Semantic Mutation Operator for Fast and Efficient Design of Bent Boolean Functions

Jakub Husa[0000−0003−0863−9952] and Lukáš Sekanina[0000−0002−2693−9011]

Faculty of Information Technology, Brno University of Technology,
Božetěchova 2/1, 612 00 Brno, Czech Republic
ihusa@fit.vut.cz

Abstract. Bent functions are a type of Boolean functions with properties that make them useful in cryptography. In this paper we propose a new semantic mutation operator for design of bent Boolean functions via genetic programming. To assess the efficiency of the proposed operator, we compare it to several other commonly used non-semantic mutation operators. Our results show that semantic mutation makes the evolutionary process more efficient, and significantly decreases the number of fitness function evaluations required to find a bent function.

Keywords: Genetic Programming · Semantic Mutation · Bent Boolean Functions.

1 Introduction

Boolean function is any function that takes some number of binary inputs and provides a single binary output. In symmetric cryptography Boolean functions often represent the only nonlinear element of a cipher [1].

To be usable for this purpose, the cryptographic function must fulfill several criteria, one of which is to have a high degree of nonlinearity (NL). The nonlinearity of a functions describes its Hamming distance to the nearest nearest affine function (which is a function that is either linear, or complementary to a linear function). To evaluate the nonlinearity of an n-input Boolean function f its truth table needs to be converted to Walsh spectrum (W_f), using the Fast Walsh-Hadamard transform [1]. Walsh spectrum shows the correlation between the Boolean function and the set of all affine functions, and defines the nonlineaity of a Boolean function as follows:

$$NL(f) = 2^{n-1} - \frac{1}{2} \max_{\boldsymbol{x} \in \mathbb{F}_2^n} (W_f(\boldsymbol{x})) \tag{1}$$

The maximum nonlinearity of a function is limited by the number of its input variables as given by the following equation:

$$NL(f_{bent}) = 2^{n-1} - 2^{\frac{n}{2}-1} \tag{2}$$

This limit is known as the covering radius bound and can only be achieved by functions with an even number of variables. Functions that achieve this limit are known as bent Boolean functions. Notably, the nonlinearity of a function is invariant under all affine transformations, meaning that all of its variables are mutually interchangeable [1].

2 Methodology

To design the bent functions we use a heuristic method known as genetic programming (GP) whose syntactic tree is built out of nodes. Each of these nodes contains three genes, one that specifies the operation that is to be executed, and two that specify its inputs. Each input of a node can point to one of the Boolean function's variables, or to a new node. To avoid bloating of the tree, the maximum tree-depth is limited and the inputs of nodes in the bottom-most layer can only point to variables [4].

The population then undergoes an evolutionary process. Following $(1+\lambda)$ evolution strategy, in each generation the fittest and newest individual is selected to be a parent that generates λ offspring through genetic mutation. Because the goal of this paper is to show the effects of genetic mutation, unlike most GP implementation, we do not use genetic crossover. Then, the fitness of the newly created individuals is evaluated, new best parent is selected, and this cycle repeats until the desired solution is found. In our case, the fitness of an individual is given by its nonlinearity (Eq. 1) and our goal is to create a function that is bent (Eq. 2).

When a gene is mutated, it is changed to a random valid value. This can mean a new randomly selected operation, a different input variable, or a newly generated node. Three commonly used (non-semantic) mutation operators are the uniform, point, and single mutation. Uniform mutation specifies a mutation rate, which is applied to every gene in the chromosome. Point mutation specifies a mutation count, and mutates the specified number of randomly selected genes. Single mutation selects a single active node and then mutates all three of its genes [4].

Our proposed semantic mutation operator is based on algebraic methods of bent function construction, and works as follows. First we randomly chose whether to mutate the structure or the function of the chromosome, and then randomly select one of two possible actions. If mutating the structure, the action can be to either activate or deactivate a new node. When activating a new node, a random input variable is selected and replaced with a new node. When deactivating a node, a random node is replaced with a variable. If mutating the function, the action can be to either shuffle all of the individual's variables or all of its operators.

When shuffling variables, the input variables of all nodes are replaced with new, randomly selected variables, with heavy bias towards selecting variables with fewest uses in order to create a more even distribution, as seen in Figure 1.

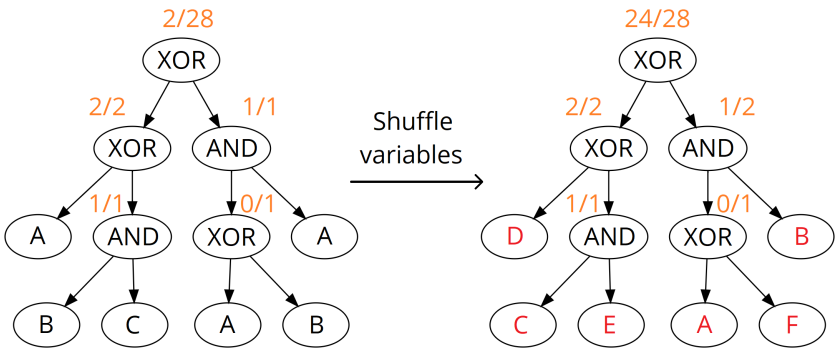

Fig. 1. Shuffle terminals - this action improves fitness by changing the distribution of terminal variables. Numbers above the nodes show their current/maximal nonlinearity.

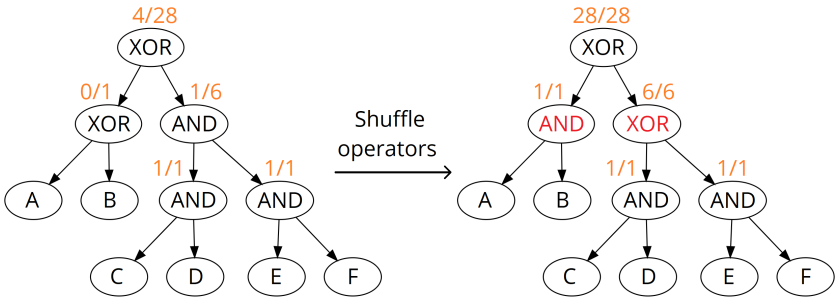

Fig. 2. Shuffle operators - this action improves fitness by changing operators of some of low-fitness nodes, while keeping the nodes with good fitness intact. Numbers above the nodes show their current/maximal nonlinearity.

When shuffling operators, the operations of all nodes are randomized with probability inversely proportional to their fitness (nonlinearity) to avoid disruptive mutations, as seen in Figure 2.

3 Results

In our experiments, we aim to design bent functions with $n = 12$ input variables, and measure the number of evaluations necessary to find the desired function. To make the comparison between the various mutation operators as fair as possible, all setting use the same set of operators {AND, XOR}, which is sufficient and highly effective for designing bent Boolean functions [2]. All settings also use the same tree depth $d = 7$, and population size $\lambda = 4$, which is unusually low for GP, but justified by the lack of genetic crossover, which makes larger populations unnecessary [3]. Based on our preliminary experiments, we select the best mutation rate $mr = 3\%$ used by uniform mutation, and $mc = 4$ used by point mutation.

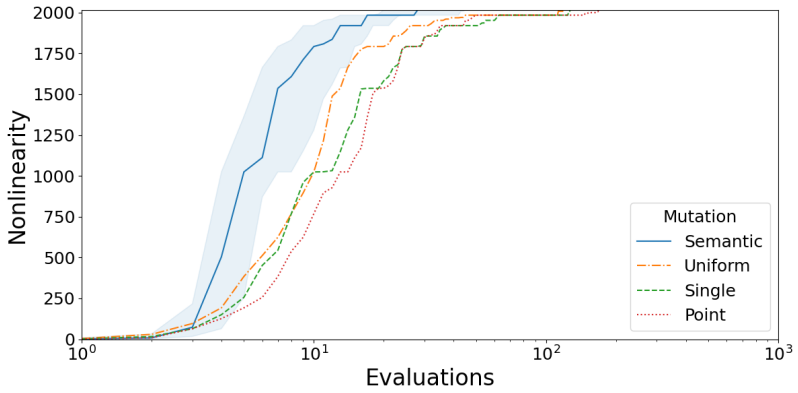

Fig. 3. Convergency curves for various types of mutation. The lines show the medians of 100 independent runs. The shaded area additionally shows the fist and third quartile of the semantic mutation. Note that the X-axis (evaluations) uses a logarithmic scale, and the Y-axis (fitness) terminates at the nonlinearity of an 12-input bent function ($NL(f_{bent}) = 2016$).

Using the four types of mutation, the process requires a median of 109 (semantic), 479 (uniform), 501 (single) or 673 (point) evaluations to find a bent function, as shown in Figure 3, which presents the convergency curves for the four examined types of mutation.

In conclusion, our results show that the proposed semantic mutation operator reduces the median number of fitness function evaluations necessary to find a bent function by 77.2% compared to the second-best operator, and thus significantly improves the efficiency of the evolutionary process, verified by non-parametric Mann-Whitney U-test at $\alpha = 0.05$.

Acknowledgements: This work was supported by the Czech Science Foundation Project 21-13001S and BUT Internal Grant Agency through project FITS-23-8141.

References

1. Carlet, C.: Boolean Functions for Cryptography and Error-Correcting Codes, p. 257397. Encyclopedia of Mathematics and its Applications, Cambridge University Press (2010). https://doi.org/10.1017/CBO9780511780448.011
2. Husa, J., Dobai, R.: Designing bent boolean functions with parallelized linear genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 1825–1832 (2017)
3. Husa, J., Sekanina, L.: Evolving cryptographic boolean functions with minimal multiplicative complexity. In: 2020 IEEE Congress on Evolutionary Computation (CEC). pp. 1–8. IEEE (2020)
4. ONeill, M.: Riccardo Poli, William B. Langdon, Nicholas F. McPhee: a field guide to genetic programming. Springer (2009)