Cryptography and Communications (2019) 11:1339–1362
https://doi.org/10.1007/s12095-019-00377-3

Upper bounds on the multiplicative complexity of symmetric Boolean functions

Luís T. A. N. Brandão[1] · **Çağdaş Çalık**[1] · **Meltem Sönmez Turan**[1] · **René Peralta**[1]

Received: 20 September 2018 / Accepted: 27 May 2019 / Published online: 17 August 2019

Abstract
A special metric of interest about Boolean functions is multiplicative complexity (MC): the minimum number of AND gates sufficient to implement a function with a Boolean circuit over the basis {XOR, AND, NOT}. In this paper we study the MC of symmetric Boolean functions, whose output is invariant upon reordering of the input variables. Based on the Hamming weight method from Muller and Preparata (J. ACM **22**(2), 195–201, 1975), we introduce new techniques that yield circuits with fewer AND gates than upper bounded by Boyar et al. (Theor. Comput. Sci. **235**(1), 43–57, 2000) and by Boyar and Peralta (Theor. Comput. Sci. **396**(1–3), 223–246, 2008). We generate circuits for all such functions with up to 25 variables. As a special focus, we report concrete upper bounds for the MC of elementary symmetric functions Σ_k^n and counting functions E_k^n with up to $n = 25$ input variables. In particular, this allows us to answer two questions posed in 2008: both the elementary symmetric Σ_4^8 and the counting E_4^8 functions have MC 6. Furthermore, we show upper bounds for the maximum MC in the class of n-variable symmetric Boolean functions, for each n up to 132.

Keywords Symmetric Boolean functions · Multiplicative complexity · Upper bounds · Logic minimization

Mathematics Subject Classification (2010) 94A60 · 06E30

This article is part of the Topical Collection on *Special Issue on Boolean Functions and Their Applications*

✉ Luís T. A. N. Brandão
luis.brandao@nist.gov

Çağdaş Çalık
cagdas.calik@nist.gov

Meltem Sönmez Turan
meltem.turan@nist.gov

René Peralta
rene.peralta@nist.gov

[1] Cryptographic Technology Group, National Institute of Standards and Technology, 100 Bureau Drive, Gaithersburg, MD 20899, USA

1 Introduction

Multiplicative complexity (MC) is an important characterizing property of Boolean functions. The MC of a Boolean function is the minimum number of fan-in 2 multiplications (AND gates) sufficient to implement the function by a Boolean circuit over the basis (AND, XOR, NOT). The cost of various secure cryptographic implementations is often proportional to the number of AND gates in Boolean circuits implementing the underlying functions. This happens, for example, with the communication complexity of zero-knowledge proofs and secure computation protocols, as well as with the amount of randomness required to implement certain protections against side-channel attacks. Despite the importance of MC, it is often computationally infeasible to measure it for arbitrary Boolean functions with a number of variables as small as seven.

In this paper we focus on symmetric Boolean functions, whose output is determined only by the Hamming weight of the input. Symmetric functions include several fundamental subclasses [1]. They are relevant in diverse areas, such as cryptography [2], logic circuit design [3, 4] and sorting networks [5]. In particular, better efficiency can stem from the use of symmetric Boolean functions, since they can be described more succinctly than arbitrary Boolean functions. We devise new techniques that enable us to obtain new upper bounds on MC, often tight, for symmetric Boolean functions with a large number of variables. This is a step towards a more comprehensive characterization of the MC of Boolean functions.

1.1 Previous work

Functions having symmetries among their input variables have more efficient implementations than those without symmetries [3]. In 1975, Muller and Preparata [5] proposed the *Hamming weight* method to implement symmetric functions using a circuit in two phases: the first phase computes the binary representation of the weight of the input; the second phase finalizes the computation with a function of the concise encoding of the weight. They also showed that it is possible to implement symmetric functions with circuits of fan-in 2 gates having linear size and logarithmic depth.

Implementations with fewer AND gates are preferable in many applications, such as secure multi-party computation (e.g., [6]), fully homomorphic encryption (e.g., [7]), and zero-knowledge proofs (e.g., [8]), where processing AND gates is more expensive than processing XOR gates. Also, 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 $\mathbb{F}_{2^n}$ [9].

The multiplicative complexity of Boolean functions is asymptotically exponential in the number of input variables [10] and it is computationally difficult to calculate even for a small number of variables [11]. Find et al. characterized the Boolean functions with multiplicative complexity 1 and 2 [12]. The classification of functions with respect to multiplicative complexity is known up to 6-variables [13, 14].

Using the Hamming weight method, and upper bounds for the MC of Boolean functions, Boyar et al. showed in 2000 that n-variable symmetric Boolean functions do not require more than $n + 3\sqrt{n}$ AND gates. They also showed that all n-variable symmetric functions can be *simultaneously* computed using at most $2n - \log_2 n$ AND gates.

In 2008, Boyar and Peralta [15] showed that the multiplicative complexity of computing the binary representation of the Hamming weight of n bits is exactly $n - hw(n)$, where $hw(n)$

is the Hamming weight of the binary representation of n. Their paper also showed that any symmetric function on seven variables has multiplicative complexity at most eight. Boyar and Peralta presented the multiplicative complexity of some of the important classes of symmetric functions, such as elementary symmetric (denoted Σ_k^n) and (exactly-k) counting functions (denoted E_k^n) for up to eight variables, in which the multiplicative complexities of two specific functions Σ_4^8 and E_4^8 were left as open problems [15]. A related question that remained unanswered was whether the multiplicative complexity of Σ_k^n is monotonic in k.

1.2 Our results

This paper improves the Hamming-weight method for constructing efficient circuits for symmetric Boolean functions. This includes alternative methods of encoding the Hamming weight, based on the arity and the degree of functions. It also includes new techniques and insights on how to optimize the second phase of the computation. Some results are further improved based on the computational ability to find MC-optimal circuits for some functions with MC up to 6. Based on these techniques, we provide new upper bounds, often tight, for the MC of a large number of symmetric functions. As concrete contributions, we present upper bounds for:

- the MC of each elementary symmetric and counting function with up to 25 variables; in particular, we answer two open questions, by showing that the exact MC of Σ_4^8 and E_4^8 is 6 (and providing concrete circuits), and that the MC of Σ_k^n is not monotonic in k.
- the maximum MC among the set of n-variable symmetric Boolean functions ($\mathcal{S}_n$), for each n up to 132; (e.g., any $f \in \mathcal{S}_n$ has MC smaller than n, for any n up to 21);

We also calculated upper bounds on the MC of all symmetric Boolean functions with up to 25 variables, and as a summary present a table with the number of n-variable functions found for each upper bound.

1.3 Organization

The remainder of the paper is organized as follows. Section 2 gives preliminary definitions and results about Boolean functions, symmetric functions and MC. Section 3 describes the Hamming weight method for constructing circuits for symmetric Boolean functions. Section 4 explains the new techniques to improve the MC upper bounds and obtain concrete circuits with low number of AND gates. Section 5 includes results of the application of the techniques. Section 6 concludes with final remarks. Appendix A shows upper bounds for the MC of concrete elementary-symmetric and exactly-counting functions. Appendix B shows upper bounds for the maximum MC within classes of fixed-arity symmetric functions.

2 Preliminaries

2.1 Boolean functions

Let $\mathbb{F}_2$ be the finite field with two elements. An n-variable Boolean function f is a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. The *arity* of f is the number n of input variables. $\mathcal{B}_n$ is the set of n-variable Boolean functions. There are $\#(\mathcal{B}_n) = 2^{2^n}$ such functions.

The *algebraic normal form* (ANF) of a Boolean function $f \in B_n$ is the multivariate polynomial representation defined by

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

where $a_{\vec{u}} \in \mathbb{F}_2$ and $x^{\vec{u}} = x_1^{u_1} \wedge x_2^{u_2} \wedge \cdots \wedge x_n^{u_n}$ is composed of the Boolean variables x_i for which $u_i = 1$.

The *degree* of a monomial $x^{\vec{u}}$ is the number of variables appearing in that monomial. The degree $deg(f)$ of a Boolean function f is the highest degree among the monomials that appear in the ANF.

The *truth table* T_f of a Boolean function f is the list, lexicographically ordered by the input vectors $\overrightarrow{v_i} \in \mathbb{F}_2^n$, of the output values of f:

$$T_f = \left(f(\overrightarrow{v_0}), f(\overrightarrow{v_1}), \ldots, f(\overrightarrow{v_{2^n-1}})\right). \tag{2}$$

2.2 Symmetric Boolean functions

A *symmetric* n-variable Boolean function f has its output invariant under any permutation of its input bits x_i, i.e.,

$$f(x_1, x_2, \ldots, x_n) = f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}), \tag{3}$$

for all permutations π of $\{1, 2, \ldots, n\}$. $\mathcal{S}_n$ denotes the set of n-variable symmetric Boolean functions. There are $\#(\mathcal{S}_n) = 2^{n+1}$ such functions.

The Hamming weight (HW), or simply *weight*, of a bit vector $\vec{x}$ is the integer number $hw(\vec{x})$ of bits with value 1.

Since the output of a symmetric function f depends only on the weight of the input, the function can be represented using the $(n+1)$-bit *value vector* $w(f) = (w_0, \ldots, w_n)$,such that $f(\vec{x}) = w_i$ if $hw(\vec{x}) = i$.

The *counting* function E_k^n is the n-variable symmetric Boolean function that outputs 1 if and only if the weight of the input is k. Any symmetric Boolean function $f \in \mathcal{S}_n$ can be expressed as a sum of a unique subset of as a sum of counting functions:

$$f(\vec{x}) = w_0 E_0^n(\vec{x}) \oplus w_1 E_1^n(\vec{x}) \oplus w_2 E_2^n(\vec{x}) \oplus \cdots \oplus w_n E_n^n(\vec{x}). \tag{4}$$

The *elementary symmetric* function Σ_k^n is the n-variable Boolean function composed of all degree-k monomials, i.e.,

$$\Sigma_k^n(\vec{x}) = \bigoplus_{1 \le i_1 < \ldots < i_k \le n} x_{i_1} \wedge \cdots \wedge x_{i_k}. \tag{5}$$

Any symmetric Boolean function $f \in \mathcal{S}_n$ can be expressed as a sum of a unique subset of as a sum of elementary symmetric functions. The *simplified* ANF of f is the $(n+1)$-bit value vector $v(f) = (v_0, \ldots, v_n)$ satisfying

$$f(\vec{x}) = v_0 \Sigma_0^n(\vec{x}) \oplus v_1 \Sigma_1^n(\vec{x}) \oplus v_2 \Sigma_2^n(\vec{x}) \oplus \cdots \oplus v_n \Sigma_n^n(\vec{x}). \tag{6}$$

The vectors $v(f)$ and $w(f)$, defining the coefficients used in the decomposition of a symmetric Boolean function f, respectively as a sum of counting functions and as a sum of elementary-symmetric functions, satisfy a linear relation:

$$w_k = \bigoplus_{i=0}^{k} \binom{k}{i} v_i \,(\mathrm{mod}\ 2), \tag{7}$$

where $\binom{k}{i}$ is the binomial coefficient k choose i.

The elementary symmetric function of degree k can be expressed as a product of elementary symmetric functions as follows [15]:

$$\Sigma_k^n = \Sigma_{2^{i_0}}^n \wedge \Sigma_{2^{i_1}}^n \wedge \cdots \wedge \Sigma_{2^{i_j}}^n, \tag{8}$$

where $(i_0, \ldots, i_j)$ is the binary representation of k.

2.3 Multiplicative complexity

With respect to multiplicative complexity (MC), the basis (`AND`, `XOR`, `NOT`) is equivalent to (`AND`, `XOR`, `1`), since evaluating a `NOT` gate is equivalent to computing an XOR with the constant 1. MC is also the number of nonlinear gates needed to implement the function even when the set of all fan-in 2 and fan-in 1 gates are available, since any nonlinear gate can be implemented using one `AND` gate and other auxiliary linear gates.

Let $C_\wedge(f)$ denote the MC of the function f. The *degree bound* [16] states that the MC of functions having algebraic degree d is at least $d-1$. A bound is denoted *tight* when it is simultaneously a lower bound and an upper bound.

Let $\mathrm{MC}_{\max}(S)$ denote the maximum MC across all the functions in a set S. Sets of interest include $\mathcal{B}_n$ and $\mathcal{S}_n$. Table 1 shows upper bounds for the MC of Boolean functions on n variables, for n between 2 and 16. For n up to 6, the bounds are tight and were obtained from prior studies that characterized the MC of Boolean functions [13, 14]. For n larger than 6, the known bounds are likely much looser. This significantly impacts the bounds that we can obtain in this work. The upper bound for $n = 7$ is derived from the relation $\mathrm{MC}_{\max}(\mathcal{B}_n) \le 1 + 2 \cdot \mathrm{MC}_{\max}(\mathcal{B}_{n-1})$, obtained from the decomposition

$$f(x_1, \ldots, x_n) = x_n \wedge f'(x_1, \ldots, x_{n-1}) \oplus f''(x_1, \ldots, x_{n-1}). \tag{9}$$

For $n \ge 8$, Table 1 uses the MC upper bound expressed in (10), as can be obtained from prior work [10, 14] (see Remark 1):

$$\mathrm{MC}_{\max}(\mathcal{B}_n) \le (2+b) \cdot 2^{(n-b)/2} - (n+4+b)/2, \tag{10}$$

where $b = \mathrm{mod}_2(n)$ is the integer 0 or 1 corresponding to the parity of n.

Remark 1 The upper bound obtained in ref. [10, Theorem 6] for the MC of functions in $\mathcal{B}_n$ resulted from a recursion that should stop at an optimal depth; a close examination reveals

Table 1 Upper bounds on the MC of Boolean functions

n	2	3	4	5	6	7	8	9[a]	10	11[a]	12	13[a]	14	15[a]	16
$MC \le$	1	2	3	4	6	13	26	41	57	88	120	183	247	374	502

[a]The upper bounds for $n \in \{9, 11, 13, 15\}$ use the improvement described in Remark 1

that for odd n the optimal depth is $d = (n-1)/2$ instead of $d = (n+1)/2$ (used therein); the former improves the upper bound by 1, as already reflected in (10).

3 Hamming weight method

This section describes the *Hamming weight method* to implement symmetric Boolean functions. Since the output of a symmetric Boolean function $f \in \mathcal{S}_n$ depends only on $hw(\vec{x})$, the method first computes the vectorial Boolean function H_{BR} that outputs the binary representation of the weight of the input; then it applies a final function g to calculate f. In essence, the output $f(\vec{x}) = g(H_{\mathrm{BR}}(\vec{x}))$ is obtained as a composition $g \circ H$ of two functions.

3.1 Phase I — computing the hamming weight

The function H_{BR} maps the input vector $(x_1, \ldots, x_n)$ to the output vector $(y_{s-1}, \ldots, y_0)$ of length $s = \lceil \log_2(n+1) \rceil$, satisfying the integer sum

$$x_1 + \cdots + x_n = 2^{s-1} y_{s-1} + \cdots + 2y_1 + y_0. \tag{11}$$

Komamiya [17] showed that the i^{th} least-significant bit of the binary representation of the Hamming weight evaluates to the elementary symmetric function $\Sigma^n_{2^{i-1}}$. Hence,

$$(y_{s-1}, y_{s-2}, \ldots, y_0) = \left(\Sigma^n_{2^{s-1}}, \Sigma^n_{2^{s-2}}, \ldots, \Sigma^n_{2^0}\right). \tag{12}$$

The main building blocks to construct circuits for H_{BR} include *half adders* and *full adders*. Each adder computes the binary representation (a pair of bits, denoted *carry* and *sum*) of the integer sum of the (respectively two or three) input bits, as follows:

– A *half adder* (HA) adds two binary inputs, x_1 and x_2, and generates a *carry* bit and a *sum* bit, satisfying $x_1 + x_2 = 2 \cdot carry + sum$, as in

$$\mathrm{HA}(x_1, x_2) = (carry, sum) = (x_1 \wedge x_2, x_1 \oplus x_2). \tag{13}$$

– A *full adder* (FA) adds three binary inputs, x_1, x_2 and x_3, and generates a *carry* bit and a *sum* bit, satisfying $x_1 + x_2 + x_3 = 2 \cdot carry + sum$, as in

$$\mathrm{FA}(x_1, x_2, x_3) = (carry, sum) = (maj(x_1, x_2, x_3), x_1 \oplus x_2 \oplus x_3) \tag{14}$$

where the majority function maj outputs 1 if at least two of its input bits are equal to 1 and outputs 0 otherwise. $C_\wedge(maj) = 1$, and it can be implemented as

$$maj(x_1, x_2, x_3) = ((x_1 \oplus x_2) \wedge (x_1 \oplus x_3)) \oplus x_1. \tag{15}$$

Muller and Preparata [5] showed that, for any given n, it is possible to compute the binary representation of the Hamming weight of n variables by using exactly $n - \lceil \log_2(n+1) \rceil$ FAs and $\lceil \log_2(n+1) \rceil - hw(n)$ (i.e., the number of 0's in the binary representation of n) HAs. Figure 1 provides an example circuit for $n = 10$, using 6 FAs and 2 HAs. Boyar and Peralta [15] provided a proof that such number is in fact optimal in terms of MC.

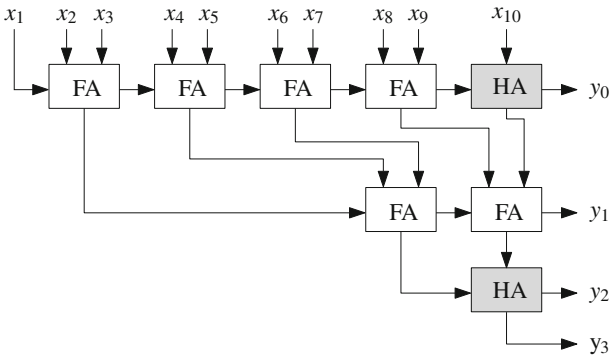


Fig. 1 An example Hamming weight circuit for $n = 10$

3.2 Phase II — computing *g*

Once the binary representation $\vec{y} = H_{\mathrm{BR}}(\vec{x})$ of the weight of the input $\vec{x}$ is computed, the computation of a symmetric Boolean function f still requires the computation of a function g satisfying $g(\vec{y}) = f(\vec{x})$ for any $\vec{x} \in \{0,1\}^n$.

In the method of Boyar et al. [8], the function f is first expressed as a sum of elementary symmetric functions Σ_i^n, and then each elementary symmetric function is written as a product (see (8)) of the output bits of $H_{\mathrm{BR}} : \sum_{2^{s-1}}^n, \sum_{2^{s-2}}^n, \ldots, \sum_{2^0}^n$. The function g is then defined constructively, by replacing the latter by the corresponding input variables of g.

Example 1 Let $f = \Sigma_9^{10} \oplus \Sigma_7^{10} \oplus \Sigma_3^{10}$. The binary representation of the Hamming weight calculated with H_{BR} outputs Σ_1^{10}, Σ_2^{10}, Σ_4^{10} and Σ_8^{10}. The function f can then be written as:

$$\begin{aligned} f &= \Sigma_9^{10} \oplus \Sigma_7^{10} \oplus \Sigma_3^{10} \\ &= \Sigma_8^{10} \wedge \Sigma_1^{10} \oplus \Sigma_4^{10} \wedge \Sigma_2^{10} \wedge \Sigma_1^{10} \oplus \Sigma_2^{10} \wedge \Sigma_1^{10}. \end{aligned} \tag{16}$$

Letting $y_i = \Sigma_{2^i}^{10}$, for $i = 0, \ldots, 3$, it follows from (12) that

$$\begin{aligned} g(y_0, y_1, y_2, y_3) &= g\left(\Sigma_1^{10}, \Sigma_2^{10}, \Sigma_4^{10}, \Sigma_8^{10}\right) \\ &= y_0 \wedge y_1 \oplus y_0 \wedge y_1 \wedge y_2 \oplus y_0 \wedge y_3. \end{aligned} \tag{17}$$

3.3 Upper bounds on the MC of symmetric functions using the HW method

The number of AND gates used in a composition of two circuits is equal to the sum of AND gates in those two circuits. Furthermore, the MC of the second part can be upper bounded by $\mathrm{MC}_{\max}(\mathcal{B}_{|H|})$, where $|H|$ is the number of output bits of the first part H. This yields the following upper bound for the MC of symmetric Boolean functions:

$$(f = g \circ H) \implies C_\wedge(f) \leq C_\wedge(H) + \mathrm{MC}_{\max}(\mathcal{B}_{|H|}). \tag{18}$$

The above expression can be refined for the case of the Hamming weight method. There, the MC of $H = H_{\mathrm{BR}}$ is exactly $n - hw(n)$ [10] and the output length is exactly $\lceil \log_2(n+1) \rceil$. Thus, an upper bound for the MC of any n-variable symmetric function f can be expressed as

$$C_\wedge(f) \leq \mathrm{MC}_{\max}(\mathcal{S}_n) \leq n - hw(n) + \mathrm{MC}_{\max}(\mathcal{B}_{\lceil \log_2(n+1) \rceil}). \tag{19}$$

Plugging (10) into (19) yields an upper bound for $\mathrm{MC}_{\max}(\mathcal{S}_n)$ that is upper bounded by a function linear in n. The exact expression is somewhat complicated; a simpler upper bound [10, Corollary 9] is

$$\mathrm{MC}_{\max}(\mathcal{S}_n) \leq n + 3\sqrt{n}. \tag{20}$$

It is not known whether or not $\mathrm{MC}_{\max}(\mathcal{S}_n)$ is $n + \Theta(\sqrt{n})$. It is conceivable that $\mathrm{MC}_{\max}(\mathcal{S}_n)$ is $n + \Theta(polylog(n))$.

4 New methods using fewer AND gates

The Hamming weight method does not always lead to MC-optimal circuits [10]. In this section we show novel computation paths, which enable improved circuit constructions in terms of the number of AND gates, often achieving MC optimality still within the paradigm $f = g \circ H$. In summary, the optimizations are categorized, based on the modifications to the HW method, in three types:

1. **Arity-based HW-encodings.** Use alternative weight encodings H that depend only on the number n of variables of f, but which can have MC smaller than $C_\wedge(H_{\mathrm{BR}})$ and lead to symmetries in the truth-table entries of the corresponding g. These alternatives allow a tradeoff between the MC and the output length of H and possibly the MC of g.
2. **Degree-based HW-encoding.** Use alternative weight encodings H, depending on the degree of f, that can eliminate unnecessary sub-computations of H_{BR} that would only be useful for other symmetric functions of higher degree.
3. **Free truth-table entries.** Depending on H, choose a g with minimal MC among a set of alternatives that can evaluate the target f.

Besides the above, some concrete MC bounds devised in this paper also take advantage of the computational ability to find the MC of functions with fewer than 7 variables [14]. For 7 and 8 variables it is also feasible to confirm whether or not a function has MC less than or equal to 5.

4.1 Arity-based HW-encodings

The first phase of the Hamming weight method computes H_{BR}, outputting the binary representation of the Hamming weight of some n-variable input. This representation, containing $s = \lceil \log_2(n+1) \rceil$ bits, is optimally concise and requires exactly $C_\wedge(H_{\mathrm{BR}}) = n - hw(n)$ AND gates. However, for some arities n there are alternative HW encodings that achieve optimal conciseness (s output bits) at the cost of fewer AND gates. Using any such encoding necessarily leads to a better upper-bound for the MC of f, if estimated based on (18). In that case, the MC for H is improved, whereas the estimated upper bound for the MC of g remains the same (since it depends only on s).

More generally, improvements can often be obtained from reasonably concise encodings, with t output bits (with $s \leq t$), even if not of minimal length s. If t is small enough, then this can sometimes be leveraged to construct a circuit for f using fewer AND gates than a circuit based on H_{BR}. This can happen when the difference (e.g., 1 or 2) in number of AND gates due to the difference $t - s$ in output length is small enough (e.g., 1 or 2, if $t - s = 1$ and t is 5 or 6, respectively) to compensate for a larger number of what would otherwise be unneeded or non-optimal AND gates used by H_{BR}. This section considers this avenue of improvements.

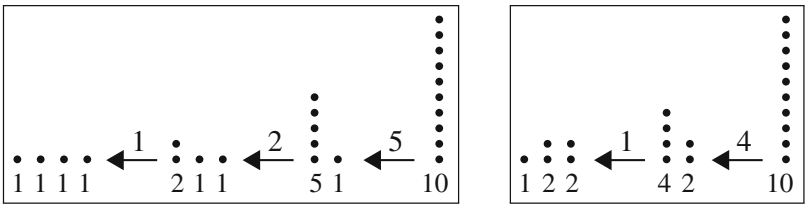
4.1.1 Intuition

In the Hamming weight method, the first phase (computing H_{BR}) reduces the number of variables while retaining full information about the Hamming weight. At each step of the process of going from n to s variables, each initial, intermediate or final variable assigns a weight 2^i to its bit. Thus, each such variable represents either a 0 or a positive integer 2^i to contribute to the overall Hamming weight. It is useful to look at full adders (FA) and half adders (HA) in this perspective:

- an FA consumes three symmetric variables of weight 2^i and outputs two variables, one of weight 2^{i+1} and another of weight 2^i;
- an HA consumes two symmetric variables of weight 2^i and also outputs two variables, one of weight 2^{i+1} and another of weight 2^i;

The input for H is an (inefficient) encoding that uses n variables (the original input), each of weight 1. The encoding process then consists of progressing over a sequence of states, each of which is itself an encoding. Each state is characterized by the tally of variables corresponding to each weight. Each step — a state transition — is induced by the application of an FA or an HA. An HA transition, associated with some weight 2^i, retains the overall number of variables: one extra variable with weight 2^{i+1} and one fewer with weight 2^i. The corresponding FA transition reduces by one the overall number of variables: one extra variable with weight 2^{i+1} and two fewer with weight 2^i. The reduction in number of variables can be concisely expressed in "dot notation", as exemplified in Fig. 2 for $n = 10$ and for two different HW encodings: H_{BR} is the encoding used in the HW method; H_{FA} is one using only full adders (further discussed in Section 4.1.2).

Dot notation The dot notation is a simple and short-hand notation useful for reflecting on the result of an encoding. Each variable is represented by a dot, and each column of dots represents all variables with the same weight. The weight associated with each column doubles between each adjacent column, from right to left. For each state, the rightmost column corresponds to weight $2^0 = 1$. In the initial state there are n variables encoding weight 1 (i.e., each variable encodes the actual input bit value). Thus, in the initial state there is a single column, and it has n dots. The i^{th} column to the left, when it exists, corresponds to weight 2^i. Each application of an HA or an FA in a column removes 1 or 2 dots, respectively, in that column, and in both cases it adds one dot to the adjacent column to the left. Figure 2 shows, for the case of $n = 10$ input variables, several intermediate states for two different



(a) Reduction via H_{BR} (b) Reduction via H_{FA}

Fig. 2 "Dots" of variable reduction states for $n = 10$ variables

encodings (H_{BR} and H_{FA}). Figure 2a shows the H_{BR} reduction, where progressively each column is reduced to a single dot. Figure 2b shows a reduction (dubbed H_{FA}) which only uses FAs, therefore not proceeding in a column when there are fewer than 3 dots. The number of AND gates used between states is shown over the arrows.

Calculating the dot configuration The final dot configuration upon applying H_{BR} is simply a row of s dots, where $s = \lceil \log_2(n+1) \rceil$.

For the H_{FA} encoding the dot configuration is slightly more complex. The resulting number of columns is $\lfloor \log_2(n+1) \rfloor$, which means it can either be s or $s-1$. If there are fewer columns, then at least one column will have two dots, since the overall number t of dots cannot be smaller than s. From each column with c dots, the reduction of that column produces exactly $\lfloor (c-1)/2 \rfloor$ dots in the adjacent column, and leaves exactly $2 - \text{mod}_2(c)$ dots (one or two) in the original column. Since one variable is reduced for each FA, the final number of dots is equal to $n - \#\text{FA}(n)$. Applying this transformation iteratively, and summing the dots in all the columns, yields the total number of dots (i.e., variables) when applying FA:

$$t = hw(n+1) - 1 + \lfloor \log_2(n+1) \rfloor . \tag{21}$$

4.1.2 HW encodings using only full adders

The first alternative encoding we propose uses only full adders and is denoted by H_{FA}. It reduces one variable for each used AND gate, and it uses as many as possible while preserving the information of the Hamming weight. When no more full-adder operations are possible in these conditions, the output is given as input to phase 2 for a corresponding function g, denoted g_{FA}.

Interestingly, H_{FA} often leads to encodings as concise as H_{BR}. Particularly, this happens exactly to all values of the form $n = 2^i + 2^j - 1$. Of these, the cases with $i \neq j$ require fewer AND gates than required by H_{BR}.

Even when H_{FA} outputs a number t of variables slightly larger than the number s output by H_{BR}, the approach may still enable better upper-bounds. The underlying intuition is that each AND gate in H_{FA} reduces the number of variables by one, whereas H_{BR} uses half adders, which do not reduce the number of variables. If the number t of variables output by H_{FA} is small enough, e.g., up to 5 or 6, then the generic upper bound $\text{MC}_{\max}(\mathcal{B}_t)$ (in Table 1) may be good enough to compensate the differential $t - s$. (Section 5 will further consider an optimized computation of g_{FA}.)

Comparing H_{BR} vs. H_{FA} Table 2 compares, for all arities up to $n = 22$, the results of applying HW encodings H_{FA} vs. H_{BR}. The table compares side-by-side the MC, output length and "dots" of the two HW encodings, as well as the generic upper bounds for the $\text{MC}_{\max}$ of the corresponding g and $\mathcal{S}_n$ (obtained by applying (18)). There are five cases to analyze:

- $\boldsymbol{n = 2^i - 1}$. For $n \in \{1, 3, 7, 15\}$, the final dots form a single row, i.e., there are no columns with two dots, meaning H_{FA} outputs the same as H_{BR}.
- $\boldsymbol{s = t}$ **and** $\boldsymbol{n \neq 2^i - 1}$. For $n \in \{2, 4, 5, 8, 9, 11, 16, 17, 19\}$, the output length t of H_{FA} is optimal ($= s$) but the final dots do not form a single row. This means that H_{FA} avoided some inefficient operations that H_{BR} would have performed. This allows a better upper-bound for $\text{MC}_{\max}(\mathcal{S}_n)$.
- $\boldsymbol{s < t \leq 5}$. For $n \in \{6, 10, 12, 13\}$, the encoding H_{FA} does not reduce the number of variables to the minimum, but the generic upper bound for the $\text{MC}_{\max}$ of g is still $t - 1$, therefore still leading to $\text{MC}_{\max}(\mathcal{S}_n) = n - 1$.

Table 2 Comparison of H_{BR} and H_{FA} encodings and results

n	Phase 1: H (HW encoding)						Phase 2: g		$f = g \circ H$	
	MC		Output length $\lvert H \rvert$		Dots		Upper bound on $\mathrm{MC}_{\max}(\mathcal{B}_{\lvert H \rvert})$		Upper bound on $\mathrm{MC}_{\max}(\mathcal{S}_n)$	
	BR	FA	BR	FA	BR	FA	BR	FA	BR	FA
1	0	0	1	1	.	.	0	0	**0**	**0**
2	1	0	2	2	..	:	1	1	2	**1**
3	1	1	2	2	..	..	1	1	**2**	**2**
4	3	1	3	3	...	.:	2	2	5	**3**
5	3	2	3	3	...	:.	2	2	5	**4**
6	4	2	3	4	...	::	2	3	6	**5**
7	4	4	3	3	...	...	2	2	**6**	**6**
8	7	4	4	4	....	..:	3	3	10	**7**
9	7	5	4	4	....	.:.	3	3	10	**8**
10	8	5	4	5	....	.::	3	4	11	**9**
11	8	7	4	4	....	:..	3	3	11	**10**
12	10	7	4	5	....	:.:	3	4	13	**11**
13	10	8	4	5	....	::.	3	4	13	**12**
14	11	8	4	6	....	:::	3	6	14	14[a]
15	11	11	4	4	....	....	3	3	**14**	**14**
16	15	11	5	5	.....	...:	4	4	19	**15**
17	15	12	5	5	.....	..:.	4	4	19	**16**
18	16	12	5	6	.....	..::	4	6	20	18[a]
19	16	14	5	5	.....	.:..	4	4	20	**18**
20	18	14	5	6	.....	.:.:	4	6	22	20[a]
21	18	15	5	6	.....	.::.	4	6	22	21[a]
22	19	15	5	7	.....	.:::	4	13	23	28[a]

[a]A better upper-bound is found for these values in Section 5

- $\boldsymbol{t = 6}$. For $n \in \{14, 18, 20, 21\}$, H_{FA} outputs an encoding with length $t = 6$, for which the generic upper bound for $\mathrm{MC}_{\max}(\mathcal{B}_t)$ is also t, therefore leading to an upper bound for $\mathrm{MC}_{\max}(\mathcal{S}_n)$ that is equal to n. In each of these cases, the techniques in this work allow us to reduce the bound to the degree bound $n - 1$.
- $\boldsymbol{t = 7}$. For $n = 22$, the $\mathrm{MC}_{\max}(\mathcal{S}_n)$ upper bound obtained with H_{FA} is worst than the one obtained with H_{BR}. This happens because the increase by 9 AND gates, between $\mathrm{MC}_{\max}(\mathcal{B}_{t=7})$ and $\mathrm{MC}_{\max}(\mathcal{B}_{s=5})$, in the second phase is larger than the decrease by 4 AND gates, between the MC 19 of H_{BR} and the MC 15 of H_{FA}, in the first phase.

Remark 2 The results mentioned in Table 2 concern the $\mathrm{MC}_{\max}(\mathcal{S}_n)$ upper bound obtained using (18), where an upper bound for $\mathrm{MC}_{\max}(\mathcal{S}_n)$ is estimated generically (Table 1). These results highlight immediate benefits from applying the H_{FA} encoding. However, better upper-bounds will be obtained in Section 5 when complementing the technique with a more refined computation of the $\mathrm{MC}_{\max}$ (and/or better upper-bound) for the set of needed functions g.

4.1.3 HW encodings using one or a few half adders

As mentioned for the cases $n \in \{14, 18, 20, 21, 22\}$ in Table 2, using an encoding H (e.g., H_{FA}) with an output length $t \geq 6$ does not yield, for $\mathcal{S}_n$, a $\mathrm{MC}_{\max}$ upper bound equal to the degree bound $(n-1)$. This happens because starting at $t = 6$ the generic upper bound for $\mathrm{MC}_{\max}(\mathcal{B}_t)$ is larger than $t - 1$. For $t = 6$ the difference between $\mathrm{MC}_{\max}(\mathcal{B}_t)$ and $\mathrm{Mc}_{\max}(\mathcal{B}_{t-1})$ is only 1, but as t increases the difference increases exponentially. For example, for $t = 7$ that difference is already 7, since the corresponding $\mathrm{MC}_{\max}$ bound is 13 (see Table 1). In such cases, a better upper-bound for $\mathrm{MC}_{\max}(\mathcal{S}_n)$, still based on (18), may be obtained by using yet a different encoding, differing from H_{FA} by using one or a few extra HAs that enable subsequent use of more FAs and therefore a corresponding further reduction in the number of variables. The tradeoff to consider is the cost of applying HAs vs. the benefit of reducing the number of variables t to enable a better upper bound for $\mathrm{MC}_{\max}(\mathcal{B}_t)$.

For example, if the use of one HA enables a subsequent use of one extra FA, then at the cost of two AND gates the number of variables is, compared with H_{FA}, further reduced by 1. This is certainly better than incurring an upper bound increase of 7, by relying on $\mathrm{MC}_{\max}(\mathcal{B}_7) = 13$ instead of $\mathrm{MC}_{\max}(\mathcal{B}_6) = 6$. More generally, if j HAs enable the overall use of i FAs, then this enables reducing i variables at the cost of $k = i + j$ AND gates. This begs the question: for each limit j on the number of HAs, what is the maximum number i of FAs that can be used in a way to reduce the number of variables from n to $n - i$?

Notation The generalized encoding using j HAs is denoted by H_j; the number of output variables is denoted by t_j; the number of used AND gates is denoted by k_j (and is equal to $n - t_j + j$). The symbol H_{HA} is used to generically denote an encoding H_j for some implicit j.

The previously explored encoding H_{FA} is the case with $j = 0$, i.e., $H_{\mathrm{FA}} = H_0$ and $t_0 = t$. It is worth exploring the cases where j is also allowed to be a small positive integer, e.g., 1, 2 and 3. With respect to the $\mathrm{MC}_{\max}(\mathcal{S}_n)$ upper bound estimated using (18) (see Remark 2), using H_j with some $j \geq 1$ is only worth over H_{FA} if $t_0 \geq 7$ and simultaneously $k_j - k_0 < \mathrm{MC}_{\max}(\mathcal{B}t_j) - \mathrm{MC}_{\max}(\mathcal{B}t_0)$. For example, if $t_0 \leq 6$, the cost of two ANDs to reduce one variable will never over-compensate the difference in $\mathrm{MC}_{\max}(\mathcal{B}_t)$. More concretely, using $j \geq 1$ is counterproductive if $t_0 \leq 5$; $j \geq 2$ is also counterproductive if $t_0 = 6$.

Comparing H_{BR} vs. H_{HA} (H_0, H_1, H_2) Table 3 shows the results upon application of up to a few half-adders, for $n \in \{22, ..., 36\}$. The table is similar to Table 2, except for replacing H_{FA} by H_j and for adding a new column for the parameter j (defining the number of HAs used in H_j). There are several cases worth analyzing:

- The arity $n = 22$ is the first for which the new technique is helpful. Using one HA directly enables for $\mathrm{MC}_{\max}(\mathcal{S}_n)$ an upper bound equal to 22, whereas H_{BR} and H_{FA} would respectively lead to upper bounds 23 and 28.
- The arity $n = 36$ is the first for which it is useful to use two HAs. In fact, using $H_{\mathrm{FA}} = H_0$ or H_1 leads to an upper bound of 42, which would be worse than the upper bound 40 possible with H_{BR}.
- The arity $n = 28$ is the first where the choice of the position of the first HA application (not detailed in the table) does not happen at the first opportunity to apply it. Doing H_{FA} would lead to $t_0 = 7$ dots with configuration "**::.:**"; then, when deciding to use one HA, instead of starting at the least-significant column (the rightmost one) with two dots, the HA starts at the second such column. The rationale for this is explained further below.

Table 3 Comparison of H_{BR} and H_{HA} encodings and results

n	j	Phase 1: H (HW encoding)						Phase 2: g		$f = g \circ H$	
		MC		Output length $\lvert H \rvert$		Dots		Upper bound on $\mathrm{MC}_{\max}(\mathcal{B}_{\lvert H \rvert})$		Upper bound on $\mathrm{MC}_{\max}(\mathcal{S}_n)$	
		BR	HA	BR	HA	BR	HA	BR	HA	BR	HA
22	**1**	19	18	5	5	.....	:...	4	4	23	22
23	0	19	18	5	5	.....	:...	4	4	23	**22**
24	0	22	18	5	6	.....	:..:	4	6	26	24
25	0	22	19	5	6	.....	:.:.	4	6	26	25
26	**1**	23	21	5	6	.....	::..	4	6	27	27
27	0	23	21	5	6	.....	::..	4	6	27	27
28	**1**	25	23	5	6	.....	....:	4	6	29	29
29	**1**	25	25	5	5	.....	.....	4	4	29	29
30	**1**	26	26	5	5	.....	.....	4	4	30	30
31	0	26	26	5	5	.....	.....	4	4	30	**30**
32	0	31	26	6	6	.....	....:	6	6	37	32
33	0	31	27	6	6	.....	...:.	6	6	37	33
34	**1**	32	29	6	6	.....	..:..	6	6	38	35
35	0	32	29	6	6	.....	..:..	6	6	38	35
36	**2**	34	32	6	6	.....	.:...	6	6	40	38

- For $n \in \{29, 30, 31\}$ the encoding H_j collapses to H_{BR}, respectively using 1, 1 and 0 HAs, leading to a dot configuration composed of a single row of dots.
- For $n \in \{23, 31\}$ the resulting upper bounds on $\mathrm{MC}_{\max}(\mathcal{S}_n)$ are equal to the respective degree lower-bound ($= n - 1$), which means they are tight. It is open for which other values $n \geq 22$ a computation for finding the actual $\mathrm{MC}_{\max}$ of the set of needed functions g_j may enable a better bound.

Where to apply HA As illustrated with the case $n = 28$, there may exist several alternatives for where to apply an HA, and not all are optimal. Indeed, $n = 28$ is the first arity for which this problem arises. For this n, applying H_{FA} would induce a dot configuration "**::.:**". Since the rightmost column does not have any other adjacent column with two dots, the use of a single HA therein would not immediately allow using another FA. However, this is possible when applying a single HA in the least significant column of a set of adjacent columns where all contain two dots. Particularly, both the two leftmost columns have two dots and are adjacent. Thus, the use of one HA in the least significant of these two allows a subsequent use of one FA, leading to a reduction to $t_1 = 6$ dots (with configuration "**....:**").

The larger the number of adjacent columns with two dots each, the better is the result when applying a single HA to the least significant column of the set. However, when more than one HA can be applied, the optimal choice becomes more complex. This observation highlights that the use of HAs must be judicious not only about the allowed number of HAs but also about the variable weights to which they are applied.

4.2 Degree-based HW-encodings

At any stage in the process of Hamming weight computation, the generated intermediate variables are functions of input variables. The degrees of these variables are related to the columns in which they appear in the dot notation, which in turn depend on the preceding processing through FAs and HAs. Specifically, the intermediate variables in the i^{th} rightmost column of the dot notation will have degrees 2^{i-1}. It is reasonable to avoid the generation of intermediate variables whose degree exceeds the degree of the symmetric function one wants to implement. This imposes the condition that FAs will only be applied up to the column corresponding to the degrees that are less than or equal to $\deg(f)$. The technique leads to circuits with fewer AND gates because it eliminates unnecessary multiplications. However, the function g might have a larger number of variables compared to the number of variables remaining in the H_{FA} encoding. However, it leads to circuits with fewer AND gates because unnecessary multiplications are eliminated.

Bounded degree case Let $1 < 2^{s-1} \leq n < 2^s$ and let f be a symmetric Boolean function on n inputs. Then f can be calculated from the values of $\Sigma^n_{2^0}, \ldots, \Sigma^n_{2^{s-1}}$. Calculating the s elementary symmetric functions above requires $n - hw(n)$ AND gates. This yields the following bound:

$$C_\wedge(f) \leq n - hw(n) + \text{MC}_{\max}(\mathcal{B}_s). \tag{22}$$

For example, if $n = 63$ then $2^5 \leq n < 2^6$ and therefore $C_\wedge(f) \leq 63$ since $hw(63) = 6$ and $\text{MC}_{\max}(\mathcal{B}_6) = 6$.

When the degree of the function is less than $n/2$, we can get a new bound on the multiplicative complexity. If f is an n-variable symmetric Boolean function of degree k, and $1 < 2^{r-1} \leq k < 2^r$ then the upper bound is

$$C_\wedge(f) \leq C_\wedge(\Sigma^n_{2^0}, \ldots, \Sigma^n_{2^{r-1}}) + \text{MC}_{\max}(\mathcal{B}_r). \tag{23}$$

Let $\gamma = n \bmod 2^r$. Then, by Lemma 11 of [15],

$$C_\wedge(\Sigma^n_{2^0}, \ldots, \Sigma^n_{2^{r-1}}) \leq \left(\frac{2^{r-1}-1}{2^{r-1}}\right)(n-\gamma) + \gamma - hw(\gamma). \tag{24}$$

For example, let f be a symmetric function of 100 variables with degree $k = 31$. Then $2^4 \leq k < 2^5$, which yields $r = 5$ and $\gamma = 100 \bmod 32 = 4$. Since $\text{MC}_{\max}(\mathcal{B}_5) = 4$ we get the bound $C_\wedge(f) \leq \left(\frac{15}{16}\right)(100-4) + 4 - 1 + 4 = 97$.

4.3 Free entries in the truth table of *g*

After phase 1 encodes the weight, phase 2 finalizes with the computation of g, which combines the bits of the weight. As described in Section 3.2 (e.g., see Example 1), Peralta and Boyar express f as a sum (6) of elementary symmetric functions, and in turn express each term of the sum as a product (8) of elementary symmetric functions of degree equal to a power of two $\Sigma^n_{2^{s-1}}, \Sigma^n_{2^{s-2}}, \ldots, \Sigma^n_{2^0}$. That method provides a unique expression for g, based on f and H_{BR}.

We observe that when n is not of the form $2^i - 1$, then for any $f \in \mathcal{S}_n$ there are several possibilities for g after applying $H = H_{\text{BR}}$. This becomes evident when representing the target symmetric function f as a sum of counting functions (see (4)).

The simplified value vector $(w_0, \ldots, w_n)$ determines the first $(n+1)$ terms of the truth table of g, as in $(w_0, w_1, \ldots, w_n, *, \ldots, *)$. The free entries, denoted as $*$, correspond to output values of g that do not matter, since they correspond to weights that never appear as input to g (after the Hamming weight encoding H_{BR}).

The number of free entries in the truth table is equal to $l = 2^s - n - 1$, where s is the number of bits in the binary representation of n. When n is of the form $2^s - 1$, the truth table for g does not have any free entries, i.e., $l = 0$. For all other values n there are free entries that allow choosing from among several possible functions g, possibly allowing to choose one with the smallest multiplicative complexity.

Example 2 Let $f = \Sigma_9^{10} \oplus \Sigma_7^{10} \oplus \Sigma_3^{10}$, which is equal to $f = E_3^{10} \oplus E_9^{10}$, in terms of counting functions. When applying the basic Hamming weight method, where $f = g \circ H_{\mathrm{BR}}$, the truth table of any corresponding 4-variable function g must be of the form

$$(0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, *, *, *, *, *). \tag{25}$$

This contains 5 free entries, which means there are 2^5 possible choices for g. In fact, for any 10-variable symmetric Boolean function f, after applying the H_{BR} encoding the missing function g can be any 4-variable function selected from within a set of 2^5 functions. While there exist 4-variable functions with MC 3, we know that for any such f there is a corresponding g with MC 2.

The Example 2 can be extended even for the case of a single free entry. It is known that 4-variable Boolean functions can be implemented using at most 3 `AND` gates, and, among those, the functions with MC 3 have degree 4. By choosing the last truth-table entry such that the parity of the truth table is even guarantees that the degree of the function is at most 3, and such functions can be implemented using at most 2 `AND` gates.

The concatenation method When the number l of free entries is large, checking the multiplicative complexity of all possible functions may be infeasible. In those cases, the truth table of an m-variable function $g \in \mathcal{B}_m$ can be interpreted as the concatenation of the truth tables of two functions g_1 and g_2, each having $m-1$ variables. This representation corresponds to a decomposition of $g(z_1, \ldots, z_m)$ as follows:

$$g_1(z_1, \ldots, z_{m-1}) \oplus (z_m \wedge (g_1(z_1, \ldots, z_{m-1}) \oplus g_2(z_1, \ldots, z_{m-1}))). \tag{26}$$

Based on this decomposition, we can get an upper bound for the MC of g:

$$C_\wedge(g) \leq C_\wedge(g_1) + C_\wedge(g_2) + 1 \tag{27}$$

In this decomposition, function g_1 has no free entries, but the last l bits of the truth table of g_2 can be selected freely. As a shorthand notation, we use G_i to denote a decomposition where $i = \lceil \log_2(2^{m-1} - l) \rceil$ is the minimum number of variables for which there is a i-variable function g_2 satisfying the defined entries. For large enough l, we can have i smaller than $m-1$.

An exceptional case (G_0) When the number of free entries is equal to $2^{m-1} - 1$ there is a single defined entry $g_2(0, \ldots, 0) = g(0, \ldots, 0, 1) \equiv b$. In this case, instead of defining g_2 as a constant function, it is advantageous to define $g_2 = g_1 \oplus b$. Then, the component $g_1(z_1, \ldots, z_{m-1}) \oplus g_2(z_1, \ldots, z_{m-1})$ in (26) becomes a constant, allowing the removal of the "+1" term in (27), meaning the use of G_0 enables $C_\wedge(g) \leq C_\wedge(g_1)$.

5 Results

5.1 Symmetric functions with up to 25 variables

In order to compute the circuit of a given symmetric function $f \in \mathcal{S}_n$, the algorithm first computes the degree of f and determines the encoding based on the degree and the arity of f, as described in Sections 4.1 and 4.2, determining up to which point to use full adders. In the second phase, the algorithm constructs a circuit for a function g, satisfying $f = g \circ H$. When only full adders have been applied in the first phase, there is a single possible function g to implement. Exceptionally for $n = 22$, we had to use half adders, since the number of variables after applying the full adders (see Section 4.2) was otherwise too high to enable the computation of the MC of g. The MC of g was computed using the techniques from ref. [14].

We have applied the method to all symmetric functions with up to $n = 25$ variables. The source code used to generate these results is publicly available online [18]. In Table 4, each cell in a row n and column B contains the number of functions $f \in \mathcal{S}_n$ for which the method provides a circuit with B AND gates. For each such function, B is thus an upper bound on the MC, but some functions may have a smaller MC.

We make a few observations about the functions accounted in Table 4:

- For each n, there are four functions with MC 0; these are the symmetric linear functions 0, 1, Σ_1^n and $\Sigma_1^n \oplus 1$.
- For each even n, there are twelve functions with MC equal to $n/2$; these are the functions Σ_2^n, Σ_3^n and $\Sigma_3^n \oplus \Sigma_2^n$, and the corresponding functions obtained by adding any of the four symmetric linear functions.
- For each odd n, there are four functions with MC equal to $(n-1)/2$; these are the functions obtained by adding Σ_2^n with any of the four symmetric linear functions.
- For each odd $n \geq 9$, there are eight functions with MC equal to $(n+1)/2$; these are the functions Σ_3^n, $\Sigma_3^n \oplus \Sigma_2^n$ and the corresponding functions obtained by adding any of the four symmetric linear functions.

Table 4 also shows that, for any $n \leq 21$ and for $n = 23$, all n-variable symmetric Boolean functions can be implemented with at most $n - 1$ AND gates. Since $n - 1$ is also the degree lower bound for functions of degree n, it follows that $n - 1$ is also the exact $\mathrm{MC}_{\max}(\mathcal{B}_n)$ for $n \leq 21$ and $n = 23$. The arity $n = 22$ is the first case for which we cannot yet settle the exact $\mathrm{MC}_{\max}$, i.e., we can present the upper bound 22, but we are not yet able to decide whether $\mathrm{MC}_{\max}(\mathcal{S}_{22})$ is 22 or 21.

Special classes of symmetric Boolean functions As a special case, in Appendix A we show MC upper bounds for all Σ_k^n (Table 5) and all E_k^n (Table 6), for $n \leq 25$.

In 2008, Boyar and Peralta posed two concrete problems related to the MC of symmetric Boolean functions [15]: is the $C_\wedge\left(\Sigma_4^8\right)$ equal to 5 or 6?; is the $C_\wedge\left(E_4^8\right)$ equal to 6 or 7? The methods described in this paper provide the implementations for Σ_4^8 and E_4^8 (see Figs. 3 and 4) with 6 ANDs, which solves the case for E_4^8. The optimality of the circuit for Σ_4^8 is verified using the methods from ref. [14], by ruling out the possibility of implementing the function with 5 AND gates.

The results also answer another question from ref. [15] — $C_\wedge\left(\Sigma_k^n\right)$ is not monotonic in k. More concretely, in Table 5 we observe that for each $n > 7$ the computed upper bound is not monotonic in k, but for each k the upper bound is non-decreasing in n.

Table 4 Number of functions in $\mathcal{S}_n$ for each obtained upper bound (B) for MC

n \ B	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	Total
1	4																										2^2
2	4	4																									2^3
3	4	4	8																								2^4
4	4		12	16																							2^5
5	4		4	24	32																						2^6
6	4			12	48	64																					2^7
7	4			4	16	104	128																				2^8
8	4				12	16	224	256																			2^9
9	4				4	8	48	448	512																		2^{10}
10	4					12	0	96	712	1224																	2^{11}
11	4					4	8	8	168	1856	2048																2^{12}
12	4						12	0	16	320	2912	4928															2^{13}
13	4						4	8	0	48	480	6048	9792														2^{14}
14	4							12	0	0	96	480	9240	22 936													2^{15}
15	4							4	8	0	8	120	1136	31 488	32 768												2^{16}
16	4								12	0	0	16	256	1344	41 440	88 000											2^{17}
17	4								4	8	0	0	48	256	2944	82 880	176 000										2^{18}
18	4									12	0	0	0	96	224	2880	91 904	429 168									2^{19}
19	4									4	8	0	0	8	104	320	9280	337 920	700 928								2^{20}
20	4										12	0	0	0	16	224	256	8608	280 496	1 807 536							2^{21}
21	4										4	8	0	0	0	48	192	576	14 960	717 456	3 461 056						2^{22}
22	4											12	0	0	0	0	4	52	80	1188	57 068	4 039 600	4 290 600				2^{23}
23	4											4	8	0	0	0	8	104	144	1264	72 704	5 411 328	11 291 648				2^{24}
24	4												12	0	0	0	0	16	224	32	1664	67 552	3 277 056	30 207 680	192		2^{25}
25	4												4	8	0	0	0	0	48	192	64	3584	100 480	6 886 976	60 116 992	512	2^{26}

$$
\begin{aligned}
t_0 &= (x_6 \oplus x_7) \wedge (x_6 \oplus x_8)\\
t_1 &= (x_4 \oplus x_5) \wedge (x_4 \oplus x_6 \oplus x_7 \oplus x_8)\\
t_2 &= (x_2 \oplus x_3) \wedge (x_2 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8)\\
t_3 &= (t_0 \oplus t_1 \oplus t_2 \oplus x_1 \oplus x_2 \oplus x_4 \oplus x_6) \wedge (x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8)\\
t_4 &= (t_0 \oplus t_1 \oplus x_1 \oplus x_4 \oplus x_6) \wedge (t_3 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8)\\
t_5 &= (t_0 \oplus t_1 \oplus x_4 \oplus x_6) \wedge (t_0 \oplus t_2 \oplus t_3 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_7 \oplus x_8)\\
\Sigma_4^8 &= t_0 \oplus t_4 \oplus t_5 \oplus x_6
\end{aligned}
$$

Fig. 3 Implementation of the elementary symmetric function Σ_4^8

5.2 Maximum MC for up to large arities

The following proposition expresses a useful observation — the $\mathrm{MC}_{\max}$ is non-decreasing as the number of variables increases. This allows framing the $\mathrm{MC}_{\max}$ for any arity n in between the $\mathrm{MC}_{\max}$ of the preceding and succeeding arities.

Proposition 1 (Non-decreasing $\mathrm{MC}_{\max}$) *Let $MC_{\max}(\mathcal{S}_n)$ denote the maximum MC of symmetric Boolean functions with n variables. Then, $MC_{\max}(\mathcal{S}_{n+1}) \geq MC_{\max}(\mathcal{S}_n)$.*

Proof Consider any function $f \in \mathcal{S}_n$. Let $v(f) = (v_0, ..., v_n)$ be the simplified ANF of f, such that v_i is 1 if and only if the elementary symmetric function Σ_i^n appears in the elementary additive decomposition of f. Let f' be another symmetric function defined as the result of replacing each $\Sigma_i^n(x_1, ..., x_n)$, in the simplified ANF of f, by $\Sigma_i^{n+1}(x_1, ..., x_{n+1})$, which contains x_{n+1} as a new variable. When in the latter we replace x_{n+1} by 0, we eliminate all terms containing x_{n+1}, leaving exactly all the terms of degree i that do not contain x_{n+1}. Thus, it follows that $\Sigma_i^n(x_1, ..., x_n) = \Sigma_i^{n+1}(x_1, ..., x_n, 0)$. Consequently, $f'(x_1, ..., x_n, 0) = f(x_1, ..., x_n)$. Let C' be an MC-optimal circuit for f'. Starting from C', for any wire carrying input x_{n+1} replace it by a constant 0, thereby getting a new circuit C. Circuit C computes f with the same number of AND gates as in C'. Therefore, $\mathrm{MC}_{\max}(\mathcal{S}_n) \leq \mathrm{MC}_{\max}(\mathcal{S}_{n+1})$, i.e., $\mathrm{MC}_{\max}(\mathcal{S}_{n+1}) \geq \mathrm{MC}_{\max}(\mathcal{S}_n)$. □

The techniques devised in this paper enable calculation of upper bounds for $\mathrm{MC}_{\max}(\mathcal{S}_n)$ for arities n larger than 25. Figure 5 shows, for n up to 132, a graphical comparison of the degree lower bound, our new upper bounds, and the upper bounds obtained from (19) and (20). Table 7 in Appendix B shows a table with the corresponding $\mathrm{MC}_{\max}$ for n up to 132, and a detailed indication of the used method.

$$
\begin{aligned}
t_0 &= (x_6 \oplus x_7) \wedge (x_6 \oplus x_8)\\
t_1 &= (x_4 \oplus x_5) \wedge (x_4 \oplus x_6 \oplus x_7 \oplus x_8)\\
t_2 &= (x_2 \oplus x_3) \wedge (x_2 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8)\\
t_3 &= (t_1 \oplus t_2 \oplus x_2 \oplus x_4) \wedge (t_0 \oplus 1 \oplus x1 \oplus x6)\\
t_4 &= (t_0 \oplus t_1 \oplus x_4 \oplus x_6) \wedge (t_0 \oplus t_3 \oplus 1 \oplus x_1 \oplus x_6)\\
E_4^8 &= (1 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8) \wedge (t_1 \oplus t_2 \oplus t_3 \oplus t_4 \oplus x_2 \oplus x_4)
\end{aligned}
$$

Fig. 4 Implementation of the counting function E_4^8

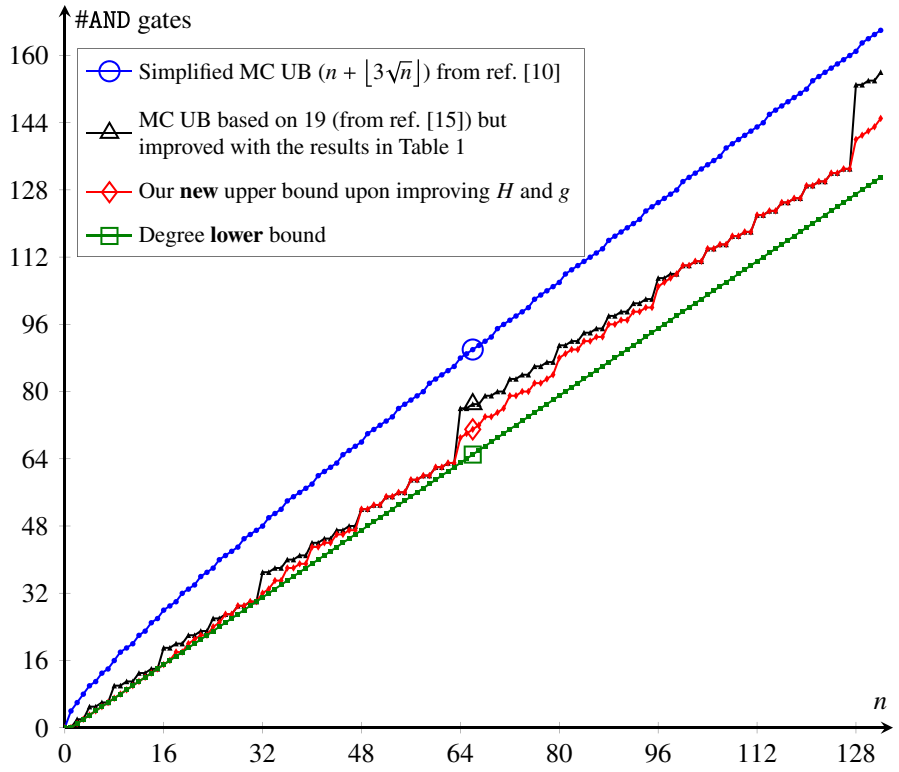


Fig. 5 Comparison of MC bounds for n-variable symmetric functions

As n increases, more often the full-adder approach ($H_{\text{FA}} = H_0$) is not sufficient to achieve a small enough t (number of variables left for phase 2). As n increases, more often more HAs may have to be applied (i.e., the larger j has to be) to achieve a low enough number t_j of variables that allow us to obtain a good upper bound for f based on a possible upper bound for g_j. If the exact $\text{MC}_{\max}$ of the set G of functions needed for phase 2 could always be computed, then a better bound could often be obtained. However, since with respect to finding the MC of arbitrary functions our computational resources are currently limited to cases with up to $t = 6$ variables, we can often not use the ideal j and respective $\text{MC}_{\max}$.

6 Conclusion and future work

Finding efficient circuits in terms of the number of AND gates for Boolean functions when $n \geq 7$ is a hard problem. In this paper, we have focused on the class of symmetric functions. The symmetries in these functions enable construction of efficient circuits with a small number of AND gates. We have provided different weight encodings that aim to optimize the number of AND gates.

Although symmetric functions constitute a small class within the set of Boolean functions, the provided bounds also hold for Boolean functions that are affine equivalent to symmetric functions. The techniques presented in this paper can potentially be applied to

larger classes of functions, such as partially symmetric functions, and rotation symmetric functions which also contain the set of symmetric functions.

A natural further direction is to understand when techniques provide optimal solutions and when they can be improved. It would be interesting to determine what is the smallest value of n such that $\mathrm{MC}_{\max}(\mathcal{S}_n)$ is greater than or equal to n.

Acknowledgments The authors thank the anonymous reviewers of the journal, and Morris Dworkin from NIST, for their useful comments and suggestions.

Appendix A: MC upper-bounds for special classes of symmetric Boolean functions

Tables 5 and 6 show MC upper bounds, respectively for all elementary-symmetric Boolean functions Σ_k^n and all exactly-counting Boolean functions E_k^n, with any number n of variables up to 25, and with k up to n.

Table 5 Upper bounds on the MC of elementary symmetric functions Σ_k^n

n \ k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	0	0	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
2	0	0	1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
3	0	0	1	2	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
4	0	0	2	2	3	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
5	0	0	2	3	3	4	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
6	0	0	3	3	4	4	5	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
7	0	0	3	4	4	5	5	6	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
8	0	0	4	4	6	5	6	6	7	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
9	0	0	4	5	6	7	6	7	7	8	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
10	0	0	5	5	7	7	8	7	8	8	9	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
11	0	0	5	6	7	8	8	9	8	9	9	10	–	–	–	–	–	–	–	–	–	–	–	–	–	–
12	0	0	6	7	9	8	9	9	10	9	10	10	11	–	–	–	–	–	–	–	–	–	–	–	–	–
13	0	0	6	7	9	10	9	10	10	11	10	11	11	12	–	–	–	–	–	–	–	–	–	–	–	–
14	0	0	7	8	10	10	11	10	11	11	12	11	12	12	13	–	–	–	–	–	–	–	–	–	–	–
15	0	0	7	8	10	11	11	12	11	12	12	13	12	13	13	14	–	–	–	–	–	–	–	–	–	–
16	0	0	8	9	12	11	12	12	14	12	13	13	14	13	14	14	15	–	–	–	–	–	–	–	–	–
17	0	0	8	9	12	13	12	13	14	15	13	14	14	15	14	15	15	16	–	–	–	–	–	–	–	–
18	0	0	9	10	13	13	14	13	15	15	16	14	15	15	16	15	16	16	17	–	–	–	–	–	–	–
19	0	0	9	10	13	14	14	15	15	16	16	17	15	16	16	17	16	17	17	18	–	–	–	–	–	–
20	0	0	10	11	15	14	15	15	17	16	17	17	18	16	17	17	18	17	18	18	19	–	–	–	–	–
21	0	0	10	11	15	16	15	16	17	18	17	18	18	19	17	18	18	19	18	19	19	20	–	–	–	–
22	0	0	11	12	16	16	17	16	18	18	19	18	19	19	20	18	19	20	20	19	20	20	21	–	–	–
23	0	0	11	12	16	17	17	18	18	19	19	20	19	20	20	21	19	20	20	21	20	21	21	22	–	–
24	0	0	12	13	18	17	18	18	21	19	20	20	21	20	21	21	22	20	21	21	22	21	22	22	23	–
25	0	0	12	13	18	19	18	19	21	22	20	21	21	22	21	22	22	23	21	22	22	23	22	23	23	24

Table 6 Upper bounds on the MC of (exactly-k) counting functions E_k^n

n \ k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	0	0	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
2	1	0	1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
3	2	2	2	2	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
4	3	2	2	2	44	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
5	4	4	3	3	54	4	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
6	5	4	5	3	64	4	5	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
7	6	6	6	6	74	6	6	6	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
8	7	6	6	6	6	6	6	6	7	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
9	8	8	7	7	7	7	7	7	8	8	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
10	9	8	9	7	8	7	8	7	9	8	9	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
11	10	10	10	10	9	9	9	9	10	10	10	10	–	–	–	–	–	–	–	–	–	–	–	–	–	–
12	11	10	10	10	11	9	9	9	11	10	10	10	11	–	–	–	–	–	–	–	–	–	–	–	–	–
13	12	12	11	11	12	12	10	10	12	12	11	11	12	12	–	–	–	–	–	–	–	–	–	–	–	–
14	13	12	13	11	13	12	13	10	13	12	13	11	13	12	13	–	–	–	–	–	–	–	–	–	–	–
15	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	–	–	–	–	–	–	–	–	–	–
16	15	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	15	–	–	–	–	–	–	–	–	–
17	16	16	15	15	15	15	15	15	15	15	15	15	15	15	15	15	16	16	–	–	–	–	–	–	–	–
18	17	16	17	15	16	15	16	15	16	15	16	15	16	15	16	15	17	16	17	–	–	–	–	–	–	–
19	18	18	18	18	17	17	17	17	17	17	17	17	17	17	17	17	18	18	18	18	–	–	–	–	–	–
20	19	18	18	18	19	17	17	17	18	17	17	17	18	17	17	17	19	18	18	18	19	–	–	–	–	–
21	20	20	19	19	20	20	18	18	19	19	18	18	19	19	18	18	20	20	19	19	20	20	–	–	–	–
22	21	20	21	19	21	20	21	18	20	19	20	18	20	19	20	18	21	20	21	19	21	20	21	–	–	–
23	22	22	22	22	22	22	22	22	21	21	21	21	21	21	21	21	22	22	22	22	22	22	22	22	–	–
24	23	22	22	22	22	22	22	22	23	21	21	21	21	21	21	21	23	22	22	22	22	22	22	22	23	–
25	24	24	23	23	23	23	23	23	24	24	22	22	22	22	22	22	24	24	23	23	23	23	23	23	24	24

Appendix B: Description of $\mathbf{MC_{max}}$ upper bounds

Table 7 shows, for each $n \leq 132$, the $\mathrm{MC}_{\max}$ upper bound we found for the set $\mathcal{S}_n$ of n-variable symmetric Boolean functions. For each n, the table identifies an encoding H of the Hamming weight, and a method G for finding an MC upper bound of the corresponding g. We checked five different practical combinations of H and G:

C_1. $H = H_{\mathrm{BR}}$ and $G =$ gen, where "gen" uses for g the MC upper bound from Table 1. This was used for $n \in \{1, 3, 7, 15, 29–31, 48–63, 99–127\}$.

C_2. $H = H_0$ (i.e., using only full adders) and $G =$ exp, where "exp" is an exhaustive computation "experimentally" determining the MC of each g corresponding to each $f \in \mathcal{S}_n$. This was used for $n \in \{14, 18, 20, 21\}$.

C_3. $H = H_j$ (possibly using some ($j \geq 0$) half adders, but not computing the full H_{BR}) and $G =$ gen. This was used for $n \in \{2, 4 - -6, 8 - -13, 16 - -17, 19, 22 - -28, 32 - -47\}$.

Table 7 Upper bounds (UB) obtained for $\mathrm{MC}_{\max}(\mathcal{S}_n)$

n	UB	D	H	G
1	**0**	0	$H_0{}^*$	gen
2	**1**	1	H_0	
3	**2**	0	$H_0{}^*$	
4	**3**	2	H_0	
5	**4**	1		
6	**5**	1		
7	**6**	0	$H_0{}^*$	
8	**7**	3	H_0	
9	**8**	2		
10	**9**	2		
11	**10**	1		
12	**11**	2		
13	**12**	1		
14	**13**	1	H_0	exp
15	**14**	0	$H_0{}^*$	gen
16	**15**	4	H_0	
17	**16**	3		
18	**17**	3	H_0	exp
19	**18**	2	H_0	gen
20	**19**	1	H_0	exp
21	**20**	2		
22	22	1	H_1	gen
23	**22**	1	H_0	
24	24	2		
25	25	1		
26	27	0	H_1	
27	27	0	H_0	
28	29	0	H_1	
29	29	0	$H_1{}^*$	
30	30	0		
31	**30**	0	$H_0{}^*$	
32	32	5	H_0	
33	33	4		
34	35	3	H_1	
35	35	3	H_0	
36	38	2	H_2	
37	38	2	H_1	
38	39	2		
39	39	2	H_0	
40	43	1	H_3	
41	43	1	H_2	
42	44	1		
43	44	1	H_1	
44	46	1	H_2	
45	46	1	H_1	gen
46	47	1		
47	47	1	H_0	
48	52	0	$H_4{}^*$	
49	52	0	$H_3{}^*$	
50	53	0		
51	53	0	$H_2{}^*$	
52	55	0	$H_3{}^*$	
53	55	0	$H_2{}^*$	
54	56	0		
55	56	0	$H_1{}^*$	
56	59	0	$H_3{}^*$	
57	59	0	$H_2{}^*$	
58	60	0		
59	60	0	$H_1{}^*$	
60	62	0	$H_2{}^*$	
61	62	0	$H_1{}^*$	
62	63	0		
63	63	0	$H_0{}^*$	
64	69	7	$H_6{}^*$	G_0
65	70	6	$H_5{}^*$	G_1
66	71	6	$H_5{}^*$	G_2
67	72	5	$H_4{}^*$	G_2
68	74	5	$H_5{}^*$	G_3
69	74	5	$H_4{}^*$	
70	75	5		
71	76	4	$H_3{}^*$	
72	79	4	$H_5{}^*$	G_4
73	79	4	$H_4{}^*$	
74	80	4		
75	80	4	$H_3{}^*$	
76	82	4	$H_4{}^*$	
77	82	4	$H_3{}^*$	
78	83	4		
79	84	3	$H_2{}^*$	
80	88	3	$H_5{}^*$	$G_{5,0}$
81	89	2	$H_4{}^*$	G_5
82	90	2		
83	90	2	$H_3{}^*$	
84	92	2	$H_4{}^*$	
85	92	2	$H_3{}^*$	
86	93	2		
87	93	2	$H_2{}^*$	
88	96	2	$H_4{}^*$	
89	96	2	$H_3{}^*$	G_5
90	97	2		
91	97	2	$H_2{}^*$	
92	99	2	$H_3{}^*$	
93	99	2	$H_2{}^*$	
94	100	2		
95	100	2	$H_1{}^*$	
96	105	2	$H_5{}^*$	$G_{6,0}$
97	106	1	$H_4{}^*$	$G_{6,1}$
98	107	1	$H_4{}^*$	$G_{6,2}$
99	108	0	$H_3{}^*$	gen
100	110	0	$H_4{}^*$	
101	110	0	$H_3{}^*$	
102	111	0		
103	111	0	$H_2{}^*$	
104	114	0	$H_4{}^*$	
105	114	0	$H_3{}^*$	
106	115	0		
107	115	0	$H_2{}^*$	
108	117	0	$H_3{}^*$	
109	117	0	$H_2{}^*$	
110	118	0		
111	118	0	$H_1{}^*$	
112	122	0	$H_4{}^*$	
113	122	0	$H_3{}^*$	
114	123	0		
115	123	0	$H_2{}^*$	
116	125	0	$H_3{}^*$	
117	125	0	$H_2{}^*$	
118	126	0		
119	126	0	$H_1{}^*$	
120	129	0	$H_3{}^*$	
121	129	0	$H_2{}^*$	
122	130	0		
123	130	0	$H_1{}^*$	
124	132	0	$H_2{}^*$	
125	132	0	$H_1{}^*$	
126	133	0		
127	133	0	$H_0{}^*$	
128	140	13	$H_7{}^*$	G_0
129	141	12	$H_6{}^*$	G_1
130	142	12	$H_6{}^*$	G_2
131	143	11	$H_5{}^*$	
132	145	11	$H_6{}^*$	G_3

C_4. $H = H_{\mathrm{BR}}$ and $G = G_i$, where G_i applies the concatenation method to function g, to obtain g_2 with only i variables (if $i \geq 1$), or to use $g_2 = g_1 + g(0, \ldots, 0, 1)$ (if $i = 0$). This was used for $n \in \{64 - -79, 81 - -95, 128 - -132\}$.

C_5. $H = H_{\mathrm{BR}}$ and $G = G_{i,j}$, where $G_{i,j}$ applies G_i to g and then applies G_j to the corresponding g_2. This was used for $n \in \{80, 96 - -98\}$.

The best combination varies with n, but sometimes several combinations yield the same best upper bound. Table 7 shows H and G only for the first best combination in the order $C_1 < C_2 < C_3 < C_4 < C_5$.

Column "H" shows the number of used half adders as a subscript j in H_j. When said encoding is H_{BR}, an asterisk is added as suffix (H_j^*). Column "D" shows the difference to the upper bound that would be obtained with the reference method C_1. Column "UB" shows the upper bound in bold when it is equal to the degree bound $(n - 1)$.

Example 3 The case $n = 72$ (using combination C_4) indicates an encoding $H = H_5^* = H_{\mathrm{BR}}$ with 5 half adders, and a method G_4 for g. The encoding H_{BR} produces an output of seven variables $(z_1, \ldots, z_7)$, upon which the function g can be written as $g_1(z_1, \ldots, z_6) \oplus z_7 \wedge (g_1(z_1, \ldots, z_6) \oplus g_2(z_1, \ldots, z_4))$. Since the MC for $H_{\mathrm{BR}}(x_1, ..., x_{72})$ is 70, the overall upper bound is equal to $79 = 70 + 6 + 1 + 3$, where 6 and 3 are the generic MC upper bounds for g_1 and g_2 (functions of 6 and 4 variables, respectively), and the extra 1 is the AND used to multiply z_7 with $(g_1 \oplus g_2)$.

Example 4 The case $n = 80$ (using combination C_5) indicates the use of $H = H_5^* = H_{\mathrm{BR}}$ and $G_{5,0}$. The H_{BR} encoding outputs 7 variables. Then, G_5 decomposes g into $g_1(z_1, \ldots, z_6) \oplus y_6 \wedge (g_1(z_1, \ldots, z_6) \oplus g_2(z_1, ..., z_5))$. Since for $n = 80$ there are 81 possible weights, the function g_2 is a 5-variable function with 17 $(= 81 - 64)$ defined entries and 15 free entries. For the second decomposition, the number of defined entries of the second component will be $1 (= 17 - 16)$. Thus, G_0 can be applied (recall the exceptional case described in Section 4.3) to decompose g_2 into $g_2'(z_1, \ldots, z_4) \oplus (z_5 \wedge b)$, where b is the constant $g(0, ..., 0, 1)$. Thus, the upper bound for the $\mathrm{MC}_{\max}$ for $n = 80$ is equal to $88 = 78 + 6 + 1 + (3 + 0)$, where 78 is the MC of H_{BR} on 80 variables, and where 6, 3 and 0 are the MC majorants for the 6-variable function g_1, the 4-variable function g_2', and the 1-variable function $b \wedge z_5$, respectively.

References

1. Wegener, I.: The complexity of symmetric Boolean functions, vol. 270 of LNCS, pp. 433–442. Springer, Berlin (1987). https://doi.org/10.1007/3-540-18170-9_185
2. Canteaut, A., Videau, M.: Symmetric Boolean functions. IEEE Trans. Inf. Theory **51**(8), 2791–2811 (2005). https://doi.org/10.1109/TIT.2005.851743
3. Sasao, T. Switching theory for logic synthesis, 1st. Kluwer Academic Publishers, Norwell (1999). https://doi.org/10.1007/978-1-4615-5139-3
4. Kerntopf, P., Szyprowski, M.: Symmetry in reversible functions and circuits. In: Proceedings of 20th ICCC/ACM international workshop on logic and synthesis — IWLS 2011, pp. 67–73 (2011)
5. Muller, D.E., Preparata, F.P.: Bounds to complexities of networks for sorting and for switching. J. ACM **22**(2), 195–201 (1975). https://doi.org/10.1145/321879.321882
6. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) 35th international colloquium — ICALP 2008 automata, languages and programming, vol. 5126 of LNCS, vol. 5126, pp. 486–498. Springer (2008). https://doi.org/10.1007/978-3-540-70583-3_40
7. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) Proceedings of 3rd innovations in theoretical computer science conference — ITCS '12, pp. 309–325. ACM (2012). https://doi.org/10.1145/2090236.2090262
8. Boyar, J., Damgård, I., Peralta, R.: Short non-interactive cryptographic proofs. J. Cryptol. **13**(4), 449–472 (2000). https://doi.org/10.1007/s001450010011

9. Carlet, C., Goubin, L., Prouff, E., Quisquater, M., Rivain, M.: Higher-order masking schemes for S-Boxes. In: Canteau, t.A. (ed.) Proceedings of 19th international workshop on fast software encryption — FSE 2012, vol. 7549 of LNCS, vol. 7549, pp. 366–384. Springer (2012). https://doi.org/10.1007/978-3-642-34047-5_21
10. Boyar, J., Peralta, R., Pochuev, D.: On the multiplicative complexity of Boolean functions over the basis ($\wedge$, $\oplus$, 1). Theor. Comput. Sci. **235**(1), 43–57 (2000). https://doi.org/10.1016/S0304-3975(99)00182-6
11. Find, M.G.: On the complexity of computing two nonlinearity measures. In: Hirsch, E.A., Kuznetsov, S.O., Pin, J.-É., Vereshchagin, N.K. (eds.) Proceedings of CSR 2014: Computer science — theory and applications, vol. 8476 of LNCS, vol. 8476, pp. 167–175. Springer International Publishing (2014). https://doi.org/10.1007/978-3-319-06686-8_13
12. Find, M.G., Smith-Tone, D., Sönmez Turan, M.: The number of Boolean functions with multiplicative complexity 2. Int. J. Inf. Coding Theory (IJICOT) **4**(4), 222–236 (2017). https://doi.org/10.1504/IJICOT.2017.086890
13. Sönmez Turan, M., Peralta, R.: The multiplicative complexity of Boolean functions on four and five variables. In: Eisenbarth, T., Öztürk, E. (eds.) Proceedings of 3rd international workshop on lightweight cryptography for security and privacy — LightSec 2014, vol. 8898 of LNCS, pp. 21–33. Springer (2015). https://doi.org/10.1007/978-3-319-16363-5_2
14. Çalık, Ç., Sönmez Turan, M., Peralta, R.: The multiplicative complexity of 6-variable Boolean functions, Cryptography and Communucations. Special Issue on Boolean Functions and Their Applications, pp. 1–15. https://doi.org/10.1007/s12095-018-0297-2 (2018)
15. Boyar, J., Peralta, R.: Tight bounds for the multiplicative complexity of symmetric functions. Theor. Comput. Sci. **396**(1-3), 223–246 (2008). https://doi.org/10.1016/j.tcs.2008.01.030
16. Schnorr, C.P.: The multiplicative complexity of Boolean functions. In: Mora, T. (ed.) Applied algebra, algebraic algorithms and error-correcting codes (AAECC 1988), vol. 357 of LNCS, pp. 45–58. Springer, Berlin (1989). https://doi.org/10.1007/3-540-51083-4_47
17. Komamiya, Y.: Theory of computing networks, Bulletin of the Electrotechnical Laboratory. In Japanese (1959)
18. Circuit minimization team at the Cryptographic Technology Group, NIST, Circuits for functions of interest to cryptography. https://github.com/usnistgov/Circuits/ (2019)