SLAP: A Supervised Learning Approach for Priority Cuts Technology Mapping

Walter Lau Neto[1], Matheus T. Moreira[2],Yingjie Li[1], Luca Amarù[3], Cunxi Yu[1], Pierre-Emmanuel Gaillardon[1]
[1]University of Utah, Salt Lake City, Utah, USA
[2]Chronos Tech, San Diego, California, USA
[3]Synopsys Inc., Design Group, Sunnyvale, California, USA

***Abstract*—Recently we have seen many works that leverage *Machine Learning* (ML) techniques in optimizing *Electronic Design Automation* (EDA) process. However, the uses of ML techniques are limited to learning forecasting models of existing EDA algorithms, instead of developing novel algorithms. In this work, we focus on designing an novel cut-based technology mapping algorithms assisted by ML techniques, which matches results of exhaustive cut exploration but preserving a small footprint of utilized cuts. The proposed approach has been demonstrated with a wide range of benchmarks with 24% reductions in number of cuts utilized compared to the state-of-the-art, while improving the circuit delay, and *Area-Delay-Product* (ADP), by average about 10%, 7%, respectively, with a 2% area penalty. Compared to the exhaustive approach, *i.e.*, considering all the cuts, we achieve similar or better results while saving over than 2× the number of considered cuts (runtime) on average. Finally, we provide a comprehensive explanation of heuristics learned by the ML model by feature ranking.**

***Index Terms*—Cut-pruning, Technology Mapping, ASIC design, Machine Learning**

I. Introduction

Recent years have seen increasing employment of *Machine Learning* (ML) techniques in EDA, which aims to reduce the manual efforts and boost the design closure process in modern tool flows [1]–[9]. For example, various of ML-based approaches have been developed to automatically optimizing tool-flow configurations for modern FPGA and ASIC design [1]–[4], fast and accurate design space exploration and Pareto-optimal analysis [5]–[8]. While these approaches have shown promising improvements compared to conventional EDA methodologies, ML techniques are limited to learning forecasting models of existing EDA algorithms and tools to tune the flow instead of producing novel algorithms. Moreover, although most of the ML models used previously demonstrate promising improvements, the *explainability* of the learned heuristics or models is rarely addressed.

Structural cut-based techniques play a major role in logic synthesis for **(i)** logic optimization in both academic and industrial environments [10]–[13], **(ii)** FPGA technology-mapping [14], [15] and **(iii)** ASIC technology mapping [16]. The main reason why cut-based algorithms have been widely used is because with carefully developed domain-specific heuristics, *e.g.*, cut sorting and filtering, such approaches can effectively achieve good *Quality-of-Results* (QoRs) with small run-time and memory usage. However, given that the number of cuts available *per node* is an exponential relation between the graph size and the number of the cut leaves [17], heuristics need to be applied to prune the search space in these algorithms [11], [14]–[16]. In technology mapping algorithms, these heuristics are mostly handcrafted through experimental analysis and based on domain-specific knowledge, being implemented based on static attributes of the cut/node, *e.g.*, number of leaves, cut level, number of nodes covered by a cut [11], [16], etc. In practice, the run-time efficiency can be estimated based on the number of cuts explored during the technology mapping, which leads to a trade-off between the number of explored cuts and QoR. However, a comprehensive and accurate understanding between the cut-pruning heuristics and QoRs remains unclear.

In this context, we revisit the sorting and filtering heuristics, showing their impact on ASIC designs' technology mapping. Alternatively, the technology mapping algorithm can be executed without heuristics, *i.e.*, exploring all possible cuts, which is believed to offer better QoRs with significant memory overhead. Specifically, we observe an average improvement in delay and area up to 20% across 14 designs while exploring all the cuts. However, it considers 56% more cuts on average, bringing a significant memory footprint. Note that the goal is to develop effective cut-based heuristics and achieve very similar results to the exhaustive approach while preserving a small number of utilized cuts. Then, we introduce the following research question: Can we learn a better cut filtering policy that achieves close to the exhaustive approach with a small footprint in the number of considered cuts?

To this end, we propose **SLAP**, a novel technology mapping framework, where the mapping algorithm is guided by ML-based cut sorting and filtering heuristics. To show our approach's implications, we benchmark it against ABC technology mapping, the state-of-the-art open-source ASIC mapper. Against the traditional heuristic implemented in *vanilla* ABC, we observe an average improvement in delay by 10% (up to 18%), with a small area penalty of 2% on average (but being improved up to 18%). Area-delay product (ADP) is improved on average by 7% (up to 22%), and the number of considered cuts is reduced by 24%. Compared to ABC considering all the cuts exhaustively, we still observe about 6% delay improvements in average and reduce the number of considered cuts by over than 2×, with a 3% area increase. While we limit our scope to ASIC tech-mapping, the findings of this work can be extended to benefit FPGA-mapping and cut-based technology-optimization [14], [15], as the nature of the problem is the same. The contributions of this paper are summarized as follows: **(i)** We present a methodology to explore the design-space in the cut selection for ASIC mapping, and illustrate it on a 128-bit Advanced Encryption Standard (AES) core, presenting the challenges and QoRs variations of cut-based heuristics. **(ii)** A ML-based approach is presented, which formulates the cut sorting heuristic as a multi-class classification problem. **(iii)** The proposed framework has been fully implemented with state-of-the-art open-source synthesis tool ABC and is publicly available [1]. **(iv)** Evaluations are conducted by comparing to state-of-the-art technology mapping algorithms implemented in ABC [16] with cut sorting and filtering heuristics, and mapping with exhaustive cut exploration over 14 arithmetic-heavy designs. **(v)** Finally, we demonstrate the *ML explainability* of the proposed approach. With comprehensive feature analysis, we have explained what heuristics have been newly learned by the proposed ML-based approach, in contrast to the state-of-the-art algorithms. Such explanations from

978-1-6654-3274-0/21/$31.00 ©2021 IEEE

[1]https://github.com/lnis-uofu/ML-Mapper

the proposed ML system feature an ultimate picture of implementing new cut-based algorithms in other future applications.

The remaining of this paper is organized as follows: Section II presents the necessary background and related works. Section III shows the problem definition, and formulates the *hypothesis* of this work. Section IV discusses the proposed approach to cast cut pruning into a multi-class classification problem. Section V presents the results achieved by employing our method with the ABC mapper. Finally, Section VI concludes this work.

II. Background and Related Works

A. Boolean Networks and Cut-based Technology Mapping

A Boolean network is a directed acyclic graph (DAG), denoted as $G = (N, E)$, where each node $n \in N$ has either no incoming (*fanin*) edges, or incoming edges. Nodes with no *fanin* are the *Primary inputs* (PIs), and nodes with incoming edges implement a Boolean function. An outgoing edge of a node n is the node's *fanout*. Let *inv* be a function from E to $\{0, 1\}$, an edge e is said to be inverted if $inv(e) = 1$. We refer to the number of fanouts in a node as $FO(n)$. The level of a node n is given by the longest structural path from any PI to the node, inclusive, and we refer to it as $lvl(n)$. Besides, we refer to the reverse level of a node n - $rLvl(n)$ - as being the longest structural path from the node to any PO. Common types of DAGs for logic manipulation include *And-Inverter Graphs* (AIGs) and *Majority-Inverter Graphs* (MIGs) [18].

Cuts: a cut c of a root node n is a pair (n, L), where L is the set of cut's leaves, such that any path from a PI to n passes through at least one leaf. A *trivial* cut of n is composed of n itself. Thus, PIs have only trivial cuts, and every logical node has at least one trivial cut. Non-trivial cuts, on the other hand, cover all the nodes found on the paths from the root to the leaves, including the root and excluding the leaves. The number of nodes covered by a cut c is said to be its volume, and we refer to it as $vol(c)$. A cut is *k*-feasible if it has up to *k* leaves. If a cut is contained, set-wise, in another cut for the same root node, the cut is dominated and usually discarded by a filtering procedure. Thus, starting from the trivial cuts in the PIs, and given an internal node n with fanins $a, b \in N$, the set of cuts for n, $\Phi(n)$, can be obtained through the set *Union* of the set of cuts for a and b as follows [15]:

$$\Phi(n) = \{\{n\}\} \cup \{u \cup v | u \in \Phi(a), v \in \Phi(b), |u \cup v| \leq k\} \quad (1)$$

Technology mapping: in our context, technology mapping consists of binding gates available in the standard cell library to cuts. Here, we briefly describe the steps performed by the ABC mapper [16], the state-of-the-art open-source ASIC mapper. The first step consists in computing *k*-feasible cuts for each node $n \in N$, where *k is usually* ≈ 5. Since each node may have in the worst case $O(n^k)$ cuts [17], pruning heuristics are applied. The cuts are first *sorted by their number of leaves*, and then *filtered by dominance*. Each node stores up to 250 cuts. Next, the cuts truth table is computed to be used for standard cell binding. Then, Boolean matching takes place. For each node, for each cut, one gate is assigned to cover this cut, if one exists. In the following, starting from the PIs, the best arrival time is updated for each node according to the found matches. Finally, a cover that minimizes delay is chosen. After that, global and exact area recovery iterations take place.

B. Related Works

Cut pruning heuristics are widely adopted for technology-mapping [14]–[16] and logic-optimization [11], [12]. Usually, these heuristics rely on a single cut attribute, which depends upon the application. Common metrics to sort cuts are the number of leaves, the number of nodes in the cover of the cut, or the number of levels in the cut [11]. Possible heuristics to cut-ranking and pruning targeting FGPA mapping are presented and discussed in [17]. Also, for FPGA mapping, in [14] the authors prioritize the cuts by the number of levels in the cut, then by the number of inputs, and then by the area estimation, in order to reduce the delay. Still, there is no comprehensive analysis of how these heuristics compare to other possible variations.

Works targeting ASIC tech-mapping focus on discussing the structural bias problem and do not thoroughly discuss the cut-ranking heuristics applied [16], [19]. In [19] the authors propose an approach to construct the subject graph by combining the result of different mappings heuristics, and do not discuss cut ranking. In [16], which presents the algorithm implemented in the state-of-the-art open-source mapper implemented in ABC [20], the authors extend the ideas presented in [19], where intermediate networks are generated by different optimization scripts and are seen as choices by the mapper. Furthermore, they introduce the concept of supergates, which consists of combining gates from the standard-cell library to build new single-output functions. The mapper sees the supergates as regular gates. These concepts make the matching process less susceptible to the subject graph structure. However, cut-sorting heuristics are not discussed, and sorting by the number of leaves is used in the implementation. Thus, in this work, we give a focus on how to have good cut-sorting and filtering heuristics for ASIC mapping. We show that current solutions let much room for improvement and propose an ML-based model that enables less-local choices. Our contributions are *orthogonal* to previous works and can be combined with previous techniques.

III. Exploring the Design Space of Cut-Based ASIC Tech-Mapping

To evaluate and show the impacts of cut-sorting and filtering heuristics, we propose modifying such a heuristic during the tech-mapping flow and keeping all the remaining steps untouched. To do so, we have relied on the state-of-the-art open-source ASIC mapper available in ABC. By default, after computing all the cuts for each node, ABC sorts on each node the list of cuts by their number of leaves and rules out dominated-cuts. Then, only the top 250 cuts for each node are exposed to the Boolean matching step. Experimentally, we evaluated sorting the cuts on each node by a different attributes other than the number of leaves. Still, there is not a single one that has consistently shown benefits to be adopted. That leads us to the following observation: depending upon each node's context, the sorting policy might be different. Also, different structural features from the cut should be considered simultaneously while evaluating good cuts.

Based on that, we explored the design-space by randomly shuffling the cut list on each node. Also, we have disabled the filtering of dominated cuts to expose all the possibilities for the mapper. We have then generated 10,000 maps with our random sampling strategy, using the open-source ASAP 7nm PDK [21]. The results of each mapping for an *Advanced Encryption Standard* (AES) core are presented in Fig. 1. The x axis presents the delay variation, whereas the y axis presents the area variation. Each data point presents the result from one mapping, and the black-start shows the result while using the default heuristic in ABC, *i.e.,* the number of leaves in the cut. Note that the initial subject graph has not gone through any optimization before mapping. The timing information is obtained with a *Static Timing Analysis* (STA). We can see that even though the ABC default result falls in the high-probability region, the filtering techniques have a huge impact on the QoR of the mapped netlist.

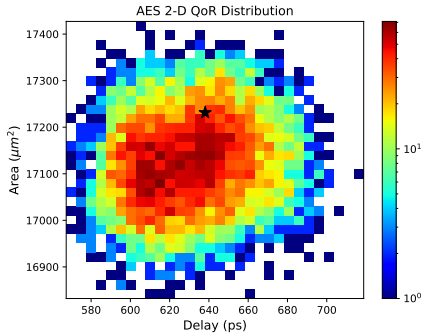

Fig. 1. 2-D QoR Distribution for an AES. Each data-point indicates the QoR of one mapping-solution by randomly shuffling the list of cuts for each node. The black star indicates the default ABC mapping result.

In this context, generating a good heuristic based on a single attribute for sorting good cuts for all the nodes is difficult, as it is local and does not give a good feeling about the context upon which the node is inserted. Also, since the *subject graph* is technology-independent, there is no precise information about cuts-timing and -area. In this context, we formulate the following ***hypothesis***: *The cut carries enough information about the graph structure that can be combined in a non-trivial way to enable less-local choices.*

IV. Approach – SLAP

To validate the hypothesis, we propose **SLAP** – a **S**upervised **L**earning **A**pproach for **P**riority-cuts technology mapping, an ML model to co-relate multiple structural features of cuts and serve as a replacement for pruning heuristics. The used features are chosen to capture a sense of the graph structure on which the cut appears. Using an ML model, we also use timing information during training, making it possible to learn filtering policies targeting the circuit delay optimization. We focus on delay as standard ABC mapper has as the goal minimizing delay as much as possible, and then recover the area. However, other metrics can equally be used. Finally, while the ML community has been applying ML to learn and replace core heuristics in other domains [22], works applying ML in logic synthesis have not yet used it to learn a new heuristic as a replacement of a core algorithm.

A. Node Embedding and Cut Embedding

We start by discussing how we embed nodes into tensors, given a circuit structurally represented as a DAG. First, to capture the node structural information, we consider features from the node itself and from its children. Here, we limit our scope to AIGs. Let e_0 be an outgoing edge from a node n, while e_1 and e_2 are two edges such that $e_1 = (c1, n)$ and $e_2 = (c2, n)$. The node embedding encompasses the features from the node itself and from its two children nodes, denoted as $c1$ and $c2$ in Table I, according to the notations defined in Section II.

TABLE I
Node Embedding Features According to the Notation Defined in Section II

Node Features	Child 1 Features	Child 2 Features
`inv(`e_0`)`	`inv(`e_1`)`	`inv(`e_2`)`
`lvl(n)`	`lvl(c1)`	`lvl(c2)`
`FO(n)`	`FO(c1)`	`FO(c2)`
`rLvl(n)`	-	-

These features present important information about the node structure and functionality (as the node Boolean function is a 2-input AND, polarities of the edges are the only information that might alter that). In this context, features are appended together into a $\mathbb{R}^{1\times 10}$ tensor. First comes the node features, followed by the features from $c1$ and $c2$. For instance, let us consider the node 13 in the AIG graph in Fig. 2, where the blue box by the node 13 represents its embedding, which is defined according to the features presented in Table I, being as follows: `nodeEmbedding`=$[[1, 3, 1, 0, 1, 2, 2, 1, 2, 1]]$. The tensors are stored in a hash-table, where the `uniqueNodeID` is used as the key for look-up. The node features are used to derive the cuts-embedding.

The first step to generate the cuts-embedding is to traverse the graph from the PIs to the POs and compute all the k-feasible cuts, for $k = 5$. We use 5-inputs cuts to allow a fair comparison with the mapper in ABC. Thus, we collect structural features for each cut. We have defined 9 features as follows: (i) a flag indicating if given a root node has an outgoing edge e such that $inv(e) = 1$; (ii) the number of leaves in the cut; (iii) the number of nodes covered by the cut, *i.e.,* $vol(c)$; (iv) the minimum level for the cut leaves; (v) the maximum level for the cut leaves; (vi) the sum of levels for all the leaves; (vii) the minimum fanout number for the cut leaves; (viii) the maximum fanout number for the cut leaves; (ix) the sum of fanout for all the leaves. These features are some of the cuts attributes that could be used as a sorting parameter, along with features that give a feeling in which context the cut appears, such as the leaves level and fanout. These features are combined with the cut root node embedding and the embedding for the nodes in the set $\{L\}$ of the cut leaves. The node embeddings are retrieved by looking-up the hash table that stores the unique tensors representing each node.

Therefore, the cut embedding has $\mathbb{R}^{i\times j}$ dimensions, where j is equal to 10 and matches the number of features representing a single node, and $i = 15$. For a cut with fewer than k (5) inputs, the invalid inputs are padded to the embedding matrix as a row of zeroes. This padding dissolves the effect of such nonexistent connections. Fig. 2 presents the node and cut embedding for a cut rooted at node 13, with leaves 10 and 12. In this case, children 3 to 5 embedding are padded with zeroes.

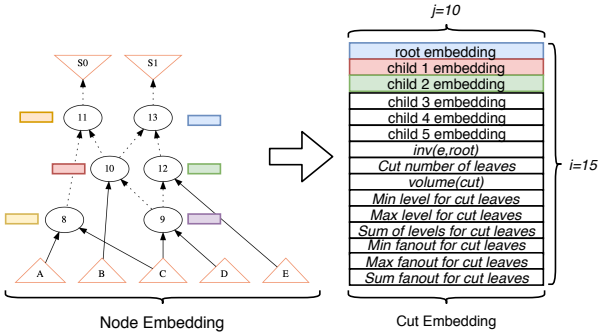

Fig. 2. The left-hand DAG represents an AIG, where each colored box represents the node embeddings. These node embeddings along with the cut features collected are used to derive the cut embedding. The first 5-rows represent the node embeddings from the root node and the cut leaves, followed by the cut features.

B. On the Training Approach and the Proposed Model

To generate the data for training, we propose to create distinct maps for a given circuit, with a wide range of QoR, where the different maps are generated by changing only the cut-sorting and filtering heuristics. Thus, we use our modified version of ABC, which randomly shuffles the cut list on each node. For each generated map, we hash the final QoR by its *area* and *delay*, to have a variety of mappings to learn from. Once the mapping is done, we dump the node embedding for each node and the feature of cuts used to deliver the

mapping. As our target is to map for performance, *i.e.,* low delay, each cut has as label the normalized circuit timing, given by the ABC *stime* command, which invokes the ABC STA tool. Thus, let S be the set of all the unique solutions possible for mapping one circuit. Let $m \in S$ be one valid mapping for the circuit, and C be the set of cuts used to implement m. Then, each $c \in C$ has as label the timing information of the implementation m normalized to the largest delay available at S. Therefore, each cut is a training data-point. Without losing generality, other metrics such as area, *Area Delay Product* (ADP), or even post-*Placement and Rounting* (PnR) could be used as a label. As we model this problem as a multi-class classification problem, the labels are in a range of 10 distinct QoR classes, where class 0 represents cuts that will minimize delay and class 9 cuts that degrade circuit performance.

Once the training set is properly generated and labeled, we train a *Convolutional Neural Network* (CNN) classifier that predicts the QoR class of cuts. Fig. 3 gives an overview of the model. The cut features matrix first goes through a Convolution layer (Conv), composed of 128 filters with dimensions 15×1 and a stride of 1. The idea behind using a convolutional layer is to correlate the node features with the cut features. The choice for the filters' dimensions is such that as the filters slide to the right with the stride of 1, a different feature of the nodes will be convolved with the features of the cut. The resultant output of the Conv layer is 128 tensors with dimensions 1×10 and passes through a flatten layer, generating a layer of 1280 neurons. This layer is fully-connected to a dense layer of 10 neurons, which go through a *softmax* function to predict the output class of a cut. The employed loss function was the sparse categorical cross-entropy due to our model's multiclass nature. The Adam optimizer was used for training. These cuts are then discarded or interfaced with ABC for Boolean matching, as explained in the next Section.

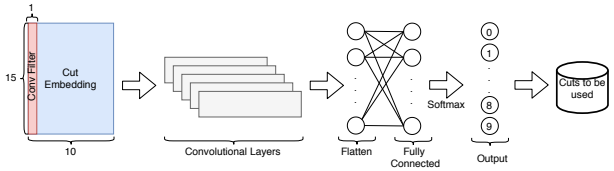

Fig. 3. Overview of the proposed model.

C. Framework Overview

Now that we have discussed the node and cut embedding, the training methodology, and the proposed model, we introduce how we put the pieces together to evaluate our method's impacts for cut selection. We interface our model with ABC to evaluate the benefits of the proposed technique for ASIC tech-mapping. To do so, we create two custom commands inside ABC. The first is called *prepare_map*, and is responsible for generating the internal AIG node embedding, as well as computing the *k*-cuts for each node and generating their initial features, which are used for the cut embedding and are represented by the last ten rows in Fig. 2. Once the features are generated, instead of applying the standard algorithm for sorting and filtering in ABC, we process the cuts through our framework. First, the nodes embedding are looked-up to finalize deriving the cuts embedding. Each cut (n, L), for each node $n \in N$, is stored as an numpy array, and goes through inference in a pre-trained CNN model. The model output is the QoR class the cut is predicted to belong. The lower the class, the better (lower) is the predicted delay to be. Here, we defined two threshold values to decide on whether a cut will be exposed to the mapper or not. Through experimentation, we divided the QoR space into three categories: the good cuts (classes 0 to 3), the average cuts (classes 4 to 6), and the bad cuts (classes greater than 6). Therefore, cuts classified to be within a QoR class ≤ 3, are considered the top options. If there is any cut predicted to be within this QoR range for a given node, use it as an option, and discard all the options greater than 3. On the other hand, if there is no cut to be within the top classes, but there are still cuts classified within the range ≥ 4 and ≤ 6, we give them as options the mapper. Otherwise, we do not pass any alternative to the mapper, and the only cut available during map is the node's trivial cut.

The model output is a list of cuts that should be considered by the mapper for each node. Thus, we created a new command called *read_cuts* inside ABC. This command takes the list of cuts to be considered by the mapper and only consider them during Boolean matching. Therefore, the only steps changed inside ABC are the cut sorting and filtering heuristics. On the other hand, Boolean matching, arrival timing update, and cover selection remain untouched. Fig. 4 presents the overall framework flow, which is interfaced with the state-of-the-art open-source logic synthesis environment.

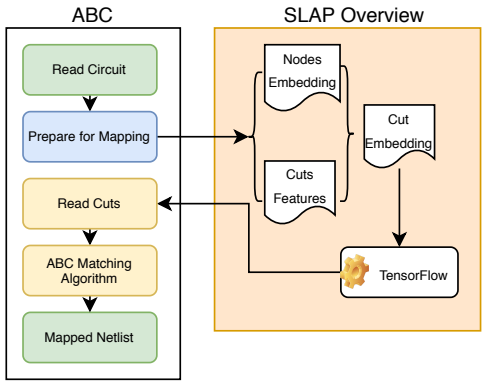

Fig. 4. SLAP framework overview.

V. Experimental Results and Discussions

This section presents and discusses the results of the proposed method. We start by discussing the proposed methodology and the benchmark selection. Then, we present the achieved accuracy for the classification model. To evaluate the implications of our approach, we interface it with ABC standard cell mapper and compare it with vanilla ABC and ABC *Unlimited*. We denote as ABC *Unlimited* a modified version of ABC that allows the mapper to try matching all possible cuts for each node and therefore takes out the cut sorting and filtering heuristics. For each scenario, *e.g.,* ABC vanilla, ABC unlimited, and our approach, we present results for *area*, *delay*, and the number of cuts exposed to the mapper, representing the memory footprint of each method. Besides, we discuss the *explainability* of our proposed framework SLAP, showing that relying on simple cut attributes is not sufficient, and many features must be combined to evaluate cuts. That can be used in future works to help designing new heuristics for cut-based techniques.

A. Methodology

To train and evaluate our approach, we focus on arithmetic designs. First, we note that arithmetic circuits are essential blocks to design complex *System on Chip* (SoCs). Also, while arithmetic circuits are widely available and can easily be generated, there are not many available control logic open-source designs. Indeed, for the EPFL benchmark suite, most of the control designs are super small. Generalizing to random logic is a step beyond of this work. Furthermore, while AIGs are efficient in handling random and control logic, they lack in efficiency to manipulate arithmetic designs [18].

TABLE II

RESULTS COMPARING OUR APPROACH AGAINST STANDARD ABC AND *Unlimited* ABC. WE HIGHLIGHT OUR METHOD CAN BEATS THE STANDARD ABC ALGORITHM IN DELAY FOR ALL THE TEST-CASES. COMPARED TO THE *Unlimited* ABC, WE IMPROVE 10 OUT OF 14 CASES. WE ALSO PRESENT RESULTS FOR AREA (μm^2), AND NUMBER OF CUTS CONSIDERED.

Circuit	ABC Original			ABC *Unlimited*			**SLAP**			Δ **SLAP**/ABC			Δ **SLAP**/*ABC Unlimited*		
	Area (μm^2)	Delay (ps)	Cuts Used	Area (μm^2)	Delay (ps)	Cuts Used	Area (μm^2)	Delay (ps)	Cuts Used	Area	Delay	Cuts	Area	Delay	Cuts
adder	898.13	3,770.65	10,118	1,009.40	3,404.62	18,228	1,031.33	**3,268.67**	1,3522	1.15	0.87	1.34	1.02	0.96	0.74
bar	2,680.39	1,114.90	42,760	2,680.39	1,114.90	43,784	3,083.23	**923.82**	10,705	1.15	0.83	0.25	1.15	0.83	0.24
c6288	3,000.45	1,265.88	95,519	3,014.68	1,228.32	197,022	3,023.54	**1,236.59**	111,609	1.01	0.98	1.17	1.00	1.01	0.57
max	2,312.27	3,809.03	39,727	2,049.60	3,980.21	41,478	2,292.44	**3,710.54**	22,521	0.99	0.97	0.57	1.12	0.93	0.54
rc256b	1,794.39	7,388.03	22,387	2,482.33	6,861.76	40,978	2,428.21	**6,807.02**	30,476	1.35	0.92	1.36	0.98	0.99	0.74
rc64b	450.70	1,844.59	5,491	601.16	1,688.38	10,066	602.33	**1,565.70**	6,397	1.34	0.85	1.16	1.00	0.93	0.64
sin	5,207.04	3,955.57	191,131	5,073.14	3,737.30	290,664	5,087.6	**3,584.79**	141,788	0.98	0.91	0.74	1.00	0.96	0.49
c7552	2,045.40	817.46	52,150	1,905.90	828.46	93,745	2,002.01	**800.09**	30,933	0.98	0.98	0.59	1.05	0.97	0.33
mul32-booth	6,802.44	1,837.68	185,406	5,623.21	1,743.56	355,106	5,597.55	**1,773.47**	147,043	0.82	0.97	0.79	1.00	1.02	0.41
mul64-booth	25,717.95	3,583.35	733,156	20,797.85	3,743.95	1,394,922	21,984.07	**3,280.47**	601,889	0.85	0.92	0.82	1.06	0.88	0.43
square	15,744.07	3,680.87	541,321	14,107.84	2,970.49	919,522	15,789.09	**3,023.01**	380,787	1.00	0.82	0.70	1.12	1.02	0.41
AES	17,321.27	638.09	264,380	16,994.21	632.07	317,827	16,489.63	**594.64**	145,532	0.95	0.93	0.55	0.97	0.94	0.46
64b_mult	25,458.31	4,649.10	833,565	24,168.27	4,144.31	1,484,104	24,021.07	**4,278.48**	892,650	0.94	0.92	1.07	0.99	1.03	0.60
Pico RISCV	12,190.05	1,782.08	181,410	12,010.19	2,126.47	197,069	12,148.52	**1,601.21**	103,174	1.00	0.90	0.57	1.01	0.75	0.52
Geomean	4,637.80	2,297.91	96,321.81	4,610.87	2,222.85	150,657.67	4,760.47	**2,090.01**	73,739.02	1.03	0.90	0.77	1.03	0.94	0.49
Improvements	1.0	1.0	1.0	0.99	0.96	1.56	1.02	**0.90**	0.76	-	-	-	-	-	-

Thus, it is essential to understand how AIGs can be more efficient in handling arithmetic designs. Finally, we focus on learning a cut sorting heuristics that minimize delay, as it is a hard optimization metric, and is also the primary goal of ABC mapper, allowing us to present a fair comparison. We chose two different 16 bits adders architectures to train our model: a ripple-carry and a carry-look-ahead.

B. Model Accuracy Results

We use $\approx 100,000$ data-points for training, collected from random maps from two distinct adders' architectures. For each adder, we generate around $\approx 50,000$ data-points. The runtime to generate these data-points and train the model for 50 epochs is ≤ 1 hour. As for the accuracy results, predicting the QoR class of a cut for 10 classes has an accuracy of around 34.01% on the validation set. However, note that we are not interested in predicting the *exact* QoR class one cut belongs to. Indeed, we are interested in considering the cuts classified to be in the top x QoR classes, wherein our case $x = 6$. Thus, when we convert the problem to a *binary-classifier*, which classifies the cuts that should be considered during tech-mapping and the cuts that should not, the achieved accuracy is 93.4% on the validation set. That supports our *hypothesis* that the cut has enough information about the graph structure that can be combined to enable timing- and graph-aware cut classification.

C. Technology Mapping QoR Results

We present the achieved QoR improvements compared to the state-of-the-art open-source technology-mapping available in ABC [16]. Results are compared against two versions of ABC: vanilla ABC and *Unlimited* ABC. The results presented in Table II are achieved by running the following flows:

a) ABC: read the AIG circuit; read the *ASAP 7nm* technology library; run technology mapping; run *stime* command to get *area* and *delay* information;

b) **SLAP:** read the AIG circuit; read the *ASAP 7nm* technology library; prepare tech-mapping; make inference; read list of cuts to be used for each node; derive the mapped netlist; run *stime* command to get *area* and *delay* information.

As for the circuit selection to evaluate our approach, we have chosen heavily arithmetic blocks from ISCAS'85 [23] benchmark suite, EPFL benchmark suite [24], and used the ABC *gen* command to generate the *ripple carry* architectures. We also validate our method over an AES core and a RISC-V architecture to see how it applies in more complex designs. In total, we evaluated our approach over 14 designs. As for the EPFL benchmark suite, we note that the biggest arithmetic blocks' results are not present as the data-frame generation with pandas takes too long. Indeed, for such huge designs, the pandas data frame generation is a bottleneck that we plan to address in a future work. Still, the inference time is very short. Also, as we discuss in this Section, our method has a smaller complexity than the default mapper, and a tight integration with ABC our any technology-mapping tool in C code will likely improve/have no impact in the runtime.

Results Discussion: Table II presents *area*, and *delay*, and number of cuts used for each case. We compare both our method and *Unlimited* ABC against vanilla ABC. We highlight in blue that we could improve delay, which is our goal, in all the circuits compared to vanilla ABC.

Comparison against vanilla ABC: Our work's main objective is to improve delay, which was achieved in all the cases compared to standard ABC. In average, our approach could improve delay by 10% (up to 18%). As for the area, we improved 8/14 blocks, achieving up to 18% improvement. On average, we have an area penalty of 2%. Finally, our method improves Area-Delay Product (ADP) in 12/14 designs, with an average improvement of 7% over all the cases (up to 22%). Also, we consider in average 24% fewer cuts than the standard ABC. Therefore, we not only achieve significant improvements in delay, but also reduce the problem complexity by significantly decreasing the memory footprint.

Comparison against Unlimited ABC: Compared to *Unlimited* ABC, we improve the delay in 10 out 14 cases. For the cases we do not improve the delay, SLAP produces very similar results to the exhaustive solutions. Delay is improved on average by 6% (up to 25%). The overall area increase is about 3%. ADP is improved in 8 out of 14 cases, with an average improvement of 3% (up to 24%). However, SLAP considers less than $2\times$ the number of cuts compared to this version in ABC, which significantly reduces the complexity of the approach and memory footprint. The reason why we can improve unlimited ABC is because its heuristic aims to reduce the arrival time in the considered node. As technology-mapping is analogous to the binate-covering problem, the choice of one cut implies other cuts must be chosen, and might eliminate future options that would contribute to the overall delay reduction. Finally, we recall the reader to the QoR distribution presented in Fig. 1. As it can be seen, our model achieves a delay of 594 ps, which falls in the low-end with respect to the delay in the distribution. Thus, we have achieved our goal, as one would not expect to learn the best data-point, but a good one in the design-space. The same trend can be observed for other designs. For the designs we do not improve compared to *Unlimited ABC*, that is because *Unlimited* ABC is an isolated data-point with

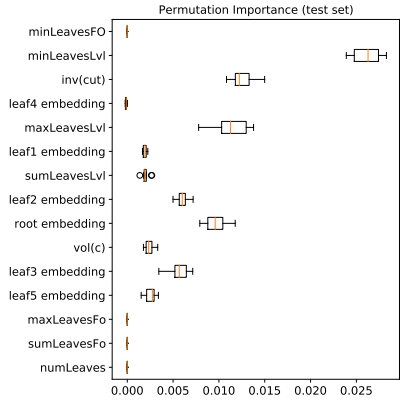

Fig. 5. Permutation test for feature importance.

very low delay for this kind of 2-D distribution.

D. Explainability of SLAP

Fig. 5 presents the feature importance evaluation results, which presents the accuracy degradation when each of the features is randomly permuted, showing the importance of each feature to select the best cuts. In other words, a higher value in Fig. 5 indicates a higher importance of a given feature. Permutation importance is a model agnostic metric, and we permute each feature for 10 rounds to evaluate how they affect our model's accuracy. Indeed, the results show that the model depends on different features, while the default feature used to sort the cuts in vanilla ABC ($numLeaves$) seems to do not have a huge impact on the quality of a cut. On the other hand, features that provided an understanding of the cut location in the graph and its surrounding structure seem to play a major role, such as the the minimum and maximum level for the leaves, and the embedding of the cut root node. Also, we observe that matching the polarity of cuts is important. That is because by default, ABC mapper takes inverters for free. However, the QoR results can be significantly affected by inverters in the post-mapping stage. That may lead to unnecessary inverters to adjust the polarity of nodes (impacting area) and more levels of logic (impacting delay). Finally, we observe that there is not a single feature that stands out and that many features need to be taken into account to enable effective cut pruning.

VI. Conclusions

In this paper, we first discuss the impact of such heuristics in the context of ASIC technology-mapping and made a *hypothesis* about cuts carrying enough information about the graph structure to enable sub-global choices. To validate our *hypothesis*, we proposed **SLAP** which formulate priority-cuts pruning as multi-class classification problem. SLAP quickly filters out candidate cuts that very likely lead to worse QoRs for technology mapping and explore the best possible cuts instead. We demonstrate substantial improvements in *delay* and ADP, while reducing the number of considered cuts by 24% compared to vanilla ABC mapping algorithm, and by over than 2× compared to *Unlimited ABC* that considers all the cuts. With comprehensive feature analysis, we have provided a thorough explanation of what heuristics have been newly learned by the proposed ML-based approach, in contrast to the state-of-the-art algorithms. Such explanations from the proposed ML system feature an ultimate picture of implementing new cut-base algorithms in other applications in the future. Future work will focus on exploring heuristics mining in end-to-end FPGA and ASIC flows.

Acknowledgment

This material is sponsored by DARPA, under agreement number FA8650-18-2-7849, and by NSF-CCF-2008144/2019336 funding agreements.

References

[1] W. L. Neto, M. T. Moreira, L. Amaru, C. Yu, and P.-E. Gaillardon, *Read Your Circuit: Leveraging Word Embedding to Guide Logic Optimization*, 2021, p. 530–535.

[2] C. Yu, H. Xiao, and G. D. Micheli, "Developing synthesis flows without human knowledge," in *In Proc. of the 55th DAC San Francisco, CA, USA, June 24-29*, 2018, pp. 50:1–50:6.

[3] A. Hosny, S. Hashemi, M. Shalan, and S. Reda, "Drills: Deep reinforcement learning for logic synthesis," *arXiv preprint arXiv:1911.04021*, 2019.

[4] C. Yu, "Flowtune: Practical multi-armed bandits in boolean optimization," in *In proc. of the IEEE ICCAD 2020*.

[5] Z. Wang, J. Chen, and B. C. Schafer, "Efficient and robust high-level synthesis design space exploration through offline micro-kernels pre-characterization," in *In proc. of the IEEE DATE*, 2020, pp. 145–150.

[6] S. Dai, G. Liu, and Z. Zhang, "A scalable approach to exact resource-constrained scheduling based on a joint sdc and sat formulation," in *In Proc. of the ACM/SIGDA FPGA*, 2018, pp. 137–146.

[7] B. C. Schafer and Z. Wang, "High-level synthesis design space exploration: Past, present and future," *In Proc. of the IEEE TCAD*, 2019.

[8] S. Liu, F. C. Lau, and B. C. Schafer, "Accelerating fpga prototyping through predictive model-based hls design space exploration," in *In proc. of the 56th ACM/IEEE DAC*, 2019, pp. 1–6.

[9] W. L. Neto, M. Austin, S. Temple, L. Amaru, X. Tang, and P. Gaillardon, "Lsoracle: a logic synthesis framework driven by artificial intelligence: Invited paper," in *IEEE/ACM ICCAD*, 2019, pp. 1–6.

[10] A. Mishchenko, S. Chatterjee, and R. Brayton, "Dag-aware aig rewriting: a fresh look at combinational logic synthesis," in *proc. 43rd ACM/IEEE DAC*, 2006, pp. 532–535.

[11] H. Riener, W. Haaswijk, A. Mishchenko, G. De Micheli, and M. Soeken, "On-the-fly and dag-aware: Rewriting boolean networks with exact synthesis," in *proc. DATE*, 2019, pp. 1649–1654.

[12] W. Yang, L. Wang, and A. Mishchenko, "Lazy man's logic synthesis," in *proc. IEEE/ACM ICCAD*, 2012, pp. 597–604.

[13] L. Amarú, P. Vuillod, J. Luo, and J. Olson, "Logic optimization and synthesis: Trends and directions in industry," in *proc. DATE*, 2017, pp. 1303–1305.

[14] A. Mishchenko, Sungmin Cho, Satrajit Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts," in *proc. IEEE/ACM ICCAD*, 2007, pp. 354–361.

[15] A. Mishchenko, S. Chatterjee, and R. K. Brayton, "Improvements to technology mapping for lut-based fpgas," *IEEE TCAD*, vol. 26, no. 2, pp. 240–253, 2007.

[16] S. Chatterjee, A. Mishchenko, R. K. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping," *IEEE TCAD*, vol. 25, no. 12, pp. 2894–2903, 2006.

[17] J. Cong, C. Wu, and Y. Ding, "Cut ranking and pruning: Enabling a general and efficient fpga mapping solution," in *In proc. of the ACM/SIGDA FPGA*, 1999, p. 29–35.

[18] L. Amaru, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A new paradigm for logic optimization," *IEEE TCAD*, vol. 35, no. 5, pp. 806–819, 2016.

[19] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, "Logic decomposition during technology mapping," *IEEE TCAD*, vol. 16, no. 8, pp. 813–834, 1997.

[20] B. L. Synthesis and V. Group, "Abc: A system for sequential synthesis and verification," 2018. [Online]. Available: http://www.eecs.berkeley.edu/ alanmi/abc/

[21] L. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics*, vol. 53, pp. 105–115, 7 2016.

[22] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *proc. of the 31st NIPS'17*. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 6351–6361.

[23] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the iscas-85 benchmarks: A case study in reverse engineering," *IEEE Design & Test*, vol. 16, no. 3, p. 72–80, Jul. 1999.

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