Logic Synthesis From Incomplete Specifications Using Disjoint Support Decomposition

Andrea Costamagna
Integrated Systems Laboratory
EPFL Lausanne, Switzerland
andrea.costamagna@epfl.ch

Giovanni De Micheli
Integrated Systems Laboratory
EPFL, Lausanne, Switzerland
giovanni.demicheli@epfl.ch

***Abstract*—Approximate logic synthesis is an emerging field that tolerates errors in the synthesized logic circuits for better optimization quality. Indeed, in many computing problems, the requirement of preserving the exact functionality either results in unnecessary overuse of resources or is hardly possible to meet. The latter case is typical of incompletely specified synthesis problems, targeting the hardware implementation of a Boolean function from a partial knowledge of its care set. The missing elements of the care set are named *don't knows*. Previous works identified information theory-based decomposition strategies as powerful synthesis tools. Nonetheless, the definition of an automatic method for approximate synthesis is an open problem, and the approximate counterpart of many logic synthesis techniques is still missing. In this paper, we extend a disjoint support decomposition algorithm to target Boolean functions in the presence of *don't knows*. Furthermore, we integrate the decomposition in an information theory-based synthesis flow. Relative experiments on the IWLS2020 benchmarks show that, on average, the addition of the designed decomposition to the flow reduces by 15.81% the number of gates and by 9.66% the depth.**

***Index Terms*—approximate logic synthesis, disjoint support decomposition, information theory.**

I. Introduction

Approximate logic synthesis (ALS) is an emerging field aiming at designing circuits for error-resilient applications. This feature is present in many computing problems, including media processing, data mining, and supervised learning [1]. All these applications highly benefit from high-performance hardware accelerators, but some of them lack a straightforward implementation due to the incompleteness of specifications.

The vast majority of the existing ALS techniques aim to devise algorithmic solutions to approximate a circuit representation of a Boolean function [2]. However, these methods cannot be applied to problems where the knowledge of the function is only partial. Incompletely specified functions typically occur in supervised learning tasks, for which it is desirable to realize dedicated hardware. Under a supervised learning lens, the incomplete specifications correspond to the *training set*, while the input-output pairs on which we test the accuracy of the devised hardware corresponds to the *test set*. Our work takes inspiration from the existing literature on the ALS subfield named *learning from examples* (LFE) [3]–[6].

978-1-6654-6700-1/22/$31.00 ©2022 IEEE

The fundamental computing problem addressed in LFE is learning a Boolean function from an incomplete knowledge of its care set, named *cover*. When the cover specifies the output for all input patterns, it corresponds to a *truth table*. Otherwise, it is called *incomplete*. The input-output pairs belonging to the care set, and missing in the cover, are named *don't knows*.

In literature, several techniques exist for learning Boolean functions from examples. Oliveira et al. [4] defined 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 [5], and Boroumand et al. [3] exploited it in a variation of the Muesli algorithm. Finally, the IWLS2020 benchmarks and the solutions proposed by the competing teams [6] constitute the main existing framework for evaluating algorithms for LFE. Overall, the main findings of the aforementioned works are the following:

- Decomposition procedures for the creation of decision trees are key strategies for tackling the problem.
- Mutual information, introduced in Sec. II, is helpful both in Muesli-like and in Fulfringe-like algorithms.
- No technique designed so far is effective for all functions.
- Many methods rely on noninterpretable hyperparameters.
- Many approaches use pruning to meet size constraints.

Therefore, the definition of an automatic synthesis method for approximate circuits remains an open problem [7].

The main contribution of this work is the generalization of a *disjoint support decomposition* (DSD) algorithm to Boolean functions represented by incomplete covers. Compared to existing literature, it does not rely on noninterpretable hyperparameters and reduces the size without pruning. The proposed procedure combines properties of mutual information and a decision tree decomposition. We show that adding the proposed DSD procedure to a synthesis technique improves its performances. On average, it reduces the size by 15.81% and the depth by 9.66%.

II. Background

Let $F : \mathbb{B}^n \mapsto \mathbb{B}$ be a single-output Boolean function, where $\mathbb{B} = \{0, 1\}$. Its *support* $\mathcal{S} = \{x_i\}_{i=0}^{n-1}$ is the set of variables on which F depends. In this paper, we represent Boolean functions as covers, i.e., lists of input patterns paired with

the corresponding outputs. We study algorithms to synthesize multilevel networks from incomplete Boolean covers.

A. Shannon Decomposition

Given a function F and a variable $x_i \in \mathcal{S}$, the *Shannon Decomposition* (SD) of F is

$$F = x_i \cdot F_{x_i} + x_i' \cdot F_{x_i'} \doteq \texttt{ite}(x_i, F_{x_i}, F_{x_i'}). \tag{1}$$

Where F_{x_i} ($F_{x_i'}$) is the positive (negative) cofactor of F, i.e., the function of domain $\mathcal{S}\backslash x_i$ whose cover is the sub-cover of F, identified by $x_i = 1$ ($x_i = 0$). When the variable is explicitly stated, we use the notation $F_1 \doteq F_{x_i}$ and $F_0 \doteq F_{x_i'}$.

B. Disjoint Support Decomposition

Performing the DSD of a function F corresponds to identifying a set of functions $\mathcal{A} = \{A_i\}_{i=0}^{m-1}$ ($m < n$) with disjoint supports and a function G such that

$$F(\mathcal{S}) = G(\mathcal{A}). \tag{2}$$

When a truth table is available, such a decomposition can be obtained by applying two procedures until convergence: *top-decomposition* and *bottom-decomposition* [8], [9].

1) Top-Decomposition: The top-decomposition identifies when the dependence of F on a variable x_i is of type

$$G = x_i \odot T(\mathcal{S}\backslash x_i), \tag{3}$$

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.3:

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

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

2) Bottom-Decomposition: Suppose that two variables x_i and x_j influence F uniquely through a 2-input function:

$$G = B(x_i \odot x_j, \mathcal{S}\backslash\{x_i, x_j\}). \tag{4}$$

Let us introduce the simplified notation for the cofactors: $F_{00} \doteq F_{x_i'x_j'}$, $F_{01} \doteq F_{x_i'x_j}$, $F_{10} \doteq F_{x_ix_j'}$ and $F_{11} \doteq F_{x_ix_j}$. For Eq.4 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) $F_{01} \neq F_{00} = F_{01} = F_{11}$: $G = \texttt{ite}(x_i < x_j, F_{01}, F_{10})$;
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) $F_{00} = F_{11}, \quad F_{10} = F_{01}$: $G = \texttt{ite}(x_i \oplus x_j, F_{01}, F_{00})$.

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

C. Entropy and Mutual Information

Let x be a random variable taking values in $\mathbb{X}$ and having a probability distribution $p_x(\cdot)$. Its Shannon entropy is

$$H(x) \doteq -\sum_{\pi \in \mathbb{X}} p_x(\pi) \log_2 p_x(\pi). \tag{5}$$

It quantifies the uncertainty on the value taken by the variable. From this quantity, *mutual information* is derived as

$$I(x; f) = H(f) - H(f|x). \tag{6}$$

It quantifies the reduction in uncertainty on the target variable f, given the knowledge of x.

D. Chatterjee's Method

Let us consider a cover and its associated nodes set. For each node variable, we know its value at each example. Once k of these variables are selected, say $\mathcal{S}_k = \{x_{s(1)}, \ldots, x_{s(k)}\}$, Chatterjee's method creates a statistically-optimal new node [5], with support $\mathcal{S}_k$. Each bit pattern $\pi \in \mathbb{B}^k$ of $\mathcal{S}_k$ can appear more than once in the examples set. Therefore, C_π^1 and C_π^0 are used to count how many times the output of F is 1 or 0, given that the variables in $\mathcal{S}_k$ take the values specified by π. The truth table of the new k-LUT node reads

$$\tilde{x}(\pi) = \begin{cases} 1 & \text{if } C_\pi^0 < C_\pi^1 \\ r & \text{if } C_\pi^0 = C_\pi^1 \\ 0 & \text{if } C_\pi^0 > C_\pi^1 \end{cases}, \pi \in \mathbb{B}^k \tag{7}$$

where r is a symmetric Bernoulli random number.

This method is helpful to learn a statistically-optimal node given its support, but it gives no information on which nodes to select. Due to the exponential size of the k-LUT nodes, Chatterjee's method only applies to covers with sufficiently small support sizes. The seminal paper [5] copes with this problem by applying it to a randomly generated k-LUT network. Instead, in this work, we use the procedure in two cases: the first one is the termination condition of a decision tree decomposition. The second one is the extraction of the 2-input function for a method we devised to detect the bottom-decomposability condition given two variables.

III. Information Theory-Based Disjoint Support Decomposition

A. Structure of the Algorithm

Algorithm 1 is the proposed synthesis technique for incompletely 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 (tautology), all output values are zeros (contradiction), or the support size is smaller than `max_sup` and Chatterjee's method can be applied. This last step is interesting because we ultimately convert the network to an And Inverter Graph (AIG) for size evaluation and accuracy checking. If the support size of the k-LUT node is sufficiently small, the k-LUT to AIG converter identifies the NPN class of the function and maps the node to the size optimum subgraph.

Algorithm 1 signal ← Decomposition($\mathcal{S}, F$)

1: **if** ($F = \top$) **then**
2: return 1
3: **else if** ($F = \bot$) **then**
4: return 0
5: **else if** ($|\mathcal{S}| \leq$ `max_sup`) **then**
6: return `Chatterjee method` $(\mathcal{S}, F)$
7: $x \leftarrow$ choose variable x from $\mathcal{S}$
8: **if** (`is top-decomposable` $(x, \mathcal{S}, F, \odot)$) **then**
9: return $x \odot$`Decomposition` $(\mathcal{S}, F)$
10: **else if** (`is bottom-decomposable` $(\mathcal{S}, F)$) **then**
11: return `Decomposition` $(\mathcal{S}, F)$
12: $f_0 \leftarrow$ `Decomposition` $(\mathcal{S} \backslash x, F_0)$
13: $f_1 \leftarrow$ `Decomposition` $(\mathcal{S} \backslash x, F_1)$
14: return $x \cdot f_1 + x' \cdot f_0$

If we ignore lines 5 to 6 and 8 to 11 in Algorithm 1, the procedure corresponds to a Shannon decomposition. Previous studies [4], [6] show that maximizing mutual information is a good criterion for selecting the variable at each SD step. We refer to the combination of this decomposition procedure with the additional termination condition of lines 5 to 6 as Information theory-based Shannon Decomposition (ISD).

With the further addition of lines 8 to 11, we name the method Information theory-based DSD (IDSD). First, the algorithm tries to perform a top-decomposition step on the selected variable. In Sec. III-B1 we motivate why, also for this case, 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 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 SD step.

B. Don't Knows-based Disjoint Support Decomposition

1) 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. We test the top-decomposition condition only on the variable maximizing the mutual information. Indeed, if Eq. 3 holds, the single 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. 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. Once chosen the variable, the designed top-decomposition follows closely the approach described for truth tables. Cases 1 to 4 listed in Subsection II-B1 amount to check if one of the cofactors is a tautology or a contradiction. On the other hand, the incompleteness of the cover complicates case 5, i.e., the top XOR-decomposition. When dealing with a truth table, the input patterns of the two sub-covers associated with the cofactors coincide, and the comparison is trivial. On the contrary, given an incomplete cover, these sets are incomplete and they might even be non-intersecting. Hence, treating the *don't knows* as *don't cares* is 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 Π_0 and Π_1 be the sets of input patterns defining the sub-covers of F_0 and F_1. The number of examples they store is $N_0 = |\Pi_0|$ and $N_1 = |\Pi_1|$. Finally, let n be the number of input variables of the original cover. If for all intersecting patterns $F_0 = F_1'$, the goal is to verify if $N_\cap = |\Pi_0 \cap \Pi_1|$, the size of the intersection, is sufficiently large to conclude that case 5 holds. Assuming a uniform sampling from the space of the input patterns, the probability of k intersections is

$$P_k = \mathbb{P}(N_\cap = k) = \frac{\binom{2^{n-1}}{k}\binom{2^{n-1}-k}{N_0-k}\binom{2^{n-1}-N_0}{N_1-k}}{\binom{2^{n-1}}{N_0}\binom{2^{n-1}}{N_1}}. \quad (8)$$

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

- More than one intersection is present, i.e., $N_\cap > 1$.
- 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, this reads $\sum_{k=0}^{N_\cap + \lceil\sigma\rceil} P_k \geq 1-\epsilon$.

Therefore, if $F_0 = F_1'$ for all intersecting patterns and both conditions hold, F is assumed to satisfy case 5. This heuristic is a statistically driven selective criterion. It limits the number of top-XOR decompositions by filtering out cases that are likely to result in not-accurate implementations.

2) Bottom-Decomposition: Analogously to the top XOR-decomposition, cases 1 to 5 listed in Sec. II-B2 require the comparison of incomplete covers. Again, a trivial 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 first need to verify if there exist a pair of variables (x_i, x_j) appearing in function F only as in Eq. 4. Such variables, if present, are equally informative. Hence, we can limit the analysis to the pairs satisfying $I(x_i; f) = I(x_j; f)$. This observation allows us to reduce considerably the number of calculations needed. For each pair, we use Chatterjee's method to find the new node $\tilde{x} = x_i \odot x_j$. Then, for Eq. 4 to be valid, it must be necessarily true that

$$I(\tilde{x}; f) = I(x_i, x_j; f), \quad (9)$$

$$I(\tilde{x}; f) = I(\tilde{x}, x_i; f) = I(\tilde{x}, x_j; f) = I(\tilde{x}, x_i, x_j; f). \quad (10)$$

Eq. 9 states that the new node contains the same information as variables x_i and x_j combined. Eq. 10 indicates that combining $\tilde{x}$ with any variable of its support does not yield

any appreciable advantage. If Eq. 9 and Eq. 10 hold, we filter out a candidate node that is likely to be redundant by requiring that $I(\tilde{x}; f) > I(x_i; f)$. Finally, we verify if $I(\tilde{x}; f) \geq \max_x I(x; f)$. Indeed, the presence of a more informative node may prevent the optimal node $\tilde{x}$ from being identified. Since the detection of the bottom-decomposability condition is followed by the irreversible substitution of (x_i, x_j) with $\tilde{x}$, it is better to first branch on $\arg\max_x I(x; f)$.

IV. Experimental Results

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 an AIG network. 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. While the one named *training set* is the cover for the synthesis phase, there is no strict indication on the other, i.e., the *validation set*. The overfit parameter is the average difference between the accuracies of the *validation* and the *test set*. We did not use the validation set for the synthesis. This choice results in a low overfit, which indicates the high generality of the networks.

We propose a comparative study of the techniques discussed in Section III. The experiments were performed on a Linux machine with 1.8 GHz i7-8550U CPU and 8 GB RAM.

As a baseline, we synthesized the IWLS2020 benchmarks using SD and ISD. In Table I we compare the average of the results of these strategies with the ones of IDSD. Adding the DSD steps to ISD gives the following average improvements: the test accuracy increases by 1.34%, the number of gates decreases by 15.81%, and the depth decreases by 9.66%. We only report the average of the improvements since the data from which we computed them is a table with 100 entries.

TABLE I
Methods Comparison

comparison	Performances				
method	***test[%]***	***gates***	***levels***	***overfit[%]***	***time[s]***
SD	64.96	6189.64	140.61	0.14	6.54
ISD	80.43	1162.23	72.54	0.18	3.36
IDSD	**81.14**	**958.19**	**68.58**	**0.11**	**73.07**
*dc*DSD	60.39	6405.97	316.39	0.02	84.04
SOA	**88.69**	**2517.66**	**39.96**	**1.86**	-

Table I also shows the worsening of the performances when performing DSD with *don't knows* treated as *don't cares* (*dc*DSD). This observation strengthens the assumption that *don't cares*-based methods are not effective in LFE tasks as they have not been built with generalization in mind [3].

Figure 1 compares the test accuracies of SD, ISD, IDSD, and *dc*DSD. Benchmark 74 is particularly informative as it corresponds to a 16-input XOR. The accuracy of ISD is 50.75%, meaning that the network performs random guesses. Instead, the application of IDSD yields the synthesis of the exact functionality, also lowering the number of gates from 6346 to 45. Hence, the top-decomposition method correctly identifies the XOR structure. In Table I, we also reported the results for the state-of-the-art (SOA) [6]. One should notice that the authors used a portfolio approach and post-optimized the network. On the contrary, since our goal is to show the effectiveness of the IDSD procedure, used as a unique technique, we did not perform any post-optimization.

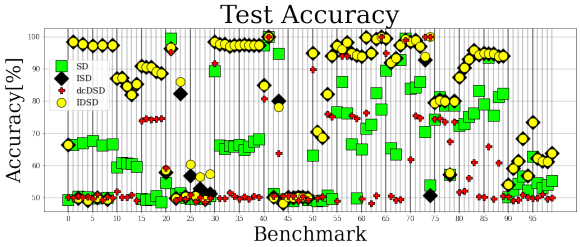

Fig. 1. Test Accuracies of SD, ISD, IDSD, and *don't cares*-based DSD.

V. Conclusions

The proposed IDSD procedure is a novel approach to extend the applicability of DSD to incomplete covers when *don't knows* are present. The method allows us to effectively perform DSD on incomplete 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 trivial 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. Future works will focus on extending IDSD to the multiple-output case, leveraging the statistical nature of the method to maximize the shared logic.

References

[1] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *2013 18th IEEE European Test Symposium (ETS)*. IEEE, 2013, pp. 1–6.

[2] I. Scarabottolo, G. Ansaloni, G. A. Constantinides, L. Pozzi, and S. Reda, "Approximate logic synthesis: A survey," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2195–2213, 2020.

[3] S. Boroumand, C.-S. Bouganis, and G. 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.

[4] A. L. Oliveira and A. Sangiovanni-Vincentelli, "Learning complex boolean functions: Algorithms and applications," in *NIPS*, 1993, pp. 911–918.

[5] S. Chatterjee, "Learning and memorization," in *International Conference on Machine Learning*. PMLR, 2018, pp. 755–763.

[6] S. Rai, W. L. Neto, Y. Miyasaka, X. Zhang, M. Yu, Q. Y. M. Fujita, G. B. Manske, M. F. Pontes, L. S. Junior, M. S. de Aguiar *et al.*, "Logic synthesis meets machine learning: Trading exactness for generalization," *arXiv preprint arXiv:2012.02530*, 2020.

[7] W. Liu, F. Lombardi, and M. Shulte, "A retrospective and prospective view of approximate computing [point of view," *Proceedings of the IEEE*, vol. 108, no. 3, pp. 394–399, 2020.

[8] Z. Chu, M. Soeken, Y. Xia, L. Wang, and G. De Micheli, "Advanced functional decomposition using majority and its applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 8, pp. 1621–1634, 2019.

[9] V. Bertacco and M. Damiani, "The disjunctive decomposition of logic functions," in *iccad*, vol. 97, 1997, pp. 78–82.