INTEGRATION, the VLSI journal 89 (2023) 248–260

Contents lists available at ScienceDirect

Integration, the VLSI Journal

journal homepage: www.elsevier.com/locate/vlsi

Accuracy recovery: A decomposition procedure for the synthesis of partially-specified Boolean functions ☆

Andrea Costamagna [*,1], Giovanni De Micheli [1]

EPFL, Lausanne, 1015, Switzerland

ARTICLE INFO

Keywords:
Accuracy recovery
Partial specifications
Binarized Neural Networks
Disjoint support decomposition

ABSTRACT

Logic Synthesis From Partial Specifications (LSFPS) is the problem of finding the hardware implementation of a Boolean function from a partial knowledge of its *care set*. The elements missing from the specifications are named *don't knows*. The exact solution of LSFPS is the minimum size circuit of the corresponding problem in which the *don't knows set* is void. Hence, in addition to the traditional objective of size minimization, the goal is to maximize the *test accuracy*, i.e., the accuracy of the circuit when evaluated over a subset of the *don't knows*. This problem is relevant because efficient solutions can lead to hardware friendly machine learning models, not relying on black-box approaches. Indeed, LSFPS maps directly to the problem of the automatic generation of optimized topologies for Binarized Neural Networks. Furthermore, combining the exact solution with modern logic synthesis techniques would unlock unprecedented optimization capabilities. Previous works proved the effectiveness of *approximate logic synthesis* (ALS) for designing circuits with high *test accuracy*. Nonetheless, these methods sacrifice accuracy on the specifications, which banishes them from the legitimate candidates for LSFPS. In this paper, we propose *accuracy recovery*, a procedure to map an approximate version of the circuit to a new one that satisfies the exact functionality of the specifications. The proposed approach relies on an extension of a disjoint support decomposition algorithm. Relative experiments on the IWLS2020 benchmarks show that, on average, the addition of the designed decomposition to a synthesis flow reduces by **17.38%** the number of gates and by **12.02%** the depth. The usage of *accuracy recovery*, based on such a decomposition, yields a **95.73%** accuracy in the binary MNIST problem, beating the state-of-the-art in ALS of **92.76%**.

1. Introduction

Integrating machine learning in IoT devices is a crucial driver of many technological advancements, including smart agriculture [1], healthcare [2], and transportation [3]. Currently, computation occurs mainly in the cloud, and bringing it to the edge would dramatically reduce the energy consumption [4]. However, embedding logic in edge devices is challenging due to resource constraints. Interestingly, some machine learning tasks admit a formulation as a fundamental problem in *logic synthesis*. Considering the extensive research done in logic synthesis for high-performance systems [5], it is worth investigating its potential role in devising hardware-aware machine learning techniques.

An *incompletely specified* Boolean function reads

$$F : \mathbb{B}^n \mapsto \{0, 1, *\}^m \tag{1}$$

where n is the number of inputs and m is the number of outputs. The *care set* of an output is the subset of $\mathbb{B}^n$ whose image is 1 or 0. The complement of the *care set* contains the *don't cares* ($*$), i.e., minterms whose output can be chosen based on optimization purposes. *Logic Synthesis From Partial Specifications* (LSFPS) introduces the additional concept of *don't knows* (?), i.e., minterms missing from the specifications. Then, a *partially-specified* Boolean function reads

$$F : \mathbb{B}^n \mapsto \{0, 1, ?, *\}^m \tag{2}$$

Fig. 1 offers a pictorial representation of these concepts. LSFPS aims at designing algorithms for inferring the *don't knows* relationships from the specifications. Ideally, the circuit synthesized from Eqs. (1) and (2) should be the same. In practice, the information on the specifications might not be sufficient for inferring the exact functionality, and the goal becomes to maximize the accuracy over an available subset of

☆ This work was supported in part by Synopsys Inc.

In this work, we discuss accuracy recovery: a logic synthesis-inspired methodology for the design of hardware-aware machine learning models.

* Corresponding author.
E-mail address: andrea.costamagna@epfl.ch (A. Costamagna).

[1] All authors have contributed equally.

https://doi.org/10.1016/j.vlsi.2022.12.008
Received 2 October 2022; Received in revised form 3 December 2022; Accepted 22 December 2022
Available online 26 December 2022

TRUTH TABLE OF THE FUNCTION

X_2	X_1	X_0	F
0	0	0	0
0	0	1	1
don't know			?
0	1	1	0
1	0	0	1
1	0	1	0
don't care			*
1	1	1	0

COVER OF THE FUNCTION

X_2	X_1	X_0	F
0	0	0	0
0	0	1	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	1	0

COVERS OF THE COFACTORS

X_1	X_0	F_0
0	0	0
0	1	1
1	1	0

X_1	X_0	F_1
0	0	1
0	1	0
1	1	0

Fig. 1. Representations of a partially specified Boolean function: truth table, cover, and covers of the cofactors.

don't knows. Under a machine learning lens, the partial specifications correspond to the *training set*, and the subset of the *don't knows* on which we test the accuracy of the devised computing system corresponds to the *test set*.

Partially-specified functions occur in supervised learning. The state-of-the-art computing system is the Deep Neural Network (DNN), for which it is desirable to realize dedicated hardware [6]. Efficient tools can map the graph structure of DNNs to Boolean circuits, later optimized with digital design procedures [7]. However, the quality of results has a bias originating from the initial topology, heuristically imposed by the DNN architect. Therefore, there is a rising interest in understanding which topology is best suited for hardware implementation [8]. Contrarily to black-box approaches, the goal is the design of DNN topologies from data [9]. Binarized Neural Networks (BNN) are DNNs with binary parameters. In BNNs, XNORs replace multiplications, yielding a superior energy efficiency of their implementation. Recently, Constantinides proved that BNNs are functionally complete [8]. Therefore, they are good candidates for high-performance hardware implementations. Due to the binary nature of the weights and the presence of XOR operators, Xor-And-Inverter graphs (XAGs) [10] are in one-to-one correspondence with BNNs. We investigate methods to synthesize BNN-topologies from data as XAGs. Furthermore, for comparisons with existing works, we sometimes consider the representation as another functionally complete representation: the And-Inverter-Graph (AIG) [11].

LSFPS is also interesting per se. Indeed, it is the core problem in *simulation-guided resubstitution*, a recently proposed optimization strategy [12]. Currently, this algorithm is limited to optimizing sub-networks with a limited input size. Hence, the exact solution of LSFPS for higher numbers of inputs would unlock unprecedented optimization opportunities. This application provides the first reason why to look for models that are 100% accurate in the specifications. Indeed, 100% accuracy on the *training set* is necessary for using the synthesis algorithm in *simulation-guided resubstitution*.

Our work takes inspiration from the existing literature in *Approximate Logic Synthesis* (ALS). Indeed, traditionally, the supervised learning problem is accounted among the error-resilient applications, and accuracy in the specifications is often sacrificed in favor of generalization. Oliveira et al. [13] proposed two algorithms based on information theory, namely `MUESLI` and `FULFRINGE`. While the former assembles Boolean networks bottom-up, the latter is a decision tree decomposition. Subsequently, Chatterjee designed a method for learning LUT nodes in a randomly generated LUT network [14], and Boroumand et al. [15] exploited it in a variation of the `MUESLI` algorithm. Finally, the IWLS2020 benchmarks and the solutions proposed by the competing teams [16] constitute the main existing framework for evaluating algorithms for LSFPS. Overall, the main findings of the aforementioned works are the following:

1. Decomposition procedures are powerful strategies.
2. Circuits can be assembled one gate at a time.
3. No technique in literature is effective for all functions.
4. Many approaches use pruning to meet size constraints.
5. Many approaches have a black-box nature.

Therefore, the definition of an automatic synthesis method in ALS remains an open problem.

The first contribution of this work is the generalization of a *disjoint support decomposition* (DSD) algorithm to the LSFPS problem. Contrarily to the existing literature, this method targets size minimization without pruning. Furthermore, it does not rely on non-interpretable hyperparameters, i.e., it has not a black-box nature. The proposed procedure combines properties of mutual information and an XAG decomposition. We show that adding the proposed DSD procedure to a synthesis technique improves its performance. On average, it reduces the size by 17.31%, the depth by 11.81%, and increases the *test accuracy* by 1.11%.

The second contribution of this work is the definition of a technique that we name *accuracy recovery*. We show that combining statistics-based decompositions with existing ALS algorithms constitutes a systematic way of assembling Boolean networks. The resulting networks are 100% accurate in the *training set*. From a supervised learning perspective, the 100% accuracy in the specifications is valuable in light of Chatterjee's analysis of the role of memorization in learning [14]. Furthermore, this feature makes the algorithm applicable to traditional logic synthesis. We show that by leveraging the concept of *accuracy recovery*, it is possible to achieve 95.73% test accuracy in the binary MNIST problem, beating the ALS state-of-the-art of 92.76%.

The rest of this paper is structured as follows. Section 2 presents the relevant background. Section 3 describes our DSD algorithm for partially-specified functions and the concept of *accuracy recovery*. Section 4 shows experimental results. Section 5 concludes the paper.

2. Background

Given $F : \mathbb{B}^n \mapsto \{0, 1, *\}$, its *support* $S_F = \{X_i\}_{i=0}^{n-1}$ is the set of variables on which F depends. In this paper, we represent Boolean functions as covers, i.e., lists of inputs/output patterns in the *care set*. We study algorithms to synthesize multilevel networks from Boolean covers.

2.1. Top-down synthesis techniques

This subsection reviews decomposition strategies for the synthesis of Boolean functions

2.1.1. Shannon decomposition

Given a function F and a variable $X_i \in S_F$, the *Shannon Decomposition* (SD) of F is

$$F = X_i \cdot F_{x_i} + X_i' \cdot F_{x_i'} \doteq \texttt{ite}(X_i, F_{x_i}, F_{x_i'}) \tag{3}$$

Where F_{x_i} ($F_{x_i'}$) is the positive (negative) cofactor of F, i.e., the function of domain $S_F \backslash X_i$ whose cover is the sub-cover of F, identified by $X_i = \top$ ($X_i = \bot$). When the variable is explicitly stated, we use the notation $F_1 \doteq F_{x_i}$ and $F_0 \doteq F_{x_i'}$.

2.1.2. Disjoint support decomposition

Performing the DSD of a function F corresponds to identifying a set of functions $\mathcal{A} = \{A_i\}_{i=1}^{m}$ ($m < n$) with disjoint supports and a function G such that

$$F(S) = G(\mathcal{A}) \tag{4}$$

When a truth table is available, such a decomposition can be obtained by applying two procedures until convergence: *top-decomposition* and *bottom-decomposition*[17,18]. Fig. 2 shows the representation of the local topological choices resulting from the considered decompositions.

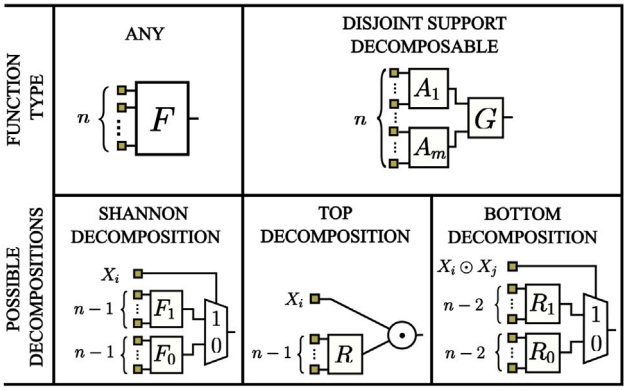


Fig. 2. Possible decompositions of a Boolean function.

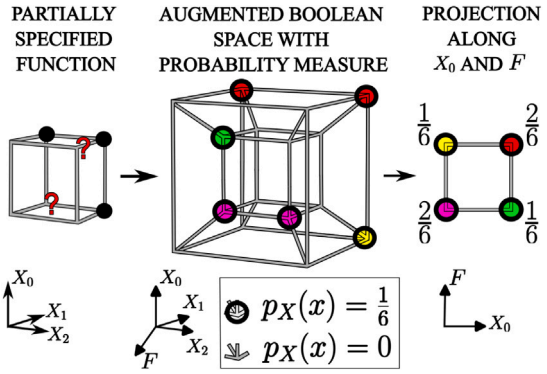


Fig. 3. From left to right: cube representation of a three variables function; augmented Boolean space considering F as a variable and labeling each point with the empirical probability of appearance; projection along $X = (X_0, F)$.

Top-decomposition:. The top-decomposition identifies when the dependence of F on a variable X_i is of type

$$G = X_i \odot T(S \backslash X_i) \tag{5}$$

with $\odot$ indicating a 2-input function. Hence, all possible top-decompositions can be obtained by listing the special cases of the SD yielding the functional form of Eq. (5):

1. $F_1 = \top$: $F = X_i + X_i' \cdot F_0 \Rightarrow G = X_i \vee F_0$;
2. $F_1 = \bot$: $F = X_i' \cdot F_0 \doteq X_i < F_0 \Rightarrow G = X_i < F_0$;
3. $F_0 = \top$: $F = X_i' + F_1 \doteq X_i \leq F_1 \Rightarrow G = X_i \leq F_1$;
4. $F_0 = \bot$: $F = X_i \cdot F_1 \Rightarrow G = X_i \wedge F_1$;
5. $F_1 = F_0'$: $F = X_i' \cdot F_0 + X_i \cdot F_0' \Rightarrow G = X_i \oplus F_0$.

In these cases, we say that the function is *top-decomposable*.

Bottom-decomposition:. Suppose that two variables X_i and X_j influence F uniquely through a 2-input function:

$$G = B(X_i \odot X_j, S \backslash \{X_i, X_j\}) \tag{6}$$

Let us introduce the simplified notation for the cofactors: $F_{00} \doteq F_{x_i' x_j'}$, $F_{01} \doteq F_{x_i' x_j}$, $F_{10} \doteq F_{x_i x_j'}$ and $F_{11} \doteq F_{x_i x_j}$. For Eq. (6) to be true, one of the following conditions must hold

1. $F_{00} \neq F_{01} = F_{10} = F_{11}$: $G = \texttt{ite}(X_i + X_j, F_{11}, F_{00})$;
2. $F_{01} \neq F_{00} = F_{10} = F_{11}$: $G = \texttt{ite}(X_i < X_j, F_{01}, F_{10})$;
3. $F_{10} \neq F_{00} = F_{01} = F_{11}$: $G = \texttt{ite}(X_i \leq X_j, F_{01}, F_{10})$;
4. $F_{11} \neq F_{00} = F_{10} = F_{01}$: $G = \texttt{ite}(X_i \cdot X_j, F_{11}, F_{00})$;
5. $F_{00} = F_{11}$, $F_{10} = F_{01}$: $G = \texttt{ite}(X_i \oplus X_j, F_{01}, F_{00})$.

The comparison of all variable pairs guarantees the detection of *bottom-decomposability* and its exploitation.

2.2. Statistical quantities in Boolean functions

Let $X \in \mathbb{B}^n$ be a Boolean random variable having as probability mass function $p_X(\cdot)$. Its Shannon entropy is

$$H(X) \doteq -\sum_{x \in \mathbb{B}^\times} p_X(x) \log_2 p_X(x) \tag{7}$$

It quantifies the uncertainty on the value taken by the variable. Given another Boolean random variable $F \in \mathbb{B}^m$, *mutual information* originates from entropy as

$$I(X;F) = H(F) - H(F|X) \tag{8}$$

It is a measure of the reduction in uncertainty on the target variable F, given the knowledge of X. In this work, we use F to indicate the Boolean variable associated with the output. Instead, we use the variable $X \in \mathbb{B}^n$ to describe any n nodes in the network, possibly including inputs and output.

Fig. 3 shows how to extract the probability mass function for the variable $X = (X_0, F) \in \mathbb{B}^2$. In words, we use the empirical frequency as an estimator of the probability.

2.3. Decomposition from partial specifications

Our work takes also inspiration from the methodologies presented in the IWLS2020 contest [16]. `TEAM 8` proposed a variation of the `C4.5` decision tree learning algorithm [19], adapted for ALS. They assemble a decision tree by splitting over the variable maximizing mutual information. If mutual information is lower than a threshold for all the variables, they perform *single variable splitting*. This is to say that they branch for the features such that, after splitting, either of the following conditions is satisfied:

1. At least one branch is constant.
2. One branch is the complement of the other.

These conditions are strictly related to the top-decomposability condition discussed in Section 3.3. As far as the second one is concerned, a rigorous procedure would require to compare all possible minterms in the specifications of the remainder functions. However, the function is partially-specified, and an exhaustive test is often impossible. The policy of `TEAM 8` was to consider the condition as satisfied, provided the absence of any counterexample.

`TEAM 8` also proposed a connection between decision tree decompositions and Espresso Minimizer [20]. They noticed that Espresso exploits *don't cares* by expanding the minterms in the SOP. The possibility to expand a minterms depends on whether the function presents the so called *nearest neighbor* property. Decision trees also leverage this property by making cuts in the input space. For this reason, they claim that neither Espresso nor decision trees are able to learn a logic function from a partial truth table if the nearest neighbor property does not hold. Our work investigates ways to go beyond the nearest neighbors limitation while using a decomposition procedure for the synthesis.

2.4. Bottom-up synthesis techniques

This section presents ALS algorithms for the synthesis of Boolean networks. Each algorithm generates the netlist by adding one gate at a time. Furthermore, all the algorithms require two steps: a support selection criterion and a gate generation method. The node addition continues until termination. The final output of the approximate circuit is the node whose simulation pattern is closer to the one of the target.

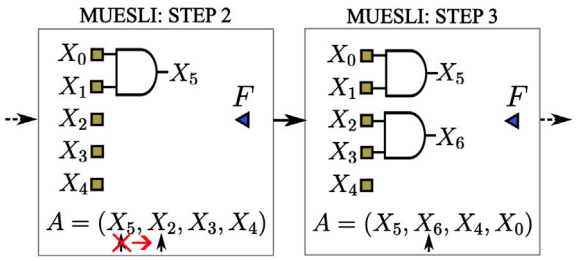


Fig. 4. MUESLI for $F = X_0X_1 + X_2X_3X_4$. Since we cannot find any two-input function having X_5 in the support, we increase the pointer. In this case, X_2X_3 enters the netlist.

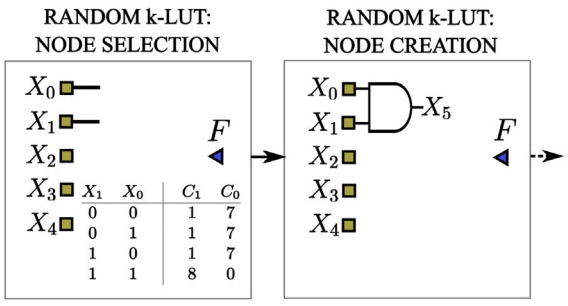


Fig. 5. Random k-LUT generation for $F = X_0X_1 + X_2X_3X_4$. We derive the counters for the randomly sampled support. Next, we insert the gate better approximating the target.

2.4.1. Muesli

MUESLI is a bottom-up ALS technique relying on mutual information [13]. The selection procedure requires filling an active list $A = (A_1, A_2, \ldots)$ recursively

$$A_k = \arg\max_{X_i \notin A} I(A_1, \ldots, A_{k-1}, X_i; F) \qquad A_0 = \emptyset \tag{9}$$

Next, a pointer starts from the most informative variable X^*. Given the desired support size $|S|$, the algorithm looks for the remaining $|S| - 1$ variables with which it is possible to synthesize the node that best approximates the target function. In the original algorithm, the node creation method employs the Kernighan–Lin algorithm [21]. If replacing X^* in the active list with the candidate node satisfies a heuristic criterion, the node is accepted. Otherwise, we increment the pointer. Fig. 4 presents two steps of an example. For more details, we refer to the seminal paper [13].

The main limitation of this method is scalability. Indeed, filling the active list and exploring potential new nodes requires multiple passes over the specifications. Furthermore, the rigorous application of MUESLI requires the new-nodes support size to be $|S| = 2$. Finally, the non-scalability limits the possible size of the nodes list, which, in turn, compromises the accuracy of the synthesized circuit.

2.4.2. Random k-LUT network

Chatterjee investigated the possibility of achieving high *test accuracy* by memorizing the specifications in a random network of k-inputs look-up tables (k-LUTs) [14]. We refer to the method as kLUTNET. The method is agnostic of size minimization, and he observed that such a model shares with DNNs the accuracy performance benefits resulting from increasing the depth. This technique is scalable since the support selection criterion is a random sampling of the nodes, removing the need for expensive heuristics. As far as node creation is concerned, the method considers all possible 2^k minterms for the support variables $S = (X_{s(1)}, \ldots X_{s(k)})$, corresponding to the input patterns that can appear at the inputs of the k-LUT. Each minterm can occur multiple times in the specifications, and each maps to an output value. The idea is to define two integers, C_0^m and C_1^m, counting the number of times the output is 0 or 1 for minterm m. Next, the method sets the output value of the k-LUT to the most frequently appearing value, and ties are broken by randomly selecting the bit-value. Fig. 5 shows the steps leading to a node creation.

The limits of this approach are that despite the goal being to memorize the specifications, achieving 100% accuracy on the specifications requires unpredictably deep topologies. Furthermore, when the size of the k-LUT networks is small, say 2, adding informative new nodes can become less likely. For instance, in the case of complete specifications, no random 2-LUT network can find the 2-inputs AND gates for a functions such as $F = X_0X_1X_2X_3$. In Section 3.1 we discuss how to overcome this limitation.

2.4.3. Modified Muesli

The method devised by Boroumand et al. [15] extends MUESLI to higher numbers of support variables. The variables considered for the support are a contiguous set of variables in the active list, starting from the one indicated by a pointer. Hence, this method corresponds to a k-LUT network assembly in which a MUESLI-inspired heuristic replaces the random selection of the support variables.

3. Synthesis from partial specifications

This section contains the contributions of the paper. Section 3.1 discusses the technique we devised for nodes generation in the investigated ALS algorithms. Section 3.2 presents the analytical considerations motivating both the methods and the experiments (Section 4). Section 3.3 discusses the decomposition technique at the basis of all the proposed synthesis methods. Finally, Section 3.4 introduces the concept of *accuracy recovery* and its usage to map approximate logic into circuits that meet the specifications.

3.1. Node creation criterion for 2-LUTs

This work targets the synthesis of XAGs due to their role in BNNs. More precisely, we consider 2-LUT networks, but these models are in one-to-one correspondence with XAGs. For some functions, Chatterjee's node creation might be incapable of finding good nodes for 2-LUT networks. An example is $F = X_0X_1X_2X_3$. On the other hand, the limited number of 2-inputs functions makes it doable to explore more nodes. We equipped the previously discussed ALS algorithms with a modified version of the node creation method. Fig. 6 shows an example of the approach.

Given two selected support variables, we consider all their possible minterms (00, 01, 10, and 11). In Fig. 6, the variables are X_3 and X_2. For each minterm m, the counters C_1^m and C_0^m report the number of times the output is 1 or 0, as in Chatterjee's case. If the cofactor of the function for a minterm is a tautology/contradiction, we fix the corresponding LUT bit to 1/0. In the example, this happens for the minterm 11. Next, we enumerate all possible functions obtainable by varying the output bit for the remaining minterms. In the example, this results in 8 possible functions. To each function, we associate a number, measuring its degree of correlation with the output. This number is the sum of the number of appearances of each minterm-output pair.

In the example, Chatterjee's method would select the contradiction function with 50% probability. Instead, we ignore *trivial* functions such as $\tilde{X}(X_i, X_j) = X_i, X_i', \top, \bot$. Furthermore, we sort the nontrivial functions based on their degree of correlation. Finally, we store the list of candidate nodes, keeping a pointer to the most correlated one not yet used. If we introduce the node in the netlist, we increment the pointer. Compared to the previously discussed methods, this approach allows us to introduce multiple nodes for the same support. Furthermore, this

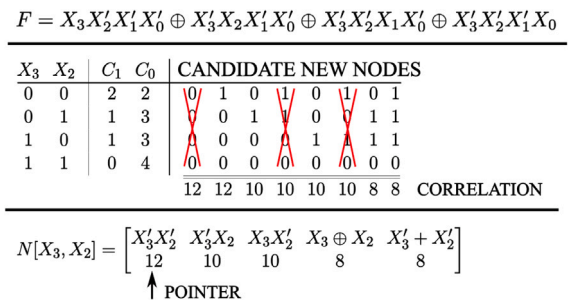


Fig. 6. Nodes creation method. We store the non-trivial candidate nodes in a list sorted by correlation with the output.

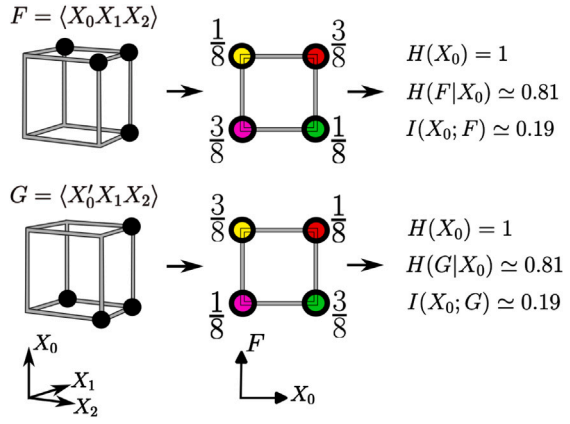


Fig. 7. Invariance of mutual information by input negation.

heuristic extends the idea of selecting nodes based on similarity with the output [14].

In a random k-LUT network, adding redundant nodes introduces buffers and inverters at the next layer. These nodes can be helpful when the nodes selection criterion proceeds in a layer-by-layer fashion, such as in the cases investigated in the seminal work. As we will see in the following sections, it makes sense to add non-redundant nodes to use node creation uniquely to introduce new information.

3.2. Analytical considerations

Mutual information is suitable for quantifying the mutual dependence of the support variables and the output during synthesis. To motivate its usage in the algorithms, and the proposed experiments, let us start with the following claim:

Claim 1. *Consider two Boolean functions $F, G : \mathbb{B}^n \mapsto \mathbb{B}$, and two subsets of their support $X \subseteq S_F$, $\tilde{X} \subseteq S_G$. If there is an inputs/output negation $\gamma : (S_F, F) \mapsto (S_G, G)$, and $\tilde{X} = \gamma(X)$, then $I(X; F) = I(\tilde{X}; G)$.*

Fig. 7 shows an example of this property in the special case in which the subset of the support is a unique variable. We prove this claim in Appendix A.

Two Boolean functions $F, G : \mathbb{B}^n \mapsto \mathbb{B}$ are *NPN equivalent* if they can be obtained from each others by means of an inputs/output negation and/or an inputs permutation. Each *NPN class* is identified by one such function, named *representative*. Claim 1 directly implies the following:

Claim 2. *Consider two NPN-equivalent Boolean functions $F, G : \mathbb{B}^n \mapsto \mathbb{B}$. For each variable X_i in the support of F, there is a variable X_j in the support of G such that $I(X_i; F) = I(X_j; G)$.*

These claims are the basis of the devised methods for determining if a function is disjoint-support-decomposable.

3.2.1. Top-decomposition

Suppose that a topological structure depends uniquely on the analysis of a variable X_i that maximizes $I(X_i; F)$. Then, for all the functions G that are NPN-equivalent to F, the same structure emerges from the analysis of the corresponding variable X_j that maximizes $I(X_j; G)$. We use this result both theoretically and experimentally.

Claim 3. *Consider a function F that is top-decomposable in X_i: $F = X_i \odot T(S_F \backslash X_i)$. For any variable $X_j \in S_F \backslash X_i$*

$$I(X_i; F) \geq I(X_j; F)$$

The proof of this claim is in Appendix B. Intuitively, if Eq. (5) holds, the top-decomposable variable alone contains as much information as a more complicated function of the others. Hence, its mutual information cannot be lower than the highest among the remaining variables.

3.2.2. Bottom-decomposition

In Appendix C, we prove the following claims:

Claim 4. *Consider a function F that is bottom decomposable in $\tilde{X} = X_i \odot X_j$. Then, $I(X_i; F) = I(X_j; F)$.*

Claim 5. *Consider a function F that is bottom decomposable in $\tilde{X} = X_i \odot X_j$. Then, $I(X_i, X_j; F) = I(\tilde{X}; F)$.*

Claim 4 states that the variables are equally informative. Claim 5 indicates that the new node contains the same information as the variables X_i and X_j combined.

3.3. Information theory-based disjoint support decomposition

In this section, we generalize a DSD procedure to LSFPS. The driving idea is to leverage the results reported in Section 3.2 to detect the presence of a DSD condition. When the identification is correct, the resulting circuit is closer to the ideal one, increasing the test accuracy. Furthermore, DSD targets reducing the size of the circuit, making it a natural choice as the synthesis algorithm.

3.3.1. Structure of the algorithm

Algorithm 1 is the proposed synthesis technique for partially-specified functions that integrates the *don't knows*-based DSD decomposition. Three exit conditions are possible, corresponding to the cases in which all output values are ones ($\top$), all output values are zeros ($\bot$), or the support size is smaller than `max_sup` and Chatterjee's method can be applied to find a termination node.

Algorithm 1 signal ← Decomposition(S, F)

```
1: if (F = ⊤) then
2:     return 1
3: else if (F = ⊥) then
4:     return 0
5: else if (|S| ≤ max_sup) then
6:     return Chatterjee method (S, F)
7: x ← choose variable X from S
8: if (is top-decomposable (X, S, F, ⊙)) then
9:     return x⊙Decomposition (S, F)
10: else if (is bottom-decomposable (S, F)) then
11:     return Decomposition (S, F)
12: f0 ← Decomposition (S\X, F0)
13: f1 ← Decomposition (S\X, F1)
14: return x · f1 + x' · f0
```

First, the algorithm tries to perform a top-decomposition step on the selected variable. In Section 3.3.2 we motivate why, in line with previous studies [13,16], the maximization of mutual information is an effective selection criterion. In case of success, `is top-decomposable` has also updated the cover, on which we can recursively apply the

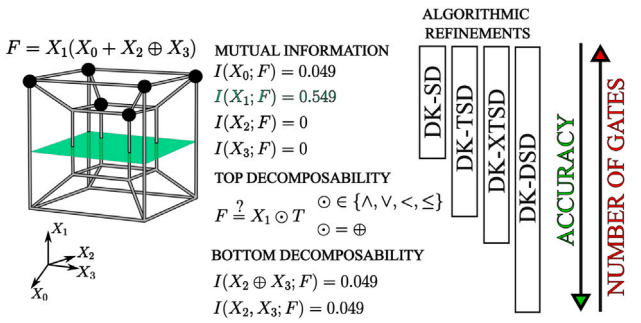


Fig. 8. Refinements of the recursion step. DK−SD performs mutual information maximization. DK−TSD also checks if there is a basic top-decomposability $\{\wedge, \vee, <, \leq\}$. The function on the left is top-and-decomposable and the algorithm would recur. DK−XTSD verifies if there is top-xor-decomposability. DK−DSD adds new nodes based on their information content.

decomposition method. In case of failure, the algorithm attempts a bottom-decomposition step. If `is bottom-decomposable` returns `true`, it has also updated the cover and the algorithm calls the recursive procedure. Finally, if none of the previous conditions occurs, the algorithm uses the selected variable to perform a Shannon decomposition step.

If we ignore lines 8 to 11 in Algorithm 1, and the selection criterion (line 7) is to take the first variable in the support, the procedure corresponds to a Shannon decomposition (SD). Instead, a powerful variable selection criterion is to take the variable maximizing mutual information. In this latter case, we say that the decomposition is *don't knows*-aware (DK−SD). With the further addition of lines 8 to 9, the method becomes a top-decomposition. We distinguish two cases: the one in which the XOR is not considered (DK−TSD), and the one in which it is considered (DK−XTSD). Finally, the further addition of lines 10 to 11 yield the *don't knows*-aware DSD (DK−DSD). Fig. 8 summarizes the key elements of each recursive step.

3.3.2. Top-decomposition

In Algorithm 1, we first check if the function is top-decomposable in variable X. If this is the case, both the support and the cover are updated, and $\odot$ contains the detected 2-input function. Thanks to Claim 3, we test the top-decomposition condition only on the variable maximizing mutual information. This choice is advantageous since, given a cover for n variables, it saves $n-1$ cofactors computations and comparisons at each new call of Algorithm 1. Cases 1 to 4 listed in Section 2.1.2 amount to check if one of the cofactors is a tautology or a contradiction.

Detecting the top-xor-decomposition is more delicate. When dealing with a truth table, the input patterns of the two cofactors coincide, and the comparison is trivial. On the contrary, given partial specifications, these sets might even be non-intersecting. Let us now assume that we treat all the *don't knows* as *don't cares*. This choice would be prone to errors since, in the case of void intersection, the algorithm could perform a top-xor-decomposition that correctly synthesize the partial specifications but is unrelated to the Boolean structure of the function.

Let $\mathcal{M}_0$ and $\mathcal{M}_1$ be the sets of input minterms defining the subcovers of F_0 and F_1. The number of inputs/output patterns they store is $N_0 = |\mathcal{M}_0|$ and $N_1 = |\mathcal{M}_1|$. Finally, let n be the number of input variables of the original cover. If for all intersecting patterns it holds that $F_0 = F_1'$, the goal is to verify if the size of the intersection $N_\cap = |\mathcal{M}_0 \cap \mathcal{M}_1|$, is sufficiently large to conclude that the function it top-xor-decomposable. Assuming a uniform sampling from $\mathbb{B}^n$, the probability of k intersections is

$$P_k^U = \mathbb{P}(N_\cap = k) = \frac{\binom{2^{n-1}}{k}\binom{2^{n-1}-k}{N_0-k}\binom{2^{n-1}-N_0}{N_1-k}}{\binom{2^{n-1}}{N_0}\binom{2^{n-1}}{N_1}} \tag{10}$$

From P_k^U we compute the standard deviation σ, that we take as an uncertainty measure for the number of intersections. Then, we check two conditions:

1. More than one intersection is present, i.e., $N_\cap > 1$.
2. The probability that the number of intersections is larger than $N_\cap$ is negligible. For a given ϵ, e.g., 0.001, and considering fluctuations, $\sum_{k=0}^{N_\cap + \lceil\sigma\rceil} P_k^U \geq 1 - \epsilon$.

Therefore, if $F_0 = F_1'$ for all intersecting patterns and both conditions hold, F is assumed to be top-xor-decomposable. This heuristic is a statistically driven filtering criterion. It limits the number of top-xor-decompositions by filtering out cases that are unlikely to result in accurate implementations.

So far, we assumed a uniform sampling over the entire Boolean space. However, portions of the Boolean space might be *don't cares*. In Appendix D, we show that P_k^U underestimates the intersection probability. Consequently, if the uniform sampling approximation satisfies the filter, so does the care-set-based sampling.

3.3.3. Bottom-decomposition

Analogously to the top-xor-decomposition, the bottom-decomposability conditions listed in Section 2.1.2 require the comparison of partial covers. Again, a viable approach is to treat the *don't knows* as *don't cares*. Nonetheless, the same consideration given for the top-xor-decomposition motivates why this approach results in suboptimal implementations. Hence, we devised a *don't knows*-aware detection strategy of the bottom-condition leveraging mutual information.

To perform a bottom-decomposition step, we must first check whether Eq. (6) is valid for at least a pair of variables (X_i, X_j). Thanks to Claim 4 we can focus on the variable pairs that are equally informative. This observation considerably reduces the number of computations needed. For each such pair, we use our node creation criterion (Section 3.1) to obtain a list of candidate nodes $\tilde{X} = X_i \odot X_j$. Claim 5 identifies another necessary condition for bottom-decomposability. Therefore, we only consider the candidate nodes satisfying $I(\tilde{X}; F) = I(X_i, X_j; F)$. Notice that, in the worst case, we need to consider a number of pairs which is quadratic in the number of nodes. Therefore, it is preferable to avoid introducing variables if not needed. For this reason, we only add the candidate node $\tilde{X}$ if it has the highest mutual information among all of the variables in the netlist, and among all of the other candidates. At the next iteration, we branch on this variable. This criterion gives a guarantee on the convergence of the algorithm.

DK−DSD relies on the estimation of the probability mass function of each variable from the data. This estimation is possible independently of the completeness of the specifications. However, the estimation of the probabilities influences the results, and the equalities in Claims 4 and 5 might fail due to the impossibility of estimating the exact probability measure. Therefore, we propose a relaxed version of the technique that we name DK−RDSD, in which we accept that Claims 4 and 5 are satisfied up to a confidence interval.

3.3.4. Time complexity and model complexity

Fig. 8 highlights that the algorithms discussed so far are progressive refinement of a unique strategy. Let n be the number of variables in the netlist and N the number of input–output patterns in the specifications. In Appendix E, we show that $T_{\text{DK−XTSD}}(n, N) = O(nN)$ and $T_{\text{DK−DSD}}(n, N) = O(n^2 N)$. The more we refine the method, the higher is the computational complexity.

To motivate the subsequent sections, we discuss the relationship between the algorithmic refinements and the complexity of the resulting model. Fig. 8 shows a result that we will discuss in Section 4.2.1. The more we refine the strategy, the more we increase the analytical information on the function. Such information has the twofold benefit of increasing *test accuracy* and reducing the network size.

Despite the superior performances of DK − DSD, computing new features on the fly results in an algorithm that is not scalable, and a better

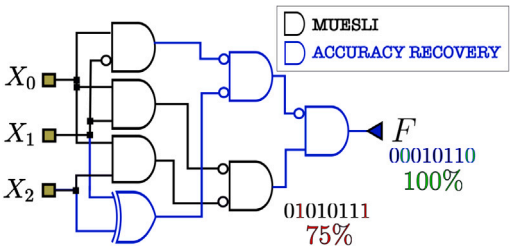


Fig. 9. Example of accuracy recovery on top of MUESLI. textttMUESLI generates the black circuit. Next, the decomposition introduces the blue portion of the circuit, recycling existing nodes to synthesize the function.

X_2	X_1	X_0	F	A	B	C
0	0	0	0	0		0
0	0	1	1	1		1
0	1	0	1	1		1
0	1	1	0	0	0	
1	0	0	1	1	1	
1	0	1	0	0	0	
1	1	0	0		0	0
1	1	1	0		0	0

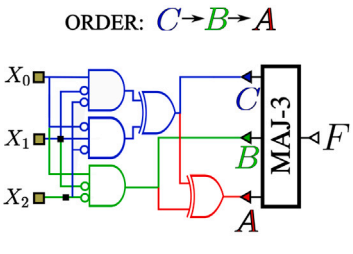


Fig. 10. Forest-like decomposition preserving memorization. We generate a set of sub-specifications and we synthesize a circuit for each sub-problem. The final model is a majority voter capable of memorizing the specifications.

approach should be based on DK – XTSD. We propose *accuracy recovery*, which is based on decoupling the two procedures: first we augment the inputs with engineered features; then we use the fastest decompositions on the new set of variables to complete the synthesis.

3.4. Accuracy recovery

Previous works in LSFPS claim that approximations are beneficial for increasing the test accuracy [16]. However, empirical practices and theoretical results in machine learning show that 100% accuracy on the specifications can lead to high test accuracy [22]. Our goal is to design an algorithm for synthesizing BNNs from data. Hence, we focus on the models capable of memorization. Furthermore, memorizing the specifications is necessary for using the algorithm in traditional logic synthesis, in which we must preserve the functionality. This section introduces *accuracy recovery*, a procedure leveraging decomposition techniques (Section 3.3) to map an approximate circuit to a new one, memorizing the specifications. The approximate circuit provides new features, and the decomposition strategy automatically selects the most informative ones among them.

3.4.1. Bridging approximate logic and memorization

ALS methods introduce informative nodes, greedily selected based on the target function. The resulting circuit is a reservoir of nodes, that is often an approximate circuit. We employ the previously discussed decomposition technique to bridge the gap between approximate synthesis and synthesis of the exact functionality. We ignore the bottom decomposition step due to its computational cost. In principle, the bottom-up assembly can substitute it. In the experimental section, we show that this technique can improve the quality of results of existing ALS techniques.

The direct application of the decomposition procedure on the approximate network can result in a size explosion due to a sub-optimal re-usage of resources. To obtain a size-aware decomposition, we assign a cost to each node. We define the cost as the number of gates in its transitive fan-in. When using a node for the synthesis, we set the cost of that node and its transitive fan-in to zero. Then we update the weights in the network. In this way, every time there is a tie of mutual information, the selection of the variable with minimum cost enables prioritizing the re-usage of resources rather than creating redundant circuitry. Fig. 9 presents an example of the use of MUESLI with our node creation strategy (Section 3.1), and followed by the accuracy recovery step. While MUESLI could only achieve a 75% accuracy on the target, *accuracy recovery* establishes the exact functionality while leveraging the precomputed circuitry.

Accuracy recovery is reminiscent of the emerging field of hyper-dimensional computing [23]. Such a field is promising for designing machine learning models in resource-constrained environments [24].

3.4.2. Multiple-output decomposition and forests

The statistical exploration of the available nodes for identifying candidate variables for the decomposition has a natural application in the synthesis of multiple-output functions. Indeed, synthesizing the sub-network for an output introduces newly available nodes in the system. Then, while synthesizing another output, the method can prefer logic sharing rather than synthesizing similar functions multiple times. Recycling pre-synthesized logic makes it possible to target logic sharing among the outputs. The discussion of the multiple-output case is out of the scope of this paper, and the strategies developed will be deepened in future works. However, it is interesting to see how we can apply this idea to a forest-like decomposition [25].

Fig. 10 represents the procedure. We split the specifications and employ the decomposition strategy to synthesize one sub-specification at a time. While synthesizing one of the sub-networks, the circuitry assembled during the previous decomposition steps is available for re-usage. Finally, we can combine the sub-networks outputs by using a majority function. We considered the case of 3 and 5 sub-networks. Note that each sub-network memorizes the associated portion of specifications. We call this model for VOTER-k. Note that the network in Fig. 10 memorizes the specification. Indeed, each input–output relationship is covered by the majority of the sub-specifications.

4. Experiments

Section 4.1 presents the experimental evidence of the analytical claims used in this research. Section 4.2 discusses the results obtained on the IWLS2020 benchmarks. Finally, Section 4.3 compares our approach with the ALS state-of-the-art on the binary MNIST classification problem.

4.1. Analytical considerations

4.1.1. Don't knows and don't cares

In this paper, we distinguish *don't knows* from *don't cares*. In logic synthesis, the *don't care* is a powerful degree of freedom that enables improving the optimization quality. Despite being conceptually different from a *don't know*, it is worth comparing the accuracy of the circuit obtained by treating the missing patterns as *don't cares* or *don't knows*. Indeed, previous works claim that decomposition strategies might be effective uniquely on functions presenting the *nearest neighbors* property [16]. This is because of the similarity between the splitting criterion of decompositions and the concept of expansion in *don't cares*-based synthesis techniques. Fig. 11 shows the result of the comparison. Each point of the x-axis is the representative function of one of the 222 4-inputs NPN classes. Claim 1, 2 and 3 legitimate us to limit the analysis to this subset of functions. Given the truth table of each such representative, we sweep over all possible numbers of inputs/output relations $N_e = 0, \dots, 16$ that could be missing. The number of ways we can choose the N_e patterns to erase is $\binom{16}{N_e}$. We uniformly sample

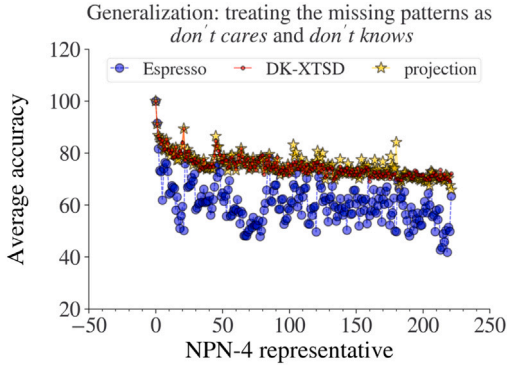


Fig. 11. Accuracy of the synthesized circuit averaged over an ensemble of partially-specified synthesis problems. Comparison of Espresso Minimizer, DK-XTSD and of accuracy recovery over the complete set of 2-inputs functions of the inputs.

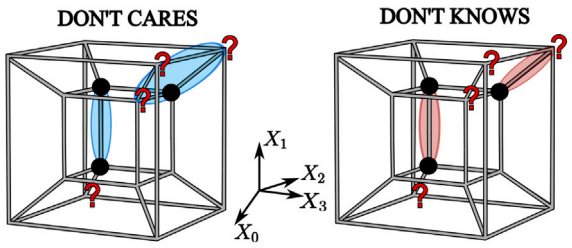


Fig. 12. Qualitative explanation of the difference between *don't know*-based and *don't care*-based synthesis: While both rely on the concept of implicants expansion, *don't knows*-awareness dictates more cautious choices.

$\min(20, \binom{16}{N_e})$ specifications among them. Given such ensemble of variations of the specifications we apply three synthesis techniques. The first one is Espresso Minimizer [20], which is *don't cares*-based. The second approach is DK-XTSD. The third approach is *accuracy recovery* over a set of variables which augments the input set. The additional variables are all possible two-input functions we can define using any pair of inputs. We call this method *projection* because the procedure maps the input space into a higher dimensional Boolean space. Overall, the *don't knows*-aware decomposition methods achieve higher accuracy, showing that the *don't knows*-aware technique is more suitable than *don't care*-based approaches for LSFPS. The reason for this is twofold. On the one hand, the possibility of detecting the top-xor-decomposition goes beyond the *nearest neighbor*'s property. On the other hand, a *don't care*-based optimizer will expand the cover over the missing pattern whenever this can lead to an optimization benefit. Instead, the *don't knows*-awareness limits the expansion to the cuts statistically selected from the data. Fig. 12 shows an example of the Boolean cover obtained using Espresso minimizer and DK-XTSD. Finally, we notice that the accuracy benefits from projecting the inputs into higher dimensional spaces. *Accuracy recovery* succeeds in automatically selecting useful variables during synthesis.

4.1.2. Top-decomposability and mutual information

To numerically validate Claim 3, we consider three cases of interest for proving it (see Appendix A): top-xor-decomposable functions ($\oplus$), top-and-decomposable functions ($\wedge$) and top-le-decomposable functions ($\leq$). We consider a representative for every NPN-4 class. For each 2-inputs gate characterizing the top-decompositions, we create a top-decomposable function in a fifth variable X_4. Figs. 13(a), 13(b), and 13(c) show that the mutual information of X_4 always satisfies Claim 3. This analysis confirms the statement for completely specified functions. However, it remains to extend the claim to the case of partial specifications.

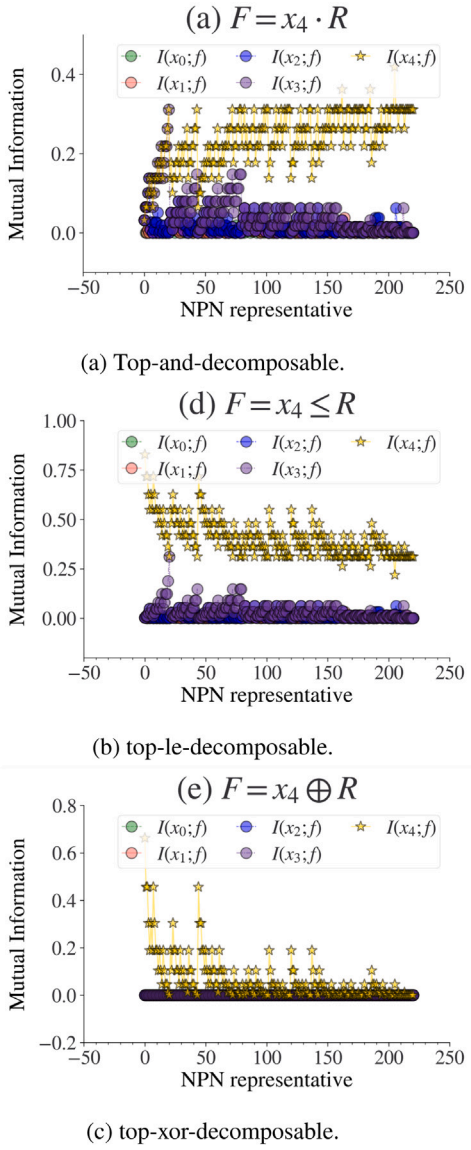


Fig. 13. Comparison of the mutual information of the variable in 4-inputs top-decomposable functions.

We numerically verify that maximizing mutual information remains an efficient filtering criterion for identifying the top-decomposable variable, even with *don't knows*. We repeat the previous experiment on functions having a smaller support size. We consider the representatives of the NPN-3 classes. For each representative, we define a top-xor-decomposable function in a 4th variable, and for each number of erasable patterns $N_e = 0, \ldots, 16$, we consider all the possible $\binom{16}{N_e}$ ways of erasing them. We use the method devised in Section 3.3.2 to classify the variable as top-xor-decomposable or not. Fig. 14 shows the average result for each function and each N_e. The higher the number of erased patterns, the lower the likelihood that the method classifies it as a top-xor-decomposable variable. This behavior is desirable because introducing a top-xor-decomposition in the topology without the data sustaining the hypothesis would result in an overcomplicated model, hardly meeting a correct counterpart in the actual functional properties. Furthermore, in the case of complete specifications, the filtering procedure always identifies the variable as top-xor-decomposable, as expected. In this way, the method extends directly to the case of complete specifications.

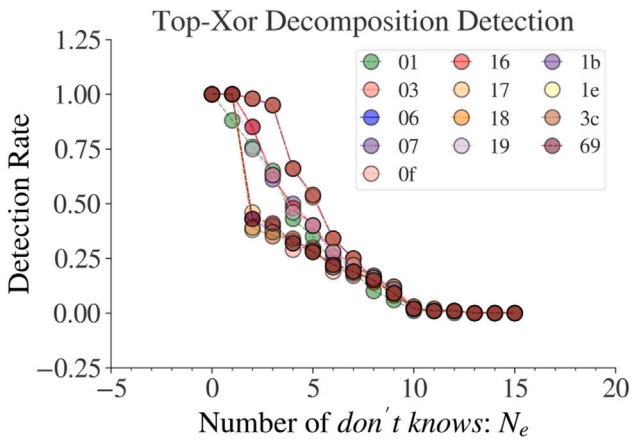


Fig. 14. Success rate for the top-xor-decomposability detection. Each line is the F_0 appearing in $F = X \oplus F_0$. For each number of *don't knows*, the point is the average top-xor detection rate over an ensemble of specifications.

4.2. IWLS2020 benchmarks

The IWLS2020 benchmarks consist of 100 single output Boolean functions from three domains: Arithmetic, Random Logic, and Machine Learning. The input size varies from 16 to 768, and, for each function, 12800 input–output pairs are available for synthesizing a circuit. Given a method, we evaluate its efficiency using the accuracy of the generated network when tested on 6400 input–output pairs not available during the synthesis phase. The 12800-dimensional list is split in two equally sized sub-lists. The sub-list named *training set* contains the specifications for the synthesis. Instead, the other one, named *validation set*, is available to check the performance of the model on unseen data. The overfit parameter is the average difference between the accuracies of the *validation* and the *test set*. A low overfit indicates high generality. We propose a comparative study of the techniques that we introduced in Section 3.3. The experiments were performed on a macOS machine with 2.2 GHz 6-Core i7 CPU and 16 GB RAM.

4.2.1. Disjoint support decomposition

In this experiment, we compare different decomposition strategies based on their performance over the IWLS2020 benchmarks. The goal is to show that introducing *don't knows* awareness in decomposition strategies effectively reduces the circuit size during synthesis while yielding higher *test accuracy*. All methods attain 100% accuracy on the training set,[2] making them candidates for *accuracy recovery*.

Shannon Decomposition (SD) is the simplest approach, and we use it as a baseline to verify the performance of the other methods. The second group of techniques comprises the *don't know*-aware decomposition strategies (Section 3.3). The main element increasing the average accuracy with respect to SD is the selection of the branching variable based on mutual information maximization [19]. Instead, by incrementally refining the DSD procedure, the number of gates decreases, as we expect from a DSD decomposition. Furthermore, we also observe an increase in the average *test accuracy*. This last improvement is associated to the benefit of detecting analytical features of the Boolean function. The DSD steps yield the following average improvements: the test accuracy increases by 1.11%, the number of gates decreases by 17.38%, and the depth decreases by 12.02%.

[2] Some benchmarks were created by compressing higher dimensional data. As a result, 100% accuracy is impossible for them due to contradicting specifications. However, all the non contradicting specifications are memorized, and the average accuracy is 100% to the second decimal digit.

The last block of the table reports the results for DK-RDSD. This algorithm is a relaxed version of DK-DSD, in which we acknowledge that the estimation of the probability mass function is penalized by incomplete knowledge of the specifications. We relaxed the filters defined by Claim 4 and 5, considering the equalities to be valid even in presence of an error of 5% and 1%, respectively. DK-RDSD often chooses to generate nodes on the fly and branch on them rather than on the primary inputs. While the relaxation in the filters induces considerable improvements in accuracy and size, the depth increases, probably due to unnecessary nodes generated at the leaves of the decomposition.

Table 1 confirms the complexity analysis of Section 3.3.4.

4.2.2. Don't cares-based decompositions

In this experiment we continue the *don't cares/don't knows* comparison. We challenge DK-XTSD with more aggressive strategies, that are *don't care* in nature. We show that treating the *don't knows* as *don't cares* can lead to memorization. However, only *don't knows*-awareness is suitable for to generalization. Table 2 summarizes the results.

First, we substitute the filter for detecting top-xor decomposability with a more aggressive criterion. DC-IXTSD is a *don't care*-oriented variant of DK-XTSD that only verifies if there is a contradiction (see Section 3.3.2). The worsening of the performances motivates the importance of the filtering criterion. Indeed, adding XORs when not needed adds layers to the network, does not simplify the function to synthesize and removes variables from the support that might be useful at later synthesis stages. In turn, this results in worse performances in terms of size, depth, accuracy, and time needed to reach the termination conditions.

The last method in Table 2 explores the entire support set, removing the variable selection by mutual information maximization. DC-TSD does not introduce XORs. We label this method as *don't care*-oriented because it does not rely on criteria to select the most statistically relevant variable. We explore the support and perform top decomposition for the first variable that allows it. As expected, such a myopic choice is detrimental to size and depth. Furthermore, the runtime shows that less informed branchings generate subproblems with slower convergence to the termination condition (see Fig. 2).

4.2.3. Muesli with accuracy recovery

In this experiment, we apply *accuracy recovery* after approximate synthesis by the MUESLI algorithm. The first row of Table 3 reports the quality of the results when using our variant of MUESLI on the benchmarks. Next, each subsequent block of the table presents the quality of the results of the networks synthesized with two approaches: a decomposition procedure, and *accuracy recovery* using the same decomposition on the approximate circuits synthesized by MUESLI. The decompositions considered are the top-xor-decomposition (DK-XTSD), the forest-like decomposition using the majority-of-3 (VOTER-3), and the forest-like decomposition using the majority-of-5 (VOTER-5). The application of *accuracy recovery* increases the *test accuracy* of both MUESLI and of the adopted decomposition procedure. Furthermore, the availability of informative nodes in the netlist results in a reduction of the number of gates. For a better comparison with the IWLS2020 contest the networks are represented as And-Inverter Graphs (AIGs).

The results show that the number of levels doubles when combining MUESLI with the decomposition procedures. The reason is that all the decomposition procedures actively use the nodes precomputed by MUESLI as branching variables. Since these nodes span an average of 32.05 levels, and the last nodes in the approximate system will be among the first ones to be selected by the decompositions, the depth of *accuracy recovery* is, on average, the sum of the depth in the two methods. We can extend this remark by noticing that VOTER-5 and VOTER-3 have almost the same runtime and VOTER-5 has a lower number of gates, despite being the combination of more subproblems. The main reason behind this phenomenon is logic sharing.

Table 1
Progressive refinement of DK-DSD.

Method	Train	Test	Gates	Levels	Overfit	Time[s]
SD	100.00	68.58	2274.08	90.66	0.02	1.21
DK-SD	100.00	80.54	1027.74	29.16	0.12	1.05
DK-TSD	100.00	80.54	873.51	25.4	0.12	0.95
DK-XTSD	**100.00**	**81.04**	**836.95**	**25.4**	**0.10**	**1.47**
DK-DSD	100.00	81.09	831.53	25.12	0.07	12.26
DK-RDSD	100.00	83.07	731.32	27.9	0.06	40.92

Table 2
Treating the missing patterns as *don't cares*.

Method	Train	Test	Gates	Levels	Overfit	Time[s]
DK-XTSD	**100.00**	**81.04**	**836.95**	**25.4**	**0.10**	**1.31**
DC-IXTSD	100.00	62.71	2669.27	85.38	0.11	4.76
DC-TSD	100.00	76.23	1455.16	84.58	0.02	19.35

Table 3
Accuracy recovery with MUESLI-based precomputation.

Method	Train	Test	Gates	Levels	Overfit	Time[s]
MUESLI	80.10	78.37	94.90	32.05	0.02	30.06
DK-XTSD	100.00	81.04	840.54	25.66	0.10	1.29
AR-MUESLI	100.00	82.49	837.95	53.99	0.05	31.15
VOTER-3	100.00	82.96	957.13	40.34	0.02	3.65
AR3-MUESLI	100.00	83.57	944.54	61.80	0.08	32.60
VOTER-5	100.00	83.09	945.52	42.75	0.08	3.69
AR5-MUESLI	100.00	83.99	939.73	63.83	0.07	31.61

The statistics-based decomposition recycles nodes, converging to a solution quickly, without creating additional circuitry, and while encoding additional information. We remark that size awareness plays a key role in size reduction (Section 3.4.1).

Table 3 also reveals that the contribution dominating time complexity is $T_{\texttt{MUESLI}}(n, N)$, i.e., the time needed to generate the initial approximate circuit (Appendix E). Therefore, the bottleneck of the method is the time-consuming ALS algorithm employed, motivating the need to investigate scalable ALS algorithms. Therefore, we consider Chatterjee's method due to its unmatched scalability.

4.2.4. Random LUT network with accuracy recovery

In this experiment, we apply *accuracy recovery* after approximate synthesis by Chatterjee's method [14]. Table 4 reports the evolution of the performance when increasing the number of nodes in a unique hidden layer ($N \times 1$), and when increasing the depth ($1024 \times L$). Both ways of introducing new nodes result in an improvement of the *test accuracy*. Furthermore, while increasing the size of the approximate logic results in an increase of the accuracy, we do not observe an explosion in the number of gates. On the contrary, more accurate models resulting from more expressive approximate circuits can even present a smaller number of gates thanks to their higher information content. More investigations are needed to better assess the potential of combining *accuracy recovery* with k-LUT networks. Contrarily to the experiments with MUESLI, in this case we see an increase of the computational time in the different settings. The reason is that MUESLI automatically generates the approximate netlist and terminates at an average of 94.90 gates. Instead, in this case, the lack of a support selection heuristic comes at the cost of fixing an initial random topology. The decomposition procedure has as number of features the number of gates in the fixed topology, which is $G \in \{1024, 2048, 4096\}$. In Appendix E we discussed that $T = O(GN)$. Table 4 confirms this claim since $T(n, N, 2G) = 2T(n, N, G)$ These results motivate the interest in developing more advanced ALS procedure, to speed up the initial calculation while increasing the accuracy.

4.3. Binary Mnist

In this experiment, we apply the devised algorithms to the binary MNIST problem. The MNIST database is a set of binary handwritten digits [26]. Each image is a 28 by 28 matrix, representing a digit from 0 to 9. The dataset consists of 60000 images for the training and 10000 images for the test. The binary MNIST problem consists in classifying each digit as belonging to one of two classes: The first class contains the digits from 0 to 4, the second class contains the digits from 5 to 9. Table 5 compares four decomposition strategies to the state-of-the-art in ALS [14,15]. The policies we used for splitting the specifications differ from VOTER-3

Already in the simple form of memorization via XAG decompositions, we achieve higher accuracy than the state-of-the-art. Moving to VOTER-3 and VOTER-5, we used two different synthesis policies. For VOTER-3, we used the complete set of attributes defined for the method, i.e., we enabled logic sharing from one sub-specification to the other. As discussed before, this increases the accuracy without leading to a size explosion. Instead, in VOTER-5, we kept each sub-problem isolated. This forces each sub-circuit to learn the specifications without leveraging existing information. This approach results in an increased number of gates. But the size remains one order of magnitude smaller than the size obtained by AR-8LUTNET.

Finally, we created a random k-LUT network composed of 5 layers, each containing 1024 8-LUTs (8LUTNET). This is the structure used by Chatterjee [14]. The application of *accuracy recovery* to the approximate circuit increases the *test accuracy*, and memorizes the specifications.

It is interesting to notice that the result of AR-8LUTNET depends on the initial topology. This topology is similar to what we would have in a classical neural network. If we compare such a model with models purely generated from data, with no structural bias enforced apriori, we see an order-of-magnitude reduction in the number of gates. Therefore, comparing the hardware-oriented decompositions with AR 8LUTNET confirms the interest in developing hardware-aware machine-learning techniques.

Table 4
Accuracy recovery with Chatterjee's based precomputation.

Method	Train	Test	Gates	Levels	Overfit	Time[s]
DK-XTSD	100.00	81.04	840.54	25.66	0.10	1.29
1024 × 1	100.00	80.93	817.15	29.69	0.07	5.85
2048 × 1	100.00	81.38	816.62	29.46	0.11	11.37
4096 × 1	100.00	82.36	790.81	28.72	0.17	32.53
1024 × 2	100.00	81.71	854.61	31.75	0.03	10.06
1024 × 4	100.00	81.44	1025.78	34.04	0.05	19.08

Table 5
Binary MNIST: XAGs compared with the state-of-the-art.

Method	Train	Test	Gates	Depth	Time[s]
VOTER-5	**100.00**	**95.73**	**13054**	**58**	**101.45**
VOTER-3	**100.00**	**93.13**	**3728**	**67**	**53.694**
DK-XTSD	**100.00**	**92.84**	**3175**	**45**	**18.00**
LSE8[15]	94.68	92.76	–	–	–
AR-8LUTNET	**100.00**	**91.55**	**257209**	**113**	**100.32**
8LUTNET[14]	99.00	90.00	–	–	–

5. Conclusions

The proposed DK-DSD procedure is a novel approach to extend the applicability of DSD to Boolean functions when *don't knows* are present. The method allows us to perform DSD on partial covers of input size as large as 768. The results show the effectiveness of using mutual information to synthesize circuits aware of the presence of *don't knows*, as opposed to the approach of treating them as *don't cares*. Additionally, our approach minimizes the number of gates while assembling the network, thus reducing the need for pruning techniques. Additionally, we introduced the concept of *accuracy recovery*, a technique for mapping an approximate circuit into a new one that exactly synthesizes the specifications. We show that this technique enables increasing the test accuracy of existing ALS methods. Possible applications of *accuracy recovery* include designing multiple-output synthesis techniques targeting logic sharing, as well as speeding up methods that require many iterations to converge, such as evolutionary algorithms [27].

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgment

The authors thank Alan Mishchenko, Satrajit Chatterjee, Alessandro Tempia Calvino, Alessandro Favero, Siang-Yun Lee, Mingfei Yu, Hanyu Wang, Rassul Bairamkulov, and Dewmini Marakkalage for the fruitful discussions.

Appendix A. Analytical considerations

In this appendix we prove the claims of Section 3.2

Claim 1. *Consider two Boolean functions $F, G : \mathbb{B}^n \mapsto \mathbb{B}$, and two subsets of their support $X \subseteq S_F$, $\tilde{X} \subseteq S_G$. If G can be obtained from F through an appropriate inputs/output negation $\gamma : (S_F, F) \mapsto (S_G, G)$, and $\tilde{X} = \gamma(X)$, then $I(X;F) = I(\tilde{X};G)$.*

Proof. Let $X \in \mathbb{B}^m$ be the combination of m Boolean variables in the support of F. Then, Eq. (8) reads

$$I(X;F) = H(X) + H(F) - H(X,F) \tag{11}$$

$$H(Z) = -\sum_{z \in \mathbb{B}^{d(Z)}} p_Z(z) \log_2 p_Z(z) \tag{12}$$

The dimensionalities of the variables are $d(X) = m$, $d(F) = 1$, and $d(X,F) = m+1$. Eq. (12) is *local*, in the sense that each contribution depends uniquely on a single $\mathbb{B}^d$ point. All the contributions have the same functional form and the sum is performed over the entire $\mathbb{B}^d$ space. Let us now consider the variable $\tilde{Z} \in \mathbb{B}^d$, obtained from Z by means of the negations rule γ, connecting G and F:

$$Z = (Z_1, \dots, Z_p) \xrightarrow{\gamma} \tilde{Z} = (\gamma_1(Z_1), \dots, \gamma_p(Z_p))$$

where $\gamma_i(Z_i)$ can be Z_i or Z_i', depending on γ. The map γ is bijective, so that for each minterm in the original space, γ identifies a minterm in the mapped space having the same probability measure $p_Z(z) = p_{\tilde{Z}}(\tilde{z})$. Claim 1 follows from the bijectivity of the map and the locality of Eq. (12)

$$H(Z) = H(\tilde{Z}) \Rightarrow I(X;F) = I(X;G) \quad \square \tag{13}$$

Claim 2. *Consider two NPN-equivalent Boolean functions $F, G : \mathbb{B}^n \mapsto \mathbb{B}$. For all $X_i \in S_F$, there is a variable $X_j \in S_G$ such that $I(X_i;F) = I(X_j;G)$.*

Proof. Claim 2 follows from the combination of Claim 1 and the concept of *NPN equivalence*. The variables X_i and X_j are related by the permutations mapping F into G. $\square$

Appendix B. Top-decomposition conditions

Claim 3. *Consider a function F that is top-decomposable in X_i: $F = X_i \odot T(S_F \backslash X_i)$. For any variable $X_j \in S_F \backslash X_i$*

$$I(X_i;F) \geq I(X_j;F)$$

Proof. A function is top-decomposable if it can be written as Eq. (5). If we consider the top-decomposability conditions we observe that we can define the following maps:

$$X_i < T = X_i' \wedge T \mapsto X_i \wedge T \tag{14}$$

$$X_i \vee T = (X_i' \wedge T')' \mapsto X_i \leq T = (X_i \wedge T')' \tag{15}$$

These formulas have both an experimental and a theoretical implication. Experimentally, since we want to understand if X_i maximizes mutual information, we can focus on the functions $\{\wedge, \leq\}$, because Claim 2 and Eqs. (14) and (15) imply that the same holds for $\{<, \vee\}$. This motivates the plots in Section 4.1.2. Theoretically, Claim 3 needs to be proven only for $\{\wedge, \oplus\}$ and considering all possible remainder functions $T : \mathbb{B}^{n-1} \mapsto \mathbb{B}$.

Top-Xor decomposable. Let $F = X_i \oplus T$, with X_j being another variable in the support. For a completely specified function, $p_{X_i}(1) = p_{X_i}(0) = 1/2$. Consequently, $H(F) = 1$ and $H(X_i) = H(X_j) = 1$. Furthermore,

$H(X_j|F) = 1$ since $p_{X_j|F} = 1/2$. Hence, using the symmetry of mutual information, we obtain

$$I(X_j; F) = H(X_j) - H(X_j|F) = 0 \tag{16}$$

The non-negativity of $I(X_j; F)$ yields $I(X_i; F) \geq I(X_j; F)$.

Top-and decomposable. Mutual information can be written as a function of four variables: $p^{0|0}_{F|X} = p_{F|X}(0|0)$, $p^{0|1}_{F|X} = p_{F|X}(0|1)$, $p^0_F = p_F(0)$ and $p^0_X = p_X(0)$. We consider the single variable probabilities as parameters, $I = I(p^{0|1}_{F|X}, p^{F|X}_{0|0}; p^0_F, p^0_X)$. By direct computation, it is possible to identify the parametric space of possible solutions in the $(p^{0|1}_{F|X}, p^{0|0}_{F|X})$ plane, that is the line

$$p^0_F = \sum_x p_{F|X}(0|x)p_X(x) \rightarrow p^{0|0}_{F|X} = \alpha - \beta p^{0|1}_{F|X} \tag{17}$$

where $\alpha = p^0_F/p^0_X$ and $\beta = (1 - p^0_F)/(1 - p^0_X)$. In this way, we obtain a 1-variable function parametrized by p^0_F and p^0_X. By direct computation of the second derivative we obtain

$$\partial^2_{p^{0|1}_{F|X}} = \frac{\beta^2 p^0_X}{(\alpha - \beta p^{0|1}_{F|X})(1 - \alpha + \beta p^{0|1}_{F|X})} + \frac{\beta^2(1 - p^0_X)}{p^{0|1}_{F|X}(1 - p^{0|1}_{F|X})} \tag{18}$$

All the quantities appearing are probabilities. Hence, as expected, $\partial^2_{p^{0|1}_{F|X}} I > 0$, that proves that mutual information is concave along the constraint. Therefore, it is maximized at the extremes of the support, that corresponds to $p^{0|1}_{F|X_i} = 1$, $p^{0|0}_{F|X_i} = 1$. Since we are proving the result for the AND function, $p^{0|0}_{F|X_i} \geq p^{0|1}_{F|X_i}$, $j = 0, \ldots, n-1$, that leads to the conclusion that the variable maximizing mutual information is the one for which $p^{0|0}_{F|X} = 1$, i.e., the top-and-decomposable variable. □

Appendix C. Bottom decomposition conditions

In this section we prove the main results used in Section 3.3.3.

Claim 4. *Consider a function F that is bottom decomposable in $\tilde{X} = X_i \odot X_j$. Then, $I(X_i; F) = I(X_j; F)$.*

Proof. Considering Eq. (6), there are two cases:

1. If $\odot \in \{\wedge, \vee, \oplus\}$, the result follows from the invariance of the function under variables swap $X_i \leftrightarrow X_j$.
2. If $\odot \in \{<, \leq\}$ we define the function G, obtained from F by the input inversion mapping $\{<, \leq\} \mapsto \{\wedge, \vee\}$. The result follows from Claim 1. □

Claim 5. *Consider a function F that is bottom decomposable in $\tilde{X} = X_i \odot X_j$. Then, $I(X_i, X_j; F) = I(\tilde{X}; F)$.*

Proof. Proving this claim corresponds to show that $H(F|\tilde{X}) = H(F|X_i, X_j)$. We start by rewriting $p_{F|\tilde{X}}$:

$$p_{F|\tilde{X}} = \frac{\sum_{x_i x_j} p_{F X_i X_j \tilde{X}}}{p_{\tilde{X}}} = \frac{\sum_{x_i x_j} p_{F X_i X_j} \delta_{\tilde{x}, x_i \odot x_j}}{p_{\tilde{X}}} \tag{19}$$

We insert this function in the conditional entropy, obtaining

$$\begin{aligned} H(F|\tilde{X}) &= \sum_{x_i, x_j, f} p_{F X_i X_j} \sum_{\tilde{x}} \delta_{\tilde{x}, x_i \odot x_j} \log p_{F|\tilde{X}} = \\ &= \sum_{x_i, x_j} p_{X_i X_j} \sum_f p_{F|X_i X_j} \log p_{F|X_i X_j} = \\ &= H(F|X_i X_j) \quad \square \end{aligned}$$

Appendix D. Worst case for the XOR filter

In this section, we prove that, in cases of interest, the devised filters for the top-xor-decomposition are based on worst-case analysis, legitimating their use. The basic assumption behind the filter for the top-xor-decomposition is that the input cubes are sampled uniformly from the entire Boolean space. When considering the cofactors for a variable, the probability of k intersections of the reduced cubes appearing in the cofactors is P^U_k, reported in Eq. (10).

On the other hand, a more precise assumption would be to consider the cubes as being sampled uniformly from the *care set* of the Boolean function. Hence, if we define $S_U = 2^{n-1}$ as the size of the Boolean space of the cofactors support, the corresponding quantity for the care set is $S_{CS} \leq S_U$, and the probabilities to compare read

$$P^\mu_k = \frac{\binom{S_\mu}{k}\binom{S_\mu - k}{N_0 - k}\binom{S_\mu - N_0}{N_1 - k}}{\binom{S_\mu}{N_0}\binom{S_\mu}{N_1}} \qquad \mu = U, CS. \tag{20}$$

Our method is legitimate if we can show that

$$\sum_{k=0}^{N} P^U_k \geq \alpha \Rightarrow \sum_{k=0}^{N} P^{CS}_k \geq \alpha \tag{21}$$

A sufficient condition for this to be true is that $P^{CS}_k \geq P^U_k$. We consider the limiting case of small intersection size, i.e., $S_U \gg N_0, N_1 \gg k$ and $S_{CS} \gg N_0, N_1 \gg k$. This condition is of practical interest since it is the one that could lead to erroneous assumptions. By approximating the Binomial coefficients as $\binom{N}{k} \approx \frac{N^k}{k!}$, we obtain

$$\frac{P^{CS}_k}{P^U_k} \rightarrow \left(\frac{S_U}{S_{CS}}\right)^k \geq 1 \qquad \forall k \geq 0. \tag{22}$$

that proves the sufficient condition.

Appendix E. Time complexity analysis

In this section, we derive the computational complexity of the main algorithms of this paper. We start from DK − XTSD. Let n be the initial number of variables in the netlist. The worst case for DK−XTSD is the synthesis of a function in which each decomposition step removes one variable from the support and splits the data into two halves. We use the time needed to compute the mutual information $I(X; F)$ as the unit time, with $X, F \in \mathbb{B}$. If N is the initially available number of input–output pairs, the recursion defining the time complexity reads

$$T(n, N) = 2T\left(n - 1, \frac{N}{2}\right) + n + N \tag{23}$$

where n is the time needed for computing mutual information for the variables in the support. Instead, N is the time needed to check top-xor-decomposability. Finally, we obtain

$$T_{\text{DK−XTSD}}(n) = O(nN) \tag{24}$$

For what concerns DK − DSD, the most expensive part is to go through all of the variable pairs. Hence, we consider a fictitious function for which, at each iteration, all of the variables have equal mutual information, a new variable is added, and the dataset is split in two halves. In this way, the recursion for time complexity becomes

$$T(n, N) = 2T\left(n, \frac{N}{2}\right) + n + N + A\binom{n}{2} + n \log n \tag{25}$$

Where A is some constant and $n \log n$ is the time needed to sort the variables by mutual information. This yields the time complexity

$$T_{\text{DK−DSD}}(n) = O(n^2 N) \tag{26}$$

Finally, *accuracy recovery* creates G nodes bottom-up ($\mathcal{B}$), and it later performs a decomposition D. Hence,

$$T_{AR}(n, N) = T_{\mathcal{B}}(n, N) + T_D(G, N) \tag{27}$$

Considering the bottom-up strategies discussed in this work, the cost of kLUTNET of a fixed topology is linear in the number of gates. Hence, if the topology has G gates

$$T_{\text{kLUTNET+DK−XTSD}}(n, N, G) = O(GN) \tag{28}$$

Instead, it is harder to define the computational complexity of MUESLI since there is no guarantee of convergence. However, if we fix a threshold to a number of gates G, the worst time complexity reads $T_{\texttt{MUESLI}} = O(G^2)$, yielding

$$T_{\texttt{MUESLI+DK-XTSD}}(n, N, G) = O(\max(G^2, GN)) \tag{29}$$

References

[1] Muhammad Ayaz, Mohammad Ammad-Uddin, Zubair Sharif, Ali Mansour, El-Hadi M Aggoune, Internet-of-Things (IoT)-based smart agriculture: Toward making the fields talk, IEEE Access 7 (2019) 129551–129583.

[2] Arijit Ukil, Soma Bandyoapdhyay, Chetanya Puri, Arpan Pal, IoT healthcare analytics: The importance of anomaly detection, in: 2016 IEEE 30th International Conference on Advanced Information Networking and Applications, AINA, IEEE, 2016, pp. 994–997.

[3] Fotios Zantalis, Grigorios Koulouras, Sotiris Karabetsos, Dionisis Kandris, A review of machine learning and IoT in smart transportation, Future Internet 11 (4) (2019) 94.

[4] Jozef Mocnej, Martin Miškuf, Peter Papcun, Iveta Zolotová, Impact of edge computing paradigm on energy consumption in IoT, IFAC-PapersOnLine 51 (6) (2018) 162–167.

[5] Giovanni De Micheli, Synthesis and Optimization of Digital Circuits, McGraw-Hill Higher Education, 1994.

[6] Jeff Heaton, Ian goodfellow, yoshua bengio, and aaron courville: Deep learning, 2018.

[7] Farah Fahim, Benjamin Hawks, Christian Herwig, James Hirschauer, Sergo Jindariani, Nhan Tran, Luca P Carloni, Giuseppe Di Guglielmo, Philip Harris, Jeffrey Krupa, et al., Hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices, 2021, arXiv preprint arXiv:2103.05579.

[8] George A. Constantinides, Rethinking arithmetic for deep neural networks, Phil. Trans. R. Soc. A 378 (2166) (2020) 20190051.

[9] Barret Zoph, Quoc V. Le, Neural architecture search with reinforcement learning, 2016, arXiv preprint arXiv:1611.01578.

[10] Giulia Meuli, Mathias Soeken, Giovanni De Micheli, Xor-and-inverter graphs for quantum compilation, Npj Quantum Inform. 8 (1) (2022) 1–11.

[11] Alan Mishchenko, Satrajit Chatterjee, Roland Jiang, Robert K Brayton, FRAIGs: A Unifying Representation for Logic Synthesis and Verification, ERL Technical Report, 2005.

[12] Siang-Yun Lee, Heinz Riener, Alan Mishchenko, Robert K Brayton, Giovanni De Micheli, A simulation-guided paradigm for logic synthesis and verification, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (2021) 1.

[13] Arlindo L. Oliveira, Alberto Sangiovanni-Vincentelli, Learning complex boolean functions: Algorithms and applications, in: NIPS, 1993, pp. 911–918.

[14] Satrajit Chatterjee, Learning and memorization, in: International Conference on Machine Learning, PMLR, 2018, pp. 755–763.

[15] Sina Boroumand, Christos-Savvas Bouganis, George A Constantinides, Learning boolean circuits from examples for approximate logic synthesis, in: Proceedings of the 26th Asia and South Pacific Design Automation Conference, 2021, pp. 524–529.

[16] Shubham Rai, Walter Lau Neto, Yukio Miyasaka, Xinpei Zhang, Mingfei Yu, Qingyang Yi Masahiro Fujita, Guilherme B Manske, Matheus F Pontes, Leomar S Junior, Marilton S de Aguiar, et al., Logic synthesis meets machine learning: Trading exactness for generalization, 2020, arXiv preprint arXiv:2012.02530.

[17] Zhufei Chu, Mathias Soeken, Yinshui Xia, Lunyao Wang, Giovanni De Micheli, Advanced functional decomposition using majority and its applications, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 39 (8) (2019) 1621–1634.

[18] Valeria Bertacco, Maurizio Damiani, The disjunctive decomposition of logic functions, in: Iccad, vol. 97, 1997, pp. 78–82.

[19] J. Ross Quinlan, C4. 5: Programs for Machine Learning, Elsevier, 2014.

[20] Patrick Mcgeer, Jagesh Sanghavi, Robert Brayton, Alberto Sangiovanni Vincentelli, ESPRESSO-SIGNATURE: A new exact minimizer for logic functions, in: Proceedings of the 30th International Design Automation Conference, 1993, pp. 618–624.

[21] Brian W. Kernighan, Shen Lin, An efficient heuristic procedure for partitioning graphs, Bell Syst. Tech. J. 49 (2) (1970) 291–307.

[22] Mikhail Belkin, Daniel Hsu, Siyuan Ma, Soumik Mandal, Reconciling modern machine-learning practice and the classical bias–variance trade-off, Proc. Natl. Acad. Sci. 116 (32) (2019) 15849–15854.

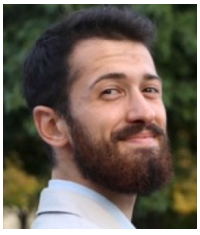
[23] Lulu Ge, Keshab K. Parhi, Classification using hyperdimensional computing: A review, IEEE Circuits Syst. Mag. 20 (2) (2020) 30–47.

[24] Partha Pratim Ray, A review on TinyML: State-of-the-art and prospects, J. King Saud Univ.-Comput. Inform. Sci. (2021).

[25] Tin Kam Ho, Random decision forests, in: Proceedings of 3rd International Conference on Document Analysis and Recognition, vol. 1, IEEE, 1995, pp. 278–282.

[26] Yann LeCun, The MNIST database of handwritten digits, 1998, http://yann.lecun.com/exdb/mnist/.

[27] Julian Francis Miller, Simon L. Harding, Cartesian genetic programming, in: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers, 2009, pp. 3489–3512.



Andrea Costamagna received the B.Sc. degree in physical engineering from the Politecnico di Torino, Turin, Italy, in 2018, the joint M.Sc. degree in physics of complex systems from Politecnico di Torino (Italy) and Université Paris-Saclay (France) in 2020, and the M.Sc. degree in nanotechnologies for ICTs from Politecnico di Torino, Turin, Italy, in 2021. Currently, he is working toward the Ph.D. degree at the Integrated Systems Laboratory at EPFL, Switzerland. His research interests include logic synthesis and statistical machine learning.

Giovanni De Micheli (Fellow, IEEE) is a research scientist in electronics and computer science. He is credited for the invention of the Network on Chip design automation paradigm and for the creation of algorithms and design tools for Electronic Design Automation (EDA). He is Professor and Director of the Integrated Systems Laboratory at EPFL Lausanne, Switzerland. Previously, he was Professor of Electrical Engineering at Stanford University. Prof. De Micheli is a Fellow of ACM , AAAS and IEEE, a member of the Academia Europaea and an International Honorary member of the American Academy of Arts and Sciences. His current research interests include several aspects of design technologies for integrated circuits and systems, such as synthesis for emerging technologies.