.

The maximum tree depth is a parameter that is selected by the user and influences the available genotype size. When GP/CGP is used, one is effectively evolving a digital circuit and then examining its truth table to assess whether the function has the desired properties (e.g. balancedness or nonlinearity). However, with a GA approach one is directly evolving a truth table, so that the question of how it is implemented is not involved. Indeed the size of the truth table determines the size of the GA genotype (bitstring) whereas in the GP/CGP approaches, the size of the genotype is not directly related to the size of the desired truth table.

3.3 Cartesian Genetic Programming

In Cartesian Genetic Programming a program is represented as an indexed graph. The graph is encoded in the form of a linear string of integers [15]. Terminal set (inputs) and node outputs are numbered sequentially. Node functions are also numbered separately [15].

CGP has three parameters that are chosen by the user; number of rows n_r, number of columns n_c and levels-back l [14]. The number of rows and number of columns make the two-dimensional grid of computational nodes and their product gives the maximum number of computational nodes. The levels-back parameter controls the connectivity of the graph, i.e. it determines which columns a node can get its input from [14].

In CGP the genotype is a list of integers that represents the program primitives and how they are connected together [16]. The genotype is mapped to the directed graph that is executed as a program. Genotypes are of fixed length while phenotypes have variable length in accordance with the number of unexpressed genes.

The maximal length of the genotype is given by the following formula:

$$max\_length = n_r n_c (n_n + l) + n_o. \tag{6}$$

In this application the number of node input connections n_n is 2 and the number of program output connections n_o is 1. The population size for CGP equals 5 in all our experiments. For CGP individual selection we use a (1 + 4)-ES evolution strategy in which offspring are favored over parents when they have a fitness less than or equal to the fitness of the parent. The mutation operator is one-point mutation where the mutation point is chosen with a fixed probability. The number of genes mutated is defined as fixed percentage of the total number of genes. Note, the single output gene is not mutated and is taken from the last node in the genotype. The genes chosen for mutation might be a node input connection or a function. For more details about CGP we refer readers to [13–16].

3.4 Fitness Functions

When searching for a balanced function with the best possible nonlinearity, we experimented with two fitness functions, both to be maximized. The first fitness function simply adds the balancedness penalty and nonlinearity values.

$$fitness = BAL + NL_f. \tag{7}$$

When a Boolean function is balanced we assign the BAL component a value of 0, and when it is unbalanced we assign it the negative difference up to the balancedness (i.e. the number of bits that need to be changed to reach balancedness) multiplied with a constant c. Based on the results from [19,20] we set that constant to 5 so that the unbalancedness penalty exceeds the values of nonlinearity.

For the second fitness function, we have used a two stage fitness in which a fitness bonus equal to the nonlinearity is awarded only to a genotype that is perfectly balanced (this occurs when $BAL = 0$); otherwise, the fitness is only the balancedness penalty. This is given in Eq. 8. The delta function $\delta_{BAL,0}$ takes the value one when $BAL = 0$ and is zero otherwise.

$$fitness = BAL + \delta_{BAL,0} NL_f. \tag{8}$$

Two stage fitness functions are commonly used in CGP when one is trying to optimize one quantity under a strict constraint; for instance, when trying to evolve a Boolean function that exactly matches a given truth table but which has the minimum number of gates [7]. Note that when Eq. 8 is used, one does not have to assign weights to the relative importance of different objectives. In Eq. 7 a nearly balanced Boolean function with high nonlinearity could receive the same fitness score as a fully balanced Boolean function with a lower nonlinearity. In the two stage fitness function described in Eq. 8 unbalanced Boolean functions are not assessed for nonlinearity at all. Note that we do not use a multiobjective approach, since the balancedness is a constraint rather than a separate objective.

An observant reader can easily notice that in Eq. 4 there is a resilience term which we know needs to be 0 so we disregard it and proceed to Eq. 5. The question is, should we disregard this property so readily? It is clear from those two formulas that the nonlinearity property changes in jumps of two and it always has an even value for Boolean functions with even number of inputs.

This means, if we reach the nonlinearity of 116, to move to the value of 118 actually a random search is performed - since there are no values between those two, the evolutionary algorithm has no means of differentiating different solutions with nonlinearity 116. To add this missing information, we may include the resilience property in the fitness function.

However, the problem is that we do not know what resilience values can lead to nonlinearity 118. It is plausible to consider it better to have the resilience as small as possible, since we know that for the best nonlinearity the resilience must be 0. However, it is possible that Boolean functions with resilience larger than 0 can lead the search towards new, unexplored areas which can eventually lead to nonlinearity 118. Since there is no research investigating those conditions at this moment, all that researchers can do is use their intuition to decide on the best approach. We take into account the first option where we add to the fitness function the constraint that the resilience must be 0 and carry out empirical experiments.

3.5 Experimental Setup

Since there are no previous results when using CGP to evolve Boolean functions with good cryptographic properties, first we need to consider how to set CGP parameters. Setting the number of rows to be 1 and levels-back to be equal to the number of columns is regarded as the best and most general choice [14]. This choice should be used when there is no specialist knowledge about the problem.

However, this still leaves open the question what should be the size of the number of columns parameter. Furthermore, CGP usually uses small population sizes and has no crossover operator [14]. Determining the best combination of maximum number of nodes (gates in this case) and mutation rate is an important step in hitting the parameter sweet spot for CGP. Indeed, it has been shown that generally very large genotypes and small mutation rates perform very well [12]. Thus some experiments were performed varying these two parameters.

Common Parameters. The following parameters of the experiments are common for all algorithms: the size of Boolean function is 8 (the size of the truth table is 256) and the number of independent runs for each experiment is 50. The function set n_f for both GP and CGP in all the experiments consists of binary Boolean primitives OR, XOR, AND, XNOR and AND with one input inverted. For the stopping condition we use the number of evaluations which we set to 500 000.

4 Results and Discussion

First we note that for the GA case, the best obtained result are balanced functions with nonlinearity value of 112 with the average of 111.8 over 50 runs. This is considerably worse than the best (and most average) solutions obtained with CGP, as shown below.

Furthermore, in all the experiments so far, we have been unable to obtain the nonlinearity of 118; only the value of 116 could be found for balanced functions. While not the maximum, this nonlinearity level is still very high for practical purposes, so we used the number of runs with 116 solution occurrences as a secondary measure of algorithm quality.

In Tables 1 and 2 we give results for CGP with fitness functions as in Eqs. 7 and 8 respectively, for different genotype sizes and mutation probabilities. The first value in each column represents the average value over all runs and the second value, in brackets, represents the number of obtained 116 nonlinearity solutions over all runs (higher is better for both values). The results for both fitness versions with GP for various tree depths are given in Table 3.

As it can be seen from the tables, CGP outperforms GA and GP quite easily. It should be noted that many additional GA and GP combinations were already experimented with in our previous research that are not shown here, which exhibit the same or worse performance than the configurations used in this work. Thus, we concentrate on the CGP efficiency which has not been previously investigated.

In addition, we can compare CGP variants with the weighted fitness and two-stage fitness. In Fig. 1 we plot the one-stage and two-stage fitness data shown in Tables 1 and 2 as a scatter graph showing the average fitness versus the genotype length for all mutation rates. We also show the number of nonlinearity 116 solutions found in both cases.

The results for the weighted fitness outperform two-stage fitness in many cases. This is a surprising result as a two-stage fitness is often used in CGP, ever since it was first described [7, 14]. It implies that more work should be done

Table 1. Results for Eq. 7 and CGP.

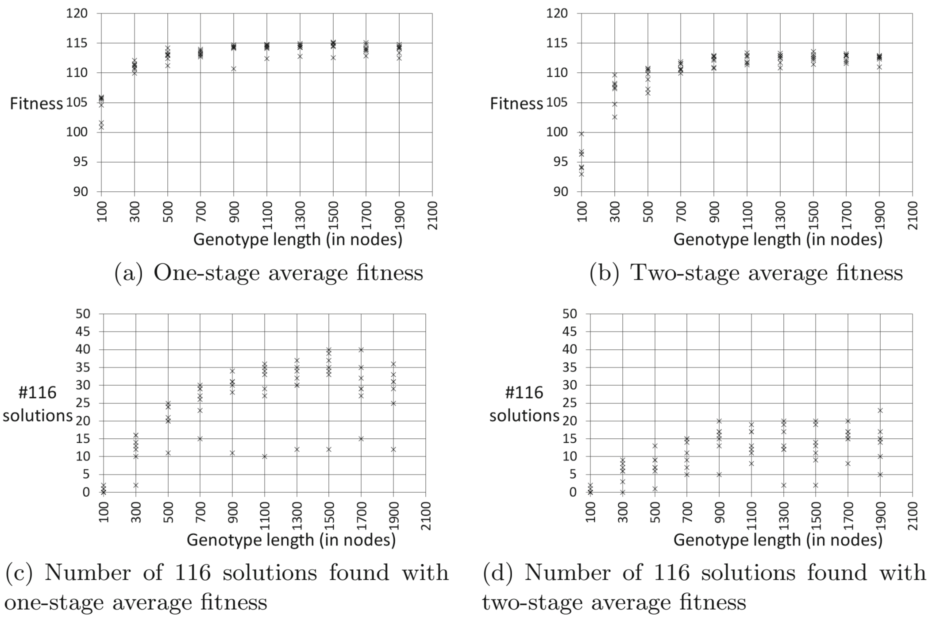
Genotype/p_m	1	3	5	7	9	11	13
100	101.58 (0)	105.78 (1)	100.9 (0)	105.52 (0)	105.94 (2)	105.68 (1)	104.58 (0)
300	110.86 (2)	110.62 (14)	111.22 (13)	111.5 (16)	109.98 (12)	112.12 (16)	111.36 (10)
500	111.26 (11)	112.94 (20)	113.04 (24)	113.5 (24)	114.18 (25)	113.16 (21)	112.42 (20)
700	112.92 (15)	112.7 (23)	113.24 (26)	113.76(27)	113.98 (29)	113.54 (29)	113.16 (30)
900	110.72 (11)	114.38(31)	114.16 (28)	114.48 (31)	114.28 (30)	114.32 (31)	114.7 (34)
1 100	112.4 (10)	114.28 (29)	114.82 (35)	114.56 (33)	114.14 (27)	114.44 (34)	114.74 (36)
1 300	112.76 (12)	114.38 (30)	114.76 (35)	114.3 (30)	114.3 (32)	114.98 (37)	114.58 (34)
1 500	112.58 (12)	114.56 (34)	114.58 (33)	115.08 (40)	114.44 (35)	114.96 (37)	115.16 (39)
1 700	112.88 (15)	113.96 (27)	114.8 (35)	113.7 (29)	114.2 (32)	113.94 (29)	115.12 (40)
1 900	112.52 (12)	114.12 (31)	114.32 (33)	114.48 (31)	114.8 (36)	114.22 (29)	113.38 (25)

Table 2. Results for Eq. 8 and CGP.

Genotype/p_m	1	3	5	7	9	11	13
100	94.16 (0)	96.8 (2)	92.96 (0)	96.32 (0)	94 (1)	99.76 (0)	96.32 (0)
300	108.28(0)	108.00 (8)	107.6 (3)	109.68 (9)	102.56 (6)	104.72 (7)	107.36 (6)
500	106.64(1)	110.8 (7)	108.92 (7)	110.4 (6)	110.64 (13)	107.28 (9)	109.84 (9)
700	111.92 (5)	109.96 (11)	111.6 (15)	110.64 (15)	110.68 (9)	111.52 (14)	110.48 (7)
900	110.8 (5)	112.32 (13)	112.76 (20)	112.08 (17)	112.72 (17)	110.96 (15)	112.92 (16)
1 100	111.64 (8)	112.96 (17)	112.96 (19)	113.36 (17)	111.84 (11)	111.40 (13)	112.72 (12)
1 300	110.88 (2)	112.84 (19)	113.28 (17)	112.96 (20)	111.72 (12)	112.48 (13)	112.56 (12)
1 500	111.48 (2)	112.48 (9)	112.20 (13)	113.60 (20)	113.12 (19)	112.76 (14)	112.52 (11)
1 700	112.16 (8)	111.6 (15)	112.88 (15)	111.88 (17)	112.92 (16)	113.04 (20)	113.20 (17)
1 900	111.0 (5)	112.96 (15)	112.76 (17)	112.6 (14)	112.64 (23)	112.8 (15)	112.36 (10)

Table 3. Results for GP.

Tree depth	5	7	9	11	13
Eq. 7	112.13 (1)	112.2 (2)	111.36(0)	111.64 (0)	111.22 (0)
Eq. 8	112.13 (1)	112 (0)	111.76 (1)	111.72 (0)	111.58 (0)



(a) One-stage average fitness

(b) Two-stage average fitness

(c) Number of 116 solutions found with one-stage average fitness

(d) Number of 116 solutions found with two-stage average fitness

Fig. 1. Comparative results for one-stage 1(a) and two-stage 1(b) fitness functions showing average fitness achieved and the number of 116 solutions found against genotype length for all mutation rates.

on a variety of problems to establish the relative merits of the two approaches. In addition in [12] it was suggested that optimal mutation rates should decrease as genotype length increases. However, the results here indicate that for the cryptographic problem studied this is not the case. Indeed fairly high mutation rates produced the best results. This is also surprising and merits further investigation.

When adding the resilience constraint to the fitness function, we observe that all Boolean functions within several generations obtain the resilience value of 0. This suggests that this condition is not hard enough objective to lead the search towards very high nonlinearity values in different parts of search space when compared with fitness functions without that objective.

When considering the average number of active nodes for CGP we give numbers for the best set of parameters and both fitness functions in Table 4. Notice that we selected the best algorithm on the basis of the number of achieved 116 nonlinearity values. In the case that two algorithms have the same number of 116 values, then we consider the average value as the second criteria.

We carried out longer runs of 10 million evaluations for the best combinations of CGP parameters considering the total number of obtained 116 nonlinearity values. We do not give similar comparison for GP since it is much slower and from that perspective is not competitive with CGP for such large number of evaluations. Figure 2 shows a boxplot comparison of best parameter

Table 4. Average number of active nodes.

Algorithm	CGP, Eq. 7	CGP, Eq. 8	CGP, Eq. 7, long run	CGP, Eq. 8, long run
Genotype, p_m	1 700, 13	1 900, 9	1 700, 13	1 900, 9
Value	84.24	76.62	81.76	86.33

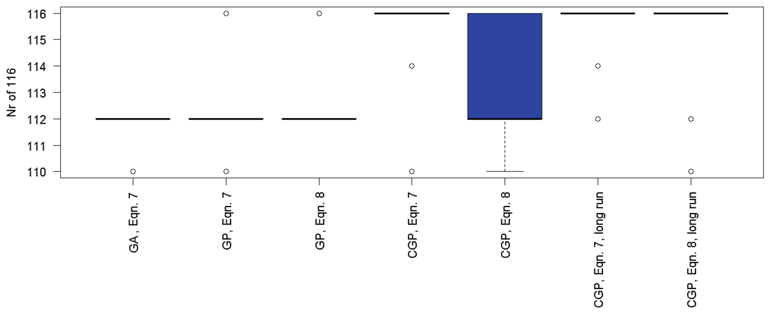


Fig. 2. Boxplot comparison of the most successful algorithms.

combinations for GA, GP and CGP with fitness functions Eqs. 7 and 8. Furthermore, we present best parameter combinations for CGP with 10 million evaluations. Note that the same parameter combinations for CGP are presented in Table 4.

5 Conclusion and Future Work

This paper describes an application of GA, GP and CGP in an evolution of cryptography relevant Boolean functions. The main contribution is the application of CGP, whose efficiency has not been previously investigated for this problem, and a comparison with two other methods. The results show that CGP is able to produce results that are clearly better than previous approaches, and is at the same time a valid choice for this kind of problem. Additionally, the described optimization problem may be considered a viable candidate as a benchmark problem for GP-related algorithms, both for its hardness as well as its real-world applicability.

Acknowledgments. This work was supported in part by the Technology Foundation STW (project 12624 - SIDES), The Netherlands Organization for Scientific Research NWO (project ProFIL 628.001.007) and the ICT COST action IC1204 TRUDEVICE.

References

1. Braeken, A.: Cryptographic Properties of Boolean Functions and S-Boxes. Ph.D. thesis, Katholieke Universiteit Leuven (2006)
2. Burnett, L.D.: Heuristic optimization of boolean functions and substitution boxes for cryptography. Ph.D. thesis, Queensland University of Technology (2005)

3. Cid, C., Kiyomoto, S., Kurihara, J.: The RAKAPOSHI stream cipher. In: Qing, S., Mitchell, C.J., Wang, G. (eds.) ICICS 2009. LNCS, vol. 5927, pp. 32–46. Springer, Heidelberg (2009)
4. Clark, J.A., Jacob, J.L., Stepney, S., Maitra, S., Millan, W.L.: Evolving boolean functions satisfying multiple criteria. In: Menezes, A., Sarkar, P. (eds.) INDOCRYPT 2002. LNCS, vol. 2551, pp. 246–259. Springer, Heidelberg (2002)
5. Crama, Y., Hammer, P.L.: Boolean Models and Methods in Mathematics, Computer Science, and Engineering, 1st edn. Cambridge University Press, New York (2010)
6. Goossens, K.: Automated creation and selection of cryptographic primitives. Master's thesis, Katholieke Universiteit Leuven (2005)
7. Kalganova, T., Miller, J.F.: Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness. In: Proceedings NASA/DoD Workshop on Evolvable Hardware, pp. 54–63. IEEE Computer Society (1999)
8. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. Chapman and Hall/CRC, Boca Raton (2008)
9. McLaughlin, J., Clark, J.A.: Evolving balanced boolean functions with optimal resistance to algebraic and fast algebraic attacks, maximal algebraic degree, and very high nonlinearity. Cryptology ePrint Archive, Report 2013/011 (2013). http://eprint.iacr.org/
10. Millan, W.L., Clark, A.J., Dawson, E.: Heuristic design of cryptographically strong balanced boolean functions. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 489–499. Springer, Heidelberg (1998)
11. Millan, W., Fuller, J., Dawson, E.: New concepts in evolutionary search for boolean functions in cryptology. Computat. Intell. **20**(3), 463–474 (2004)
12. Miller, J., Smith, S.: Redundancy and computational efficiency in cartesian genetic programming. IEEE Trans. Evol. Comput. **10**(2), 167–174 (2006)
13. Miller, J.F.: An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In: Banzhaf, W., Daida, J.M., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M.J., Smith, R.E. (eds.) GECCO, pp. 1135–1142. Morgan Kaufmann (1999)
14. Miller, J.F. (ed.): Cartesian Genetic Programming. Natural Computing Series. Springer, Heidelberg (2011)
15. Miller, J.F., Thomson, P.: Cartesian genetic programming. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) EuroGP 2000. LNCS, vol. 1802, pp. 121–132. Springer, Heidelberg (2000)
16. Miller, J.F., Harding, S.L.: Cartesian Genetic Programming. In: Proceedings of the 10th Annual Conference Companion on Genetic and Evolutionary Computation, GECCO 2008, pp. 2701–2726. ACM, New York (2008)
17. Nyberg, K.: Perfect nonlinear S-Boxes. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 378–386. Springer, Heidelberg (1991)
18. Picek, S., Batina, L., Jakobovic, D.: Evolving DPA-resistant boolean functions. In: Bartz-Beielstein, T., Branke, J., Filipič, B., Smith, J. (eds.) PPSN 2014. LNCS, vol. 8672, pp. 812–821. Springer, Heidelberg (2014)
19. Picek, S., Jakobovic, D., Golub, M.: Evolving cryptographically sound boolean functions. In: GECCO (Companion), pp. 191–192 (2013)
20. Picek, S., Marchiori, E., Batina, L., Jakobovic, D.: Combining evolutionary computation and algebraic constructions to find cryptography-relevant boolean functions. In: Bartz-Beielstein, T., Branke, J., Filipič, B., Smith, J. (eds.) PPSN 2014. LNCS, vol. 8672, pp. 822–831. Springer, Heidelberg (2014)

21. Prouff, E.: DPA Attacks and S-Boxes. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 424–441. Springer, Heidelberg (2005). http://www.iacr.org/cryptodb/archive/2005/FSE/3172/3172.pdf
22. Sarkar, P., Maitra, S.: Nonlinearity Bounds and Constructions of Resilient Boolean Functions. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 515–532. Springer, Heidelberg (2000)