Exploiting Symmetrization and D-reducibility for Approximate Logic Synthesis

Anna Bernasconi, Valentina Ciriani, and Tiziano Villa

Abstract—Approximate synthesis is a recent trend in logic synthesis where one changes some outputs of a logic specification, within the error tolerance of a given application, to reduce the complexity of the final implementation. We attack the problem by exploiting the allowed flexibility in order to maximize the regularity of the specified Boolean functions. Specifically, we consider two types of regularity: *symmetry* and *D-reducibility*, and contribute two algorithms to find, respectively, a symmetric and a D-reducible approximation of a given target function f, within the given error rate threshold if possible. When targeting symmetry, we characterize and compute polynomially the closest symmetric approximation, i.e., the symmetric function obtained by injecting the minimum number of errors in the original incompletely specified Boolean function, with an unbounded number of errors; then, we discuss strategies to achieve partial symmetrization of the original specification while satisfying given error bounds. Finally, we present a polynomial heuristic algorithm to compute a D-reducible approximation of an incompletely specified target function, under a bit error metric. Experimental results on classical and new benchmarks confirm the effectiveness of the proposed approaches.

Index Terms—Logic synthesis, Approximate synthesis , Regular Boolean functions.

1 Introduction

In *approximate logic synthesis*, the error tolerance of specific applications allows to produce erroneous outputs for some inputs, within a specified error threshold [31], [42]. Applications domains where approximations are acceptable include image, video, audio, machine learning, pattern recognition, and error-correcting codes for wireless communication. This flexibility can be exploited to synthesize circuits with smaller area, delay, or lower power consumption. In recent years there has been an increasing flow of publications to investigate approximate logic synthesis, from one side adapting to the new scenario the conceptual tools already developed for standard exact logic synthesis, from the other side addressing the specific issues such as modeling the acceptable errors. We refer to the next section on previous work for a tentative survey of relevant literature (see for instance [8], [15], [26], [27], [28], [29], [34], [35], [39]).

Partial flexibility to implement a given Boolean function appears in exact synthesis too, due both to incomplete specification from the onset and to controllability/observability conditions arising in a specific chosen architecture, which can be summed up as don't care computation and exploitation. However, exploitation of don't cares is harder in approximate synthesis than in exact synthesis, because in the latter case we are required to optimize under a given set of don't cares, i.e., to assign them to obtain the best final implementation, whereas in approximate synthesis we know only an upper bound on the number of don't cares, and we are free to choose which inputs to set as don't cares. This point-of-view is highlighted in [28], [29], where the authors first address the problem of approximate logic synthesis under arbitrary error magnitude by casting it as a Boolean relation minimization [2]; then, they present an efficient heuristic algorithm for refining iteratively the magnitude-constrained solution to yield a solution that satisfies also the error rate constraint.

The key question that each approach must answer is how to exploit the flexibility in changing the value of the function on a subset of input points. In this paper we exploit the network tolerance to errors to maximize the *regularities* of Boolean functions and so to reduce the complexity of their final logic implementations. Regularity of a Boolean function is a generic notion that aims to capture intrinsic properties of the function easing the minimization process or suggesting the architecture of a simplified implementation. Some well-known classes of regular functions are: unate functions, self-dual functions, linear functions, totally and partially symmetric functions, threshold functions, D-reducible and autosymmetric functions. E.g., unateness is leveraged in ESPRESSO, which is based on building cofactoring trees of the given function until we reach unate leaves whose minimum is unique and easy to compute [13]; synthesis procedures based on autosymmetry can be found in [9], [10]. Here we bring to a new height the idea of exploiting regularities for synthesis of Boolean functions, since we apply it within the context of approximate synthesis, where we have the freedom of choosing which output values to toggle to improve the regularity of the function, whereas in the exact case we are left only with the task of exposing the regularity if it exists, and exploiting it in the final implementation.

In this paper, we consider two types of regularity: *sym-*

- *A. Bernasconi is with the Department of Computer Science, Università di Pisa, Italy.*
  E-mail: anna.bernasconi@unipi.it
- *V. Ciriani is with the Department of Computer Science, Università degli Studi di Milano, Italy.*
  E-mail: valentina.ciriani@unimi.it
- *T.Villa is with the Department of Computer Science, Università degli Studi di Verona, Italy.*
  E-mail: tiziano.villa@univr.it

Manuscript received ...; revised ...

metry [24], [41] and *D-reducibility* [5], [6], [7], and present two algorithms that find, respectively, a symmetric and a D-reducible approximation of a given target function f to be synthesized, within the given error rate threshold if possible. These regularity-based approaches to approximate synthesis do not refer to any particular logic form, and can therefore be used on top of any minimizer, targeting e.g. two-level logic, XOR-AND-OR forms, multi-level logic.

When targeting symmetrization, we contribute a *polynomial algorithm* for computing the *closest totally symmetric approximation* of a given *incompletely specified multi-output* Boolean function with an unbounded number of errors. This is achieved by exploiting properties of the disjoint covers of a Boolean function and efficient BDD constructions, so that we can build polynomially the characteristic vector of the closest symmetric function and its BDD. The proposed algorithm is designed for single-output functions, but it can be applied to multi-output ones, treating each output as a separate Boolean function. In fact, we prove that once each output has been replaced with its closest symmetric approximation, the overall multi-output function becomes the totally symmetric function closest to the original one. We then propose heuristic strategies to relax total symmetrization in order to guarantee a bounded approximation scheme. When targeting D-reducibility, we contribute a *polynomial heuristic algorithm* to compute a D-reducible approximation of an incompletely specified function, under a bit error metric.

Experimental results on classical and new benchmarks have confirmed the efficacy of the proposed approaches. For symmetrization, the average gain in the BDD size for the unbounded approximation scheme is 69% with an average error rate of about 15%, while in the bounded context, with a fixed error rate of 5%, the average gain is about 31%. For D-reducibility we obtain a reduction of almost 19% on the number of variables for an approximation threshold of 5%.

This paper is an extended version of the conference paper presented in [12] and is organized as follows. Previous work is reviewed in Sec. 2. Sec. 3 introduces background information on symmetric functions, D-reducible functions and error metrics. Sec. 4 describes the theory and algorithms to characterize and compute the symmetric function closest to a given one, under unbounded approximation, whereas heuristics for partial symmetrization under a bounded error rate are discussed in Sec. 5. Theory and algorithms for approximations targeting D-reducible functions are described in Sec. 6. Experimental results are discussed in Sec. 7, and conclusions outlined in Sec. 8.

2 Related work

In this section we review some representative previous work on approximate logic synthesis.

The problem of approximate two-level logic synthesis has been studied in [34], where the authors design a two-level approximate logic synthesis algorithm with the objective of synthesizing an SOP circuit with fewer literals under a constrained error rate (i.e., how often the circuit can produce erroneous outputs). The main idea of this algorithm is to identify heuristically output values that can be complemented from 0 to 1 in order to expand products and reduce the number of literals in the final SOP representation. This approach has been generalized to three-level logic synthesis in [8], where the authors propose an approximate logic synthesis heuristic for synthesizing 2-SPP circuits, i.e., three-level EXOR-AND-OR forms with EXOR gates with fan-in 2.

For the approximate synthesis of multi-level circuits, Shin and Gupta propose in [35] a scheme where a node in the circuit is assumed to have a stuck-at-fault and the circuit is simplified by propagating this redundancy.

In [39] the authors describe a framework called SALSA (Systematic methodology for Automatic Logic Synthesis of Approximate circuits) to synthesize approximate circuits under a given error constraint. Given the description of a logic circuit and a constraint on the errors that could be tolerated, SALSA synthesizes an approximate version of the circuit that adheres to the quality constraint. SALSA encodes the quality constraints using logic functions and captures the flexibility that they allow as Approximation Don't Cares (ADCs), which are used for circuit simplification using traditional don't care based optimization techniques.

A tool called SASIMI has been presented in [40]. The key insight behind SASIMI is to identify signal pairs in the circuit that assume the same value with high probability, and substitute one for the other. While these substitutions introduce functional approximations, they might result in some logic to be eliminated from the circuit while also enabling downsizing of gates on critical paths. The authors thus propose an automatic synthesis framework that performs substitution and simplification iteratively, while ensuring that a user-specified quality constraint is satisfied.

Another contribution to this field is described in [28] and [29], where the authors address the problem of approximate logic synthesis under arbitrary error magnitude (i.e., the maximum amount by which the numerical value at the outputs of a circuit can deviate from the exact value) and error rate constraints. In particular, in [28] the authors focus on two-level approximate synthesis and propose a two-phase approach. First, they study the problem when constrained only by the error magnitude and they reduce it to minimization of Boolean relations, the latter being performed by a fast solver [2]. Then, they describe an efficient heuristic algorithm to refine iteratively the magnitude-constrained solution until when the error rate constraint is satisfied. This algorithm finds the optimal set of minterms on which the exact outputs must be enforced, and systematically corrects, in a greedy fashion, erroneous outputs of the Boolean relation solution that lead to the smallest cost increase until the error rate constraint is met. Furthermore, in [29] the authors address the multi-level approximate logic synthesis problem and develop a heuristic that synthesizes approximate Boolean networks with reduced gate count by using external don't care sets. Boolean relations are again used to formulate the error magnitude constrained problem, then the more general error magnitude and frequency constrained problem is solved with an iterative and greedy algorithm.

A statistically certified framework based on parallelized stochastic optimization was presented in [27]. The framework performs statistical testing to certify the quality of the optimized circuits with high confidence. This is accomplished by continuously monitoring the quality of the generated designs.

Other papers on approximate logic synthesis have been published over the past three years. In [26], the authors present an interesting integration between threshold logic and approximate computing and propose a synthesis algorithm to obtain cost-efficient approximate threshold logic circuits with an error rate guarantee. A new approximate logic synthesis technique based on Boolean matrix factorization is proposed in [19]. Moreover, a reinforcement learning based approximate logic synthesis framework called Q-ALS has been presented in [32]. Q-ALS exploits Boolean difference calculus to estimate the maximum error rate that each node of the given network can tolerate so that the total error rate at each output does not exceed the predetermined maximum error rate, and the worst case delay and the total area are minimized. Maximum Hamming Distance (MHD) between exact and approximate truth tables of cuts of each node is used as error metric.

Finally, a novel heuristic search method for two-level approximate logic synthesis under the error rate constraint was proposed recently in [37]. Here the key idea is to search for an optimal set of input combinations for 0-to-1 output complement. The experiments reported in the paper show that the proposed search method is more effective than previous state-of-art methods, especially for multiple-output circuits.

3 Preliminaries

In this section we introduce and review the two classes of regular Boolean functions considered in our work.

3.1 Symmetric Boolean functions

The concept of symmetry has been extensively studied and applied in several contexts, such as function classification, functional decomposition in technology-independent logic synthesis, Boolean matching in technology mapping, formal verification, and binary decision diagram (BDD) minimization (see for instance [21], [23], [25], [30], [33], [41]).

Let $f : \{0,1\}^n \rightarrow \{0,1\}$ be a completely specified Boolean function, and $X = \{x_1, x_2, \ldots, x_n\}$ be the set of its input variables, then the function f is called *symmetric*, or *totally symmetric*, if it is invariant under all permutations of the input variables; f is said to be *partially symmetric* if it remains invariant under any permutation of a proper subset of the input variables, of size at least 2.

A function can be partially symmetric with respect to different subsets of variables. More precisely, symmetry of a completely specified Boolean function f on pairs of input variables leads to an equivalence relation on the set X. Thus, there exists a unique minimal partition P of X into disjoint subsets $S_1, S_2, \ldots, S_k$, with $k \leq n$, given by the equivalence classes of this relation. These sets are called *symmetry sets*. If $k = 1$, then f has only one symmetry set which coincides with X, thus f is totally symmetric, while if $k = n$, f presents no symmetries. In all other cases $1 < k < n$, f is partially symmetric. In this case, at least one symmetry set contains two or more variables.

Observe that a totally symmetric function f depends only on the number of ones in the input minterm, thus it can be described by its *value vector* $v = [v_0, v_1, \ldots, v_n]$, where $f(x_1, x_2, \ldots, x_n) = v_i$ if $x_1 + x_2 + \ldots + x_n = i$, i.e., on all minterms of Hamming weight i.

In the next sections, we describe an efficient algorithm for computing the symmetric function closest to a given Boolean function, and we exploit it in the approximation framework. The concept of the minimal distance between a Boolean function and the set of symmetric functions has been considered in some recent papers [14], [36], where the authors have introduced the notion of *asymmetry* of a function f, defined as the minimum number of function values that must be changed so that f becomes totally symmetric. In particular, in [36] the authors study and characterize the set of functions that are *maximally asymmetric*, i.e., those that are maximally distant from symmetric functions.

Very efficient algorithms are known in literature for detecting various type of symmetries and computing the symmetry sets for Boolean functions [30], [33]. Algorithms are usually based on checking the equality of two-variable cofactors of the function, in order to find all pairs of symmetric variables. Using this information, larger sets of symmetric variables can be constructed by applying the transitivity of the symmetry relation. This basic approach has been improved in several ways, as reviewed in [30], where an improved method with worst-case complexity cubic in the size of the BDD representing the input function (but close to linear for practical benchmarks) has been presented.

Symmetric functions have the following features:

1) They have compact realizations in some logic architectures; e.g., a totally symmetric Boolean function with n input variables has a BDD of size $O(n^2)$ (independent from the variable order) which maps into a network of MUXes of the same size (see [3], [16], [20]).
2) They can be implemented with circuits of linear size. Indeed, the output of a totally symmetric function only depends on the number $0 \leq k \leq n$ of ones in the input vector, which can be represented using $\lceil \log_2(n+1) \rceil$ bits. Thus, the idea is to compose a circuit that counts the number k of ones in the original n-bit input vector with a $\lceil \log_2(n+1) \rceil$-input circuit that outputs the value of f on vectors with k ones. The counter component can be realized by a circuit of linear size using n full adders (see [18], [38]), and the second circuit, which depends on $\lceil \log_2(n+1) \rceil$ variables, can be implemented in linear size in n.

3.2 D-reducible functions

Dimension-reducible functions (shortly, *D-reducible functions*, [5], [6], [7]) are functions whose on-set minterms are contained in a linear (or affine) space A strictly smaller than the whole Boolean space $\{0,1\}^n$. Therefore, a D-reducible function f can be represented as $f = \chi_A \cdot f_A$, where χ_A is the characteristic function of A and f_A is the projection of f onto A. Notice that D-reducibility is different from degeneracy (not being sensitive to some variables): e.g., the function $f(x_1, x_2, x_3) = x_2 + x_3$ is degenerate, but not D-reducible since its onset includes the points $001, 010, 011, 101, 110, 111$ which do not fit in the Boolean space B^2; instead the function $f(x_1, \ldots, x_n) = x_1 \oplus x_2 \oplus \cdots \oplus x_n$ is D-reducible since its on-set is contained in a Boolean subspace of dimension $n-1$, but is not degenerate since it depends on all n variables [7].

The D-reducibility of a function f can be exploited in the minimization process: the idea is to minimize the projection f_A of f onto A, instead of f. In particular, the characteristic function χ_A can be always represented with an AND of EXORs of literals (see [5], [7]), and the projection f_A can be synthesized in any framework of logic minimization, e.g., two level logic, three-level logic, or general multi-level minimization. Observe that the synthesis of f_A could be easier than the synthesis of f, since f_A depends on a reduced number n_A of variables, where $n_A < n$. Moreover, the size of the network for f_A can be smaller than the size of the corresponding network for f. Indeed f and f_A have the same number of on-set minterms, but f_A is defined in a smaller space and its on-set minterms are less sparse. This approach thus requires two steps: *(i)* deriving the space A and the projection f_A; *(ii)* minimizing f_A in a given logic framework.

In the SOP framework this method is particularly convenient because if we project a function onto a smaller Boolean space (depending on fewer variables) we have the chance of reducing the Hamming distances among its minterms in order to merge them forming larger cubes in the final SOP form (as also discussed in [4]).

We refer the reader to [7] for a comprehensive and formal introduction to D-reducibility. Here, we just give an intuitive presentation through an example. Consider the function f in the Karnaugh map on the left side of Figure 1, f is D-reducible since its on-set is entirely contained in the three-dimensional space A marked with circles in the Karnaugh map. We can therefore study the new function f_A that depends only on three variables, represented in the Karnaugh map on the right side of the figure. Notice that f and f_A have the same number of on-set minterms, but these are now compacted in a smaller space, whose description requires less variables (in this case three instead of four). If we synthesize f and f_A in the classical SOP framework we obtain $f = \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 + \overline{x}_1x_2x_3\overline{x}_4 + x_1\overline{x}_3x_4 + x_1x_2x_4$, and $f_A = x_1x_2 + x_2x_3 + \overline{x}_2\overline{x}_3$, respectively. The overall number of products has decreased from 4 to 3. As proposed in [5], [7], if we multiply the SOP for f_A by the characteristic function χ_A of the space A, we can derive a new and more compact form describing the original function f. For the current example, as A is represented by the EXOR $(x_1 \oplus \overline{x}_4)$, we get the form

$$f = (x_1 \oplus \overline{x}_4)(x_1x_2 + x_2x_3 + \overline{x}_2\overline{x}_3)\,.$$

This form contains 8 literals, while the SOP for f contains 14 literals.

The test that establishes whether a function f is D-reducible and the computation of the smallest space A containing f and of the projection f_A can be easily performed in polynomial time starting from any SOP representation of f. As shown in [5], [7], this kind of regularity if quite common: a large percentage (about 70%) of the functions in the classical ESPRESSO benchmark suite have at least a D-reducible output.

3.3 Error metrics for approximate synthesis

Approximate logic synthesis has been studied under different error metrics, and therefore different cost functions. The two main error constraints that have been considered are *Error magnitude (EM)* and *Error rate (ER)*. The *error magnitude* for a set of outputs is defined as the maximum amount by which the numerical value at the outputs of a circuit can deviate from the exact value, and it is used typically in arithmetic circuits to quantify the numerical error. Instead, the *error rate* represents the percentage of all input vectors that produce the erroneous outputs in the approximate circuits. Composite metrics have also been defined using ER and EM. In this paper we use metrics based on the error rate.

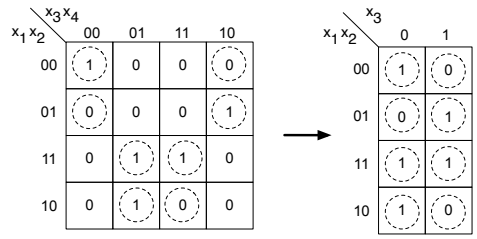

Fig. 1. Karnaugh maps of a D-reducible function f (on the left) and its corresponding projection f_A (on the right).

The concept of error rate has been specialized in [8], following the approach from [34], into the notions of *bit threshold* B_t and *minterm threshold* M_t. The first metric evaluates the overall number of complemented (wrong) output bits, while the second metric evaluates the number of input vectors on which the output computed by the circuit differs from the exact one by *at least one bit*. These two error measures coincide for single-output functions, while $B_t \geq M_t$ for multi-output ones. For practical implementations, the most adequate error measure should be selected depending on the specific application under study.

In this work, we consider only the bit threshold metric B_t, which can be computed starting from the error rate threshold as follows. Let $f : \{0,1\}^n \rightarrow \{0,1,-\}^m$ be a multi-output Boolean function with n inputs and m outputs, and let r denote the error rate, defined as the maximum percentage allowed of erroneous output bits. Then, $B_t = r \cdot m \cdot 2^n$.

In our study, we will also adopt a slightly different approach. In Sec. 4, we approximate the function f by the totally symmetric function with a minimal Hamming distance from f. Thus, instead of fixing an error rate r, we will first compute the minimum number e of output bits that must be complemented in order to transform f into a totally symmetric function. Then, from e, we will derive the error rate r induced by this transformation as follows. If the target function f is a single output function, with n inputs, then $r = e/2^n$. If the target function f is a multi-output function with n inputs and m outputs, then $r = \frac{\sum_{i=1}^{m} e_i}{m\,2^n}$, where e_i denotes the minimum number of output bits that must be complemented to transform the i-th output function into a totally symmetric one. The value of r should then be compared with the error rate that is considered acceptable for the application under study, to verify whether the approximation is feasible or not.

In Sec. 5, we discuss different strategies that can be

adopted to *enhance* the symmetry of the target function within a given error rate r defined in advance.

4 Unbounded approximation with a totally symmetric function

In the context of approximate logic synthesis, we now study how to modify some outputs of a Boolean function f in order to maximize its structural regularity and derive a *regular approximation* f' whose logic implementation might be of reduced complexity, smaller area and delay, as discussed in Section 1. The first type of regularity that we consider is symmetry. To this aim, we propose here an algorithm that, given a single output Boolean function f, computes the *minimum number* of outputs that must be changed in order to transform f into a totally symmetric function. In other words, the proposed algorithm computes the totally symmetric function f' closest to f, implicitly assuming an unbounded error rate threshold. We will discuss in Section 5 some strategies that can be adopted to enhance the symmetry of the target function f within a given error rate r defined in advance. Finally, in Section 6 we will analyze the approximation with D-reducible functions and propose a polynomial heuristic algorithm to find a D-reducibiile approximation of an incompletely specified function.

4.1 Completely specified functions

Let $f : \{0,1\}^n \to \{0,1\}$ be a completely specified Boolean function depending on n binary variables. Our algorithm for computing the closest symmetric approximation of f starts from a minimal disjoint sum of products form (DSOP) representing f [11]. Recall that in a DSOP representation, each minterm in the on-set of f is covered by exactly one product. This representation can be derived by applying an efficient heuristic algorithm, as for instance the one described in [11], or by building the BDD representation of f as proposed in [17]. In fact, a DSOP form can be extracted in a straightforward way from a BDD, as different one-paths correspond to disjoint cubes. We refer the reader to [11] for more details and references on DSOP forms and their applications.

The first step of the algorithm consists in computing, for each $0 \le w \le n$, the number of minterms in the on-set of f with Hamming weight w, where the Hamming weight of a minterm $x_1 x_2 \cdots x_n$ is defined as the number of ones among $x_1, x_2, \ldots, x_n$.

Proposition 1. Let p be a product in a DSOP representation of f, containing $d \le n$ literals, k of which are positive. Let $h = n - d$ be the number of don't care variables, i.e., the variables that do not appear in p. Then, for each $0 \le w \le n$, the number $T_p(w)$ of on-set minterms with Hamming weight w covered by p is given by

$$T_p(w) = \begin{cases} \binom{h}{w-k} & k \le w \le k+h \\ 0 & o/w\,. \end{cases}$$

Proof. The Hamming weight of the 2^h minterms covered by p is at least k and at most $k+h$, as p contains k positive literals and h don't cares variables. Then, the proposition follows immediately since, for any $0 \le i \le h$, there are exactly $\binom{h}{i}$ ways to choose the i variables to set to 1, out of the h variables that do not appear in p. ■

For example, in the Boolean space B^4, consider the function f with DSOP representation $\overline{x}_2\overline{x}_3x_4 + \overline{x}_1\overline{x}_3\overline{x}_4 + \overline{x}_1\overline{x}_2x_3\overline{x}_4 + x_2x_3\overline{x}_4$, and its product $p = \overline{x}_2\overline{x}_3x_4$. For p we have that $k = 1$ and $h = 1$. Therefore, $T_p(0) = T_p(3) = T_p(4) = 0$, $T_p(1) = \binom{1}{0} = 1$, and $T_p(2) = \binom{1}{1} = 1$. In fact, p covers two minterms: one, 0001, with Hamming weight 1 (i.e., $T_p(1) = 1$) and one, 1001, with Hamming weight 2 (i.e., $T_p(2) = 1$).

Summing the contribution $T_p(w)$ of each product p in the DSOP representation of f, we then derive the exact number $T(w)$ of on-set minterms of Hamming weight w, for each $0 \le w \le n$. Observe that, f is a totally symmetric function if and only if, for all $0 \le w \le n$, $T(w) = 0$ or $T(w) = \binom{n}{w}$.

For instance, considering all the products in the previous example, we have that $T(0) = 1$, $T(1) = 3$, $T(2) = 2$, $T(3) = 1$, and $T(4) = 0$. We can note that f is not symmetric since some of the non-zero values of T are not equal to the corresponding binomial: i.e., $T(1) \ne \binom{4}{1} = 4$, $T(2) \ne \binom{4}{2} = 6$, $T(3) \ne \binom{4}{3} = 4$.

The second phase of the algorithm consists in deriving the totally symmetric function f' closest to f. For all $0 \le w \le n$, we compute the minimum number $E(w)$ of minterms on which the output bit of f must be complemented to make the function constant on all inputs of Hamming weight w. Observe that $E(w)$ is given by

$$E(w) = \min\left\{T(w), \binom{n}{w} - T(w)\right\}.$$

In particular, if $E(w) = T(w)$, then f' is derived from f by a 1 to 0 complement of the output bit on the on-set minterms of Hamming weight w, otherwise f' is derived by a 0 to 1 complement on the off-set minterms of Hamming weight w. The overall minimum number of output bits that must be changed in order to transform f into a totally symmetric function, is then given by $e = \sum_{w=0}^{n} E(w)$.

Following the previous example, we have that $E(0) = \min\{1, \binom{4}{0} - 1\} = 0$, $E(1) = \min\{3, \binom{4}{1} - 3\} = 1$, $E(2) = \min\{2, \binom{4}{2} - 2\} = 2$, $E(3) = \min\{1, \binom{4}{3} - 1\} = 1$, and $E(4) = \min\{0, \binom{4}{4} - 0\} = 0$. In particular, the points with Hamming weight 1 are: 3 in the on-set and 1 (i.e., 1000) in the off-set of f. With 1 error we set all of them to 1. Moreover, the points with Hamming weight 2 are: 2 in the on-set and 4 in the off-set. With 2 errors we set all of them to 0. Finally, the points with Hamming weight 3 are: 1 in the on-set and 3 in the off-set, with 1 error we set all of them to 0. This means that the value vector of the closest symmetric function f' is $v = [1, 1, 0, 0, 0]$, and $e = 4$.

The overall computation is depicted in Table 1, where the first two columns report the weigh w and the number of points that have weigh w (i.e., $\binom{n}{w}$), respectively. The third and forth columns show the number of points in the on-set ($T(w)$) and in the off-set ($\binom{n}{w} - T(w)$) having weight w, respectively. The column labeled $E(w)$ is the computed error. The column labeled "Approximation", in presence of an error, indicates what kind of approximation has been applied, i.e., a 0 to 1 or a 1 to 0 approximation. The last column represents the value vector (v_w) of the computed symmetric function f'.

TABLE 1
Example of error computation for a completely specified function.

w	$\binom{n}{w}$	$T(w)$	$\binom{n}{w} - T(w)$	$E(w)$	Approximation	v_w
0	1	1	0	0	no	1
1	4	3	**1**	**1**	$\mathbf{0 \to 1}$	**1**
2	6	**2**	4	**2**	$\mathbf{1 \to 0}$	**0**
3	4	**1**	3	**1**	$\mathbf{1 \to 0}$	**0**
4	1	0	1	0	no	0

TABLE 2
Example of error computation for an incompletely specified function.

w	$\binom{n}{w}$	$T(w)$	$D(w)$	$\binom{n}{w} - T(w) - D(w)$	$E(w)$	Approximation	v_w
0	1	0	0	1	0	no	0
1	4	3	1	0	0	no	1
2	6	**1**	1	4	**1**	$\mathbf{1 \to 0}$	**0**
3	4	2	1	**1**	**1**	$\mathbf{0 \to 1}$	**1**
4	1	1	0	0	0	no	1

Observe that the symmetric approximation f' of f is not necessarily unique. Indeed, whenever $T(w) = \frac{1}{2}\binom{n}{w}$, f' can be derived either by a 1 to 0 complement of f on the on-set minterms of Hamming weight w, or by a 0 to 1 complement of f on the off-set minterms of weight w.

4.2 Incompletely specified functions

We now discuss how to transform an incompletely specified Boolean function in a totally symmetric one, introducing the minimum number of errors. Since any don't care condition can represent a 0 or a 1, often, the generalization of a problem to an incompletely specified Boolean function implies a growth of the complexity of the resolution algorithm. Fortunately, we can show that, in this case, the resolution procedure is still polynomial. The intuition behind this fact is that don't cares with the same Hamming weight w should be all set to 0 or all set to 1. Therefore, the choice is performed for the whole subset of don't cares with the same weight w, which ranges between 0 and n, and not for any single don't care point.

More formally, let $f : \{0,1\}^n \to \{0,1,-\}$ be an incompletely specified Boolean function, and $X = \{x_1, x_2, \ldots, x_n\}$ be the set of its input variables. The minimum number of errors is given by

$$E(w) = \min\left\{T(w), \binom{n}{w} - T(w) - D(w)\right\},$$

where $D(w)$ is the number of don't care minterms of Hamming weight w, for each $0 \le w \le n$, computed in the same way as $T(w)$, considering a DSOP representation of the don't care minterms of f. Note that if $E(w) = T(w)$, then f' is derived from f assigning value 0 to all on-set and don't care-set points of Hamming weight w and the number of errors is given only by the number $T(w)$ of on-set minterms of weight w, otherwise f' is derived assigning value 1 to all off-set and don't care minterms of Hamming weight w. In this last case, the number of errors is given by the number of off-set minterms of Hamming weight w, i.e., $\binom{n}{w} - T(w) - D(w)$.

Let us consider, for example, the incompletely specified function f with on-set $f^{on} = \{0001, 0010, 0100, 0011, 1110, 1101, 1111\}$ and don't-care set $f^{dc} = \{1000, 0101, 1011\}$.

Table 2 shows the computation of the error and the corresponding value vector. The first two columns report the weigh w and the number of points that have weigh w (i.e., $\binom{n}{w}$), respectively. The third, fourth and fifth columns show the number of points in the on-set ($T(w)$), in the don't care set ($D(w)$), and in the off-set ($\binom{n}{w} - T(w) - D(w)$) having weight w, respectively. The column labeled $E(w)$ is the computed error. The column labeled "Approximation", in presence of an error, indicates which kind of approximation has been applied, i.e., a 0 to 1 or a 1 to 0 approximation. The last column (v_w) represents the value vector of the computed symmetric function f'. In this case, the overall error is 2. Observe that the error computation does not take into account the don't care set. In the example, while the don't care points with weight 2 are set to 0, the don't care points with weights 1 and 3 are set to 1.

4.3 Algorithms to represent totally symmetric functions

Algorithm 1 describes, in pseudocode, the strategy discussed in the previous subsections, in the general case of incompletely specified Boolean functions.

Let $f : \{0,1\}^n \to \{0,1,-\}$ be an incompletely specified Boolean function, and let f^{on} and f^{dc} denote its on-set and don't care-set, respectively. For the sake of simplicity, suppose that $f^{on} \cap f^{dc} = \emptyset$; otherwise, following the usual semantics, we consider $f^{on} \setminus f^{dc}$ as the on-set of f. The proposed algorithm starts from the DSOP representations of the on-set and of the don't care-set of the target function f.

Algorithm 1. Algorithm for computing the totally symmetric function closest to an incompletely specified target function under the bit threshold metric.

SymmetricApproximation (function f)

INPUT: An incompletely specified function $f = (f^{on}, f^{dc})$, with on- and don't care-set represented in DSOP form
OUTPUT: The value vector v of the totally symmetric function f' closest to f, and the number e of output bits of f that must be complemented to derive f'

```
e = 0
v = new array of n + 1 integers, initialized to 0
T = new array of n + 1 integers, initialized to 0
D = new array of n + 1 integers, initialized to 0
forall product p in DSOP(f^on) or in DSOP(f^dc)
   compute the number k of positive literals in p
   compute the number h of don't care variables in p
   for w = k to k + h do
         if (p ∈ DSOP(f^on))  T[w] = T[w] + C(h, w-k)
         else D[w] = D[w] + C(h, w-k)
for w = 0 to n do
      if (T[w] > C(n, w) - T[w] - D[w])
         v[w] = 1
         e = e + C(n, w) - T[w] - D[w]
      else
         v[w] = 0
         e = e + T[w]
return v, e
```

The algorithm has a time complexity polynomial in the number n of input variables and in the number t of products in the DSOP forms representing f^{on} and f^{dc}. More precisely, the time complexity is $O(n^2 + t \cdot n)$, where the term n^2 accounts for the cost of the computation (via dynamic programming) of all the binomial coefficients, while the term $t \cdot n$ represents the overall cost of the nested for-cycles.

The correctness is proved in the following theorem.

Theorem 1. Let f be a Boolean function represented in DSOP form. The proposed algorithm computes the totally symmetric function closest to f, i.e., a totally symmetric function f' that can be derived complementing the minimum number e of output bits of f.

Proof. The correctness of the algorithm follows from Proposition 1 and from the fact that each on-set minterm is covered by one and only one product in the DSOP representation of f^{on}, as well as each don't care-set minterm is covered by one and only one product in the DSOP representation of f^{dc}. Thus, the numbers $T(w)$ and $D(w)$ of the on-set and of the don't care-set minterms of Hamming weight w, for each $0 \leq w \leq n$, can be correctly computed summing the contribution of each product in the DSOP representations of f^{on} and f^{dc}. ■

Observe that in an approximate logic synthesis scenario, the minimum number e of output bits that must be changed to make f totally symmetric provides a sort of lower bound to the bit threshold B_t required in order to allow the approximation of f with f'. That is, whenever $e \leq B_t$, we can synthesize f' instead of f. In terms of error rate, we can consider the approximation feasible if the error rate $r = e/2^n$ induced by the substitution of f with f', is less or equal to the error rate considered acceptable for the application under study.

We finally describe a dynamic programming procedure, based on the recursive approach described in [22] (pp. 65-66), which can be used to build the BDD representation of the totally symmetric function f' computed by Algorithm 1, starting from the value vector v of f'. The idea is to first build an $(n+1)\times(n+2)$ matrix M of BDDs, where $M[i][j+1]$ is the BDD representing the set of all i-dimensional vectors of Hamming weight j, and then to select, according to the value vector v, the BDDs in the last row of the matrix M whose union corresponds to f'.

Algorithm 2. Algorithm for computing the BDD representation of a totally symmetric function, starting from its value vector.

ValueVectorToBDD (function f)

INPUT: The value vector v of a totally symmetric function with n inputs
OUTPUT: The reduced BDD representation of f

```
/* Dynamic programming computation of the matrix M */
M = new matrix of dimension (n + 1) × (n + 2)
M[0][1] = the BDD terminal node 1
for i = 0 to n do
    M[i][0] = the BDD terminal node 0
for i = 0 to n do
    for j = i + 2 to n + 1 do
        M[i][j] = the BDD terminal node 0
for i = 1 to n do
    for j = 1 to i + 1 do
        M[i][j] = BDD-ITE(x_i, M[i − 1][j − 1], M[i − 1][j])
/* BDD construction */
SymDD = the BDD terminal node 0
for i = 0 to n do
  if (v[i] == 1)  SymDD = BDD-OR(SymmDD, M[n][i + 1])
return SymDD
```

The time complexity of this algorithm is polynomial in the number n of input variables. Indeed, the first dynamic programming phase accounts for a cost $\Theta(n^2)$, since each entry in the table can be derived in constant time using the entries already computed. Then, the algorithm computes at most n unions of two BDDs representing symmetric functions, each of size $O(n^2)$. The cost of each union is upper bounded by the product of the sizes of the two BDDs. Moreover, the BDDs resulting from the union operations are always symmetric and their size remains $O(n^2)$. Thus, the computational cost of this second phase is at most $O(n^5)$, and the overall cost of the algorithm is $O(n^5)$.

4.4 Multi-output functions

The algorithms that we designed for single-output functions can be applied also to multi-output functions, by simply considering each output as a separate Boolean function. Indeed, once each output function has been transformed into a totally symmetric function, the overall multi-output function becomes totally symmetric as well, as proved in the following Proposition 2.

Let f be an incompletely specified Boolean function with n inputs and m outputs. For each i, $1 \leq i \leq m$, let f_i denote the single output function corresponding to the i-th output of f, and let f'_i be the totally symmetric function derived complementing the minimum number e_i of output bits of f_i, i.e., applying Algorithm 1 to each output of f. Finally, let f' be the multi-output function obtained substituting each output function f_i in f with the totally symmetric function f'_i, $1 \leq i \leq m$.

Proposition 2. The multi-output function f' is a totally symmetric function.

Proof. A multi-output function is totally symmetric if and only if it computes the same output vector on all minterms with the same Hamming weight. Now consider the function f' and observe that each single output of f' is totally symmetric, thus each single output is constant on all minterms with weight w, $0 \leq w \leq n$. As a consequence, f' outputs the same vectors on all minterms with Hamming weight w and the thesis immediately follows. ■

We now prove that not only f' is totally symmetric, it is also the totally symmetric function closest to f.

Proposition 3. The function f' derived applying Algorithm 1 to each output of f is the totally symmetric function that can be derived complementing the minimum number of output bits of f.

Proof. For each i, $1 \leq i \leq m$, let e_i denote the number of output bits complemented in order to transform the i-th output function of f into a totally symmetric one applying Algorithm 1. Moreover, let $e = \sum_{i=1}^{m} e_i$ denote the overall number of output bits complemented to make the whole f totally symmetric. Suppose that there exists a totally symmetric multi-output function g' closer to f. Then, there exists at least one output of f, say f_i, that can be made totally symmetric complementing $e'_i < e_i$ output bits, in contradiction with the minimality of e_i proved in Theorem 1. ■

Finally, the error rate r induced by the approximation of f with the totally symmetric function f' can be computed

as $r = \frac{e}{m\,2^n} = \frac{\sum_{i=1}^{m} e_i}{m\,2^n}$, where m is the number of outputs of f.

5 Symmetry-based approximation under a bounded error rate

In this section we discuss some strategies that can be adopted to enhance the symmetry of a function f within an error rate r defined in advance.

Let $f : \{0,1\}^n \rightarrow \{0,1,-\}^m$ be an incompletely specified function with n inputs and m outputs, and let r denote the error rate. From r, we can compute the bit threshold $B_t = r \cdot m \cdot 2^n$, i.e., the maximum number of output bits that we are allowed to complement to derive a *symmetric* approximation f' of f.

A first greedy strategy consists in simply sorting the outputs of the multi-output function with respect to the number of errors needed to transform each of them into a totally symmetric function. Starting from the output function with the lowest number of errors, we transform th outputs in symmetric functions till we reach the given bit threshold B_t. In this way, we enhance the symmetry of the overall function making some of its outputs totally symmetric. Observe that two possible approaches could be considered in this context: we could assign the same threshold to each output, splitting the bit threshold B_t in m equal parts, or apply the overall threshold value B_t to the function as a whole. In the first case, we approximate each output function independently, complementing at most $r2^n$ of its output bits.

An alternative method could make a function constant on all minterms with the same Hamming weight. The idea is basically to apply Algorithm 1 to f, and to stop the computation as soon as the number of complemented output bits reaches the bit threshold B_t. But instead of processing the subsets of minterms in order of increasing Hamming weight, we first compute, for each weight w and each output function f_i, the minimum number $E_i(w)$ of minterms on which the output bit of f_i must be complemented to make this output constant on all inputs of weight w. Then, we start from the weight w and from the output f_i on which f can be made constant (0 or 1) complementing the *least number* of output bits, and we proceed in a greedy way in ascending order of $E_i(w)$, until we reach the threshold B_t. At the end of the process, we obtain a function f' with some outputs that assume a constant value on minterms with equal weight for a subset of possible weights. A limitation of this approach is that making a function constant only for some weights does not necessarily enhance its overall symmetry and may not provide realizations of reduced complexity. This limitation has been confirmed by our experiments, showing that only the greedy strategy based on the total symmetrization of some outputs of the function provides interesting results.

We finally discuss a completely different approach based on the idea of enhancing the partial symmetry of a function, instead of its total symmetry, increasing the number of variables that can be permuted without changing the output. This task can be accomplished, for instance, increasing the cardinality of the biggest symmetry set adding a new variable to it. This new variable should be selected in a greedy way, as the variable whose insertion in the currently biggest symmetry set causes the least number of erroneous output bits.

More precisely, consider the partition P of the set of input variables $X = \{x_1, x_2, \ldots, x_n\}$ into the disjoint symmetry sets S_1, S_2, ..., S_k, with $k \leq n$. Let $S \subset \{x_1, x_2, \ldots, x_n\}$ be a symmetry set with maximal cardinality t. Our goal is to increase the cardinality of S adding a new variable to it.

Observe that f can be represented with a *cofactor vector*, i.e., a sequence of $t+1$ cofactors (or subfunctions) each depending on the $n-t$ variables outside S, obtained assigning all possible values to the t variables in S. Since f is symmetric in all variables in S, the number of different cofactors is $t+1$ instead of 2^t, as each cofactor depends only on the number of variables in S with value 1. If f is totally symmetric, then $t = n$ and the cofactor vector becomes the usual value vector v.

Let us denote with $f_{|S,w}$ the cofactor obtained assigning the value 1 to exactly w of the variables in S, for $0 \leq w \leq t$. In order to add a new variable y to S we need to change the value of the function in such a way that all cofactors corresponding to the $t+1$ variables in the set $S' = S \cup \{y\}$, depend only on the number of variables in S' with value 1.

For each w between 1 and t, we consider the two cofactors $f_{|S,w,y=1}$ and $f_{|S,w,y=0}$ obtained from $f_{|S,w}$ assigning to y the value 1 and the value 0, respectively. We leave the first cofactor unchanged, while we change the second one in order to make it equal to $f_{|S,w-1,y=1}$, so that all cofactors based on $S' = S \cup \{y\}$ and with the same weight become equal to each other[1]: $\forall\, w \in [1,t], \;\; f_{|S,w,y=0} \leftarrow f_{|S,w-1,y=1}$. The number of output bits that must be complemented to transform $f_{|S,w,y=0}$ in $f_{|S,w-1,y=1}$ is given by the Hamming distance between the two cofactors, which can be computed as $|f_{|S,w,y=0} \oplus f_{|S,w-1,y=1}|$, where for a function g, $|g|$ denotes the number of on-set minterms. Thus, the overall number of erroneous output bits induced by adding y to S can be computed as

$$E(y) = \sum_{w=1}^{t} \binom{t}{w} \; |f_{|S,w,y=0} \oplus f_{|S,w-1,y=1}| \,,$$

since the cofactor $f_{|S,w,y=0}$ occurs $\binom{t}{w}$ times in the original function f. If $E(y) \leq B_t$, y can be added to S. The same procedure can then be applied to the new function obtained, in order to further increase the cardinality of the biggest symmetry set within the bound $B_t - E(y)$. As before, the variable y should be selected in a greedy way, as the variable whose insertion in the biggest symmetry set S' causes the minimum number of erroneous output bits.

The correctness of this procedure is proved in the following theorem. Let f' be the function obtained from the target function f applying the procedure described above.

***Theorem* 2.** The function f' is symmetric in all variables in the set $S' = S \cup \{y\}$.

Proof. We show that the cofactors defined by S' depend only on the number of variables in S' with value 1, i.e., all cofactors with the same weight are equal to each other. Consider the $\binom{t+1}{w}$ value assignments to the $t+1$ variables

1. Alternatively, we could change the cofactor $f_{|S,w,y=1}$ making it equal to $f_{|S,w+1,y=0}$, for $0 \leq w < t$.

in S' of weight w, for $0 \leq w \leq t+1$, and observe that they can be partitioned into two subsets: the $\binom{t}{w-1}$ assignments with $y = 1$, and the $\binom{t}{w}$ assignments with $y = 0$. The cofactors defined by all assignments in the first group coincide, as they are all equal to $f_{|S,w-1,y=1}$. The thesis then follows from the fact that, by construction, all cofactors obtained from the assignments in the second group have been changed and made equal to $f_{|S,w-1,y=1}$. ■

6 Approximation to D-reducible functions

We now consider a different kind of approximation towards regularity. We consider the class of D-reducible Boolean functions, and discuss how to modify some outputs of a given target function f, in order to derive a *D-reducible approximation* of f, for a given error rate r. To ease the computation and, at the same time, to be technologically feasible, we only consider spaces A that can be represented by an AND of single literals and/or EXORs of two literals (2-EXOR factors).

For example, consider the D-reducible Boolean function f depicted on the left side of Figure 1 and its on-set $f_{on} = \{0000, 0110, 1001, 1101, 1111\}$. Let us define a new completely specified Boolean function f' with on-set $f'_{on} = f_{on} \cup \{1010\}$. The function f' is not D-reducible, since the point 1010 is not in the space A marked with circles in the Karnaugh map of Figure 1, and there is no other space $A' \subset \{0,1\}^n$ that entirely contains the on-set of f'. If we synthesize f' in the classical SOP framework we obtain $f' = \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 + \overline{x}_1x_2x_3\overline{x}_4 + x_1\overline{x}_3x_4 + x_1x_2x_4 + x_1\overline{x}_2x_3\overline{x}_4$. If we introduce an error and we insert the point 1010 in the off-set of f' we obtain the D-reducible function f whose representation is much more compact, i.e., $(x_1 \oplus \overline{x}_4)(x_1x_2 + x_2x_3 + \overline{x}_2\overline{x}_3)$.

So we propose an algorithm that, given a single output Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$, computes the minimum number of outputs that must be changed in order to transform f into a D-reducible function f_S whose projection space S can be described by an AND of literals and 2-EXOR factors. The algorithm starts from a minimal DSOP representing f, and looks for a $(n-1)$-dimensional space, described by a single literal or by an EXOR of two distinct literals, onto which to project the on-set of f with the least number of errors. The projection of f onto this space defines a D-reducible approximation f_S. If the distance d between f and f_S (i.e., the number of outputs on which the two functions differ) is smaller than the bit threshold $B_t = r \cdot 2^n$, we consider the approximation f_S instead of f.

Moreover, if the distance d is less than the bit threshold B_t, the algorithm proceeds iteratively on the projection f_S, i.e., it looks for a $(n-2)$-dimensional subspace $T \subseteq S$ onto which to project f_S, and so on. The overall projection space A computed by the algorithm is then given by the product (AND) of the single literals or 2-EXOR factors representing the spaces identified during each iterative step. The approximation f' of f can be computed projecting f onto the space A, and setting to 0 all on-set minterms that do not belong to A.

The proposed algorithm is thus based on a greedy heuristic, which proceeds iteratively making the locally optimal choice for the projection space at each phase. Observe that, while the first step of the algorithm, i.e., the projection onto a $(n-1)$-dimensional space S described by a single literal or by an EXOR of two literals, actually provides the D-reducible function *closest* to f, we have no guarantees that the overall approximation f', computed by the set of all projections, defines the closest D-reducible approximation of f. For example, consider a function f and suppose that the first step of the algorithm selects the space S where $x_i = 1$ as the most convenient one for the projection, and chooses the cofactor $f_{|x_i=1}$ as D-reducible approximation of f. Observe that the other cofactor $f_{|x_i=0}$ contains less on-set minterms than $f_{|x_i=1}$, otherwise the projection onto the space $x_i = 0$ would have been more convenient. Now suppose that $f_{|x_i=0}$ is a D-reducible function, while $f_{|x_i=1}$ is not. Then, any other projection starting from the chosen cofactor $f_{|x_i=1}$ increases the overall number of errors. On the other hand, with the initial choice of projecting (with a greater number of errors) f onto the space where $x_i = 0$, the subsequent projections would not introduce any additional error, possibly leading to a closer overall D-reducible approximation of f.

Let us now describe how at each step the algorithm finds the space better suited for the projection. Given a DSOP representation of the function f that must be projected, the algorithm computes the following parameters, for all $1 \leq i \leq n$ and for all $i < j \leq n$:

- the number $E_0(i)$ of on-set minterms that do not belong to the subspace S whose characteristic function is $\chi_S = \overline{x}_i$, i.e., the space where the variable x_i is always equal to 0;
- the number $E_1(i)$ of on-set minterms that do not belong to the subspace S whose characteristic function is $\chi_S = x_i$, i.e., the space where the variable x_i is always equal to 1;
- the number $E_=(i,j)$ of on-set minterms that do not belong to the subspace S whose characteristic function is $\chi_S = (x_i \oplus \overline{x}_j)$, i.e., the space where $x_i = x_j$;
- the number $E_\neq(i,j)$ of on-set minterms that do not belong to the subspace S whose characteristic function is $\chi_S = (x_i \oplus x_j)$, i.e., the space where $x_i \neq x_j$.

These parameters represent the number of outputs that must be changed (from 1 to 0) in order to transform f into a D-reducible function f' with projection space S described by $\overline{x}_i, x_i, (x_i \oplus \overline{x}_j)$, and $(x_i \oplus x_j)$, respectively. They can be computed as follows. Let p be a product representing a cube of dimension d in a DSOP of f:

- if the product contains $\overline{x}_i$, then we add 2^d to $E_0(i)$;
- if the product contains x_i, we add 2^d to $E_1(i)$;
- if the product does not contain the variable x_i, then we add 2^{d-1} to both $E_0(i)$ and $E_1(i)$.
- if x_i and x_j appear both as negative or both as positive literals in p, we add 2^d to $E_=(i,j)$;
- if x_i and x_j appear with a different complementation in p, we add 2^d to $E_\neq(i,j)$;
- finally, if at least one of the two variables x_i and x_j does not appear in p, we add 2^{d-1} to both $E_=(i,j)$ and $E_\neq(i,j)$.

Once these parameters have been evaluated for all n variables and all $\binom{n}{2}$ pairs of variables, the algorithm projects the input function f onto the space that con-

TABLE 3
Example of error computation D-reducible approximation for the on-set $\{0000, 0110, 1010, 1001, 1101, 1111\}$.

$E_0(1)$	$E_0(2)$	$E_0(3)$	$E_0(4)$	$E_1(1)$	$E_1(2)$	$E_1(3)$	$E_1(4)$	$E_=(1,2)$	$E_=(1,3)$	$E_=(1,4)$	$E_=(2,3)$	$E_=(2,4)$	$E_=(3,4)$	$E_\neq(1,2)$	$E_\neq(1,3)$	$E_\neq(1,4)$	$E_\neq(2,3)$	$E_\neq(2,4)$	$E_\neq(3,4)$
4	3	3	3	2	3	3	3	3	3	**1**	2	2	4	3	3	5	4	4	2

tains the majority of on-set minterms, i.e., the space corresponding to the parameter with the smallest value d that does not exceed the bit threshold B_t: $d = \min_{\substack{1\leq i\leq n\\ 1\leq i<j\leq n}} \{E_0(i), E_1(i), E_=(i,j), E_\neq(i,j)\}$. In particular, if the parameter of minimum value d corresponds to a space S described by a single variable, e.g., $\overline{x}_i$ or x_i, then the projection f_S is obtained simply by assigning the correct value, 0 or 1, to the variable x_i in the DSOP for f. Instead, if the parameter with minimum value corresponds to a space S described by an EXOR of two variables, e.g., $(x_i \oplus \overline{x}_j)$ or $(x_i \oplus x_j)$, then the projection f_S can be computed substituting in the DSOP for f each occurrence of x_j with x_i or $\overline{x}_i$, respectively. In both cases, the approximation of f is given by the function $f' = \chi_S f_S$, and is obtained from f by complementing from 1 to 0 all on-set minterms that do not belong to S. In case of more parameters with the same minimum value d, the algorithm heuristically chooses, if possible, the space described by just a single literal, to simplify the algebraic description and derive an implementation of smaller area. If $d > B_t$, the algorithm stops the computation without performing the projection, otherwise it computes f_S and proceeds iteratively trying to project f_S onto a subspace $T \subseteq S$ within the updated minterm threshold $B_t - d$. At the end of the computation, the algorithm outputs the D-reducible approximation f' of the target function f, the characteristic function of its projection space, and the residual minterm threshold that can be exploited for further approximations.

For example, consider again the D-reducible Boolean function f depicted on the left side of Figure 1 and its on-set $f_{on} = \{0000, 0110, 1001, 1101, 1111\}$, and the non-D-reducible function f' with on-set $f'_{on} = f_{on} \cup \{1010\} = \{0000, 0110, 1010, 1001, 1101, 1111\}$ described in the previous example.

Let us now compute the number $E_0(i)$ of on-set minterms in f'_{on} that do not belong to the subspace whose characteristic function is $\overline{x}_i$, i.e., the space where the variable x_i is always equal to 0. For instance, when $i = 1$, the on-set minterms with variable x_1 equal to 1 (i.e., they do not belong to the subspace $\overline{x}_i$) are 4 (i.e., $\{1010, 1001, 1101, 1111\}$), thus $E_0(1) = 4$. The number $E_1(i)$ can be computed in a similar way, considering now the on-set minterms f'_{on} that do not belong to the subspace whose characteristic function is x_i. For instance, when $i = 1$, the on-set minterms with variable x_1 equal to 0 (i.e., they do not belong to the subspace x_i) are 2 (i.e., $\{0000, 0110\}$), thus $E_0(1) = 2$. The overall computation for $E_0(i)$ and $E_1(i)$ is shown in the first 8 columns of Table 3. $E_=(i,j)$ is the number of on-set minterms that do not belong to the subspace whose characteristic function is $(x_i \oplus \overline{x}_j)$, i.e., it is the number of on-set minterms where $x_i \neq x_j$. For example, the on-set minterms where $x_1 \neq x_2$ are $\{0110, 1010, 1001\}$. Thus, $E_=(1,2) = 3$. Similarly, $E_\neq(i,j)$ is the number of on-set minterms that do not belong to the subspace $(x_i \oplus x_j)$, i.e., it is the number of on-set minterms where $x_i = x_j$. For example, the on-set minterms where $x_1 = x_2$ are $\{0000, 1101, 1111\}$. Thus, $E_\neq(1,2) = 3$. The overall computation for $E_=(i,j)$ and $E_\neq(i,j)$ is shown in the last 12 columns of Table 3. Table 3 shows that the minimum number of errors is 1 that corresponds to the space with characteristic function $(x_1 \oplus \overline{x}_4)$. The approximate function is then composed by the minterms in f'_{on} such that $(x_1 = x_4)$, i.e., $\{0000, 0110, 1001, 1101, 1111\}$. Note that the approximated D-reducible function is the one of Figure 1.

Algorithm 3. Algorithm for computing a D-reducible approximation of an incompletely specified target function under the bit threshold metric.

D-reducibleApproximation (function f)

INPUT: An incompletely specified function $f = (f^{on}, f^{dc})$ on n variables, with on-set in DSOP form, and a bit threshold B_t
OUTPUT: A D-reducible function f', approximating f within the threshold B_t, or f is such approximation is not possible

```
function AppSynDred (f, n, B_t)
  if n = 0 then return 1
  E_0 = new array of n integers, initialized to 0
  E_1 = new array of n integers, initialized to 0
  E_≠ = new array of (n 2) integers, initialized to 0
  E_= = new array of (n 2) integers, initialized to 0
  /* Error evaluation */
  forall product p in DSOP(f^on)
    compute the number d of don't care variables in p
    for i = 1 to n do
      if x_i occurs in p then
        E_0[i] = E_0[i] + 2^d
        for j = i + 1 to n do
          if x_j occurs in p then E_≠[i,j] = E_≠[i,j] + 2^d
          else if x̄_j occurs in p then E_=[i,j] = E_=[i,j] + 2^d
          else
            E_≠[i,j] = E_≠[i,j] + 2^(d-1)
            E_=[i,j] = E_=[i,j] + 2^(d-1)
      else if x̄_i occurs in p then
        E_1[i] = E_1[i] + 2^d
        for j = i + 1 to n do
          if x_j occurs in p then E_=[i,j] = E_=[i,j] + 2^d
          else if x̄_j occurs in p then E_≠[i,j] = E_≠[i,j] + 2^d
          else
            E_≠[i,j] = E_≠[i,j] + 2^(d-1)
            E_=[i,j] = E_=[i,j] + 2^(d-1)
      else
        E_0[i] = E_0[i] + 2^(d-1)
        E_1[i] = E_1[i] + 2^(d-1)
        for j = i + 1 to n do
          E_≠[i,j] = E_≠[i,j] + 2^(d-1)
          E_=[i,j] = E_=[i,j] + 2^(d-1)
  /* Selection of the projection space S */
  e_1 = min({E_0, E_1})
  e_2 = min({E_≠, E_=})
  if e_2 < e_1 then e = e_2    else e = e_1
  /* Computation of χ_S and f_S */
  if e < B_t then
    compute the characteristic function χ_S of the projection space
    compute the projection f_S = (f_S^on, f_S^dc) of f onto the space S
    derive a DSOP for f_S^on from DSOP(f^on)
    return χ_S · AppSynDred (f_S, n − 1, B_t − e)
  else return f
```

The time complexity of this algorithm is again polynomial in the number of input variables and in the number t of products in the DSOP representation of the on-set of f. More precisely, the overall cost of the algorithm is $O(t \cdot n^3)$. Indeed, the algorithm performs at most n recursive calls, each of cost $O(t \cdot n^2)$. Observe that, inside each recursive call, the computations of χ_S, f_S, and DSOP($f_S^{(on)}$) require time linear in the input size, and are therefore dominated by the cost $O(t \cdot n^2)$ of the nested for-cycles.

We finally discuss how to approximate an incompletely specified Boolean function with a D-reducible one. First of all, observe that for this kind of approximation, it is never convenient to move a minterm from the don't care-set to the on-set, as this may increase the cost of the projections (i.e., the number of 1 to 0 complementations required) and it never decreases it. For this reason, all don't care minterms can be considered as off-set minterms when computing the parameters $E_0(i), E_1(i), E_=(i,j), E_\neq(i,j)$, for all $1 \leq i \leq n$ and for all $i < j \leq n$, that measure the costs of the projections. Then, when the best space S has been selected, the projection f_S can be computed setting to 0 all don't care minterms that do not belong to S, while leaving unspecified those that belong to S. Algorithm 3 describes, in pseudocode, the strategy discussed for the general case of incompletely specified Boolean functions.

The time complexity of the proposed heuristic is polynomial in the number n of input variables and in the number of products in the DSOP form representing the input function f.

7 Experimental results

In this section we discuss the experimental results for the proposed techniques. We first evaluate the method proposed for symmetric functions and then we show the experimental results for D-reducible functions.

7.1 Approximation to Symmetric Functions

We first discuss the experimental results obtained by applying the algorithms described in Sections 4 and 5. We considered both the classical ESPRESSO benchmark suite [43] and the new EPFL combinational benchmark suite [1]. The computational experiments were performed on a Linux Intel Core i7-7700 CPU with 8 GB of RAM. The algorithms have been implemented in C, using the CUDD library for BDDs to represent Boolean functions.

Table 4 compares the BDD dimension for the benchmark functions and for their closest symmetric approximations in the unbounded and bounded error models. We report a significant subset of the experiments. The first column reports the name of the benchmarks and the number of their inputs and outputs. The second column shows the dimension (number of nodes) of the BDDs representing the on-set of the benchmark functions. The first group of columns, labeled *Sym Unbounded*, refers to the closest totally symmetric function in the unbounded model (Section 4). The first column in this group shows the BDD size of the symmetric function, the second column contains the required error rate, and the third column reports the percentage gain of the BDD of the closest symmetric function. The second group, labeled *Sym Bounded*, provides the results for the bounded model based on the symmetrization of a subset of the outputs (Section 5) with a fixed error rate $r = 5\%$. The last column provides the sum of computational times (in seconds) of the two approaches. In general, the average gain for the unbounded model is 69% with an average error rate 15%. In the bounded context, with a fixed error rate 5%, the average gain is about 31%. Note, however, that there are a few cases where the symmetrization of the benchmark produces BDDs of larger size (e.g., $i2c$ and $router$). Due to the polynomial nature of the algorithms, the computational times are very small. The second bounded approach described in Section 5, which makes a function constant only for some weights, has been tested on the same benchmarks with low quality results, i.e., the BDD size increases by 150%. This confirms the theoretical intuition that fixing a constant value for just some weights does not give an interesting "symmetry" property to the function.

In order to assess the effectiveness of our approach, we compared the BDDs generated by our experiments with those of the approximated functions derived by the approximation method proposed in [34], keeping $r = 5\%$ as error rate. The experiments, conducted on the classical ESPRESSO benchmark suite [43], show that our bounded model generates an average reduction of the number of nodes of about 20%, and our unbounded model gives an average reduction of the number of nodes of about 81%. These results show that an approximation synthesis method designed for augmenting the symmetry of a function can significantly help in deriving more compact BDD representations.

7.2 Approximation to D-reducible Functions

We now discuss the experimental results obtained by applying the technique proposed in Section 6 on the classical ESPRESSO benchmark suite [43]. In particular, we are interested in evaluating experimentally the amount of variables that are not needed to describe the projection f_A, due to the approximation procedure. In fact, a function that depends on fewer variables is, in general, less complex and easier to be synthesized. For this purpose, we first consider an approximation threshold of 5%. In this case, we get an average reduction of the number of variables of about 40.5%. This result seems very encouraging.

On the other hand, we gather from the experiments that an approximation threshold of 5% corresponds to an average 36.4% of errors in the on-set. This means that, in many benchmarks, the on-set is very small with respect to the entire Boolean space. In these cases, the number of swapped outputs, from 1 to 0 in the on-set, may end up as being too high. To address this issue, we set the thresholds of our approximations by considering only the on-set dimension of the benchmarks. We then obtain the results reported in Table 5, which shows that we can get an interesting reduction in the number of variables already with a low approximation threshold (on the dimension of the on-set). Indeed, we have an high reduction in the number of variables even when considering high approximation thresholds. Table 5 shows that an approximation threshold of about 5% is a good trade-off to obtain a significant decrease on the number of variables (i.e., 18.45%).

TABLE 4
Experimental comparison with the closest-symmetric in the unbounded and bounded models.

bench.(in/out)	BDD	Sym Unbounded BDD	ER (%)	gain	Sym Bounded (5%) BDD	gain	time (s)
add6 (12/7)	309	122	42.2	60.5	226	26.9	0.01
alcom (15/38)	95	131	6.9	-37.9	83	12.6	0.01
alu2 (10/8)	242	42	9.7	82.6	177	26.9	0.01
alu3 (10/8)	237	43	9.7	81.9	170	28.3	0.01
apla (10/12)	212	40	1.2	81.1	40	81.1	0.01
b2 (16/17)	4424	79	29.0	98.2	4215	4.7	0.15
b9 (16/5)	173	80	37.9	53.8	173	0.0	0.01
bc0 (26/11)	590	490	32.9	17.0	635	-7.6	0.02
bca (26/46)	1428	52	0.1	96.4	52	96.4	0.02
bcb (26/39)	1268	52	0.1	95.9	52	95.9	0.01
bcc (26/45)	1116	27	0.1	97.6	27	97.6	0.02
bcd (26/38)	843	27	0.1	96.8	27	96.8	0.01
cavlc (10/11)	508	37	10.3	92.7	291	42.7	0.01
chkn (29/7)	742	234	7.8	68.5	577	22.2	0.02
cps (24/109)	1710	48	2.1	97.2	48	97.2	0.03
ctrl (7/26)	101	27	13.4	73.3	65	35.6	0.01
dec (8/256)	510	16	0.4	96.9	16	96.9	0.01
dk48 (15/17)	189	1	0.0	99.5	1	99.5	0.01
ex5 (8/63)	268	32	5.5	88.1	57	78.7	0.01
exps (8/38)	521	30	14.8	94.2	336	35.5	0.01
i2c (147/142)	2873	8306	22.1	-189.1	7287	-153.6	2.24
in0 (15/11)	518	89	23.2	82.8	410	20.9	0.01
in1 (16/17)	4424	79	29.0	98.2	4215	4.7	0.10
in2 (19/10)	2361	101	12.1	95.7	1195	49.4	0.05
in5 (24/14)	492	144	10.6	70.7	200	59.4	0.01
in7 (26/10)	235	229	23.0	2.6	166	29.4	0.01
int2float (11/7)	359	51	25.5	85.8	345	3.9	0.01
mark1 (20/31)	253	153	0.0	39.5	153	39.5	0.01
max1024 (10/6)	261	93	43.2	64.4	261	0.0	0.01
max128 (7/24)	130	78	26.0	40.0	144	-10.8	0.01
max46 (9/1)	75	1	12.1	98.7	75	0.0	0.01
max512 (9/6)	148	65	38.0	56.1	168	-13.5	0.01
opa (17/69)	513	172	8.0	66.5	118	77.0	0.01
pdc (16/40)	695	102	4.0	85.3	102	85.3	0.06
router (60/30)	231	551	0.2	-138.5	231	0.0	2.77
t2 (17/16)	149	81	7.9	45.6	44	70.5	0.01
test2 (11/35)	4817	77	9.9	98.4	2694	44.1	0.14
test3 (10/35)	2619	79	9.8	97.0	1445	44.8	0.07
test4 (8/30)	941	89	8.5	90.5	476	49.4	0.02
tial (14/8)	1307	153	37.7	88.3	1029	21.3	0.01
ts10 (22/16)	4391	1	6.3	100.0	1331	69.7	0.09
vg2 (25/8)	219	213	21.2	2.7	175	20.1	0.01
vtx1 (27/6)	241	228	14.9	5.4	182	24.5	0.01
x9dn (27/7)	271	229	12.8	15.5	186	31.4	0.01
Z9sym (9/1)	25	25	0.0	0.0	25	0.0	0.01

TABLE 5
Average number of reduction of variables.

Approximation threshold	5%	10%	30%
Reduction (% number of variables)	18.45%	19.02%	22.58%

8 Conclusions

In this paper we describe a new approach to approximate synthesis, based on changing the outputs of a given function to obtain either the closest symmetric function, or a D-reducible function, where the optimality criterion is given by the chosen error metrics. To achieve symmetrizazion, we change the original function within the tolerance allowed by the given error bounds to yield a totally symmetric function with the fewest changes, and then we relax total symmetrization when the error bounds are not met, until we reach a feasible solution. The core technical contribution is a polynomial algorithm for computing the closest symmetric approximation of an incompletely specified multi-output Boolean function with an unbounded number of errors. Then we present the first polynomial heuristic algorithm to compute a D-reducible approximation of an incompletely specified target function, under a bit error metric.

Experimental results confirm the efficacy of the proposed approaches. For symmetrization, the average gain in the BDD size for the unbounded approximation scheme is 69% with an average error rate of about 15%, while in the bounded context, with a fixed error rate of 5%, the average gain is about 31%. For D-reducibility, we obtain a reduction of almost 19% on the number of variables for an approximation threshold of 5% of the on-set of the function only.

Future work includes a comparison of the proposed techniques with other approximation methods. Finally, to improve the applicability of our approach, which currently requires, for both symmetrization and D-reducibility, a DSOP or a BDD representation of the input function that can be hard to obtain for large benchmarks, we plan to design a new approximation heuristic starting from a more scalable representation of the input function, such as AIGs.

References

[1] "The EPFL Combinational Benchmark Suite." [Online]. Available: http://lsi.epfl.ch/benchmarks

[2] D. Bañeres, J. Cortadella, and M. Kishinevsky, "A Recursive Paradigm to Solve Boolean Relations," *IEEE Transactions on Computers*, vol. 58, no. 4, pp. 512–527, 2009.

[3] B. Becker, "Synthesis for testability: Binary decision diagrams," in *STACS 92, 9th Annual Symposium on Theoretical Aspects of Computer Science, Cachan, France, February 13-15, 1992, Proceedings*, 1992, pp. 501–512.

[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 and V. Ciriani, "DRedSOP: Synthesis of a New Class of Regular Functions." in *Euromicro Conference on Digital Systems Design: Architectures, Methods and Tools (DSD)*, 2006, pp. 377–384.

[6] ——, "Logic synthesis and testability of d-reducible functions," in *IFIP/IEEE VLSI-SoC 2010 - International Conference on Very Large Scale Integration of System-on-Chip,*, 2010, pp. 280–285.

[7] ——, "Dimension-reducible Boolean functions based on affine spaces," *ACM Trans. Design Autom. Electr. Syst.*, vol. 16, no. 2, p. 13, 2011.

[8] ——, "2-SPP approximate synthesis for error tolerant applications," in *17th Euromicro Conference on Digital System Design, DSD 2014, Verona, Italy, August 27-29, 2014*, 2014, pp. 411–418.

[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] ——, "Compact DSOP and partial DSOP forms," *Theory Comput. Syst.*, vol. 53, no. 4, pp. 583–608, 2013.

[12] A. Bernasconi, V. Ciriani, and T. Villa, "Approximate logic synthesis by symmetrization," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019*, 2019, pp. 1655–1660.

[13] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.

[14] J. T. Butler and T. Sasao, "Maximally asymmetric multiple-valued functions," in *2019 IEEE 49th International Symposium on Multiple-Valued Logic (ISMVL), Fredericton, NB, Canada, May 21-23, 2019*, 2019, pp. 188–193.

[15] A. Chandrasekharan, M. Soeken, D. Große, and R. Drechsler, "Approximation-aware rewriting of AIGs for error tolerant applications," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2016, pp. 1–8.

[16] R. Drechsler and W. Günther, *Towards One-Pass Synthesis*. Kluwer Academic Publishers, 2002.

[17] G. Fey and R. Drechsler, "Utilizing BDDs for disjoint SOP minimization," in *The 2002 45th Midwest Symposium on Circuits and Systems, 2002. MWSCAS-2002.*, vol. 2, 2002.

[18] C. C. Foster and F. D. Stockton, "Counting responders in an associative memory," *IEEE Transactions on Computers*, vol. C-20, no. 12, pp. 1580–1583, Dec 1971.

[19] S. Hashemi, H. Tann, and S. Reda, "Approximate logic synthesis using boolean matrix factorization," in *Approximate Circuits, Methodologies and CAD*, S. Reda and M. Shafique, Eds. Springer, 2019, pp. 141–154. [Online]. Available: https://doi.org/10.1007/978-3-319-99322-5_7

[20] M. A. Heap, "On the exact ordered binary decision diagram size of totally symmetric functions," *J. Electronic Testing*, vol. 4, no. 2, pp. 191–195, 1993.

[21] L. Heinrich-Litan and P. Molitor, "Least upper bounds for the size of OBDDs using symmetry properties," *IEEE Trans. Computers*, vol. 49, no. 4, pp. 360–368, 2000. [Online]. Available: https://doi.org/10.1109/12.844348

[22] T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli, *Synthesis of Finite State Machines: Functional Optimization*. Kluwer Academic Publishers, now Springer, 1996.

[23] B. Kim and D. L. Dietmeyer, "Multilevel logic synthesis of symmetric switching functions," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 10, no. 4, pp. 436–446, 1991.

[24] Z. Kohavi, *Switching and Finite Automata Theory*. McGraw Hill, 1970.

[25] V. N. Kravet and K. A. Sakallah, "Constructive library-aware synthesis using symmetries," in *Proceedings Design, Automation and Test in Europe Conference and Exhibition 2000 (Cat. No. PR00537)*, March 2000, pp. 208–213.

[26] Y. Lai, C. Lin, C. Wu, Y. Chen, and C. Wang, "Efficient synthesis of approximate threshold logic circuits with an error rate guarantee," in *2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018*, 2018, pp. 773–778.

[27] G. Liu and Z. Zhang, "Statistically certified approximate logic synthesis," in *2017 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2017, Irvine, CA, USA, November 13-16, 2017*, 2017, pp. 344–351. [Online]. Available: https://doi.org/10.1109/ICCAD.2017.8203798

[28] J. Miao, A. Gerstlauer, and M. Orshansky, "Approximate logic synthesis under general error magnitude and frequency constraints," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2013, pp. 779–786.

[29] ——, "Multi-level approximate logic synthesis under general error constraints," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2014, pp. 504–510.

[30] A. Mishchenko, "Fast computation of symmetries in Boolean functions," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 22, no. 11, pp. 1588–1593, 2003.

[31] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62:1–62:33, Mar. 2016. [Online]. Available: http://doi.acm.org/10.1145/2893356

[32] G. Pasandi, S. Nazarian, and M. Pedram, "Approximate logic synthesis: A reinforcement learning-based technology mapping approach," in *20th International Symposium on Quality Electronic Design, ISQED 2019, Santa Clara, CA, USA, March 6-7, 2019*. IEEE, 2019, pp. 26–32.

[33] C. Scholl, D. Möller, P. Molitor, and R. Drechsler, "BDD minimization using symmetries," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 18, no. 2, pp. 81–100, 1999.

[34] D. Shin and S. K. Gupta, "Approximate logic synthesis for error tolerant applications," in *DATE*, 2010, pp. 957–960.

[35] D. Shin and S. Gupta, "A new circuit simplification method for error tolerant applications," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2011, pp. 1–6.

[36] P. Stnica, T. Sasao, and J. Butler, "Distance duality on some classes of Boolean functions," *Journal of Combinatorial Mathematics and Combinatorial Computing*, vol. 107, pp. 181–198, 2018.

[37] S. Su, C. Zou, W. Kong, J. Han, and W. Qian, "A novel heuristic search method for two-level approximate logic synthesis," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 39, no. 3, pp. 654–669, 2020. [Online]. Available: https://doi.org/10.1109/TCAD.2018.2890532

[38] E. E. Swartzlander, "Parallel counters," *IEEE Trans. Comput.*, vol. 22, no. 11, pp. 1021–1024, Nov. 1973. [Online]. Available: https://doi.org/10.1109/T-C.1973.223639

[39] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "SALSA: Systematic logic synthesis of approximate circuits," in *49th ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2012, pp. 796–801.

[40] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: a unified design paradigm for approximate and quality configurable circuits," in *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013*, 2013, pp. 1367–1372.

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

[42] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design Test*, vol. 33, no. 1, pp. 8–22, Feb 2016.

[43] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0," Microelectronic Center, User Guide, 1991.

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.

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 90 research papers, published in international journals, conference proceedings, and books chapters.

Tiziano Villa received a Laurea degree in Mathematics from the University of Milano, took the Part III of Mathematical Tripos at the University of Cambridge, completed a M.S. in CS at U.C. Berkeley, and was awarded a Ph.D. in EECS in 1995 by U.C. Berkeley. In 1997 he joined as a Research Scientist the PARADES Labs, Rome, Italy. In 2002 he became an Associate Professor at Università di Udine, Italy. Since 2006 he is a Professor with the Department of Computer Science (DI), Università di Verona, Italy. His research interests are in formal methods for electronic design automation, including logic synthesis, formal verification, automata theory and models of computation, discrete-event dynamic systems, supervisory control, cyber-physical and embedded systems. He co-authored the books "Synthesis of FSMs: Functional Optimization" (Kluwer/Springer, 1997, reprint 2010), "Synthesis of FSMs: Logic Optimization" (Kluwer/Springer, 1997, reprint 2012), "The Unknown Component Problem: Theory and Applications" (Springer, 2012), and co-edited the book "Coordination Control of Distributed Systems", Springer, 2015.