The Multiplicative Complexity of 6-variable Boolean Functions

Çağdaş Çalık · Meltem Sönmez Turan ·
René Peralta

Abstract The multiplicative complexity of a Boolean function is the minimum number of AND gates that are necessary and sufficient to implement the function over the basis (AND, XOR, NOT). Finding the multiplicative complexity of a given function is computationally intractable, even for functions with small number of inputs. Turan et al. [1] showed that n-variable Boolean functions can be implemented with at most $n - 1$ AND gates for $n \leq 5$. A counting argument can be used to show that, for $n \geq 7$, there exist n-variable Boolean functions with multiplicative complexity of at least n. In this work, we propose a method to find the multiplicative complexity of Boolean functions by analyzing circuits with a particular number of AND gates and utilizing the affine equivalence of functions. We use this method to study the multiplicative complexity of 6-variable Boolean functions, and calculate the multiplicative complexities of all $150\,357$ affine equivalence classes. We show that any 6-variable Boolean function can be implemented using at most 6 AND gates. Additionally, we exhibit specific 6-variable Boolean functions which have multiplicative complexity 6.

Çağdaş Çalık
National Institute of Standards and Technology
Tel.: +1-301-975-4024
Fax: +1-301-975-8670

Meltem Sönmez Turan
National Institute of Standards and Technology &
Dakota Consulting Inc.

René Peralta
National Institute of Standards and Technology

1 Introduction

Multiplicative complexity is a complexity measure defined as the minimum number of multiplications (AND gates) that are necessary and sufficient to implement a function with a circuit over the basis (AND, XOR, NOT). In many protocols for multi-party computation (e.g., [2]), fully homomorphic encryption (e.g., [3]), and zero-knowledge proofs (e.g., [4]), processing AND gates is more expensive than processing XOR gates. Moreover, the cost of some of the countermeasures against side channel attacks is related to the number of AND gates in the implementation. For example, the complexity of higher-order masking schemes for S-boxes mainly depends on the *masking complexity*, which is defined as the minimum number of nonlinear field multiplications required to evaluate a polynomial representation of an (n, m)-bit S-box over $\mathbb{F}_{2^n}$ [5].

The multiplicative complexity of a random n-variable Boolean function is at least $2^{n/2} - O(n)$ with high probability [6]. Determining the multiplicative complexity of a given function is computationally intractable, even for small number of variables. It is known that if one-way functions exist, then given a truth table for a Boolean function of n bits, it is not possible to compute the multiplicative complexity in polynomial time in the size of the truth table [7]. Turan and Peralta [1] showed that n-variable Boolean functions can be implemented with at most $n - 1$ AND gates for $n \leq 5$. Using a counting argument, Codish et al. [8] showed that there exist n-variable Boolean functions with multiplicative complexity at least n, for $n \geq 7$. It is also known that the multiplicative complexity of functions having algebraic degree d is at least $d-1$. This is called the *degree bound*. There are very few classes of functions for which lower bounds better than the degree bound are known (see [9]). Prior to our present work, no specific n-variable function had been proven to have multiplicative complexity larger than $n - 1$.

We propose a method to find the multiplicative complexity of Boolean functions. We use this method to study the multiplicative complexity of 6-variable Boolean functions. The method used by Turan and Peralta in [1] uses heuristics that do not provide optimal solutions for 6-variable Boolean functions. Here, the problem of finding the multiplicative complexity distribution of 6-variable Boolean functions is reduced to finding the multiplicative complexities of the 150 357 affine equivalence classes constructed in [10]. The multiplicative complexity of each class is determined by processing all circuits with a particular number of AND gates and then identifying the classes that could be generated by those circuits. We give a complete distribution of the multiplicative complexities of 6-variable Boolean functions and show that they can be implemented with at most 6 AND gates. Our techniques also enable us to exhibit specific 6-variable functions which have multiplicative complexity 6.

The organization of the paper is as follows. Section 2 gives definitions and preliminary information about Boolean functions and affine equivalence relations. Section 3 provides algorithms to construct and evaluate circuits. Section 4 focuses on the multiplicative complexity of 6-bit Boolean functions. Section 5 concludes the paper.

2 Preliminaries on Boolean Functions

Let $\mathbb{F}_2$ be the finite field with two elements and let $\mathbb{F}_2^n$ denote the n-dimensional vector space over $\mathbb{F}_2$. An n-variable Boolean function f is a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. Let B_n be the set of n-variable Boolean functions, clearly $|B_n| = 2^{2^n}$. A Boolean function $f \in B_n$ can be represented uniquely by the multivariate polynomial defined by

$$f(x_1, \ldots, x_n) = \bigoplus_{u \in \mathbb{F}_2^n} a_u x^u \tag{1}$$

where $a_u \in \mathbb{F}_2$ and $x^u = x_1^{u_1} x_2^{u_2} \cdots x_n^{u_n}$ is a *monomial* composed of the variables for which $u_i = 1$. This polynomial is called the *algebraic normal form* (ANF) of f. The *degree* of a Boolean function is the highest number of variables in a monomial for which $a_u = 1$ in its ANF representation. The Boolean functions of the form $f(x) = a_1 x_1 + \ldots + a_n x_n + a_0$, where $a_i \in \mathbb{F}_2$, are called *affine* functions. If $a_0 = 0$, f is called a *linear* function.

The *Walsh transform* of an n-variable Boolean Function f at point $w \in \mathbb{F}_2^n$ is defined as

$$W_f(w) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus w \cdot x}, \tag{2}$$

where $w \cdot x$ is the inner product $w_1 x_1 + \ldots + w_n x_n$. The vector $[W_f(\overline{0}), \ldots, W_f(\overline{2^n - 1})]$ is called the *Walsh spectrum* of f.

The *autocorrelation* of an n-variable Boolean Function f at point $w \in \mathbb{F}_2^n$ is defined as

$$R_f(w) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus f(x \oplus w)}. \tag{3}$$

The vector $[R_f(\overline{0}), \ldots, R_f(\overline{2^n - 1})]$ is called the *autocorrelation spectrum* of f.

Definition 1 An *affine transformation* (A, a, b, c) from f to g in B_n is a mapping of the form $g(x) = f(Ax + a) + b \cdot x + c$, where A is a non-singular $n \times n$ matrix over $\mathbb{F}_2$, and $a, b \in \mathbb{F}_2^n$ and $c \in \mathbb{F}_2$. We call (A, a) the *inner* transformation and (b, c) the *outer* transformation.

Two functions f, g are *affine equivalent* if there exist affine transformations between them. Affine equivalence is an equivalence relation. An algorithm to check whether two functions are equivalent is given in [10]. This algorithm also outputs an affine transformation between the input functions, if one exists. Constructing all equivalence classes is feasible for $n \leq 6$. In 1972, Berlekamp and Welch [11] described the 48 classes on 5-variable Boolean functions. In 1991, Maiorana [12] classified the 150 537 classes on 6-variable Boolean functions. This result was independently verified by Fuller [10] and Braeken et al. [13]. For $n = 7$, Hou [14] determined the number of equivalence classes to be $\approx 2^{65.78}$.

Properties of Boolean functions such as multiplicative complexity, algebraic degree, the set of absolute values in the Walsh spectrum, and the set of absolute

values in the autocorrelation spectrum remain unchanged after applying an affine transformation. These properties are said to be *affine invariant* [15] and they provide a useful tool for showing whether two functions are affine equivalent or not.

3 Boolean Circuits and Topologies

Definition 2 A *Boolean circuit* C with n inputs and 1 output is a directed acyclic graph where the inputs and the gates are the nodes, and the edges correspond to the Boolean-valued *wires*. The *fanin* and *fanout* of a node is the number of wires going in and out of the node, respectively. The nodes with fanin zero are called the *input nodes* and are labeled with an input variable from $X_n = \{x_1, \ldots, x_n\}$. The circuits we consider here have exactly one node with fanout zero, which is called the *output node*. AND gates have fanin two. XOR gates have fanin <u>one or more</u>.

Each Boolean circuit C with n input nodes computes a Boolean function $f \in B_n$. When a Boolean vector $\bar{x} \in \{0,1\}^n$ is fed to the input nodes, the logic gates compute the function where the output node gets the value $f(\bar{x})$. Any Boolean function can be evaluated using the basis $\Omega = \{$AND $(\wedge)$, XOR $(\oplus)$, NOT $(\neg)$ $\}$. Since $\neg x = x \oplus 1$, it is also possible to replace the NOT gates with XOR gates, when the constant **1** is allowed to be used as an input node.

It is not hard to verify that a circuit computing the function f can be put in the following canonical form without changing the number of AND gates (see figure 3b for an example).

1. the circuit output is an XOR gate;
2. outputs of AND gates are inputs to XOR gates, and never inputs to AND gates;
3. inputs of AND gates are outputs of XOR gates;
4. inputs of XOR gates are either inputs to the circuit or outputs of AND gates;
5. there are no negation gates;
6. if $f(0) = 0$ then the constant **1** is not used;
7. if $f(0) = 1$ then the constant **1** is an input to the output XOR gate. This is the only use of the input **1** in the circuit;
8. the set of AND gates is labeled as $A = \{a_1, \ldots, a_k\}$;
9. the AND gates are numbered topologically, with no gate being an ancestor of a lower-numbered gate;
10. XOR gates have fanout one (for intermediate gates) or zero (for the output gate);
11. the set of XOR gates is labeled as $B = \{b_1, \ldots, b_{2k+1}\}$;
12. a_i has inputs b_{2i-1} and b_{2i}, and the output node is b_{2k+1}.

Fig. 1: **Canonical form for a circuit computing the function** f

Unless otherwise specified, we will be assuming that circuits are in canonical form. The notation defined in Figure 2 will be useful in the rest of this paper.

Let S_i be the set of AND gates that are inputs to the ith XOR gate $\mathtt{b}_i$. Let L_i be the set of inputs to $\mathtt{b}_i$ not in S_i. Note that the elements of L_i are either input nodes or the constant $\mathbf{1}$.

Fig. 2: **The L and S sets**

Notation: Given a set V of nodes, let $\mathcal{X}_V$ denote the Boolean function computed as $\bigoplus_{v \in V} v$. [1] The output of the i-th XOR gate is $F_{\mathtt{b}_i} = \mathcal{X}_{L_i} \oplus \mathcal{X}_{S_i}$, and the output of the i-th AND gate is

$$F_{\mathtt{a}_i} = (\mathcal{X}_{L_{2i-1}} \oplus \mathcal{X}_{S_{2i-1}}) \wedge (\mathcal{X}_{L_{2i}} \oplus \mathcal{X}_{S_{2i}}). \tag{4}$$

Definition 3 Given a circuit, the ordered list $(L_1, \ldots, L_{2k+1}, S_1, \ldots, S_{2k+1})$ is called the *trace of the circuit*. The ordered list $[(S_1, S_2), (S_3, S_4) \ldots, (S_{2k-1}, S_{2k})]$ shows the relations between the AND gates, and is called the *topology* of the circuit. The ordered list $(L_1, \ldots, L_{2k+1})$ shows the linear inputs to the XOR gates, and is called the *input to the topology*. S_{2k+1} is called the *output mask*.

Note that the trace of a circuit does not contain all the information about the circuit. So it is not possible to reconstruct a circuit from its trace.

By using the gate numbering given in Figure 1, the topology of the circuit can be expressed in graphical form. We use the procedure of Algorithm 1.

---

input : $\{(S_{2i-1}, S_{2i})\}_{1 \leq i \leq k}$
output: A digraph

1 for *i from 1 to k* **do**
2   add node a_i to graph;
3   if S_{2i-1} is not empty, create an XOR node with output to a_i and inputs the elements of S_{2i-1};
4   repeat the previous step with S_{2i};
5 end
6 bypass XOR gates with fan-in 1.

Algorithm 1: Mapping topologies to directed graphs

Our depictions of topologies will not show the direction of edges (they point down) or the labels of gates (we are only interested in the structure of each graph). Table 1 contains topologies with 3 AND gates, along with their graphical representation.

Example 1 The function $f = x_1 x_2 x_3 + x_1 x_3 + x_1 x_4 + x_2 x_3 + x_4$ has multiplicative complexity 2, as is clear from $f = x_1(x_2 x_3 + x_3 + x_4) + x_2 x_3 + x_4$. A circuit in canonical form that computes f is depicted in figure 3b. The trace for that circuit is

$$(\{x_3\}, \{x_2\}, \{x_3, x_4\}, \{x_1\}, \{x_4\}, \emptyset, \emptyset, \{a_1\}, \emptyset, \{a_1, a_2\}).$$

---

[1] We abuse notation here, identifying a node with the function it computes.

| k	Topology	Graph Representation
1	$[(\emptyset, \emptyset)]$	
2	$[(\emptyset, \emptyset), (\emptyset, \emptyset)]$	
2	$[(\emptyset, \emptyset), (\emptyset, \{a_1\})]$	
3	$[(\emptyset, \emptyset), (\emptyset, \emptyset), (\emptyset, \emptyset)]$	
3	$[(\emptyset, \emptyset), (\emptyset, \emptyset), (\emptyset, \{a_2\})]$	
3	$[(\emptyset, \emptyset), (\emptyset, \emptyset), (\emptyset, \{a_1, a_2\})]$	
3	$[(\emptyset, \emptyset), (\emptyset, \emptyset), (\{a_1\}, \{a_2\})]$	
3	$[(\emptyset, \emptyset), (\emptyset, \{a_1\}), (\emptyset, \{a_1\})]$	
3	$[(\emptyset, \emptyset), (\emptyset, \{a_1\}), (\emptyset, \{a_2\})]$	
3	$[(\emptyset, \emptyset), (\emptyset, \{a_1\}), (\emptyset, \{a_1, a_2\})]$	
3	$[(\emptyset, \emptyset), (\emptyset, \{a_1\}), (\{a_1\}, \{a_2\})]$	

Table 1: List of topologies with up to 3 AND gates

The topology of the circuit is $[(\emptyset, \emptyset), (\{a_1\}, \emptyset)]$. The input to the topology is $(\{x_3\}, \{x_2\}, \{x_3, x_4\}, \{x_1\}, \{x_4\})$, and the output mask is $\{a_1, a_2\}$. The graphical representations of the circuit and its topology are given in Figure 3.

Definition 4 Given a circuit in canonical form, a numbering of the AND gates is a *proper numbering*, if no gate is an ancestor of a lower-numbered gate (i.e. if the numbering does not violate topological ordering).

Definition 5 Two topologies are said to be *isomorphic* if one results from a proper re-numbering of the AND gates of the other.

Definition 6 A Boolean function f is *computable* by a topology T if it is computable by a circuit whose topology is T. The set of Boolean functions that are computable by a topology T is denoted $B(T)$.

Clearly, if two topologies are isomorphic, then the sets of functions computable by each are the same. We state this as a proposition.
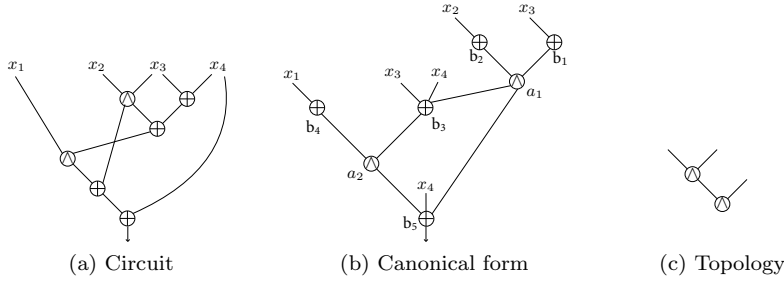
Fig. 3: Circuit and topology computing f.

Proposition 1 *If topologies T and T' are isomorphic, then $B(T) = B(T')$.*

3.1 Evaluating Topologies

Without loss of generality, the remainder of this paper only considers functions f for which $f(0) = 0$. These functions have negation-free circuits that are optimum with respect to multiplicative complexity. Any function for which $f(0) = 1$ is of the form $f(\bar{x}) = g(\bar{x}) + 1$ where $g(0) = 0$.

The aim of this section is to construct the set of Boolean functions that are computable by a topology $T = [(S_1, S_2), \ldots, (S_{2k-1}, S_{2k})]$. The set $B(T)$ can be obtained by exhaustively evaluating the following family of circuit traces

$$(L_1^*, \ldots, L_{2k+1}^*, S_1, \ldots, S_k, S_{2k+1}^*),$$

where L_i^* is any subset of $\{\mathtt{x}_1, \ldots, \mathtt{x}_n\}$ and S_{2k+1}^* is any subset of $\{\mathtt{a}_1, \ldots, \mathtt{a}_k\}$. However, going over all possible L_i^*'s for $i = 1, \ldots, 2k + 1$ and S_{2k+1}^* quickly becomes inefficient, since there are $(2^n)^{2k+1} \times 2^k = 2^{2kn+k+n}$ possible choices for these sets (e.g., there are 2^{71} choices for a topology with 5 AND gates when $n = 6$).

Theorem 1 *Let $f \in B_n$ be computable by a topology T. If f' is affine equivalent to f, then f' is also computable by T.*

Proof Let $C = (L_1, \ldots, L_{2k+1}, S_1, \ldots, S_{2k+1})$ be the trace of a circuit with topology T that computes f. If f' is affine equivalent to f, then there exists an affine transformation (A, a, b, c) satisfying $f'(x) = f(Ax + a) + bx + c$. The circuit generated by applying the inner transformation $Ax + a$ to the inputs $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, and adding outer transformation $bx + c$ to L_{2k+1} constructs f'. Since the topology of the circuit is not affected by the affine transformation, f' is computable by T. □

Theorem 1 implies that either all or none of the functions in an equivalence class are computable by a topology T. We say that the equivalence class of $f \in B_n$ is *computable by a topology T*, if f is computable by T. Hence, in order to construct $B(T)$, we may construct the list of equivalence classes that are computable by T.

Corollary 1 *Consider two circuits with traces*

$$(L_1, \ldots, L_{2k+1}, S_1, \ldots, S_k, S_{2k+1}) \text{ and}$$

$$(L'_1, \ldots, L'_{2k+1}, S_1, \ldots, S_k, S_{2k+1}).$$

If there exists an inner transformation (A, a) that transforms $\mathcal{X}_{L_i} \to \mathcal{X}_{L'_i}$ for $i = 1, \ldots, 2k$, then the circuits compute affine equivalent functions.

To see why the corollary holds, note that the outer transformation (b, c) affecting $\mathcal{X}_{L'_{2k+1}}$ is a specific case of an affine transformation and hence does not change the equivalence class of a function.

Corollary 1 implies that it is not necessary to evaluate an input to a circuit if a respectively affine equivalent input has already been evaluated, because the computed functions will be from the same equivalence class. Thus, the problem is to find a maximal set of inputs $(L_1, \ldots, L_{2k})$ such that no two are affine equivalent with respect to an inner transformation. This problem corresponds to enumerating all m-dimensional subspaces in a $2k$-dimensional vector space, where m varies depending on k. The number of subspaces of dimension m in a vector space of dimension $2k$ over a finite field of 2 elements is equal to the Gaussian binomial coefficient $\binom{2k}{m}_2$ [16]. This is approximately 2^{26} inputs for $m = 6$, $k = 5$. The algorithm to construct the subspaces is given in [17]. In its simplest form, to generate a sufficient set of inputs of dimension m, for each component in an input we either introduce a new variable, or choose a linear combination of the previously used variables, by taking into account that the exact number of variables to be used is m.

Example 2 Consider the vector space consisting of all linear functions on variables x_1, x_2, x_3, x_4. Denote by (a_1, a_2, a_3, a_4) the subspace generated by all linear functions $\sum_{i=1}^{4} a_i x_i$ where $a_i \in \{0, 1\}$. A maximal set of 3-dimensional subspaces in a 4-dimensional vector space ($m = 3$, $2k = 4$) is as follows:

$$
\begin{array}{lll}
(a_1, a_2, a_3, 0), & (a_1, a_2, a_3, a_1), & (a_1, a_2, a_3, a_2), \\
(a_1, a_2, a_3, a_1 + a_2), & (a_1, a_2, a_3, a_3), & (a_1, a_2, a_3, a_1 + a_3), \\
(a_1, a_2, a_3, a_2 + a_3), & (a_1, a_2, a_3, a_1 + a_2 + a_3), & (a_1, a_2, 0, a_3), \\
(a_1, a_2, a_1, a_3), & (a_1, a_2, a_2, a_3), & (a_1, a_2, a_1 + a_2, a_3), \\
(a_1, 0, a_2, a_3), & (a_1, a_1, a_2, a_3), & (0, a_1, a_2, a_3)
\end{array}
$$

Notation: $I_{n,k}$ denotes the set of subspaces of $\{0, 1\}^{2k}$ of dimension at most $min\{n, 2k\}$

3.2 Constructing Topologies

All topologies with k AND gates have the form $[(S_1, S_2), \ldots, (S_{2k-1}, S_{2k})]$, where $S_{2i-1}, S_{2i} \subseteq \{a_1, \ldots, a_{i-1}\}$. The sets S_{2i-1} and S_{2i} can take 2^{i-1} different values, so the total number of topologies with k AND gates is

$$\prod_{i=1}^{k} (2^{i-1})^2 = 2^{\sum_{i=1}^{k} 2(i-1)} = 2^{k^2 - k}.$$

However, this formula counts many topologies redundantly in the sense that there are many topologies computing the same set of Boolean functions. In addition to isomorphic topologies, redundant topologies can result from some transformations on pairs (S_{2k-1}, S_{2k}) corresponding to the AND gates that define the output of AND gate a_i (see equation 4). Next, we derive some of these transformations.

Notation: Given two sets $S_i, S_j \subseteq \{\mathtt{a}_1, \ldots, \mathtt{a}_k\}$ of AND gates, we denote by $S_i \oplus S_j$ the symmetric difference of S_i and S_j, i.e., the set of elements that are contained in S_i or S_j but not in both.

Remark 1 Let $h = (f + l_1)(g + l_2)$ with $f, g, l_1, l_2 \in \mathcal{B}_n$. Let $l_3 = 1 + l_1 + l_2$. Then it follows that

$$h = (f + l_1)(f + g + l_3)$$
$$h = (f + g + l_3)(g + l_2)$$

Proposition 2 *Let $T = [(S_1, S_2), \ldots, (S_{2k-1}, S_{2k})]$ be a topology. Let T' be the topology formed by replacing (S_{2i-1}, S_{2i}) by $(S_{2i-1}, S_{2i-1} \oplus S_{2i})$, and let T'' be the topology formed by replacing (S_{2i-1}, S_{2i}) by $(S_{2i-1} \oplus S_{2i}, S_{2i})$. Then $B(T) = B(T') = B(T'')$.*

Proof Let $F_{\mathtt{a}_i} = (\mathcal{X}_{L_{2i-1}} \oplus \mathcal{X}_{S_{2i-1}}) \wedge (\mathcal{X}_{L_{2i}} \oplus \mathcal{X}_{S_{2i}})$ be a function generated by AND gate a_i. Then,

$$F_{\mathtt{a}_i'} = (\mathcal{X}_{L_{2i-1}} \oplus \mathcal{X}_{S_{2i-1}}) \wedge (1 \oplus \mathcal{X}_{L_{2i}} \oplus \mathcal{X}_{L_{2i-1}} \oplus \mathcal{X}_{S_{2i}} \oplus \mathcal{X}_{S_{2i-1}})$$

and

$$F_{\mathtt{a}_i''} = (1 \oplus \mathcal{X}_{L_{2i}} \oplus \mathcal{X}_{L_{2i-1}} \oplus \mathcal{X}_{S_{2i}} \oplus \mathcal{X}_{S_{2i-1}}) \wedge (\mathcal{X}_{L_{2i}} \oplus \mathcal{X}_{S_{2i}})$$

are the same functions by Remark 1. □

Proposition 2 implies that for any pair (S_{2i-1}, S_{2i}) in a topology, any choice of two input pairs from the set $\{S_{2i-1}, S_{2i}, S_{2i-1} \oplus S_{2i}\}$ for AND gate a_i will not change the set of functions computable by the topology. This motivates the following definition.

Definition 7 The following six pairs are said to be equivalent.

$$(S_{2i-1}, S_{2i})$$
$$(S_{2i}, S_{2i-1})$$
$$(S_{2i-1}, S_{2i-1} \oplus S_{2i})$$
$$(S_{2i-1} \oplus S_{2i}, S_{2i-1})$$
$$(S_{2i-1} \oplus S_{2i}, S_{2i})$$
$$(S_{2i}, S_{2i-1} \oplus S_{2i})$$

From now on we will identify the pair (S_{2i-1}, S_{2i}) with AND gate a_i where doing so will not cause confusion.

We represent a set $S \subseteq \{a_1, \ldots, a_k\}$ by a k-bit mask $(v_1, \ldots, v_k)$, each v_i denoting the existence of a_i in S. Since each gate has two sets of inputs, (S_{2i-1}, S_{2i}) is represented by a $2k$-bit vector.

Definition 8 The *minimal representation* of an AND gate $a_i = (S_{2i-1}, S_{2i})$ in a topology is the lexicographically smallest among the set of equivalent gates listed in Definition 7.

We use Algorithm 2 to construct the set of topologies with k AND gates. The exhaustive list of topologies having up to 3 AND gates after removing the redundant ones is shown in Table 1.

---

    input : k
    output: Set of topologies with k AND gates, T_k

1 $T_0 = \emptyset$;
2   **for** *(i = 1 to k)* **do**
3 $S \leftarrow \emptyset$;
4   **for** *each topology t in T_{k-1}* **do**
5   **for** *all choices of (S_{2k-1}, S_{2k})* **do**
6   If (S_{2k-1}, S_{2k}) is minimal representation;
7 $t' \leftarrow$ add $a_k = (S_{2k-1}, S_{2k})$ to t;
8   If t' is not isomorphic to any of the topologies in S, $S \leftarrow S \cup t'$;
9   **end**
10 $T_k \leftarrow S$.
11   **end**
12 **end**

Algorithm 2: Iterative construction of topologies

3.3 Finding the Multiplicative Complexity of a Boolean Function

In this section, we propose an algorithm to find the multiplicative complexity of a given Boolean function. Let $f \in B_n$ be a Boolean function with degree d. It is known that the multiplicative complexity of f is at least $d - 1$. The first step of the algorithm is to use Algorithm 2 to construct the topologies with $k = d - 1$ AND gates. Then, the set of $\binom{2k}{m}_2$ inputs for the topologies are constructed as explained in Section 3.1. If f is computable by any of the topologies, the algorithm outputs k, else k is incremented by one, and a new

k	1	2	3	4	5	6
$|T_k|$	1	2	8	84	3 170	475 248

Table 2: The number of topologies up to 6 AND gates

```
     input  : Boolean function f
     output: Multiplicative complexity of f
  1  d ← degree of f;
  2  k ← d − 1;
  3  while (true) do
  4  │    Construct T_k using Algorithm 2;
  5  │    Construct I_{n,k};
  6  │    for each topology t in T_k do
  7  │    │    for each input (L_1, …, L_{2k}) in I_{n,k} do
  8  │    │    │    for each S_{2k+1} ⊆ {a_1, …, a_k} do
  9  │    │    │    │    C ← (L_1, …, L_{2k}, 0, S_1, …, S_{2k+1});
 10  │    │    │    │    f' ← the Boolean function that C computes;
 11  │    │    │    │    if f' is affine equivalent to f then
 12  │    │    │    │    │    return k;
 13  │    │    │    │    end
 14  │    │    │    end
 15  │    │    end
 16  │    end
 17  │    k = k + 1;
 18  end
```

Algorithm 3: Finding the multiplicative complexity of a Boolean function

set of topologies is evaluated, until f is computable by one of the topologies, as given in Algorithm 3.

The algorithm is practical when the number k of AND gates is small. The algorithm can also be used for providing lower bounds on the multiplicative complexity of a given function, when it does not find a solution up to a particular value of k, establishing a lower bound of $k + 1$.

4 Multiplicative Complexity of 6-bit Boolean Functions

In this section, our aim is to find the multiplicative complexities of all Boolean functions in 6-variables. The number of affine equivalence classes in $\mathcal{B}_6$ is known to be 150 357. In this work, we use the representative functions computed in [10].

Our method can be summarized as follows: We iteratively construct and evaluate topologies until at least one function from each affine equivalence class is computed (See Algorithm 4). The algorithm identifies the set of affine equivalence classes generated by a topology, by going over all linear inputs and the output masks. Given a topology, a function is computed by providing a set of linear function inputs to this topology and combining the outputs of the AND gates with an output mask. For each function that is computed from a topology, the algorithm checks whether the function is affine equivalent to a Boolean function from the input set. If so, it means that this is the first time a function from this equivalence class is computed and the multiplicative complexity of this class is assigned the value k, and the function is removed

from the set of input functions. The construction of topologies, described in Section 3.2, is independent of the number of variables the topology will be evaluated for. The input generation and determining the affine equivalence classes of functions is described in Sections 4.2 and 4.1, respectively.

It should be noted that an equivalence class may be generated by distinct topologies having a different number of AND gates. An equivalence class is assigned a multiplicative complexity value k, if it was generated by a topology having k AND gates and was not previously generated with a topology that had less than k AND gates. This is guaranteed by processing the topologies in order with respect to the number of AND gates.

---

     input : n, Set of representatives of $\mathcal{B}_6$ $\{f_1, \ldots, f_{150357}\}$
     output: Multiplicative complexity of $MC[f_i]$

  1 $k = 1$;
  2 $S \leftarrow \{f_1, \ldots, f_{150357}\}$;
  3   **while** $(S \neq \emptyset)$ **do**
  4   Construct T_k using Algorithm 2;
  5   Construct $I_{n,k}$;
  6   **for** *each topology* $[(S_1, S_2), \ldots, (S_{2k-1}, S_{2k})]$ *in* T_k **do**
  7   **for** *each input* $(L_1, \ldots, L_{2k})$ *in* $I_{n,k}$ **do**
  8   **for** *each* $S_{2k+1} \subseteq A$ **do**
  9 $C \leftarrow (L_1, \ldots, L_{2k}, 0, S_1, \ldots, S_{2k+1})$;
 10 $f' \leftarrow$ the Boolean function that C computes;
 11   **if** *f' is affine equivalent an $f_i \in S$* **then**
 12 $MC[f_i] = k$;
 13   Remove f_i from S;
 14   **if** $S = \emptyset$ **then**
 15   **return** $MC[f_1], \ldots, MC[f_{150357}]$;
 16   **end**
 17   **end**
 18   **end**
 19   **end**
 20   **end**
 21 $k = k + 1$;
 22 **end**

Algorithm 4: Finding Multiplicative Complexity of 6-bit Boolean Functions

---

4.1 Determining the Equivalence Classes of Functions

An important step in finding the set of equivalence classes that is computable by a topology is determining the equivalence class of a specific function that is generated from that topology. It is crucial that this process be performed efficiently, since it will be repeated for each set of inputs supplied to a topology. An equivalence class is represented by a representative function from that class whose selection can be arbitrary.

Deciding on whether or not two given functions are affine equivalent is done by making use of the affine transformation invariant properties of functions, i.e., algebraic degree, absolute Walsh spectrum distribution, and absolute autocorrelation spectrum distribution. We call the values these three metrics take the *signature* of a function. For $n \leq 5$, the signature of a function is sufficient to determine its class because the signatures of functions in different classes are different. However, for $n = 6$ there are a total number of 30 883 distinct signatures, which is less than the number of equivalence classes, implying that some equivalence classes have the same signature. Indeed, 17 234 of the signatures belong to a unique class, whereas the remaining 13 649 signatures are shared by two or more classes, with the maximum number of classes having the same signature being 564.

Suppose we want to determine the equivalence class of a function f. That is, we want to know which of the 150 357 representatives is affine equivalent to f. If the signature of f is among the 17 234 signatures that uniquely identifies its class, then the representative is the one associated with the signature. Otherwise, it is the case that f is affine equivalent to one of the representatives, say $f_1, ..., \ldots, f_m$, $m \geq 2$, whose signatures are equal to that of f's. Then, we use the methods described in [10], namely the local connectivity and indicator functions to eliminate the representatives that could not be affine equivalent to f until we're left with a single candidate, which will be the representative that f is affine equivalent to.

4.2 Evaluating Topologies

As described in Section 3.1, for a given n, the inputs to a topology with k AND gates are the functions that correspond to the subspaces of dimension m in a vector space of dimension $2k$, where $1 \leq m \leq \min(2k, n)$. In [1], it was shown that any Boolean function on n-variables has multiplicative complexity at most $n - 1$ for $n \leq 5$. This implies that if we want to evaluate a topology with $k \leq 5$ AND gates, the functions generated by this topology must be affine equivalent to a function that has at most $k + 1$ variables. This observation lets us reduce the number of dimensions m that must be considered when generating the inputs. Table 3 provides the list of dimensions m in a vector space of dimension $2k$ for $n = 6$.

Table 3: Size of the input combinations for evaluating topologies.

k	$(m, 2k)$	Number of input vectors
1	(2,2)	1
2	(3,4), (4,4)	16
3	(4,6), (5,6), (6,6)	715
4	(5,8),(6,8)	107 950
5	(6,10)	53 743 987
6	(6,12)	230 674 393 235

4.3 Results

The algorithms for constructing and evaluating the topologies have been implemented in the C++ programming language. The construction of topologies up to 5 AND gates took less than a minute on a standard desktop PC, whereas topologies with 6 AND gates could be generated in approximately one day. The task of identifying the equivalence classes generated by the topologies demanded more computation due to the large number of input combinations to be evaluated. We exploited the inherent parallelism of this process by running it on a cluster, with each node processing a certain subset of the topologies. The nodes in the cluster had dual Intel Xeon E5-2630 v3 processors[2] (3.20 GHz, 8 cores) and 64 GB memory. The total computation took 38 422 core hours.

Table 4 shows the results of the computation. The number of functions having a particular multiplicative complexity was calculated by adding the total number of functions in each equivalence class. After running Algorithm 4 up to 5 AND gates, there were still 931 equivalence classes on 6-variables that could not be generated, implying that their multiplicative complexity should be at least 6. The task of evaluating all topologies having 6 AND gates with all possible inputs was not feasible due to the large number of topologies and input combinations. Therefore, we randomly picked a subset of those topologies, and computed the remaining set of equivalence classes. This process completed in the order of hours as it was sufficient to generate one function from each class to conclude that these functions had multiplicative complexity 6.

The results can be accessed online at [18], which contains one representative from each affine equivalence class, along with a circuit for that function with a minimum number of multiplications.

Table 4: Multiplicative complexity distribution of the equivalence classes and functions for $n = 6$.

MC	#classes	#functions	$\log_2(\#functions)$
0	1	128	7.00
1	1	83 328	16.34
2	3	73 757 184	26.13
3	24	281 721 079 808	38.03
4	914	7 944 756 861 878 272	52.81
5	148 483	18 344 082 080 963 133 440	63.99
6	931	94 716 954 089 619 456	56.39

---

[2] Commercial equipment and software referred to in this paper are identified for informational purposes only, and does not imply recommendation of or endorsement by the National Institute of Standards and Technology, nor does it imply that the products so identified are necessarily the best available for the purpose.

Among the set of functions having multiplicative complexity 6, we list below all representatives with degree 4:

$f_1 = x_3x_5 + x_3x_4 + x_2x_4 + x_2x_3x_6 + x_2x_3x_4x_5 + x_1x_6 + x_1x_5 + x_1x_4x_5 + x_1x_3 + x_1x_3x_4$
$\qquad + x_1x_3x_4x_6 + x_1x_2x_5x_6$

$f_2 = x_4x_5 + x_3x_4x_5 + x_2x_5 + x_2x_4 + x_2x_4x_6 + x_1x_5x_6 + x_1x_4 + x_1x_3 + x_1x_2x_4x_5$
$\qquad + x_1x_2x_3x_6$

$f_3 = x_3x_5 + x_3x_4 + x_2x_4 + x_2x_3x_6 + x_2x_3x_4x_5 + x_1x_4 + x_1x_4x_5 + x_1x_3 + x_1x_3x_4$
$\qquad + x_1x_3x_4x_6 + x_1x_2x_5x_6$

$f_4 = x_3x_5 + x_3x_4 + x_2x_4 + x_2x_3 + x_2x_3x_6 + x_2x_3x_4x_5 + x_1x_6 + x_1x_5 + x_1x_4x_5$
$\qquad + x_1x_3x_4 + x_1x_3x_4x_6 + x_1x_2 + x_1x_2x_5x_6$

$f_5 = x_4x_6 + x_4x_5 + x_3x_5 + x_2x_3x_6 + x_2x_3x_4x_5 + x_1x_5 + x_1x_3 + x_1x_3x_5 + x_1x_3x_4$
$\qquad + x_1x_3x_4x_6 + x_1x_2 + x_1x_2x_5x_6 + x_1x_2x_4$

5 Conclusion

We explored the multiplicative complexity of 6-variable Boolean functions and showed that they can be implemented using at most 6 AND gates. More specifically, we computed the distribution of affine equivalence classes and the number of Boolean functions with multiplicative complexity k for $k = 0, \ldots, 6$. We were able to exhibit specific 6-variable functions which have multiplicative complexity 6.

6 Acknowledgments

References

1. Meltem Turan Sönmez and René Peralta. *The Multiplicative Complexity of Boolean Functions on Four and Five Variables*, pages 21–33. Springer International Publishing, Cham, 2015.
2. Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498. Springer, 2008.
3. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 309–325. ACM, 2012.

4. Joan Boyar, Ivan Damgård, and René Peralta. Short non-interactive cryptographic proofs. *J. Cryptology*, 13(4):449–472, 2000.

5. Claude Carlet, Louis Goubin, Emmanuel Prouff, Michaël Quisquater, and Matthieu Rivain. Higher-order masking schemes for s-boxes. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 366–384. Springer, 2012.

6. Joan Boyar, René Peralta, and Denis Pochuev. On the multiplicative complexity of Boolean functions over the basis $(\wedge, \oplus, 1)$. *Theor. Comput. Sci.*, 235(1):43–57, 2000.

7. Magnus Gausdal Find. On the complexity of computing two nonlinearity measures. In *Computer Science - Theory and Applications - 9th International Computer Science Symposium in Russia, CSR 2014, Moscow, Russia, June 7-11, 2014. Proceedings*, pages 167–175, 2014.

8. Michael Codish, Luís Cruz-Filipe, Michael Frank, and Peter Schneider-Kamp. When six gates are not enough. *CoRR*, abs/1508.05737, 2015.

9. J. Boyar and R. Peralta. Tight bounds for the multiplicative complexity of symmetric functions. *Theoretical Computer Science*, 396(1-3):223–246, 2008.

10. Joanne Elizabeth Fuller. *Analysis of affine equivalent boolean functions for cryptography*. PhD thesis, Queensland University of Technology, 2003.

11. Elwyn R. Berlekamp and Lloyd R. Welch. Weight distributions of the cosets of the (32, 6) Reed-Muller code. *IEEE Transactions on Information Theory*, 18(1):203–207, 1972.

12. James A. Maiorana. A classification of the cosets of the Reed-Muller code $\mathcal{R}(1,6)$. *Math. Comput.*, 57(195):403–414, 1991.

13. An Braeken, Yuri Borissov, Svetla Nikova, and Bart Preneel. *Classification of Boolean Functions of 6 Variables or Less with Respect to Some Cryptographic Properties*, pages 324–334. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

14. Xiang-Dong Hou. AGL (m, 2) acting on R (r, m)/R (s, m). *Journal of Algebra*, 171(3):927–938, 1995.

15. Claude Carlet. Boolean functions for cryptography and error correcting codes. In Y. Crama and P.L. Hammer, editors, *Boolean Models and Methods in Mathematics, Computer Science and Engineering*, chapter 8. Cambridge Univ. Press, Cambridge, UK, 2010.

16. Donald E Knuth. Subspaces, subsets, and partitions. *Journal of Combinatorial Theory, Series A*, 10(2):178 – 180, 1971.

17. Jay Goldman and Gian-Carlo Rota. On the foundations of combinatorial theory iv finite vector spaces and eulerian generating functions. *Studies in Applied Mathematics*, 49(3):239–258, 1970.

18. NIST Computer Security Division. *SLP's for 6-variable predicates*, https://github.com/usnistgov/Circuits/tree/master/slp.