Multiplicative Complexity of XOR Based Regular Functions

Anna Bernasconi, Stelvio Cimato, Valentina Ciriani, and Maria Chiara Molteni

Abstract—XOR-AND Graphs (XAGs) are an enrichment of the classical AND-Inverter Graphs (AIGs) with XOR nodes. In particular, XAGs are networks composed by ANDs, XORs, and inverters. Besides several emerging technologies applications, XAGs are often exploited in cryptography-related applications based on the multiplicative complexity of a Boolean function. The multiplicative complexity of a function is the minimum number of AND gates (i.e., multiplications) that are sufficient to represent the function over the basis {AND, XOR, NOT}. In fact, the minimization of the number of AND gates is important for high-level cryptography protocols such as secure multiparty computation, where processing AND gates is more expensive than processing XOR gates. Moreover, it is an indicator of the degree of vulnerability of the circuit, as a small number of AND gates corresponds to a high vulnerability to algebraic attacks. In this paper we study the multiplicative complexity of Boolean functions characterized by two particular regularities, called autosymmetry and D-reducibility. Moreover, we exploit these regularities for decreasing the number of AND nodes in XAGs. The experimental results validate the proposed approaches.

Index Terms—Logic synthesis, Multiplicative complexity, XOR-AND Graphs, Regular Boolean functions.

1 Introduction

Nowadays, AND-Inverter Graphs (AIGs) are one of the most studied and exploited data structure in Logic Synthesis. An AIG is a directed acyclic graph of 2-input AND nodes, with possibly inverted edges, that represents a Boolean function. Recently, AND-Inverter Graph logic networks, implemented in the academic state-of-the-art logic synthesis tool ABC [19], have evolved into the new *XOR-AND Graph* (XAG) representations [26], [27], [36], [37]. An XAG is an AIG enriched with 2-input XOR nodes.

The introduction of XOR nodes in AIGs is mainly due to two different requirements. On one hand, several proposed emerging technologies exploit XOR gates [13], [27], [35], [38]. On the other hand, the growing relevance of cryptography-related applications has revived the interest in XOR gates [22], [26], [27], [33], [36], [37], [38]. For example, in high-level cryptography protocols such as secure multiparty computation, processing XOR gates is convenient since their evaluation is possible without any communication cost [28]. In this context, it is thus important to consider network representations that assume XOR gates explicitly. Thus, cryptographic applications often consider Boolean functions represented over the basis {AND, XOR, NOT}.

In the context of logic synthesis for emerging technologies, the minimization of an XAG mainly aims at reducing the number of AND/XOR nodes. On the contrary, in the cryptography-related applications, we are typically interested in reducing the number of AND nodes, only. For example, for secure multiparty computation AND nodes are the only nodes in an XAG with a communication cost (observe that the NOT operation can be implemented with a XOR). In this paper, we are interested in this second type of XAG applications, where the minimization cost depends only on the number of AND nodes. Therefore, our main aim is the minimization of the number of AND gates in an XAG [22], [36], [37], [38]. The number of AND nodes in an XAG implementation of a function is called the *multiplicative complexity of the XAG*, while the minimum number of ANDs that are sufficient to represent a function with an XAG defines the *multiplicative complexity of the function*. This complexity measure plays a crucial role in cryptography-related applications: not only it is important for high-level cryptography protocols, but it also represents an indicator of the degree of vulnerability of the circuit, as a small number of AND nodes corresponds to a high vulnerability to algebraic attacks.

In this paper, we study the multiplicative complexity of two classes of Boolean functions exhibiting regularities that can be expressed using the XOR operation: 1) *autosymmetric* functions [3], [10], [11], [30]; and 2) *D-reducible* functions [6]. Our motivation is based on the following two observations. First, we are interested in functions' regularities that allow to fully exploit XOR gates. Second, the presence of XOR gates often guarantees more compact representations [4], [12], [13], [26], [27], [36], [37], [38].

Intuitively, a Boolean function f over n variables is *k-autosymmetric* if it can be projected onto a smaller function f_k that depends on $n-k$ variables only, and has a smaller on-set. The XOR operation comes into play as the new $n-k$ variables are XOR combinations of some of the original ones. The new function f_k is called *restriction* of f and can be identified in time polynomial in the dimension of some standard representations of the input instance. For example, the restriction f_k can be computed in time polynomial in

- *A. Bernasconi is with the Department of Computer Science, Università di Pisa, Italy.*
  E-mail: anna.bernasconi@unipi.it
- *S. Cimato, V. Ciriani, and M.C. Molteni are with the Department of Computer Science, Università degli Studi di Milano, Italy.*
  E-mail: {stelvio.cimato, valentina.ciriani, maria.molteni}@unimi.it

Manuscript received ...; revised ...

the size (i.e., the number of nodes) of the initial Reduced Ordered Binary Decision Diagram (ROBDD) representation of the original function f, as proved in [10]. Obviously, note that the dimension of the ROBDD of f might be exponential in the number n of input variables. Observe that, even if autosymmetric functions depend in general on all their n input variables, they can be studied in a $n-k$ dimensional space; i.e., they are in general non-degenerated, whereas all degenerated functions are autosymmetric.

The interest in autosymmetric functions in this context is motivated by the fact that an XAG representation of an autosymmetric function f can be easily obtained composing an XAG for the restriction f_k with an additional layer of XOR nodes (as discussed in Sections 2.3 and 3). As a result, the multiplicative complexity of f can be estimated from the multiplicative complexity of the restriction f_k. Actually, a stronger result holds: f and f_k have exactly the same multiplicative complexity. This result is a consequence of the fact that f and its restriction f_k embedded in an n-dimensional Boolean space are affine-equivalent, and affine equivalent functions have the same multiplicative complexity, as discussed in [21], [24]. Since f_k is a smaller function, depending on less variables, computing an XAG representation and minimizing the number of AND nodes become easier problems, whose solutions allow to better assess the actual multiplicative complexity of the original function (note that determining the exact value of the multiplicative complexity of a function is a computationally intractable problem [20]).

Dimension-reducible (*D-reducible*) functions are Boolean functions whose minterms are all contained in an affine space A strictly smaller than the whole Boolean space $\{0,1\}^n$ [6], [7], [8], [14]. A D-reducible function f contained in an affine space A can be represented as $f = \chi_A \cdot f_A$, where χ_A is the characteristic function of A and $f_A \subseteq \{0,1\}^{\dim A}$ is the projection of f onto A. We show that an XAG representation for f can be derived combining an XAG representation for the affine space A with an XAG representation for the smaller function f_A. The interest on this special regularity is due to the fact that affine spaces can be represented by an AND of XORs of literals. In this paper, we study and formally estimate the multiplicative complexity of a D-reducible function f starting from the multiplicative complexities of χ_A and of the function f_A. We validate the proposed approach through three sets of experiments: for autosymmetric functions, for D-reducible functions, and for functions that are both autosymmetric and D-reducible.

First of all, we observe that autosymmetry is a property that is frequent enough within Boolean functions to be worth studying, indeed about 24% of the functions in the classical ESPRESSO benchmark suite [41] have at least one non-degenerate autosymmetric output [11]. For this set of functions, we are able to get a better estimate of the multiplicative complexity in about 52% of the cases, with an average reduction of the number of ANDs of about 44%.

The experiments for D-reducible functions show that the XAG minimization can benefit from the D-reducible decomposition of the function in about the 43% of the D-reducible benchmarks, with an average reduction of the number of ANDs of about 35%. Moreover, the computational time for XAG synthesis is reduced in average by the 24%.

Finally, for functions that are both autosymmetric and D-reducible, we get a better estimate of the multiplicative complexity in about 90% of the cases, with an average reduction of the number of ANDs of about 24%.

This paper is an extended version of the conference paper presented in [5] and it is organized as follows. Preliminaries on multiparty computation, multiplicative complexity, XAG networks, autosymmetric and D-reducible functions are described in Section 2. Section 3 discusses the multiplicative complexity of autosymmetric functions. Section 4 analyzes the multiplicative complexity of D-reducible functions. Section 5 provides an analysis of the multiplicative complexity of Boolean functions that are both D-reducible and autosymmetric. Section 6 reports the experimental results. Finally, Section 7 concludes the work.

2 PRELIMINARIES

2.1 Secure Multiparty Computation

The distributed scenario, where a number of computing devices would collaborate in the computation of a given function, has been investigated for the last forty years, dealing with different problems caused by non-cooperating parties, due to faults or deliberate misbehavior. Secure Multiparty Computation (SMC) protocols have been devised to allow untrusted parties to cooperate in a secure manner, keeping their own inputs private and sharing only the result of the computation. At the end of the execution, *privacy* and *correctness* requirements are guaranteed, meaning that the parties receive correctly their output, while they are enabled to learn nothing more than the results from the computation. A typical example of secure computation is the millionaires' problem, where two millionaires want to run a computation to establish which one of them is richer, without the help of any trusted third party and knowing at the end nothing more than the output of the computation.

In literature, different techniques have been used to realize SMC protocols, generally deploying fully homomorphic encryption, garbled circuits, or linear secret sharing, and a theoretical result has been reported proving that any function can be securely computed [43]. In addition, specialized application cases have been studied giving origin to research fields such as private set intersection or privacy preserving benchmarking. Recently, impressive improvements in the performance of SMC have been reported, showing how complex applications can be resolved under this paradigm, with important practical consequences. In [2], for example, the computation of the AES cipher, where the private inputs are the symmetric key and the data to be encrypted, is analyzed reporting execution time ranging from 50 microseconds to 20 seconds for block. In general SMC can be now considered as a practical solution to various real-life problems, and some deployments have been registered such as in the case of a public auction among sugar beets producers [16] or to analyze and drive public investments [15].

The Garbled Circuit (GC) protocol, introduced by Yao in 1982 [42], is one of the possible solutions for the two-party secure computation problem. In this approach the computed function is represented using a Boolean circuit, and the evaluation is performed gate per gate by the collaborating parties.

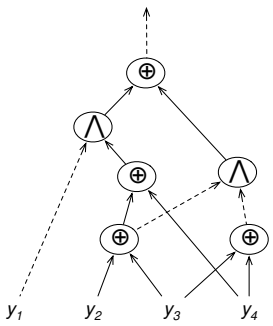


Fig. 1. XAG representation of the 4-input function corresponding to the Karnaugh map in Figure 3.

More in detail, one of the parties, say Alice, acts as the garbler, who encodes the inputs to the circuit and encrypts the truth table of each gate. The encryption consists in randomly selecting strings and keys, representing the input and output values, i.e. the 0s and 1s obtained according to the truth table of the selected gate. Using the correct keys associated to the input values, the decryption enables the recovering of the output result. In the protocol, Alice provides the evaluator, say Bob, with her encrypted input values and the garbled tables of all the gates included in the circuit. Bob retrieves his encoded input values interacting with Alice using an oblivious transfer protocol, without in this way revealing anything about his input to Alice. At the end, Bob is enabled to compute the output values for all the gates in the circuit and to share the final result with Alice.

An enhancement of the basic protocol has been presented by Koleshnikov et al [28], where they enable the computation of garbled XOR gates without any interaction between Alice and Bob, saving then the communication costs for the execution of the oblivious transfer protocol.

A direct consequence of this result, is that using logic synthesis algorithms, it is possible to optimize the Boolean circuit for the garbled evaluation and minimize the multiplicative complexity of the circuit. In this direction, authors in [37] present a technique based on a cut rewriting algorithm, which improves in benchmarks related to secure multiparty computation applications.

2.2 Multiplicative Complexity and XOR-AND graphs

The *multiplicative complexity* of a Boolean function is a complexity measure defined as the minimum number of AND gates (i.e., multiplications) that are sufficient to represent the function over the basis {AND, XOR, NOT}, a basis widely used to represent Boolean functions for cryptographic applications [18], [20], [37], [36], [39]. More precisely:

Definition 1. The *multiplicative complexity $M(f)$ of a Boolean function f* is the number of AND gates, with fan-in 2, that are necessary and sufficient to implement f with a circuit over the basis {AND, XOR, NOT}.

Definition 2. The *multiplicative complexity $M_C(f)$ of a circuit C* implementing a Boolean function f over the basis {AND, XOR, NOT} is the actual number of AND gates in C.

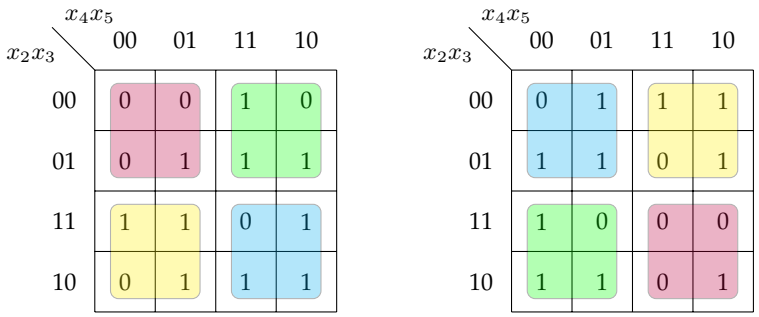
Observe that the multiplicative complexity of a circuit for f only provides an upper bound for the multiplicative complexity of f, i.e., $M(f) \leq M_C(f)$. As already discussed in Section 1, the multiplicative complexity measure plays a crucial role in cryptography-related applications for various reasons. First of all, the minimization of the number of AND gates is important for high-level cryptography protocols such as zero-knowledge protocols and secure multiparty computation, where processing AND gates is more expensive than processing XOR gates [1], as discussed in the previous section. Moreover, the multiplicative complexity is an indicator of the degree of vulnerability of the circuits, as a small number of AND gates in an {AND, XOR, NOT} circuit corresponds to a high vulnerability to algebraic attacks [20], [25], [39]. Unfortunately, determining the exact value of the multiplicative complexity of a function f is a computationally intractable problem [20]. Thus, the minimization of the number of AND gates in any circuit implementation over the basis {AND, XOR, NOT} becomes very important to assess the actual multiplicative complexity of the function.

In this work, we consider Boolean functions represented in *XOR-AND graphs* (XAGs) form [26], [37], [36], and use the multiplicative complexity $M_X(f)$ of an XAG implementation of a function f to provide an upper bound for its real multiplicative complexity $M(f)$. XAGs are logic networks which contain only binary XOR nodes, binary AND nodes, and inverters. In particular, we refer to the XAG model described in [36], where regular and complemented edges are used to connect the gates. Complemented edges indicate the inversion of the signals and replace inverters in the network. An example of XAG is shown in Figure 1, where complemented edges are denoted by dashed lines.

2.3 Autosymmetric Functions

Commonly, the "regularities" of Boolean functions are exploited with the purpose to derive, in shorter synthesis time, more compact circuits. It is not always clear whether a function is "regular", and which type of regularity could be exploited for its synthesis. Some previous papers on logic synthesis have focused on structural regularities of Boolean functions based on the notion of affine spaces and easily expressed using XORs. In this context, we study a particular regularity, i.e., autosymmetry [10], [11], [30], in order to decrease the multiplicative complexity of an XAG.

Intuitively, a Boolean function f over n variables is *k-autosymmetric* if it can be projected onto a smaller function f_k that depends on $n-k$ variables. The regularity of a Boolean function f is then measured computing its *autosymmetry degree* k, with $0 \leq k \leq n$, where $k = 0$ means no regularity. For $k \geq 1$ the Boolean function f is said to be *autosymmetric*, and a new function f_k depending on $n-k$ variables only, called the *restriction* of f, is identified in time polynomial in the size of the initial ROBDD representation of f [10]. Moreover, an expression for f can be simply built from f_k: $f(x_1, x_2, \ldots, x_n) = f_k(y_1, y_2, \ldots, y_{n-k})$, where f_k is a Boolean function on $n-k$ variables $y_1 = \oplus(X_1), y_2 = \oplus(X_2), \ldots, y_{n-k} \oplus (X_{n-k})$ and each $\oplus(X_i)$ is a XOR whose input is a set of variables X_i with $X_i \subseteq \{x_1, x_2, \ldots, x_n\}$. Note that $\oplus(X_i)$ can be a single variable, i.e., $X_i = \{x_j\}$ and $\oplus(X_i) = x_j$. The autosymmetry test consists of finding



(a) $x_1 = 0$ (b) $x_1 = 1$

Fig. 2. Karnaugh map of the second output of *rd53* depending on the 5 Boolean variables x_1, x_2, x_3, x_4, x_5.

the value of k, the restriction f_k, and each single XOR with its input variables X_i (reduction equations). Note that a degenerate function, i.e., a function that does not depend on all the variables in the Boolean space, is autosymmetric.

Since the algorithms for the autosymmetry test would require a fairly long and detailed explanation, we refer the reader to [3], [9], [10]. In particular, [3] and [10] describe implicit procedures for the test, working on BDD representations of incompletely and completely specified Boolean functions, respectively.

The restriction f_k is "equivalent" to, but smaller than f, and has $|S(f)|/2^k$ minterms only, where $S(f)$ denotes the support of f, and thus $|S(f)|$ is the number of minterms of f. The synthesis of f can be reduced to the synthesis of its restriction f_k. As the new $n-k$ variables are XOR combinations of some of the original ones, the reconstruction of f from f_k can be obtained with an additional logic level of XOR gates, whose inputs are the original variables, and the outputs are the new $n-k$ variables and their complementations given as inputs to a circuit for f_k. The restricted function f_k can be synthesized in any framework of logic minimization, in this paper we derive an XAG representation of it. The overall representation of an XAG for the function f using the XAG for f_k and the XOR nodes for the reduction equations is represented in Figure 5.

Consider, for example, the second output of the benchmark function *rd53* of the LGSynth'89 benchmark suite [41], i.e., the Boolean function f depicted in Figure 2. The "regularity" of the function is highlighted by the colors in the figure. The autosymmetry degree of f is 1 (i.e., $k = 1$) and the reduction equations are $y_1 = x_1 \oplus x_2$, $y_2 = x_1 \oplus x_3$, $y_3 = x_1 \oplus x_4$, $y_4 = x_1 \oplus x_5$ (for details on the computation see [10]). Thus, the restriction f_1 depends on 4 variables and it is depicted in Figure 3. Note that each point of the restriction corresponds to two points of the original function, as indicated by the colors in the maps. For example, the point 0000 in the Karnaugh map of Figure 3 corresponds to the two points 00000 and 11111 in the Karnaugh map of Figure 2. This is due to the reduction equations. In fact, considering the point $(x_1, x_2, x_3, x_4, x_5) = 00000$, through the reduction equations we get $(y_1, y_2, y_3, y_4) = (x_1 \oplus x_2, x_1 \oplus x_3, x_1 \oplus x_4, x_1 \oplus x_5) = 0000$. Exactly the same holds for the point $(x_1, x_2, x_3, x_4, x_5) = 11111$. It is easy to verify that we can perform a similar computation for any couple of corresponding points depicted in Figure 2, obtaining the Karnaugh map of Figure 3. For this example we have the XAG representation described in Figure 6.

We now recall some properties of autosymmetric functions and of their restrictions, that will be useful for the analysis of their multiplicative complexity. As shown in [10], [11], any k-autosymmetric function f is associated to a k-dimensional vector space L_f, defined as the set of all minterms α s.t. $f(x) = f(x \oplus \alpha)$ for all $x \in \{0,1\}^n$. The k variables that are truly independent onto L_f are called *canonical variables*, while the other variables are called *non-canonical*. Informally, the canonical variables are the ones that assume all the possible combinations of $\{0,1\}$ values in the vectors of the vector space L_f, meanwhile the non-canonical variables are the variables that, on L_f, have a constant value or are a linear combination of the canonical ones. Moreover, the restriction f_k corresponds to the projection of f onto the subspace $\{0,1\}^{n-k}$ where all the canonical variables assume value 0 (see [10], [11] for more details), while the reduction equations correspond to the linear combinations that define each non-canonical variable in terms of the canonical ones. For instance, consider again the 1-autosymmetric benchmark *rd53* (second output). Its associated vector space is the 1-dimensional space $L_f = \{00000, 11111\}$, whose canonical variable is x_1 (all other variables are non-canonical and must be equal to x_1 on L_f), and the restriction corresponds to the projection of the function onto the space where $x_1 = 0$, as it can be noted from Figures 2 and 3.

Autosymmetric functions are just a subset of the total number of Boolean functions. Indeed, while the number N_B of the Boolean functions of n variables is $N_B = 2^{2^n}$, the number of autosymmetric ones is $N_A = (2^n - 1)2^{2^{n-1}}$ [11]. Therefore, the set of autosymmetric functions is much smaller than the one containing all the Boolean functions. Nevertheless, a considerable amount of standard Boolean functions of practical interest falls in the class of autosymmetric functions. Indeed, about 24% of the functions in the classical ESPRESSO benchmark suite [41] have at least one truly (i.e., non degenerate) autosymmetric output [10], [11].

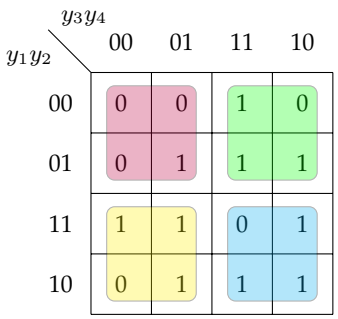


Fig. 3. Karnaugh map of the restriction of the second output of *rd53* depending on the 4 Boolean variables y_1, y_2, y_3, y_4.

Thus, the interest on autosymmetric functions is motivated by 1) their compact (in term of number of AND gates) representation, which consists of a XOR layer that is the input to an XAG for the restriction; 2) the frequency of autosymmetric functions in the set of benchmark functions.

We point out that autosymmetric functions have been independently characterized in the theoretical framework concerning the analysis of affine equivalent Boolean functions [21], [24], [32]. Two Boolean functions f and g, depending on n binary variables, are affine-equivalent if f can be written as $f(x) = g(Ax + a) + b^T x + c$, for all $x \in \{0,1\}^n$, where A is a non-singular $n \times n$ matrix over $\{0,1\}$; a, b are column vectors in the Boolean vector space $(\{0,1\}^n, \oplus)$, and $c \in \{0,1\}$. The parameters (A, a, b, c) constitute an *affine transformation* that maps g to f. Affine transformations are used to partition Boolean functions into equivalence classes, and it turns out that many relevant cryptographic properties of Boolean functions, including the multiplicative complexity, are invariant under affine transformations.

In this context, autosymmetric functions depending on n variables represent a subset of functions with *linearity dimension* [32] strictly less than n. We recall here that the linearity dimension is defined as the dimension $d_l(f)$ of the vector space of the *linear structures* of f, i.e., of all minterms $\alpha \in \{0,1\}^n$ such that $f(x \oplus \alpha) \oplus f(x)$ is a constant function, i.e., $f(x \oplus \alpha) \oplus f(x) = c$, with $c \in \{0,1\}$. Thus, we can observe that the autosymmetry degree k of a function f is upper bounded by its linearity dimension $d_l(f)$. Indeed, any vector $\alpha \in L_f$ is a linear structure, but not viceversa, as the definition of autosymmetry requires that $f(x)$ and $f(x \oplus \alpha)$ are always equal, for all $x \in \{0,1\}^n$, i.e., $f(x) \oplus f(x \oplus \alpha) = 0$.

The theory discussed in [21], [32], together with the properties of autosymmetric functions, can be exploited to prove that f and its restriction f_k, embedded into an n-dimensional Boolean space, are affine equivalent, and therefore have the same multiplicative complexity. First of all, observe that for any $x \in \{0,1\}^n$, f can be written in terms of its restriction f_k as $f(x) = f_k(Lx)$, where L is the $(n-k) \times n$ matrix that defines the reduction equations. This matrix has rank $n-k$, as each of the $(n-k)$ columns corresponding to the non-canonical variables contains only one entries equal to 1, while all other entries are equal to 0 [10], [11]. Now, we recall from [24] the definition of embedding of a function.

Definition 3. Let $g : \{0,1\}^\ell \to \{0,1\}$. The *embedding* of g in the n-dimensional space $\{0,1\}^n$, $n \geq \ell$, is defined as the n-variables Boolean function $g^{(n)}$ that satisfies $g^{(n)}(x_1, \ldots, x_\ell, x_{\ell+1}, \ldots, x_n) = g(x_1, \ldots, x_\ell)$.

Observe that, whenever $\ell < n$, $g^{(n)}$ is a degenerate function, depending only on the first ℓ variables. We now show that f and the embedding $f_k^{(n)}$ of its restriction are affine equivalent.

Proposition 1. Let f be a k-autosymmetric function depending on n binary variables, and let $f_k^{(n)}$ be the embedding of its restriction f_k in $\{0,1\}^n$. Then f and $f_k^{(n)}$ are affine equivalent.

Proof. We show that there exists a non-singular $n \times n$ matrix A such that, for any $x \in \{0,1\}^n$, $f(x) = f_k^{(n)}(Ax)$. Consider the matrix L representing the reduction equations. As already observed, L has rank $n-k$. The idea is to simply add k new rows to L, linearly independent from the others, in order to obtain a non-singular matrix of maximum rank n. This can be accomplished by choosing as new rows the n-dimensional vectors containing only one entry equal to one, corresponding to one of the canonical variables of f (recall that a k-autosymmetric function has exactly k canonical variables). The resulting matrix A is non-singular as it is a permutation of a triangular one, with all entries equal to 1 on the main diagonal, thanks to the initial structure of L.

Now, for all $x \in \{0,1\}^n$, let $y = Ax$. We have $f_k^{(n)}(Ax) = f_k^{(n)}(y) = f_k^{(n)}(y_1, \ldots, y_{n-k}, y_{n-k+1}, \ldots, y_n) = f_k(y_1, \ldots, y_{n-k}) = f_k(Lx) = f(x)$, and the thesis follows. ■

Finally, we can conclude that autosymmetry is a special and restricted case of affinity equivalence. We study this particular case since this regularity can be efficiently tested and autosymmetric functions can be better minimized in XAG forms, as shown by the experiments shown in Section 6.

2.4 D-Reducible Functions

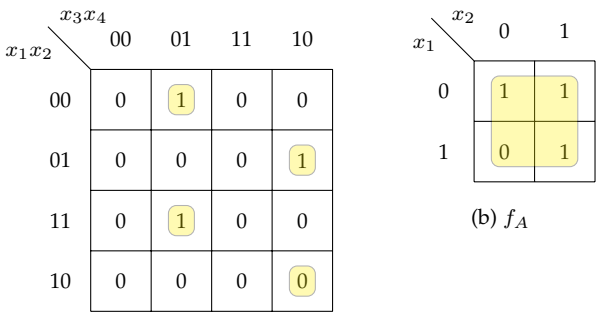
In this section we summarize the definition and the major properties of Dimension Reducible Boolean functions (i.e., DRed functions). For this purpose, we recall that the Boolean space $\{0,1\}^n$ is a vector space with respect to the exclusive sum $\oplus$ and the multiplication with the scalars 0 and 1. Moreover, an affine space is a vector space or a translation of a vector space, as described in the following definition [6].

Definition 4. Let V be vector subspace of the Boolean vector space $(\{0,1\}^n, \oplus)$ and α be point in $\{0,1\}^n$, then the set $A = \alpha \oplus V = \{\alpha \oplus v \mid v \in V\}$ is an *affine space* over V with *translation point* α.

D-reducible functions are Boolean functions whose minterms are all contained in an affine space A strictly smaller than the whole Boolean space $\{0,1\}^n$.

Definition 5. A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is *D-reducible* if $f \subseteq A$, where $A \subset \{0,1\}^n$ is an affine space of dimension strictly smaller than n.

Observe that the notion of dimension exploited in the definition of D-reducibility refers to the dimension of the smallest vector, or affine space, that contains the whole on-set of a function f, and it is defined as the number of on-set minterms, considered as vectors of the Boolean space



(a) f (b) f_A

Fig. 4. *Karnaugh maps of a D-reducible function f and its corresponding projection f_A.*

$(\{0,1\}^n, \oplus)$, that are linearly independent. Thus, this notion is different from the notion of *dimension of a Boolean function* f ($\dim(f)$) defined as the minimum number of *essential variables* (i.e., variables on which a function truly depends) of all functions that are affine equivalent to f [21], [24]. For instance, the dimension of the smallest affine space that contains the on-set of the AND of n binary variables is $d = 1$, while $\dim(AND) = n$. In general, the same affine space A may contain functions with very different dimensions. The minimal affine space A containing a D-reducible function f is unique and it is called the *associated affine space* of f. The function f can be represented in the following way: $f = \chi_A \cdot f_A$, where $f_A \subseteq \{0,1\}^{\dim A}$ is the projection of f onto A and χ_A is the characteristic function of A. Moreover, as shown in [23], an affine space can be represented by a simple expression, called *pseudoproduct*, consisting in an AND of XORs or literals. In particular, an affine space of dimension $\dim A$ can be represented by a pseudoproduct containing $(n - \dim A)$ XOR factors.

Consider, for instance, the function $f : \{0,1\}^4 \to \{0,1\}$ with on-set $\{0001, 0110, 1101\}$ represented in the Karnaugh map on the left side of Figure 4. The smallest affine space containing the on-set of f, depicted with dotted circles on the map, is $A = \{0001, 0110, 1010, 1101\}$. This affine space has dimension $\dim A = 2$ and can be represented by the pseudoproduct $(x_1 \oplus x_2 \oplus \overline{x}_3)(x_1 \oplus x_2 \oplus x_4)$ that contains exactly $n - \dim A = 2$ XOR factors (see Section 4 for more details on how to derive such representation). If we project f onto the smaller space A, we obtain the function $f_A = \{00, 01, 11\}$, represented in the Karnaugh map on the right side of the figure.

The D-reducibility of a function f can be exploited in the minimization process. The projection f_A is minimized instead of f. This approach requires two steps: *(i)* deriving the affine space A and the projection f_A; *(ii)* minimizing f_A in any logic framework.

The D-reduciblity test, which establishes whether a function f is D-reducible, and the computation of A can be performed efficiently. In particular, the D-reduciblity test described in [6] is based on the *Gauss-Jordan elimination* procedure [29], which is used to find the on-set minterms of f that are linearly independent. The number ℓ of linearly independent minterms defines the dimension of the smallest affine or vector space A covering the on-set of f, and if $\ell < n$, then the function f is D-reducibile. The Gauss-Jordan elimination procedure can also be used to derive an algebraic expression for the characteristic function of A. Most importantly, as described in [6], this procedure can be applied starting from any SOP representation of f without generating all its minterms, i.e., in time polynomial in the initial representation of f. Finally, the projection f_A of f onto A can be simply derived from f by deleting $n - \dim A$ variables from the SOP representation of f.

3 Multiplicative Complexity of Autosymmetric Functions

In this section we analyze the relationships between the multiplicative complexity of an autosymmetric function and the multiplicative complexity of its restriction. As already mentioned in Section 2.3, any autosymmetric function f is affine-equivalent to the function obtained embedding the restriction f_k in an n-dimensional Boolean space, and this implies that f and f_k have the same multiplicative complexity [21]. In order to provide a comprehensive presentation and discussion of the multiplicative complexity of autosymmetric functions, we present here an alternative and constructive proof of this result, that requires only the properties of autosymmetric functions recalled in Section 2.3.

First of all, observe that an XAG representation of a k-autosymmetric function f can be easily obtained composing an XAG for the restriction f_k with an additional layer of XOR gates implementing the reduction equations. The inputs to the new layer are the original variables $x_1, x_2, \ldots, x_n$ and the outputs are the new variables $y_1, y_2, \ldots, y_{n-k}$, that become the inputs to the XAG for f_k, as shown in Figure 5. Since the new layer contains only XOR gates, we immediately conclude that $M(f) \leq M(f_k)$, as formally stated in the following proposition.

Proposition 2. The multiplicative complexity of an autosymmetric function f is less or equal to the multiplicative complexity of its restriction f_k.

Proof. First recall that the multiplicative complexity of an XAG implementation for a function f provides an upper bound for the multiplicative complexity of the function itself, i.e., $M(f) \leq M_{XAG}(f)$. Thus, the thesis follows since we can construct an XAG for f with exactly $M(f_k)$ AND nodes. This can be done adding to an XAG for f_k, containing a minimum number $M(f_k)$ of AND nodes, a layer consisting only of XOR nodes, as shown in Figure 5. ■

We can now exploit the characterization for the restriction of an autosymmetric function recalled in Section 2.3 to prove the following result.

Theorem 1. Let f be a k-autosymmetric function, and let f_k be its restriction. Then, $M(f) = M(f_k)$.

Proof. We already argued that $M(f) \leq M(f_k)$. Thus, it is enough to show that $M(f) \geq M(f_k)$. By contradiction suppose that the multiplicative complexity of the whole function f is strictly less than the multiplicative complexity of its restriction f_k, i.e., $M(f) < M(f_k)$. This assumption means that any XAG for f_k requires strictly more than $M(f)$

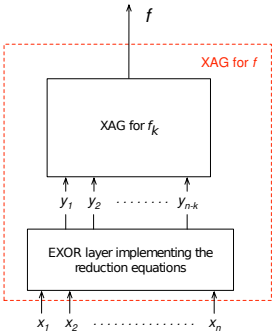


Fig. 5. An XAG for an autosymmetric function f obtained adding a XOR level implementing the reduction equations to an XAG for the restriction f_k.

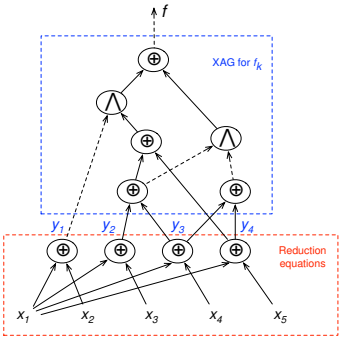


Fig. 6. XAG representation for the benchmark *rd53* (second output), derived exploiting the autosymmetry of the function.

AND nodes, i.e., $M_X(f_k) > M(f)$, where $M_X(f_k)$ denotes the multiplicative complexity of an XAG for f_k. Since the restriction f_k corresponds to the projection of f onto the subspace $\{0,1\}^{n-k}$ where all the canonical variables of f have value 0, we can derive an XAG representation for f_k starting from an XAG for f and substituting all canonical input variables with the constant value 0. Note that the constant value 0 can be obtained computing the XOR of any non-canonical variable with itself. Such a transformation can only decrease the number of ANDs in the original XAG, as all AND nodes that receive in input the constant value 0 can be removed from the circuit, and substituted with the value 0. Therefore, if we start from an XAG implementation of f with the minimum number $M_X(f) = M(f)$ of AND nodes, we can derive an XAG for f_k with $M_X(f_k) \leq M(f)$ ANDs, in contradiction with the initial assumption $M(f) < M(f_k)$. ■

Since f_k is a smaller function, depending on less variables, computing an XAG representation and minimizing the number of ANDs become easier problems, whose solutions allow to better assess the actual multiplicative complexity of the original function f. For instance, for our running example concerning the benchmark *rd53* already discussed in Section 2.3, we can derive the XAG representation shown in Figure 6 simply adding four XOR nodes to the XAG for f_k of Figure 1, that contains 4 XORs and only 2 ANDs. Notice that the direct XAG minimization of *rd53* (second output), performed using the software from [36], would produce a bigger circuit, with 5 ANDs.

4 Multiplicative Complexity of D-Reducible Functions

We now focus on the class of D-reducible functions with the aim of verifying if the decomposition that characterizes these functions can be exploited to estimate their multiplicative complexity, in analogy to what we have seen for autosymmetric functions.

Recall, from Section 2.4, that a D-reducible function f, with associated affine space A, can be decomposed in the form $f = \chi_A \cdot f_A$, where χ_A is the characteristic function of A and $f_A \subseteq \{0,1\}^{\dim A}$ is the projection of f onto A. Thus, an XAG representation for f can be derived combining, via an AND gate, an XAG representation for the affine space A with an XAG representation for f_A (as shown in Figure 7).

This decomposition immediately allows to upper bound the multiplicative complexity of f with the sum of the multiplicative complexity of χ_A and of f_A. More precisely we have:

$$M(f) \leq M(\chi_A) + M(f_A) + 1\,, \tag{1}$$

where we also account for the AND gate on top of the overall XAG for f. The advantages of this approach should derive from the fact that f_A is a function that depends on fewer variables, so we can reasonably expect its XAG representation to be more compact, and also easier to derive from a computational point of view. Moreover, χ_A is not just any function, but the characteristic function of an affine space, thus we can take advantage of its structural properties to derive an XAG possibly optimal in number of AND gates. While the first aspect can only be verified and evaluated experimentally, the XAG representation of affine spaces can be fully investigated from a theoretical point of view. To this aim, we first mention a result concerning the relation between the multiplicative complexity of a function and its *algebraic degree*. Recall that any function f depending on n binary variables can be represented as a multilinear polynomial over $GF(2)$, i.e., a XOR of conjunctions of literals.

This representation, which is unique, can be obtained taking the modulo 2 sum of all the minterms in the on-set of the function, each represented as a conjunction of literals (see [40] for more details). Literals corresponding to negated variables are replaced with the modulo-2 sum of the variable and the constant 1, e.g., $\overline{x}_i = (1 \oplus x_i)$.

Definition 6. The *algebraic degree* $deg(f)$ of a Boolean function f is the degree of the unique multilinear polynomial that represents f over $GF(2)$.

Observe that $deg(f)$ corresponds to the number of variables in the longest products of this polynomial.

Consider for instance the function $f : \{0,1\}^4 \rightarrow \{0,1\}$ with on-set $\{0001, 0110, 1101\}$ represented in the Karnaugh map on the left side of Figure 4. The polynomial representation of f over $GF(2)$ is given by the expression $f = x_4 \oplus x_1x_4 \oplus x_2x_3 \oplus x_2x_4 \oplus x_3x_4 \oplus x_1x_2x_3 \oplus x_1x_3x_4 \oplus x_1x_2x_3x_4$. Thus, f has algebraic degree $deg(f) = 4$.

The algebraic degree turns out to be related to several complexity measures, and in particular to the multiplicative complexity [17], [34].

Lemma 1. ([34]) Any circuit computing a Boolean function f over the basis {AND, XOR, NOT} contains at least $deg(f) - 1$ AND gates with fan-in 2.

Corollary 1. ([34]) If a function f has algebraic degree $def(f)$, then its multiplicative complexity is at least $deg(f) - 1$.

Let us now focus on the multiplicative complexity of affine spaces. Consider an affine subspace A of $\{0,1\}^n$. As discussed in [6], [23], we can partition the set of binary variables $\{x_1, x_2, \ldots, x_n\}$ into two subsets: the subset of the *canonical variables* and the subset of the *non-canonical variables*. The canonical variables are the truly independent variables in the space A, in the sense that they can assume all possible combinations of 0-1 values, and their number is exactly $\dim A$. On the contrary, the remaining $n - \dim A$ non-canonical variables are not independent of A because they can be defined as linear combinations (i.e., XORs) of the canonical ones. This fact is clearly expressed by the characteristic function χ_A of the affine space A, which can always be represented by a special pseudoproduct containing exactly $(n - \dim A)$ XOR factors such that: 1) each non-canonical variable appears in exactly one XOR factor; 2) each XOR factor is composed by one non-canonical variable and possibly some canonical ones. This pseudoproduct representation is called the *canonical (CEX) expression* of A.

Consider, for example, the function f represented in Figure 4, and its affine space $A = \{0001, 0110, 1010, 1101\}$. We can observe that the first two variables, x_1 and x_2 assume on A all possible combinations of values, i.e., 00, 01, 10, and 11. On the contrary, x_3 and x_4 can be defined on A in terms of x_1 and x_2 such that $x_3 = x_1 \oplus x_2$ and $\overline{x}_4 = x_1 \oplus x_2$. Thus, the two canonical variables of A are x_1 and x_2, while the non-canonical ones are x_3 and x_4. The CEX expression of this affine space is given by the pseudoproduct $(x_1 \oplus x_2 \oplus \overline{x}_3)(x_1 \oplus x_2 \oplus x_4)$, which encodes the two linear combinations defining x_3 and x_4 on A. Indeed, this pseudoproduct states that on A the two factors $(x_1 \oplus x_2 \oplus \overline{x}_3)$ and $(x_1 \oplus x_2 \oplus x_4)$ must be equal to 1, i.e., $x_1 \oplus x_2 \oplus \overline{x}_3 = 1$ and $x_1 \oplus x_2 \oplus x_4 = 1$, from which we can derive the two equalities (1) and (2). Note that each non-canonical variable occurs in one and only one XOR factor. We now show how the CEX expression allows to characterize the multiplicative complexity of an affine space.

Theorem 2. The multiplicative complexity of the characteristic function χ_A of an affine subspace $A \subseteq \{0,1\}^n$ is exactly $M(\chi_A) = n - \dim A - 1$.

Proof. We first prove that $M(\chi_A) \geq n - \dim A - 1$. To this aim, let us consider a CEX representation for A. This expression is composed by an AND of $n - \dim A$ XOR factor. Each XOR factor contains a different non-canonical variable. Thus, the polynomial representation of χ_A, that can be immediately derived from the CEX expression, has degree $\deg(\chi_A) = n - \dim A$ since it certainly contains the term corresponding to the product of all the non-canonical variables. Thus, Corollary 1 immediately implies that $M(\chi_A) \geq n - \dim A - 1$. Now, observe that we can immediately derive an XAG representation from the CEX expression: we use XOR gates for each XOR factor, and then exactly $n - \dim A - 1$ AND gates to compute the product of all factors. Thus, we have $M(\chi_A) \leq n - \dim A - 1$, and the thesis immediately follows. ■

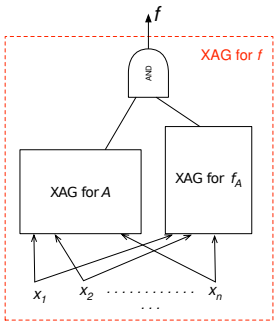


Fig. 7. An XAG representation for a D-reducible function f obtained combining an XAG for the affine space A with an XAG for the projection f_A. Notice that only $\dim A$ of the n input variables are inputs for f_A.

Thus, in the overall XAG representation for D-reducible functions proposed in Figure 7, we can always represent the affine space A with the XAG derived from its CEX expression, which is optimal in the number of AND gates. We therefore derive the following upper bound for the multiplicative complexity of any D-reducible function:

Corollary 2. Let $f : \{0,1\}^n \rightarrow \{0,1\}$ be a D-reducible function with affine space A, and let f_A be its projection onto A. Then $M(f) \leq n - \dim A + M(f_A)$.

Proof. Follows immediately from Equation (1) since, by Theorem 2, we have $M(\chi_A) = n - \dim A - 1$. ■

Given a D-reducible function f and an XAG representation for its projection f_A, with $M_X(f_A)$ AND gates, we can then exploit the decomposition of Figure 7 to derive the estimate

$$M_{dec}(f) = M(\chi_A) + M_X(f_A) + 1 = n - \dim A + M_X(f_A) \quad (2)$$

for the multiplicative complexity of f.

5 Multiplicative Complexity of D-reducible Autosymmetric Functions

In this section we provide an analysis of the multiplicative complexity of Boolean functions that are both D-reducible and autosymmetric. First of all, we prove that if a function f is autosymmetric and D-reducible with associated affine space A, then its projection onto A is also autosymmetric.

Theorem 3. Let f be a k-autosymmetric Boolean function depending on n binary variables. If f is D-reducible with associate affine space A, then the projection f_A of f onto A is k-autosymmetric.

Proof. Recall that any k-autosymmetric function f is associated to a k-dimensional vector space L_f, defined as the set of all minterms α s.t. $f(x) = f(x \oplus \alpha)$ for all $x \in \{0,1\}^n$. We prove that f_A is k-autosymmetric by showing that for any $\alpha \in L_f$ it holds that $f_A(y \oplus \alpha_A) = f_A(y)$ for all $y \in \{0,1\}^{\dim A}$, where α_A denote the projections of α onto A, i.e., the minterm obtained from α by keeping only the literals corresponding to the canonical variables of A.

First of all, we observe that the set L_f is a subspace of the linear vector space V associated to A. Indeed, let $\alpha \in L_f$,

and let x be any on-set minterm of f. Then, $f(x \oplus \alpha) = f(x) = 1$, and therefore both x and $x \oplus \alpha \in A$. This in turns implies that $\alpha \in (x \oplus A)$, i.e., $\alpha \in V$, since $x \oplus A = V$ for any $x \in A$ (we refer the reader to [23] for more details on affine spaces and their properties).

Since $L_f \subseteq V$, we have that for all $x \in \{0,1\}^n$ and for all $\alpha \in L_f$, $x \in A$ if and only if $x \oplus \alpha \in A$. Indeed,

$$\begin{aligned} x \in A &\Leftrightarrow x = a \oplus v\,, \text{ for some } a \in A \text{ and some } v \in V \\ &\Leftrightarrow x \oplus \alpha = a \oplus v \oplus \alpha \\ &\Leftrightarrow x \oplus \alpha \in A \end{aligned}$$

since $v \oplus \alpha \in V$. Now, let α be any vector in L_f. Then, for all $x \in \{0,1\}^n$, we have $f(x \oplus \alpha) = f(x)$, i.e., $\chi_A(x \oplus \alpha) f_A(x_A \oplus \alpha_A) = \chi_A(x) f_A(x_A)$, where x_A and α_A denote the projections of x and α onto A. If we consider only the vectors $x \in A$, the fact that $x \oplus \alpha$ is also in A implies $\chi_A(x) = \chi_A(x \oplus \alpha) = 1$. Thus, we have $f_A(x_A \oplus \alpha_A) = f_A(x_A)$, which in turns implies that $\alpha_A \in L_{f_A}$. ■

This result is interesting from a practical point of view, it implies that we can run the autosymmetry test onto f_A instead of f, with a reduced computational effort guaranteed by the fact that f_A depends on less variables than f.

As for the estimation of the multiplicative complexity of f, we can observe that, since f is autosymmetric and D-reducible, we can upper bound its multiplicative complexity by first projecting f onto A, and then by estimating the multiplicative complexity of the restriction $f_{A,k}$ of f_A. Indeed, we have that $M(f) = M(f_k) \leq (n - \dim A) + M(f_A)$ and, since $M(f_A) = M(f_{A,k})$, we finally obtain $M(f) \leq (n - \dim A) + M(f_{A,k})$. Thus, we can estimate the multiplicative complexity of f by computing and minimizing the restriction $f_{A,k}$ of f_A in XAG form. Since $f_{A,k}$ contains $|S(f)|/2^k$ minterms only (recall that $|S(f)|$ is the number of minterms of f), and depends on $\dim A - k < n - k$ variables only, its XAG minimization should be easier and might provide a final circuit with a reduced number of AND gates. This expectation has been confirmed by our experiments.

6 Experimental results

In this section we report and discuss the experimental results of the evaluation of the multiplicative complexity of the classes of autosymmetric and D-reducible functions[1].

6.1 Autosymmetric functions

The approach presented in Section 3 has been applied to the ESPRESSO and LGSynth'89 benchmark suite [41] and to some functions from cryptography benchmarks in the context of multi-party computation (MPC) and fully homomorphic encryption (FHE) [36], [37]. Notice that autosymmetry is a property of single outputs, i.e., different outputs of the same benchmark can have different autosymmetry degrees. Thus, we perform the autosymmetry test on each single output of the considered benchmark suites. The experiments have been run on a Pentium INTEL(R) CORE(TM) i5-5200U 2.20 GHz processor with 4.00 GB RAM, on a virtual machine running OS Ubuntu 64-bit.

1. A GitHub repository with the experimental data can be found at `https://github.com/MariaChiaraMC/IEEETC-experiments`

TABLE 1
Experimental comparison of autosymmetric benchmarks, considering an XAG after the autosymmerty test and the standard XAG computed without the autosymmetry test.

			standard XAG		XAG with autosym. test		
Benchmark	**in**	**k**	$\mathbf{M_X(f)}$	**time (s)**	$\mathbf{M_X(f_k)}$	**time (s)**	**gain**
add6(5)	12	1	8	0.97	5	1.77	38%
addm4(5)	9	2	32	3.98	32	3.93	0%
amd(0)	14	2	53	10.19	53	10.03	0%
apla(5)	10	1	11	0.96	8	0.68	27%
b2(15)	16	1	118	22.00	118	22.32	0%
ex5(21)	8	1	7	0.59	6	0.43	14%
exep(46)	30	9	20	0.92	20	0.87	0%
exps(19)	8	1	10	2.30	6	0.45	40%
in0(3)	15	1	14	3.03	14	2.77	0%
mainpla(20)	27	5	119	21.37	119	21.80	0%
max46(1)	10	2	36	6.74	10	1.80	72%
max46(5)	10	1	305	50.10	156	26.22	49%
opa(26)	17	9	16	2.51	16	2.44	0%
opa(48)	17	3	22	4.21	22	5.74	0%
p1(6)	8	1	6	1.03	8	0.79	-33%
pdc(10)	16	3	12	2.05	14	1.66	-17%
rd53(1)	5	1	5	0.38	2	0.11	60%
spla(39)	16	3	16	1.27	12	0.80	25%
tial(2)	14	2	174	29.69	174	29.83	0%
xparc(6)	41	16	83	10.90	21	1.80	75%
z5xp1(6)	7	3	2	0.00	0	0.00	100%
ctrl(13)	7	3	3	0.02	3	0.02	0%
dec(39)	8	1	6	0.53	6	0.42	0%
voting_N_2_M_2(1)	8	1	81	9.88	22	3.04	73%
voting_N_2_M_3(1)	16	1	9261	739.29	5208	468.86	44%
int2float(6)	11	2	8	0.59	8	0.55	0%

The experiments consider the subset of single outputs that are autosymmetric. The main aim of the experiments is to compare the synthesized XAG computed starting from an autosymmetric function f and the synthesized XAG computed starting from the corresponding restriction f_k, after the autosymmetry test. Recall that the autosymmetry test computes the autosymmetry degree k of a Boolean function and outputs: 1) the reduction equations, which form the XOR layer, and 2) the corresponding restriction f_k. We performed the autosymmetry test described in [10], [11] considering the on-set of the benchmarks. The functions f and f_k are first minimized in SOP form (using Espresso [31]), and then synthesized in XAG form using the heuristic approach proposed in [36] and briefly described at the end of Section 2.2. We then compare the number of AND nodes of the XAGs for f and f_k in order to understand how the autosymmetry test can enable the XAG minimization of autosymmetric functions.

For the sake of briefness, we report in Table 1 only a significant subset of the results. The first column reports the name of the function considered (benchmark function and output number). The following one provides its input size. Next column refers to the autosymmetry degree (i.e., k) of the function. The following two pairs of columns report the multiplicative complexity of the XAG (M_X) after applying the heuristic in [36] and the time in seconds required to obtain it, for the entire function f (first couple) and for the corresponding restriction f_k (second couple). Finally, the last column reports the gain in applying the autosymmetry test before XAG synthesis. Note that there are also some (rare) cases in which the application of the autosymmetry test before the XAG synthesis does not imply a gain; in Table 1, two representative functions are reported, i.e., *p1(6)* and *pdc(10)*. This unexpected result is due to the heuristic nature of the XAG minimizer.

Table 2 shows a summary of the overall results, i.e., the results obtained for all the circuits that have been processed in our experimental evaluation. We first consider

TABLE 2
Summary of the experimental evaluation, considering the number of ANDs in the XAGs for autosymmetric functions and non-degenerate autosymmetric functions.

	$M_X(f_k) < M_X(f)$	$M_X(f_k) = M_X(f)$	$M_X(f_k) > M_X(f)$
autosymmeric functions	6.06%	92.31%	1.63%
non-degenerate autosymmeric functions	52.09%	40.10%	7.81%

TABLE 3
Experimental comparison of D-reducible benchmarks, considering XAGs computed exploting the d-reducibility property with standard XAGs.

		standard XAG		XAG with D-red. test				
Benchmark	**in**	$\mathbf{M_X(f)}$	**time (s)**	$\mathbf{M_X(f_A)}$	$\mathbf{M(\chi_A)}$	$\mathbf{M_{dec}(f)}$	**time (s)**	**gain**
alu2(6)	10	6	2.69	5	0	6	2.29	0%
amd(15)	14	6	2.51	0	5	6	2.33	0%
amd(16)	14	7	2.47	0	6	7	2.28	0%
apla(8)	10	17	3.97	4	2	7	2.31	59%
b10(2)	15	17	3.78	13	4	18	3.19	-6%
dk48(16)	15	16	3.58	1	10	12	2.20	25%
in2(6)	19	66	9.19	48	1	50	6.27	24%
in5(8)	24	46	8.09	47	0	48	7.66	-4%
m181(6)	15	8	2.89	6	1	8	2.56	0%
newcond(1)	11	2	2.28	0	1	2	2.22	0%
spla(2)	16	14	2.90	0	13	14	2.24	0%
spla(16)	16	10	2.95	1	8	10	2.26	0%
spla(29)	16	19	3.47	5	9	15	2.33	21%
spla(42)	16	10	2.88	0	6	7	2.19	30%
t1(1)	21	10	3.89	9	0	10	3.08	0%
t2(0)	17	14	3.44	12	1	14	3.14	0%
t2(4)	17	5	2.65	3	1	5	2.44	0%
t3(2)	12	16	3.72	9	1	11	3.10	31%
t4(0)	12	17	3.87	5	0	6	2.42	65%
vg2(2)	25	30	4.26	14	5	20	3.34	33%

the set of all autosymmetric functions (degenerate and non-degenerate), we then study the truly autosymmetric (i.e., the non-degenerate) ones. Recall that degenerate functions are, by definition, autosymmetric. In Table 2, we denote with $M_X(f_k) < (=, >,$ resp.$)\ M_X(f)$ the number of benchmarks where the number of ANDs of the XAG for f_k is less than (equal to, greater than, resp.) the number of ANDs of the XAG for f.

We notice that the XAG minimization algorithm proposed in [36] is sensible to degenerate functions as shown in the first row of Table 2, where the number of benchmarks where f_k and f have the same number of ANDs is the majority (i.e., about 92.31%) and only 6.06% of them is such that $M_X(f_k)$ is less than $M_X(f)$. However, when we concentrate on non-degenerate autosymmetric functions (i.e., second row of the table), we notice that the number of benchmarks where $M_X(f_k) < M_X(f)$ considerably increases, reaching about the 52.09%, and the number of benchmarks where f_k and f have the same number of ANDs remains high (about 40.1%). Moreover, in this set the average gain is about 44%. Interestingly enough, the two compared approaches have similar synthesis times. From these experiments, we can conclude that, when a function is truly autosymmetric (i.e., non-degenerate), we can obtain better results computing the XAG on the restriction f_k instead of computing the XAG directly on the function f.

6.2 D-reducible functions

We now analyze the experimental results conducted in order to evaluate the multiplicative complexity of D-reducible functions exploiting the XAG decomposition discussed in Section 4. The experiments have been run on a CPU Intel i7 2.60GHz processor, with a virtual machine running Ubuntu 18.04, and benchmarks are taken from the ESPRESSO and LGSynth'89 benchmark suite [41]. We considered each output as a separate Boolean function, and analyzed a total of 406 D-reducible functions. As before, the functions and their projections have been synthesized in XAG form using the heuristic approach proposed in [36].

We report in Table 3 a significant subset of functions as representative indicators of our experiments. The first two columns report the name and the number of the considered output of each benchmark, and the number of its input variables. The following pair of columns reports the multiplicative complexity of the XAG for the entire function f ($M_X(f)$), obtained running the heuristic in [36], and the time in seconds required to obtain it. The next group of four columns reports *(i)* the multiplicative complexity of the XAG for the projection f_A ($M_X(f_A)$); *(ii)* the multiplicative complexity of the affine space χ_A computed applying Theorem 2 ($M(\chi_A)$); *(iii)* the overall estimate of the multiplicative complexity of f ($M_{dec}(f)$) derived using Equation (2); and *(iv)* the time in seconds required to obtain this overall estimate. Finally, the last column reports the gain (or loss) in applying the proposed decomposition method.

In Table 4, we report a summary of the overall experimental results conducted on all the 406 D-reducibile benchmarks' outputs. We denote with $M_{dec}(f) < M_X(f)$, $M_{dec}(f) = M_X(f)$, and $M_{dec}(f) > M_X(f)$ the number of benchmarks' outputs where the number of ANDs obtained applying the decomposition based on the D-reducibility property is less than, equal to, and greater than the number of ANDs of the XAG for f. The number of functions where the XAG minimization can benefit from the D-reducible decomposition is about the 43% of the whole set of functions, with an average reduction of the number of ANDs of about 35%; the number of functions where the estimates of the multiplicative complexity are the same is about 49%, while for the remaining 8% of the functions the method provides a worst estimate. We finally observe how the proposed decomposition method guarantees a reduction of the average computational time for XAG synthesis of about 24%.

6.3 Autosymmetric and D-reducible functions

We conclude this experiments section analysing the results reached applying both the autosymmetry test and the D-reducible decomposition to Boolean functions in the benchmarks from ESPRESSO and LGSynth'89 benchmark suite [41]. These last experiments have been run on a Pentium INTEL(R) CORE(TM) i5-5200U 2.20 GHz processor with 4.00 GB RAM, on a virtual machine running OS Ubuntu 64-bit.

Table 5 shows a summary of the results for functions that are both autosymmetric and D-reducible (about 9% on the total). We applied the autosymmetry test to the D-reducible functions discussed in Section 6.2. We note that

TABLE 4
Summary of the experimental evaluation, considering the number of ANDs in the XAGs for D-reducible functions.

	$M_{dec}(f) < M_X(f)$	$M_{dec}(f) = M_X(f)$	$M_{dec}(f) > M_X(f)$
D-reducible functions	42.61%	49.02%	8.37%

TABLE 5
Summary of the experimental evaluation, considering the number of ANDs in the XAGs for autosymmetric and D-reducible functions.

	$M_X(f_k) < M_X(f)$	$M_X(f_k) = M_X(f)$	$M_X(f_k) > M_X(f)$
autosymmeric and D-reducible functions	89.54%	3.27%	7.19%

we did not find D-reducible benchmarks that are also non-degenerate and autosymmetric. From Table 5, we note that the functions where the XAG minimization can benefit from autosymmetry and D-reducibility are about 90%, with an average reduction of the number of ANDs of about 24%; the number of functions where the estimates of the multiplicative complexity are the same is about 3%, while for the 7% of the functions the method provides a worst result.

7 Conclusion

In this paper we have considered the classes of autosymmetric and D-reducible functions and we have shown how these regularities can be exploited to better estimate their multiplicative complexity. Moreover, the experimental results show that these tests of regularity can enable the XAG minimization of Boolean functions.

In this work, we have considered XOR-based regularities that are frequent in classical benchmark functions. Nevertheless, it would be interesting to study the multiplicative complexity of other regular functions such as, for example, symmetric Boolean functions or self-dual functions, in order to better understand the multiplicative complexity and further improve the XAG minimization. Moreover, we plan to extend the autosymmetry algorithms in order to detect all linear structures of a function, thus generalizing our approach to all the functions with linearity dimension strictly less than n. This could further enable the XAG minimization of the corresponding circuits. Finally, all proposed methods could be extended to multi-output functions, potentially sharing the XOR layer among multiple outputs.

Acknowledgment

We are in debt to our students Alessandro Poggiali and Nicolò Pierami who carried out the benchmark experimentation. Moreover, we are grateful to the anonymous reviewers for their very constructive comments.

References

[1] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner, "Ciphers for MPC and FHE," in *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, 2015, pp. 430–454.

[2] D. W. Archer, D. Bogdanov, B. Pinkas, and P. Pullonen, "Maturity and performance of programmable secure computation," *IEEE Security Privacy*, vol. 14, no. 5, pp. 48–56, Sep. 2016.

[3] A. Bernasconi and V. Ciriani, "Autosymmetry of Incompletely Specified Functions," in *Design Automation and Test in Europe (DATE)*, 2021.

[4] A. Bernasconi, V. Ciriani, G. Trucco, and T. Villa, "Logic Synthesis by Signal-Driven Decomposition," in *Advanced Techniques in Logic Synthesis, Optimizations and Applications*, K. Gulati, Ed. Springer New York, 2011, pp. 9–29.

[5] A. Bernasconi, S. Cimato, V. Ciriani, and M. C. Molteni, "Multiplicative complexity of autosymmetric functions: Theory and applications to security," in *57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, July 20-24, 2020*. IEEE, 2020.

[6] A. Bernasconi and V. Ciriani, "Dimension-Reducible Boolean Functions Based on Affine Spaces," *ACM Trans. Design Autom. Electr. Syst.*, vol. 16, no. 2, pp. 13:1–13:21, 2011.

[7] ——, "Autosymmetric and dimension reducible multiple-valued functions," *J. Multiple Valued Log. Soft Comput.*, vol. 23, no. 3-4, pp. 265–292, 2014.

[8] A. Bernasconi, V. Ciriani, L. Frontini, and G. Trucco, "Synthesis on switching lattices of dimension-reducible boolean functions," in *2016 IFIP/IEEE International Conference on Very Large Scale Integration, VLSI-SoC 2016, Tallinn, Estonia, September 26-28, 2016*. IEEE, 2016, pp. 1–6.

[9] A. Bernasconi, V. Ciriani, F. Luccio, and L. Pagli, "Three-Level Logic Minimization Based on Function Regularities," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 22, no. 8, pp. 1005–1016, 2003.

[10] ——, "Exploiting regularities for boolean function synthesis," *Theory Comput. Syst.*, vol. 39, no. 4, pp. 485–501, 2006.

[11] ——, "Synthesis of autosymmetric functions in a new three-level form," *Theory Comput. Syst.*, vol. 42, no. 4, pp. 450–464, 2008.

[12] A. Bernasconi, V. Ciriani, G. Trucco, and T. Villa, "Using flexibility in p-circuits by boolean relations," *IEEE Trans. Computers*, vol. 64, no. 12, pp. 3605–3618, 2015.

[13] ——, "Boolean minimization of projected sums of products via boolean relations," *IEEE Trans. Computers*, vol. 68, no. 9, pp. 1269–1282, 2019.

[14] A. Bernasconi, V. Ciriani, and T. Villa, "Exploiting symmetrization and d-reducibility for approximate logic synthesis," *IEEE Trans. Computers*, 2021.

[15] D. Bogdanov, L. Kamm, B. Kubo, R. Rebane, V. Sokk, and R. Talviste, "Students and taxes: a privacy-preserving study using secure computation," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 3, pp. 117–135, 2016.

[16] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter *et al.*, "Secure multiparty computation goes live," in *International Conference on Financial Cryptography and Data Security*. Springer, 2009, pp. 325–343.

[17] J. Boyar, M. G. Find, and R. Peralta, "On various nonlinearity measures for boolean functions," *Cryptogr. Commun.*, vol. 8, no. 3, pp. 313–330, 2016.

[18] J. Boyar, R. Peralta, and D. Pochuev, "On the multiplicative complexity of boolean functions over the basis (cap, +, 1)," *Theor. Comput. Sci.*, vol. 235, no. 1, pp. 43–57, 2000.

[19] R. K. Brayton and A. Mishchenko, "ABC: an academic industrial-strength verification tool," in *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, 2010, pp. 24–40.

[20] Ç. Çalik, M. S. Turan, and R. Peralta, "The multiplicative complexity of 6-variable boolean functions," *Cryptography and Communications*, vol. 11, no. 1, pp. 93–107, 2019.

[21] ——, "Boolean functions with multiplicative complexity 3 and 4," *Cryptogr. Commun.*, vol. 12, no. 5, pp. 935–946, 2020.

[22] S. Cimato, V. Ciriani, E. Damiani, and M. Ehsanpour, "An obdd-based technique for the efficient synthesis of garbled circuits," in *Security and Trust Management - STM*, 2019, pp. 158–167.

[23] V. Ciriani, "Synthesis of SPP Three-Level Logic Networks using Affine Spaces," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 22, no. 10, pp. 1310–1323, 2003.

[24] M. G. Find, D. Smith-Tone, and M. S. Turan, "The number of boolean functions with multiplicative complexity 2," *Int. J. Inf. Coding Theory*, vol. 4, no. 4, pp. 222–236, 2017.

[25] D. Goudarzi and M. Rivain, "On the multiplicative complexity of boolean functions and bitsliced higher-order masking," in *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, Proceedings*, 2016, pp. 457–478.

[26] I. Halecek, P. Fiser, and J. Schmidt, "Are xors in logic synthesis really necessary?" in *20th IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems, DDECS 2017, Dresden, Germany, April 19-21, 2017*, 2017, pp. 134–139.

[27] I. Hálecek, P. Fiser, and J. Schmidt, "Towards AND/XOR balanced synthesis: Logic circuits rewriting with XOR," *Microelectron. Reliab.*, vol. 81, pp. 274–286, 2018.

[28] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free xor gates and applications," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2008, pp. 486–498.

[29] R. Liebler, *Basic Matrix Algebra with Algorithms and Applications*. Chapman & Hall/CRC., 2003.

[30] F. Luccio and L. Pagli, "On a New Boolean Function with Applications," *IEEE Transactions on Computers*, vol. 48, no. 3, pp. 296–310, 1999.

[31] P. McGeer, J. Sanghavi, R. Brayton, and A. Sangiovanni-Vincentelli, "ESPRESSO-SIGNATURE: A New Exact Minimizer for Logic Functions," *IEEE Transactions on VLSI*, vol. 1, no. 4, pp. 432–440, 1993.

[32] K. Nyberg, "On the construction of highly nonlinear permutations," in *Advances in Cryptology - EUROCRYPT '92, Workshop on the Theory and Application of of Cryptographic Techniques, Balatonfüred, Hungary, May 24-28, 1992, Proceedings*, ser. Lecture Notes in Computer Science, R. A. Rueppel, Ed., vol. 658. Springer, 1992, pp. 92–98.

[33] M. S. Riazi, M. Javaheripi, S. U. Hussain, and F. Koushanfar, "Mpcircuits: Optimized circuit generation for secure multi-party computation," Cryptology ePrint Archive, Report 2019/275, 2019, https://eprint.iacr.org/2019/275.

[34] C. Schnorr, "The multiplicative complexity of boolean functions," in *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 6th International Conference, AAECC-6, Rome, Italy, July 4-8, 1988, Proceedings*, ser. Lecture Notes in Computer Science, T. Mora, Ed., vol. 357. Springer, 1988, pp. 45–58.

[35] M. Soeken, L. G. Amarù, P. Gaillardon, and G. D. Micheli, "Exact synthesis of majority-inverter graphs and its applications," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 36, no. 11, pp. 1842–1855, 2017.

[36] E. Testa, M. Soeken, H. Riener, L. Amaru, and G. D. Micheli, "A logic synthesis toolbox for reducing the multiplicative complexity in logic networks," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 568–573.

[37] E. Testa, M. Soeken, L. G. Amarù, and G. D. Micheli, "Reducing the multiplicative complexity in logic networks for cryptography and security applications," in *Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA*, 2019, p. 74.

[38] ——, "Logic synthesis for established and emerging computing," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 165–184, 2019.

[39] M. S. Turan and R. Peralta, "The multiplicative complexity of boolean functions on four and five variables," in *Lightweight Cryptography for Security and Privacy - Third International Workshop, LightSec 2014, Istanbul, Turkey*, 2014, pp. 21–33.

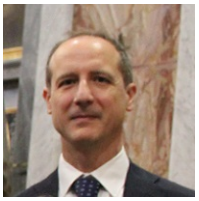
[40] I. Wegener, *The Complexity of Boolean Functions*. John Wiley & Sons, 1987.

[41] S. Yang, "Logic synthesis and optimization benchmarks user guide version 3.0," Microelectronic Center, User Guide, 1991.

[42] A. C. Yao, "Protocols for secure computations," in *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*. IEEE, 1982, pp. 160–164.

[43] A. C.-C. Yao, "How to generate and exchange secrets," in *Foundations of Computer Science, 1986., 27th Annual Symposium on*. IEEE, 1986, pp. 162–167.

Anna Bernasconi received the Laurea degree in Physics from the University of Pavia, Italy, in 1992, and the Ph.D. in Computer Science from the University of Pisa, Italy, in 1998. Her doctoral dissertation Mathematical techniques for the analysis of Boolean functions received the Doctoral Dissertation Thesis Award 1998 from the Italian Chapter of the European Association for Theoretical Computer Science (EATCS). She is currently an Associate Professor with the Department of Computer Science of the University of Pisa, where she teaches fundamental courses in the Computer Science Program. Her research interests include algorithms and complexity, Boolean function complexity, as well as combinational logic synthesis. She has authored or coauthored more than 80 research papers, published in international journals, conference proceedings, books and books chapters.



Stelvio Cimato is an Associate Professor at the Computer Science Department of the Università degli Studi di Milano. He got his Ph.D in Computer Science at University of Bologna, Italy in 1999. His main research interests are in the area of cryptography, network security, and Web applications. He has published several papers in the field and is active in the community, serving as member of the program committee of several international conferences in the area of cryptography and data security. He is Senior Member of IEEE since 2017.

Valentina Ciriani received the Laurea degree and the Ph.D. degree in Computer Science from the University of Pisa, Italy, in 1998 and 2003, respectively. In 2003 and 2004, she was with the Department Computer Science at University Pisa, Italy as a Ph.D. fellow. From January 2005 to February 2015, she was an assistant professor with the Department Information Technologies and the Department of Computer Science, Università degli Studi di Milano, Italy. She is currently an Associate Professor in Computer Science with the Department of Computer Science, Università degli Studi di Milano. Her research interests include algorithms and data structures, as well as combinational logic synthesis, VLSI design of low power circuits and testing of Boolean circuits. She has authored or coauthored more than 100 research papers, published in international journals, conference proceedings, and books chapters. She is Senior Member of IEEE since 2019.

Maria Chiara Molteni received the Laurea degree in Mathematics from the Università degli Studi di Trento (Italy) in 2016. She is currently a Ph.D student with the Computer Science Department of the Università degli Studi di Milano (Italy). Her interests include probing security and side-channel attacks, as well as multi-party computation and multi-valued logic.