Information Graph-Based Resubstitution For Networks of Look-Up Tables

Andrea Costamagna
EPFL
Lausanne, Switzerland

Alessandro Tempia Calvino
EPFL
Lausanne, Switzerland

Alan Mishchenko
UC Berkley
Berkley, California

Giovanni De Micheli
EPFL
Lausanne, Switzerland

***Abstract*—This paper addresses the problem of minimizing the area of circuits represented as networks of look-up tables. This optimization problem is crucial for achieving high-quality synthesis for FPGA designs. Furthermore, any combinational logic cone can be seen as an interconnection of look-up tables, increasing the impact of the proposed optimization. Resubstitution is a powerful logic minimization method that can identify non-local logic dependencies and restructure sub-networks to reduce area. Existing resubstitution-based algorithms for look-up table networks rely heavily on SAT solving, limiting the number of optimization attempts and the size of the resubstitution sub-networks to 1 node [1]. The methods discussed in this paper rely on circuit simulation to increase the number of considered optimization candidates and enable resubstitution using sub-networks with more than 1 node. The experiments show that the proposed heuristics can identify optimization opportunities missed by state-of-the-art methods, improving 11 out of 23 best-known results in the ongoing EPFL synthesis competition, and resulting in a 4% smaller area, compared to state-of-the-art approaches for the EPFL benchmarks.**

***Index Terms*—FPGA, logic synthesis, area optimization**

I. Introduction

OVER the last decades, configurable hardware has grown in importance because its versatility enabled rapid prototyping, adaptability to changing requirements, energy efficiency, resource optimization, and overall democratization of electronic development. Field-programmable gate arrays (FPGAs) are configurable hardware components, in which combinational circuits are represented as networks of look-up tables (LUTs). Minimizing the LUT count is a crucial for meeting the resource constraints of FPGAs.

In the design flow, logic synthesis tools contribute to area minimization by optimizing technology-independent representations and mapping them to a target technology. However, as Figure 1 shows, technology-independent logic minimization does not always guarantee an improved area after LUT mapping. This insight motivates us to develop algorithms aimed at optimizing mapped networks. Since any combinational logic cone can be seen as an LUT network, efficient algorithms for optimizing LUT networks can broadly impact logic synthesis.

Resubstitution is a powerful logic minimization method that tries to resynthesize the function of a node using a set of candidate nodes in the network called *divisors*. Resubstitution can identify non-local logic restructuring opportunities missed by even high-effort area-oriented technology mappers.

Existing resubstitution-like algorithms for LUT networks, such as `mfs` [1], optimize circuit parts called *windows*. For each node of the network, called *pivot*, `mfs` solves a *support selection* problem, i.e., it identifies a subset of divisors that can be used to express the function of the *pivot*. The larger the window size, the more global information is used, resulting in higher optimization quality. The capability of `mfs` to perform logic restructuring is a key component of design space exploration commands such as `&deepsyn` in ABC [2]. However, `mfs` heavily relies on SAT solving for support selection, limiting the window size that can be used, and only resynthesizes sub-networks by introducing 1 node (1-resub).

This research was supported in part by Synopsys inc., in part by SRC Contract 3173.001 "Standardizing Boolean transforms to improve quality and runtime of CAD tools".

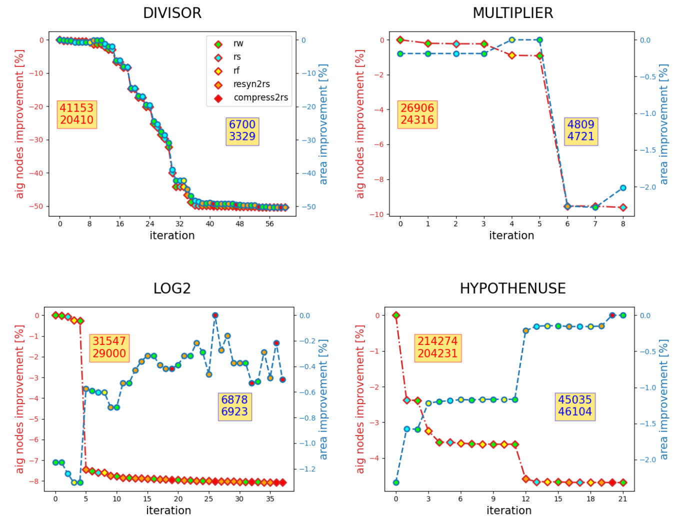

Fig. 1: Correlation between technology-independent size optimization and area after technology mapping to 6-input LUTs for different networks extracted during logic optimization.

Simulation-guided resubstitution [3], [4] is an optimization paradigm aiming at improving scalability beyond that of the conventional resubstitution methods [1]. Rather than computing support by SAT solving and resynthesizing the pivot afterward, a simulation-guided algorithm relies on partial simulation to solve the support computation problem and to resynthesize the pivot. Due to the incomplete specification used for simulation, a SAT-based verification of the correctness of the replacements is needed. This method replaces SAT-based support selection with SAT-based equivalence checking, which is considerably more efficient.

In this paper, we adopt the *simulation-guided paradigm* [3] for optimizing LUT networks and propose a number of novel solutions to address the limitations of `mfs`. In particular, two related problems are solved:

1) How to efficiently do *support selection* without SAT?
2) How to enable resubstitution with more than 1 LUT?

First, we propose an algorithm to address the support selection problem. Next, we define a new heuristic for decomposing large look-up tables into smaller ones, while taking *don't-cares* into account. Finally, we combine these methods to develop a novel resubstitution engine, which focuses on extracting global information to enable high-effort design space exploration.

The experiments show that our heuristics identify optimization opportunities missed by state-of-the-art engines, improving 11 "best results" in the EPFL competition. Furthermore, our heuristic proves effective for design space exploration, resulting in a 4% smaller area for the EPFL benchmarks, when replaced to `mfs2` (a variant of `mfs`) in an optimization flow.

II. Preliminaries

This section introduces the background and formalizes the theoretical notions underlying the proposed algorithms.

A. Logic Synthesis and Optimization Basics

A *k-input look-up table* (k-LUT) network is a directed acyclic graph, in which nodes represent k-input logic functions, and edges represent wires. If there is a path from a node x_i to a node x, x_i is in the *transitive fanin* (TFI) of x, and x is in the *transitive fanout* (TFO) of x_i. The *maximum fanout free cone* (MFFC) of a node x is the set of nodes, including x and those TFI nodes whose fanouts are in the MFFC. The primary inputs (PIs) are nodes without fanins in the network.

A *structural cut* of a node x is a set of nodes, called *leaves*, such that any path from a PI to x passes through at least one leaf. The Boolean sub-network defined by a node x and a cut $\mathcal{C}$ of size k identifies the Boolean function of the cut of node x, also called *the structural cut*. If the cut contains only PIs, the function of the cut is the global function of the node. The possible input patterns are called *minterms*. An approximate function of node x computed by simulating a subset of all possible input patterns is called the *simulation signature*.

The *don't-care set* of a cut $\mathcal{C}$ is the set of the input patterns never appearing at the leaves of the cut. The *don't-care* minterms can be characterized by the *care function*, which takes value 0 for *don't-care* minterms and 1, otherwise.

The presence of *don't-cares* in logic circuits is related to the existence of *reconvergences*, i.e., portions of the circuit where the paths from a set of leaf nodes to a target node diverge before reaching the target node. A *reconvergence-driven cut* of size k is a structural cut of size k constructed to include as many reconvergencies as can be captured within the depth-limited TFI/TFO cone [5].

Given a node x, characterized by its global function, or by an approximation of its global function called *simulation signature*, x_M is the value of the function at minterm M. This identifies three sets:

- *Onset* : $\mathtt{ON}_x = \{M \in \mathbb{B}^n : \; x_M = 1\}$
- *Offset* : $\mathtt{OF}_x = \{M \in \mathbb{B}^n : \; x_M = 0\}$
- *Don't-care set* : $\mathtt{DC}_x = \{M \in \mathbb{B}^n : M \notin \mathtt{ON}_x \cup \mathtt{OFF}_x\}$

The main advantage of using *simulation signatures* is that they contain global information, and include *satisfiability don't-cares*. The *don't-care set* of a simulation signature might include *care values* of the global function. Some authors name these minterms *don't-knows*, so that a signature is a function $x : \mathbb{B}^n \rightarrow \{0, 1, *, ?\}$ [6].

B. Resubstitution and Functional Cuts

Boolean resubstitution is a logic optimization method that restructures the TFI of the node to reduce the area of the network. The new node is the output of a newly generated TFI having a smaller area than the current MFFC. We name the inputs to this sub-network *functional cut*.

Definition II.1. Given a target node x, a *functional cut* $\mathcal{C}$ is a set of nodes not in the TFO of the target such that there is a function $f : \mathbb{B}^{|\mathcal{C}|} \rightarrow \{0, 1, *\}$ realizing $x = f(\mathcal{C})$.

This function exists if, for each pattern appearing at the leaves of the functional cut, there is a unique value of the global function of the target. This intuitive statement is formalized as the *dependency theorem* [7]:

Theorem 1. *Given a function x and a set of functions $\mathcal{C} = \{x_i\}_{i=1}^k$, $x, x_i : \mathbb{B}^n \rightarrow \mathbb{B}$, there is a function $f : \mathbb{B}^k \rightarrow \{0, 1, *\}$ such that $x = f(\mathcal{C})$ iff $\forall M_i, M_j \in \mathbb{B}^n$,*

$$x_{M_i} \neq x_{M_j} \Rightarrow \exists x_l \in \mathcal{C} \text{ s.t. } x_{l,M_i} \neq x_{l,M_j}. \tag{1}$$

If such a function exists, it is called a dependency function.

Finding a *functional cut* is the *support selection problem*. Finding the dependency sub-network for a *functional cut* is the *resynthesis problem*. A *resubstitution engine* consists of a *support selection* heuristic and a *resynthesis* heuristic. This paper proposes a new *resubstitution engine* for LUT networks.

A key observation is that a *structural cut* is also a *functional cut*, but a *functional cut* is not necessarily a *structural cut*.

C. Previous Works on Optimizing Look-Up Table Networks

Several existing algorithms address area optimization of LUT networks and are implemented in the tool `abc` [2].

Command `mfs` [1] uses resubstitution with don't-cares to re-express the target node using a single k-LUT with reduced fanins number. The support selection problem is solved as an instance of a satisfiability problem, and an SAT solver verifies if the dependency theorem is satisfied by a subset of k or fewer divisors. Unlike `mfs`, our approach is not limited to a single k-LUT. Furthermore, our support selection method is entirely based on *simulation signatures*, while SAT solving is only used to verify functional equivalence.

Command `lutpack` [8] employs Boolean decomposition to re-express LUT sub-networks using fewer k-LUT. However, the support selection strategy consists of finding *structural cuts* and is agnostic of global *don't-cares*. Our technique considers global *don't-cares* and relies on the notion of *functional cuts*. Furthermore, our resynthesis algorithm enhances Boolean decomposition with a new strategy based on *don't-cares*.

D. Information Graphs and Edge Covering

This work relies on *information graphs* introduced by Józwiak [9], and making intuitive the concept of *sets of pairs of functions to be distinguished* (SPFDs) [10]–[13].

Definition II.2. An *Information graph* (IG) of a Boolean function $x : \mathbb{B}^n \rightarrow \{0, 1, *\}$ is a set of *bipartite graphs* in which a vertex is a minterm, and the parts of each element are the *onset* ($\mathcal{P}_1^x = ON_x$) and the *offset* ($\mathcal{P}_0^x = OF_x$).

$$\Upsilon_x = \{(\mathcal{P}_0^x | \mathcal{P}_1^x)\} \tag{2}$$

Our definition preserves the information of the Boolean attributes of the minterms. The top-part of Figure 2 shows the

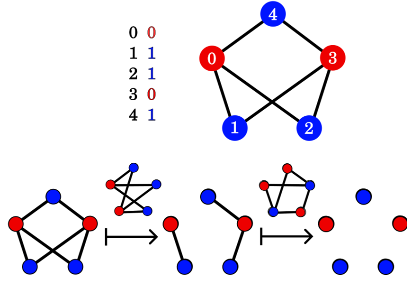

Fig. 2: Example of an IG and a covering process. At each step, the edges in common with the covering graph are removed.

IG for a simulation signature, where the vertices associated to the offset and onset minterms are colored in red and blue.

Given two Boolean functions x, x_i, and the corresponding IGs $\Upsilon_x, \Upsilon_{x_i}$, we define the *edge covering* of Υ_x by Υ_{x_i} as

$$\Upsilon_x \succ \Upsilon_{x_i} = \{(\mathcal{P}_0^x \cap \mathcal{P}_0^{x_i} | \mathcal{P}_1^x \cap \mathcal{P}_0^{x_i}), (\mathcal{P}_0^x \cap \mathcal{P}_1^{x_i} | \mathcal{P}_1^x \cap \mathcal{P}_1^{x_i})\}$$

The covering operation returns an IG whose edges are all the edges contained in Υ_x, but not in Υ_{x_i}, inducing a partition of an IG into non-connected bipartite sub-graphs. The left-hand side of Figure 2 shows how removing from the initial IG the edges in common with another IG gives a 1-covered IG, in which we can recognize the two complete bipartite sub-graphs.

Definition II.3. Given a set of functions $\mathcal{C} = \{x_i\}_{i=1}^T$, the *covering process* induced by $\mathcal{C}$ $\Upsilon_x^0 \to \cdots \to \Upsilon_x^T$ reads

$$\Upsilon_x^0 = \{\mathcal{P}_0^{x,0} = (\mathcal{P}_{0,0}^{x,0} | \mathcal{P}_{0,1}^{x,0})\} \doteq \Upsilon_x \tag{3}$$

$$\Upsilon_x^t = \{\mathcal{P}_i^{x,t} \ i \in [0, 2^t)\} \tag{4}$$

$$\begin{cases} \mathcal{P}_{2m}^{x,t} = (\mathcal{P}_{m,0}^{x,t-1} \cap \mathcal{P}_0^{x_t} | \mathcal{P}_{m,1}^{x,t-1} \cap \mathcal{P}_0^{x_t}) \\ \mathcal{P}_{2m+1}^{x,t} = (\mathcal{P}_{m,1}^{x,t-1} \cap \mathcal{P}_1^{x_t} | \mathcal{P}_{m,1}^{x,t-1} \cap \mathcal{P}_1^{x_t}) \end{cases} \tag{5}$$

At step t, the covering process removes the edges in common with Υ_{x_t}: we say that Υ_x^t is *t-covered*. The number of bipartite sub-graphs in Υ_x^t is upper-bounded by 2^t.

Remark 1. *Given a set $\mathcal{C} = \{x_i\}_{i=1}^T$, and a target node x, Theorem 1 is satisfied iff $\mathcal{C}$ fully covers Υ_x [14].*

For instance, in Figure 2, the two divisors whose IG is used for the covering process would satisfy Theorem 1.

E. Information Graph Representations

Practically useful simulation signatures have size $p \sim 2^{10}$. Hence, representing IGs using their adjacency matrix is infeasible from the memory size perspective, as it would require $\mathcal{O}(p^2)$ bits, assuming one bit per an onset/offset minterm pair. Recent advances in resubstitution [14] were possible by observing that IGs can be constructively represented during a covering process. This corresponds to a data structure tailored for performing covering processes on IGs.

Given a p-dimensional simulation signature for node x, and a care function μ, the bitstring pair (x, μ) uniquely distinguishes its onset from its offset:

$$\Upsilon_x = \{(\mathcal{P}_0^x | \mathcal{P}_1^x)\} \to \{(\mu, x)\} \tag{6}$$

The edge covering of Υ_x by Υ_{x_i} yields an IG whose edges are the minterm pairs that have not yet been covered:

$$\Upsilon_x^1 = \begin{cases} \mathcal{P}_0^{x,1} = (\mathcal{P}_0^x \cap \mathcal{P}_0^{x_i} | \mathcal{P}_1^x \cap \mathcal{P}_0^{x_i}) \to (x_i' \mu, x) \\ \mathcal{P}_1^{x,1} = (\mathcal{P}_0^x \cap \mathcal{P}_1^{x_i} | \mathcal{P}_1^x \cap \mathcal{P}_1^{x_i}) \to (x_i \mu, x) \end{cases}$$

Similarly, after the covering process $\Upsilon_x \succ \Upsilon_{x_i} \succ \Upsilon_{x_j}$

$$\Upsilon_x^2 \to \{(x_i' x_j' \mu, x), (x_i' x_j \mu, x), (x_i x_j' \mu, x), (x_i x_j \mu, x)\}$$

Essentially, every partition $\mathcal{P}_i^{x,t}$ of a t-covered IG can be identified by two bitstrings of length p: a mask μ_i^t and the signature of the target node. The mask is the result of intersecting the care set of the initial target with the literals of the covering divisors identifying the sub-graph in the partition. We name this representation the *covering representation*, and it allows us to evaluate the number of edges as

$$||\Upsilon_x^t||_E = \sum_{i=0}^{2^t-1} ||\mathcal{P}_{i,0}^{x,t}||_1 ||\mathcal{P}_{i,1}^{x,t}||_1 = \sum_{i=0}^{2^t-1} ||\mu_i^t x'||_1 ||\mu_i^t x||_1 \tag{7}$$

The memory requirement for storing a t-covered information graph is $M[\Upsilon_x^t] = p(2^t+1)$, so it is preferable to the *adjacency matrix* representation as long as $2^t < p$, i.e., $t < 10$.

Recent research on support selection has proposed to construct *functional cuts* via a covering process, where at each step a divisor is sampled from a probability distribution

$$p(x_i; \beta, t) \propto e^{-\beta ||\Upsilon_x^t \succ \Upsilon_{x_i}||_E} \tag{8}$$

This distribution and its parameters was given as an ansatz in the original paper [14]. This paper shows how to infer models for the probability distribution from experimental data.

III. The Resubstitution Engine

To devise a simulation-guided resubstitution algorithm for LUT networks, we must address two sub-problems:

1) How to find a *functional cut*?
2) How to synthesize LUT networks using *don't-cares*?

This section discusses the algorithm's structure.

A. The Simulation-Guided Algorithmic Structure

Algorithm 1 illustrates our resubstitution engine. Following the *simulation guided paradigm*, we randomly sample p minterms from the Boolean space $\mathbb{B}^n$, where n is the number of PIs of the circuit. Next, we simulate the input patterns, generating a simulation signature of size p for each node.

For each node x, a structural exploration of the network identifies the MFFC and the candidate divisors by computing a *reconvergent driven cut*. The cut is then expanded to include a fixed number of nodes that are not in the TFI of x, but whose fanins are candidates divisors.

The algorithm's core (Section III-B) relies entirely on simulation signatures to identify a resubstitution candidate. If the area of this sub-network is smaller than the area of the MFFC, we check functional equivalence and commit the result.

In the case of failure, the counter-example returned by the SAT solver can be used to update the simulation signatures, as suggested by the *simulation guided paradigm* [3]. While present in our framework, in this work we only use the random signatures generated at the beginning of optimization, and we investigate how much we can improve the quality of the results by refining the support selection heuristics.

Algorithm 1 `LUT_RESUB(ntk, K )`

1: `generate random simulation patterns`
2: `simulate the network`
3: **for all** $x \in$ `ntk` **do**
4: $\quad \mathcal{D} \leftarrow$`collect some divisors`
5: $\quad \mathcal{R} \leftarrow$`collect the mffc`
6: $\quad$ **for** `a number of trials` **do**
7: $\quad\quad \mathcal{C} \leftarrow$ `functional support from` $\mathcal{D}$ $(|\mathcal{C}| \leq K)$
8: $\quad\quad f \leftarrow$ `dependency function from` $\mathcal{C}$
9: $\quad\quad$ **if** `database.has(`f`)` **then**
10: $\quad\quad\quad$ `subntk`$\leftarrow$ `database[`f`]`
11: $\quad\quad$ **else if** `on the fly is enabled` **then**
12: $\quad\quad\quad$ `subntk`$\leftarrow$ `synthesize on the fly`
13: $\quad\quad$ **if** `area(subntk)` $\leq$ `area(`$\mathcal{R}$`)` **then**
14: $\quad\quad\quad$ **if** `subntk is equivalent to` x **then**
15: $\quad\quad\quad\quad$ `substitute` x `with subntk`
16: $\quad\quad\quad$ **else**
17: $\quad\quad\quad\quad$ `save the counter example`
18: $\quad\quad\quad\quad$ **if** 64 `counter-examples` **then**
19: $\quad\quad\quad\quad\quad \sigma \leftarrow$`update the simulations`

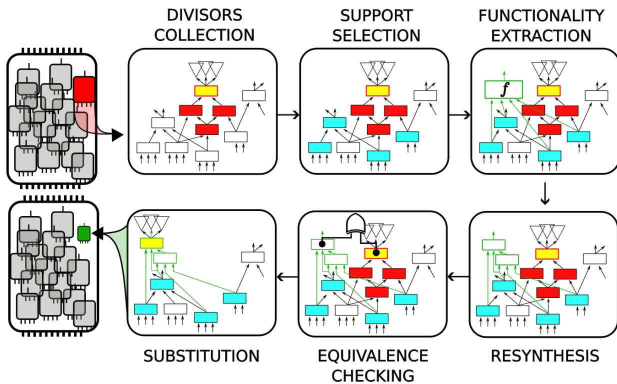

Fig. 3: Template of simulation-guided resubstitution for LUT networks, with a detail on LUT-decomposition.

B. The Resubstitution Engine

Figure 3 illustrates the core of the engine. First, we solve the *support selection problem*. Section IV discusses our approach, which relies on sampling *functional cut*s according to a probability distribution inferred from experimental data.

Given a *functional cut* $\mathcal{C}$ for a *pivot* node x, we extract $f : \mathbb{B}^{|\mathcal{C}|} \rightarrow \{0, 1, *\}$ by considering each minterm $M \in \mathbb{B}^{|\mathcal{C}|}$, and assigning it to $\texttt{ON}_x$, $\texttt{OF}_x$, or $\texttt{DC}_x$ based on its appearance and the value of the target signature.

In the next step, the algorithm generates a k-LUT sub-network representing the *dependency function* f. Section V discusses the proposed approach, which consists of a decomposition algorithm which can take *don't-cares* into account. If the area of this sub-network is smaller than the area of the current MFFC, we verify the functional equivalence of the target node with the new sub-network, and we commit the result if all the checks have passed.

C. Support Selection, Resynthesis, and Generalization

Remark 1 states that the *support selection* is a set covering-problem. Our goal is to find an *optimal functional cut*:

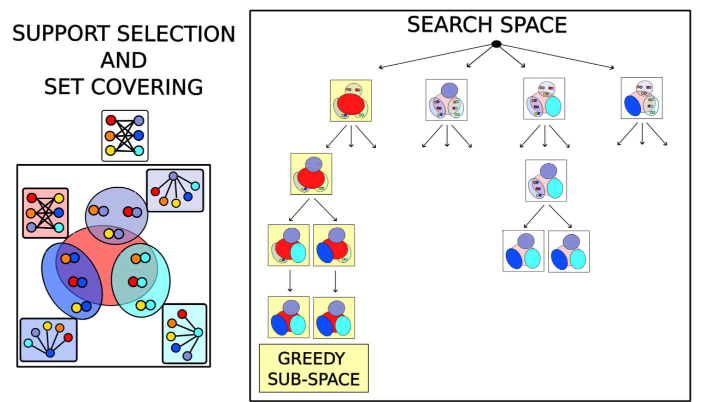

Fig. 4: Representation of the support selection problem as a set covering problem over the minterm pairs, and some sub-spaces of the search space. On the right, each node is a copy of the drawing on the left, where the set of the divisor added at the node is colored more brightly.

Definition III.1. A *functional cut* found from the analysis of *simulation signatures* is *optimal* if it yields the smallest resynthesis sub-network generalizing to the *don't-knows*.

There is no guarantee that the minimum-size solution to the set covering problem using *simulation signatures* yields the optimal *functional cut*. However, we claim that targeting the minimization of the *functional cut* size is an educated guess. Indeed, on average, smaller supports yield smaller resynthesis sizes, with higher chances of meeting the size constraints imposed by the MFFC. Furthermore, by adopting the *Minimum Description Length principle* (MDL), smaller supports should be chosen because they correspond to compact descriptions of the observations, which have higher likelihood of generalizing to unseen patterns [15].

Previous works on supervised learning with Boolean networks showed that, in the presence of input-output correlations, the Boolean network is more likely to generalize to the *don't-knows* [6], [16], [17]. A Boolean function is correlated (anti-correlated) to one of its variables when its IG is highly covered by the IG of the variable. This motivates investigating greedy support selection algorithms, where at each step we add a highly correlated variable to the support. However, single-variable information can be misleading since groups of variables could collectively contain more effective information. The next section addresses this issue.

IV. Support Selection

K-SUPPORT SELECTION

Given: 1) A target node x.
2) A set of candidate divisors $\mathcal{D} = \{x_i\}_{i=1}^{D}$.

Find a subset $\mathcal{C} \subseteq \mathcal{D}$, $|\mathcal{C}| \leq K$ satisfying Theorem 1.

A. Set Covering And Limitations Of Greedy Approaches

The *covering representation* of IGs was introduced to build *functional cuts*, one divisor at the time. *Greedy support selection* (GSS) is the simplest approach of this type: at each step,

TABLE I: ISCAS benchmarks: average percentage area variation after applying `mfs`, `mfs2`, and algorithm 1 with random, GSS, and enumeration support selection.

	`mfs`	`mfs2`	`rnd`	`gss`	`enu`
$\langle\delta_{100}\rangle[\%]$	-1.35	-1.46	-0.43	-1.94	-2.61

we select the divisor whose IG covers most of the remaining edges. Ties are broken at random [14]. GSS is fast but can fail to find some solutions. For instance, Figure 4 shows that greedy support selection cannot find a solution when imposing a size constraint of 3. Indeed, GSS can only explore a subspace of the search space, highlighted in yellow in the example of Figure 4. Consequently, it fails to find the existing solution with 3 divisors, for which non-greedy choices should be made.

A computationally expensive alternative to GSS is enumeration. We consider all possible combinations of K divisors, increasing K from 1. For each combination we verify if the divisors satisfy Theorem 1. After considering all possible combinations of size K, if no solution is found we increase K by one. Enumeration continues until finding one support satisfying Theorem 1, or until K reaches a threshold value for the support size. We use enumeration as a reference to test our algorithms and explore the properties of valid solutions.

We consider the ISCAS benchmarks, after technology-independent optimization and 4-LUT mapping (`resyn2rs; fraig; st; dch; if -a -C 12 -K 4`). For each of them, we run Algorithm 1 setting the number of support sampling attempts to 100, for each pivot node. In the enumeration case, the number of attempts denotes the number of permutations of the search spaces, while in the GSS case, they denote the ways of breaking ties at random. As a baseline, we also report the result of choosing the next divisor at random at each branching point of the search tree. We limit the support size K to 4 to compare the performances with `mfs` and `mfs2`. Table I reports the average results. For a fair comparison, in both `mfs` and `mfs2` we activate high-effort resubstitution, we set the number of windows to the number of nodes in the mapped network, and we allow 200 levels of depth increase for aggressive area optimization. The key observations are:

1) GSS can beat the state-of-the art in area optimization.
2) GSS misses optimization opportunities.

The result for enumeration gives an idea of what quality we can hope to achieve by refining the tree search exploration, but it does not scale to industrial designs. The challenge is to achieve the enumeration quality with the GSS scalability, i.e., to find more efficient ways to explore the search space.

B. Empirical distribution

Runtime constraints force us to limit the exploration to a small portion of the search space. Hence, it would be desirable to have a metric estimating which step to take at each branch of the search tree. This leads to the following ansatz:

Ansatz 1. *Let Υ^t_x be a t-covered IG, and $\mathcal{D} = \{x_i\}_{i=1}^D$ a set of divisors. Then, the normalized cost*

$$\mathcal{H}(x_i, \Upsilon^t_x) = \frac{||\Upsilon^t_x \succ \Upsilon_{x_i}||_E - \min_{x_j \in \mathcal{D}} ||\Upsilon^t_x \succ \Upsilon_{x_j}||_E}{||\Upsilon^t_x||_E - \min_{x_j \in \mathcal{D}} ||\Upsilon^t_x \succ \Upsilon_{x_j}||_E} \quad (9)$$

is a good metric to guide tree search exploration.

The normalized cost is a real number in the range $[0, 1]$. The divisors with the normalized cost at 0 are the ones covering most of the edges, i.e., the ones that GSS would choose. The divisors with 1 normalized cost are the ones not covering any edge. This metric can be effective only if it allows us to model the cases in which the enumeration succeeded.

We repeat the experiment in the previous section, but rather than committing the valid resubstitutions, we keep saving the *functional cuts* found by enumeration. For each *cut*, we artificially define a covering process that, at each step, takes the divisors with the smallest normalized cost from the *functional cut*, and covers the graph. If the greedy approach can find the solution, the normalized cost of the divisor chosen at each step is 0. Otherwise, the normalized cost is some value higher than 0. By plotting the frequency of the normalized costs, we obtain the empirical frequency associated with the probability that a divisor has a normalized cost, given that it is chosen as the next support divisor: $\hat{P}(\mathcal{H}(x_i, \Upsilon^t_x)|VALID)$. Figure 5 shows that most of the divisors in the valid support can be identified by GSS. However, the normalized cost is larger than 0 in many cases, in correspondence with supports missed by GSS.

Figure 5 also shows the empirical distribution of the normalized costs $\hat{P}(\mathcal{H}(x_i, \Upsilon^t_x))$. Using Bayes' Theorem, we estimate the posterior probability that a divisor should be included in the solution, given its normalized cost (Figure 5).

$$\hat{P}(VALID|\mathcal{H}(x_i, \Upsilon^t_x)) \propto \frac{\hat{P}(\mathcal{H}(x_i, \Upsilon^t_x)|VALID)}{\hat{P}(\mathcal{H}(x_i, \Upsilon^t_x))} \quad (10)$$

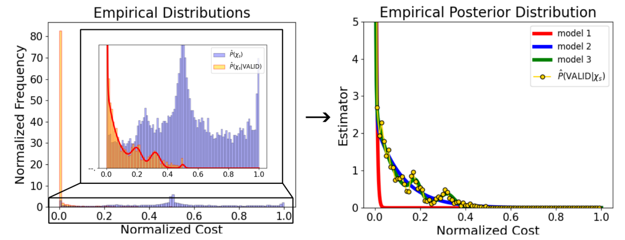

Fig. 5: Empirical distributions of the normalized cost, of the likelihood of a normalized cost given that the divisor belongs to a valid *functional cut*, and the estimator of the probability that a divisor is valid, given its normalized cost.

C. Models For The Transition Probability

We fit the posterior distribution with three models. The first model mirrors the ansatz of the paper [14].

$$p_1(VALID|\mathcal{H}) = \alpha_1 e^{-\beta_1 \mathcal{H}}$$

As Figure 5 shows, this model fits well with the main behavior but underestimates the probability that an *functional cut* might distribute the information more evenly among the divisors. The second model is an *hyperexponential* [18]:

$$p_2(VALID|\mathcal{H}) = \sum_{i=1}^{2} \alpha_i e^{-\beta_i \mathcal{H}}$$

This model takes into account that the distinguishing power of a valid support divisor either highly correlates with that of the target function or shares the information with other divisors.

TABLE II: EPFL and IWLS benchmarks: comparing GSS with `mfs`, `mfs2`, and the three models p_1, p_2 and p_3. We used 10 and 100 support samplings for each node. We report the average area improvement, and the average optimization time.

	`mfs`	`mfs2`	`gre`	p_1	p_2	p_3
$\langle\delta_{100}\rangle[\%]$	-0.92	-1.07	-1.17	-1.40	-1.78	-1.99
$\langle T_{100}\rangle[s]$	53.96	1.72	22.93	21.74	19.74	20.33
$\langle\delta_{10}\rangle[\%]$	-0.92	-1.07	-1.06	-1.26	-1.55	-1.53
$\langle T_{10}\rangle[s]$	53.96	1.72	5.30	5.72	5.84	6.08

The third model encodes the distribution of this information with peaks at specific values of the normalized cost.

$$p_3(VALID|\mathcal{H}) = \sum_{i=1}^{2} \alpha_i e^{-\beta_i \mathcal{H}} + \sum_{i=3}^{4} \alpha_i e^{-\frac{(\mathcal{H}-\mu_i)^2}{2\sigma_i^2}}$$

This model accounts for the fact that some values of the normalized cost are more likely than others (Figure 5).

Algorithm 2 shows how to use these probability distributions to guide the tree exploration for support selection. To avoid overfitting the parameters, we infer them on the ISCAS benchmarks and validate the models on the IWLS and EPFL benchmarks. These models for the posterior distributions are

Algorithm 2 `SUPPORT SELECTION<MODEL>`$(x, \mathcal{D}; K)$

1: **while** trials < maxtrials AND !solution found **do**
2: $\quad \mathcal{C} \leftarrow \varnothing \; t \leftarrow 0$
3: $\quad$ **while** $|\mathcal{C}| < K$ **do**
4: $\quad\quad$ evaluate the normalized costs $\mathcal{H}$
5: $\quad\quad$ sample a divisor $d \sim p_{\text{MODEL}}(VALID|\mathcal{H})$
6: $\quad\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{d\}$
7: $\quad\quad \Upsilon_x^{t+1} \leftarrow \Upsilon_x^t \succ \Upsilon_d$
8: $\quad\quad t \leftarrow t+1$
9: $\quad\quad$ **if** $\mathcal{C}$ satisfies Theorem 1 for the signatures **then**
10: $\quad\quad\quad$ return $\mathcal{C}$

not guaranteed to extend to other circuit representations and optimization objectives. This paper only shows that fitting them with some benchmarks and for some target objectives generalizes to other benchmarks when optimizing them for the same target objectives. Instead, the fact that the normalized cost is a good optimization metric is a general observation.

Table II reports the results of the three methods for the IWLS and EPFL benchmarks, and for different sampling numbers per node: 10 and 100. We observe a monotonic improvement when going from greedy to the second model. Instead, the third model performs better for high numbers of samplings but loses accuracy for 10 samplings. Again based on the MDL, we prioritize the second model due to its simplicity.

V. Look-Up Table Synthesis With Don't-Cares

Given a *functional cut*, it is possible to obtain the dependency function $f : \mathbb{B}^K \to \{0, 1, *\}$ by looking at the simulation signatures and filling in the entries of a k-LUT based on the patterns appearing at the leaves of the *cut*. If a pattern does not appear, we treat it as a *don't-care*. This section discusses how to decompose a K-LUT into a k-LUT network, with $k < K$ while taking *don't-cares* into account.

LUT SYNTHESIS WITH DON'T-CARES

Given: 1) A function $f : \mathbb{B}^K \to \{0, 1, *\}$.
2) A maximum fanin size is k

Find a k-LUT network synthesizing f.

A. Information Graph Transformations

We start by defining some IG transformations needed to understand the proposed decomposition.

Definition V.1. Let Υ_x^t be a t-covered IG, and A be the adjacency matrix of the IG. The *adjacency preserving transformations* $\{\Delta_m\}_{m=0}^{2^t-1}$ are the transformations

$$\Delta_m(\Upsilon_x^t) = \{\Delta_m(\mathcal{P}_i^{x,t}) \; i \in [0, 2^t)\}$$

$$\Delta_m(\mathcal{P}_i^{x,t}) = ({}^m\mathcal{P}_{i,0}^{x,t}|{}^m\mathcal{P}_{i,1}^{x,t}) = \begin{cases} (\mathcal{P}_{i,0}^{x,t}|\mathcal{P}_{i,1}^{x,t}) & \text{if } m_i = 1 \\ (\mathcal{P}_{i,1}^{x,t}|\mathcal{P}_{i,0}^{x,t}) & \text{if } m_i = 0 \end{cases}$$

m_i is the i-th bit of the binary representation of m.

These transformations are the IGs obtained by inverting the Boolean attributes of the partitions of the covered information graph, which are not detached from the IG. The adjacency matrix A is an invariant of this transformation, the number of partitions is upper-bounded by 2^t.

Definition V.2. Let Υ_x^t be a t-covered IG, and Δ_m an adjacency-preserving transformation. The *projection* operator Π is the operator collapsing the IG $\Delta_m(\Upsilon_x^t)$ into a Boolean function $\pi = \Pi(\Delta_m(\Upsilon_x^t))$. π_M is set by the Boolean attributes of the minterm M, and the *don't-care set* reads

$$\mathtt{DC}_\pi = \{\mathcal{P}_{i,a}^{x,t} \text{ s.t. } (\mathcal{P}_{i,a}^{x,t}|\varnothing) \in \Upsilon_f^t \vee (\varnothing|\mathcal{P}_{i,a}^{x,t}) \in \Upsilon_f^t \; i \in [0, 2^t)\}$$

The result of the covering process is twofold:

1) From one covered IG, many functions can be derived, all sharing the same IG.
2) $\mathtt{DC}_x \subseteq \mathtt{DC}_\pi$, i.e., the *don't-care set* of the projected function can be enlarged by the process.

$\mathtt{DC}_x \backslash \mathtt{DC}_\pi$ is the set of minterms detached during the process.

Definition V.3. Let $x : \mathbb{B}^K \to \{0, 1, *\}$ be a Boolean function, $\Upsilon_x^t \succ \cdots \succ \Upsilon_{x_t}$ be a covering process, Υ_x^t the resulting t-covered IG, and Δ_m an adjacency-preserving transformation. These transformations on Υ_x are *support-reducing* if the support size after the projection is smaller than n.

The term *reduced support* is used for the support of the function obtained after a support-reducing transformation.

B. Decomposition of Look-Up Tables

Remark 2. *Let $x : \mathbb{B}^n \to \{0, 1, *\}$ be a Boolean function, and $\mathcal{C} = \{x_i\}_{i=1}^n$ be a set of functions satisfying Theorem 1, and k be a desired fanin size. If it is true that:*

1) $n \leq 2k - 1$.
2) *There is a subset $\mathcal{C}_T \subset \mathcal{C}$, named top subset, $|\mathcal{C}_T| = k-1$ for which there is a support-reducing transformation.*
3) *The reduced support $\mathcal{C}_B \subset \mathcal{C}$ satisfies $|\mathcal{C}_B| \leq k$.*

Then, x can be decomposed using two k-input functions.

$$x = g(\mathcal{C}_T, h(\mathcal{C}_B)) \qquad g, h : \mathbb{B}^k \to \mathbb{B} \tag{11}$$

Proof. By definition of support-reducing transformation, the IG of the function $h(\mathcal{C}_B)$ covers the $(k-1)$-covered IG obtained with the covering process defined by the variables $\mathcal{C}_T$. Hence, the set $\mathcal{C}_F \cup \{h\}$ satisfies Theorem 1, implying the existence of a dependency function $g : \mathbb{B}^k \to \mathbb{B}$. □

This remark provides an operational definition of the key engine of our decomposition, which is reported in Algorithm 3.

Algorithm 3 `2_decompose`(x)

1: $\chi \leftarrow$`sort_by_coverage`$(x_1, \dots, x_n)$
2: **for all** $\binom{n}{k-1}$ support subsets $\mathcal{C}_T = \{x_i\}_{i=1}^{k-1} \subset \mathcal{C}$ **do**
3: $\quad \Upsilon_x^{k-1} \leftarrow \Upsilon_x \succ \Upsilon_{x_1} \succ \Upsilon_{x_2} \succ \cdots \succ \Upsilon_{x_{k-1}}$
4: $\quad m \leftarrow$`get_m_minimizing_ones()`
5: $\quad$`iter`$\leftarrow 0$
6: $\quad$**while** `iter` $\leq 1 +$ `effort` $\cdot 2^{k-2}$ **do**
7: $\quad\quad h \leftarrow \Pi(\Delta_m(\Upsilon_x^{K-1}))$
8: $\quad\quad \mathcal{C}_B \leftarrow$`get_support`$(h)$
9: $\quad\quad$**if** $|\mathcal{C}_B| \leq k$ **then**
10: $\quad\quad\quad$`return` $g(\mathcal{C}_F, h(\mathcal{C}_B))$;
11: $\quad\quad$`iter`$\leftarrow$mod(`iter`,2^{k-1}) $\quad m{+}{+}$

The first step sorts the divisors by coverage of Υ_x, so that in the enumeration of all possible top subsets, we will first consider the input variables leading to the highest IG coverage, i.e., correlating the most with the target function. This choice was empirically verified to speed up synthesis. Next, we consider all the possible combinations of $k-1$ divisors as the input of the top LUT, and we cover the IG with them. Using remark 2, a decomposition composed of 2 LUTs exists if and only if there is a k-input function covering the remaining edges of the IG. This function can be the projection of any adjacency-preserving transformation. Iterating through all the possible transformations would allow us to find a solution when present, but it requires an exponential number of trials in the worst case. To reduce the runtime effort, we introduce the parameter `effort`, in the range $[0, 1]$, where 0 corresponds to only one adjacency preserving transformation, and 1 considers all of them. To maximize the chances that a solution is found in the first few iterations, the first adjacency-preserving transformation we consider minimizes the number of ones in the projected function. This choice detects many decompositions with the lowest effort.

We attempt this decomposition every time the support size is $K \leq 2k-1$. When the decomposition fails, or when $K > 2k-1$, we try performing one step of top disjoint-support decomposition, and fallback to Shannon decomposition in case of failure. We use as the branching variable the one covering most edges in the IG, and we recursively apply the decomposition to the cofactors, after updating their care-sets with the branching variable.

VI. Experiments

A. The LUT Decomposition

Table III shows the success rate of our decomposition when trying to decompose a K-LUT in a sub-network composed of two 4-LUTs. The table considers *practical functions*, i.e., Boolean functions frequently appearing in modern hardware designs. The results refer to the case in Algorithm 3 when the effort is set to the maximum value. Except for the 6-variables case, our algorithm gets the best result faster than a SAT formulation. The failure in the 6-variables case is because, for runtime reasons, our support reduction algorithm is not exact but heuristic, and thus it misses some optimization opportunities. Due to the small optimality gap, our heuristic achieves better results than *disjoint support decomposition* and other methods used in `lutpack` [8].

TABLE III: Decomposition of the practical functions.

	5 vars (1233)		6 vars (7351)		7 vars (41071)	
	success	time[s]	success	time[s]	success	time[s]
DSD	55.31%	0.25	23.30%	1.81	16.52%	11.8
lpack	91.08%	0.34	45.65%	2.11	18.70%	12.72
IG44	**96.67%**	**0.45**	**63.77%**	**31.64**	**20.86%**	**740.27**
SAT	96.67%	1.47	64.22%	34.98	20.86%	766.11

TABLE IV: Quality and effort in decomposition.

effort	5 vars (1233)		6 vars (7351)		7 vars (41071)	
	success	time[s]	success	time[s]	success	time[s]
0%	95.94%	0.033	60.94%	0.52	18.90%	10.26
20%	96.59%	0.046	62.88%	1.64	19.32%	47.71
40%	96.67%	0.073	63.44%	2.77	19.77%	83.28
60%	96.67%	0.063	63.69%	3.97	20.26%	122.29

To ensure scalability we try to reduce the effort as much as possible. Table IV shows that the heuristics of Algorithm 3 allow us to identify most opportunities for 2-decomposition in a single attempt. By increasing effort, other optimizations are identified. However, already in this configuration, the success rate is higher than both DSD and `lutpack`.

Finally, we test our algorithm in the presence of *don't-cares*. We take all the practical functions, for which there is no 2-decomposition, and we randomly generate a care-set. We compare our *don't-cares*-aware heuristic IG44$_*$ with IG44$_0$, which sets to 0 all the *don't-cares*, and IG44$_p$, which assigns a random value to them. Table V shows that leveraging *don't-cares* yields superior resynthesis quality.

B. Resubstitution Statistics

In this experiment, we investigate the statistics of an optimization run. We consider the IWLS and EPFL benchmarks with less than 300000 AIG nodes, optimize them with one round of `resyn2rs`, and map them using the area-oriented LUT mapper with structural choices in ABC (`dch; if -a -K 4`). Table VI reports the optimization statistics, including the number of attempted substitutions, the sizes of the resynthesis sub-networks, the number of successes, and the average number of *don't-cares* exploited by the accepted substitutions.

C. High Effort Optimization of 4-LUT networks

This experiment analyzes the design-space exploration capabilities of our algorithm. The test cases are the EPFL combinational benchmarks. Since we are interested in dramatically reducing the area without relying on AIG optimization, we use only one run of the ABC script `resyn2rs`. Next, we perform SAT sweeping to remove combinationally equivalent nodes using command (`fraig`). Then, we map the networks into

TABLE V: Successful 2-decomposition with *don't-cares*.

	5 vars (123300)	6 vars (128350)	7 vars (260620)
IG44$_p$	28.39%	0.00%	0.00%
IG44$_0$	70.91%	4.94%	0.46%
IG44$_*$	**99.88%**	**92.60%**	**38.39%**

TABLE VI: Statistics on the EPFL and IWLS benchmarks for the two models p_1 and p_2 described in section IV.

		1-resub	2-resub	3-resub	4-resub	5-resub
gre	valid	37606	2441	0	4	1
	trial	40697	3712	1	7	1
	$\langle \lvert \mathrm{DC}_x \rvert \rangle$	10%	28%	0%	32%	25%
p_1	valid	33419	2919	1	9	1
	trial	36447	4298	1	17	2
	$\langle \lvert \mathrm{DC}_x \rvert \rangle$	11%	28%	25%	27%	13%
p_2	valid	36281	3923	5	9	3
	trial	39306	5466	13	25	5
	$\langle \lvert \mathrm{DC}_x \rvert \rangle$	13%	30%	31%	26%	17%

TABLE VII: Design space exploration for 4LUT-networks. The area is a, The depth is d, and the time is t.

design	a_{ABC}	a_{IRS}	d_{ABC}	d_{IRS}	t_{ABC}	t_{IRS}
div	4323	4268	2121	2118	1.00	177.00
log2	9752	9613	144	150	88.00	629.00
max	949	928	111	111	1.00	11.00
multiplier	7231	7116	130	130	1.00	396.00
sin	1849	1718	81	94	6.00	634.00
sqrt	6506	5246	2558	2118	3.00	616.00
square	5310	4897	123	123	7.00	611.00
arbiter	4139	4138	31	30	5.00	13.00
cavlc	261	244	9	10	3.00	33.00
ctrl	47	45	5	5	0.00	4.00
i2c	381	351	10	10	1.00	17.00
int2float	74	76	8	8	1.00	8.00
mem_ctrl	13065	11513	58	53	59.00	618.00
priority	183	187	30	30	1.00	5.00
router	55	56	12	12	1.00	4.00
voter	2475	2028	19	22	4.00	387.00
	−5.49%	**−9.50%**				

4-LUTs using the area-oriented LUT mapper with structural choices (`dch; if -a -K 4`). This is our baseline. We compare two simple design-space exploration flows. The first one runs `mfs2` and `lutpack` until convergence, or until a runtime limit of 10 minutes per benchmark is met. The second one replaces `mfs2` with our resubstitution engine. In both cases, we require no depth increase to avoid depth explosion during area optimization. Table VII shows the results for the EPFL benchmarks. The higher runtime of the second algorithm is partly due to the higher effort imposed, and partly because new optimizations are found at each attempt: the algorithm terminates when the runtime limit is reached.

D. Restructuring 6-LUT networks

We now examine the engine's behavior under stricter runtime constraints. We focus on 6-LUT networks, comparing the previously discussed flows with a 60-second runtime limit, allowing processes started before the end time to complete. The results are shown in Table VIII. Higher performances are achieved, thanks to superior restructuring capabilities. Combined results from this and the previous section demonstrate that our engine enables higher restructuring capabilities.

Future research could explore filtering strategies to speed up optimization and improve scalability. Nonetheless, as shown in the next section, this engine is significant for its restructuring capabilities, allowing us to identify optimization opportunities beyond the reach of state-of-the-art engines.

E. EPFL Best Results

The *EPFL Combinational Benchmark Suite* consists of 23 combinational circuits used to benchmark logic optimization

TABLE VIII: Design space exploration for 6LUT-networks. The area is a, The depth is d, and the time is t.

design	a_{ABC}	a_{IRS}	d_{ABC}	d_{IRS}	t_{ABC}	t_{IRS}
div	4118	4076	2028	1994	3.00	66.00
log2	7465	7344	141	142	68.00	276.00
max	731	730	88	87	0.00	10.00
multiplier	5682	5567	126	126	8.00	99.00
sin	1455	1399	70	70	6.00	62.00
sqrt	4911	4263	2252	1850	3.00	123.00
square	3985	3579	122	122	9.00	64.00
i2c	268	244	8	7	1.00	24.00
int2float	45	46	5	5	1.00	2.00
mem_ctrl	9395	9724	45	44	50.00	99.00
priority	143	141	28	28	1.00	7.00
router	39	40	9	9	1.00	1.00
voter	1885	1726	17	17	13.00	67.00
	−1.04%	**−3.20%**				

tools. The challenge includes deriving a 6-LUT network with the smallest LUT count. If a heuristic can optimize these benchmarks, then it introduces novel optimization capabilities in design space exploration, compared to the state-of-the-art. We apply a flow to the best LUT networks available in 2023. We iteratively apply our heuristic using `model 2` for support selection, and setting the support size to 6. In the case of failure, we increase the support size to 8. Table IX shows that this flow allows us to improve the best-known results for 11 out of 23 test cases from the competition. We report two different results. The first one stops optimization if, at the beginning of an optimization cycle more than 10 minutes have passed. The second one involves randomly varying the parameters indefinitely, until optimization is possible.

TABLE IX: Best area results for the EPFL benchmarks [19].

	OLD BEST		FLOW$_{\leq 600s}$		FLOW$_{\infty}$	
Design	6-LUTs	Depth	6-LUTs	Depth	6-LUTs	Depth
div	3090	1100	3090	1100	3085	1102
hyp	36836	4384	36814	4535	36491	4633
log2	6076	243	6043	252	6012	257
mul	4330	178	4316	211	4314	208
sin	1053	86	1025	112	1023	110
sqrt	2983	1382	2978	1180	2966	1185
square	2959	170	2939	206	2935	200
i2c	177	9	176	10	176	9
memctrl	1708	14	1696	14	1694	14
priority	93	30	92	30	92	30
voter	1180	28	1178	30	1175	29

VII. Conclusion

This paper presentes a new resubstitution algorithm for combinational logic represented as a LUT network. The algorithm has two novel features:

1) An algorithm with customizable runtime effort to select divisors during resubstitution.
2) A decomposition strategy for synthesizing LUTs into networks of smaller LUTs.

The resubstitution engine combining these heuristics enables new optimization opportunities, compared to state-of-the-art engines. Future works will investigate the implications of the non-local restructuring capabilities of our method on other network representations, including resubstitution on networks mapped to standard cell, beyond existing methods [20].

REFERENCES

[1] A. Mishchenko, R. Brayton, J.-H. R. Jiang, and S. Jang, "Scalable don't-care-based logic optimization and resynthesis," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 4, no. 4, pp. 1–23, 2011.

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

[3] S.-Y. Lee, H. Riener, A. Mishchenko, R. K. Brayton, and G. De Micheli, "A simulation-guided paradigm for logic synthesis and verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 8, pp. 2573–2586, 2021.

[4] S.-Y. Lee and G. De Micheli, "Heuristic logic resynthesis algorithms at the core of peephole optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[5] A. M. R. Brayton and A. Mishchenko, "Scalable logic synthesis using a simple circuit structure," in *Proc. IWLS*, vol. 6, 2006, pp. 15–22.

[6] A. Costamagna and G. De Micheli, "Accuracy recovery: A decomposition procedure for the synthesis of partially-specified boolean functions," *Integration*, vol. 89, pp. 248–260, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167926022001791

[7] J.-H. R. Jiang and R. K. Brayton, "Functional dependency for verification reduction," in *Computer Aided Verification: 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004. Proceedings 16*. Springer, 2004, pp. 268–280.

[8] A. Mishchenko, R. Brayton, and S. Chatterjee, "Boolean factoring and decomposition of logic networks," in *International Conference of Computer-Aided Design (ICCAD)*. IEEE, 2008, pp. 38–44.

[9] L. Józwiak, "Information relationships and measures: an analysis apparatus for efficient information system synthesis," in *EUROMICRO 97. Proceedings of the 23rd EUROMICRO Conference: New Frontiers of Information Technology (Cat. No. 97TB100167)*. IEEE, 1997, pp. 13–23.

[10] J. S. Zhang, S. Sinha, A. Mishchenko, R. K. Brayton, and M. Chrzanowska-Jeske, "Simulation and satisfiability in logic synthesis," *computing*, vol. 7, p. 14, 2005.

[11] S. Sinha, *SPFDs: A new approach to flexibility in logic synthesis*. University of California, Berkeley, 2002.

[12] R. K. Brayton, "Understanding SPFDs: A new method for specifying flexibility," in *Notes of International Workshop on Logic Synthesis (IWLS'97), May*, 1997.

[13] S. Muroga, Y. Kambayashi, H. C. Lai, and J. N. Culliney, "The transduction method-design of logic networks based on permissible functions," *IEEE Transactions on Computers*, vol. 38, no. 10, pp. 1404–1424, 1989.

[14] A. Costamagna, A. Mishchenko, S. Chatterjee, and G. De Micheli, "An enhanced resubstitution algorithm for area-oriented logic optimization," 2024, accepted at the *International Symposium On Circuits And Systems (ISCAS)*.

[15] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, no. 5, pp. 465–471, 1978. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0005109878900055

[16] S. Rai et al, "Logic synthesis meets machine learning: Trading exactness for generalization," in *2021 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2021, pp. 1026–1031.

[17] A. Oliveira and A. Sangiovanni-Vincentelli, "Learning complex boolean functions: Algorithms and applications," *Advances in Neural Information Processing Systems*, vol. 6, 1993.

[18] A. Feldmann and W. Whitt, "Fitting mixtures of exponentials to long-tail distributions to analyze network performance models," *Performance evaluation*, vol. 31, no. 3-4, pp. 245–279, 1998.

[19] L. Amarú, P. E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *Proc. IWLS*, 2015.

[20] L. Amarú, M. Soeken, P. Vuillod, J. Luo, A. Mishchenko, J. Olson, R. Brayton, and G. De Micheli, "Improvements to boolean resynthesis," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 755–760.