MIDAS: Mutual Information Driven Approximate Synthesis

Sina Boroumand
Imperial College London
London, UK
s.boroumand18@imperial.ac.uk

Christos-Savvas Bouganis
Imperial College London
London, UK
christos-savvas.bouganis@imperial.ac.uk

George A. Constantinides
Imperial College London
London, UK
g.constantinides@imperial.ac.uk

***Abstract*—Applications ranging from the Internet of Things (IoT) to high-performance computing demand energy-efficient hardware for processing and storage. Reducing computation accuracy has shown the potential to achieve high energy efficiency in hardware implementations. In recent years, several automatic approximate logic synthesis techniques have been proposed to build an approximate circuit systematically, trading off accuracy for hardware cost. In this paper, we propose a novel approximate logic synthesis technique to simplify circuits using mutual information by considering the input distribution. Our experimental result shows that our proposed methodology demonstrates improvements in terms of area, delay, and error compared to the state-of-the-art.**

***Index Terms*—logic synthesis, approximate computing, mutual information**

I. Introduction

Decades of research have focused on the design of integrated circuits for processing and storage of high-performance computer systems, from massive cloud data centers to mobile phones. However, with the exponential growth of data, designing smaller and faster, and thus energy-efficient circuits becomes ever more challenging. Fortunately, many applications, such as signal processing, computer vision, machine learning, etc., are intrinsically error-resilient which allows hardware designers to deliberately relax the accuracy of computations in order to achieve significant efficiency gains. Techniques exploiting this trade-off are known as *approximate computing*, and can be implemented from algorithm to circuit in a cross-layer manner [1].

Over the years, various approximate computing techniques have been proposed for the circuit layer. Circuit approximation, which is the focus of this paper, refers to the problem of building a circuit by trading off accuracy for circuit complexity. These techniques are commonly categorized into two classes: *manual* and *automatic* techniques. Significant progress has been made on manually crafting approximate arithmetic units; however, due to the scalability and re-usability, it is preferable to exploit automatic approximate techniques, known as *approximate logic synthesis* (ALS). These techniques synthesize a circuit with an arbitrary logical structure from a hardware description language, and they have become interesting for modern system designers since they can be employed in conjunction with an electronic design automation (EDA) flow to design efficient hardware accelerators.

Approximate logic synthesis techniques are categorized into two main classes [2]: Boolean rewriting and netlist approximation. Boolean rewriting techniques do not make use of any original gate-level representation of the circuit, but they act on the truth table of the Boolean function in order to simplify the circuit. Netlist approximation techniques transform the gate-level representation of the circuit by pruning and/or substituting gates and nets in order to simplify the circuit by introducing a user-defined acceptable amount of error. However, due to the design space explosion, heuristics must be taken to perform these techniques efficiently.

In this paper, we present a novel heuristic for netlist approximation that uses mutual information to simplify combinational circuits by trading off accuracy for reduced circuit complexity. We aim to approximate the behavior of the circuit by considering the information that the sub-circuits carry about the output. In other words, we try to answer the question of what small sub-circuits give sufficient information about the output and how to use them to simplify the circuit. Measures from information theory, such as entropy and mutual information, are thoroughly used to quantify the amount of information and uncertainty; thus, we suggest that utilizing these measures helps to explore the approximation design space effectively. We call our technique MIDAS: Mutual Information Driven Approximate Synthesis. We target combinational logical and arithmetic circuits, and tune the approximation to commonly-occurring portions of the input space by considering the input distribution. We summarize the contribution of this paper as follows:

- We propose a heuristic based on mutual information to steer the approximate design space exploration. We use a set of I/O examples to find sub-circuits that have low impact on the accuracy and substitute them with approximations, ensuring that the maximum error tolerance is not violated.
- We show that our proposed method scales well with the problem size in arithmetic and logical benchmarks. Our approach also demonstrated improvements in most cases when comparing against a state-of-the-art ALS technique. Also, evaluation on separate test data shows that our technique performs well with unseen inputs.

II. Related Works

For netlist level approximation, the simplest way to simplify a circuit is to truncate the least significant bits and set constant values for the truncated bits [3]. However, the error increases exponentially by truncating each bit. Schlachter *et al.* [4] proposed a gate-level pruning by iteratively removing gates of a netlist starting from the least significant bits until reaching the error threshold. One major disadvantage of this method is that the optimization solution would be stuck in a local optimum. Therefore, Scarabottolo *et al.* have proposed Circuit Carving [5], which exhaustively labels circuit nodes by exploring the circuit tree and estimates the maximum error based on removing sub-circuits. Liu *et al.* [6] proposed an algorithm that applies statistical measures on input distributions to guarantee an approximate transformation of a netlist. This approach splits the netlist into smaller sub-netlists, performs an independent approximation on each, and recombines them. SASIMI [7], identifies all possible signal pairs which are nearly identical by calculating the probability of their difference and substitutes one for another to simplify the circuit. However, the complexity of this method grows significantly with the size of the problem.

For Boolean rewriting approaches, Hashemi *et al.* developed a formal method, BLASYS [8], by simplifying the truth table of a circuit using a method inspired by Boolean Matrix Factorization to generate approximate circuits where the trade-off between circuit accuracy and complexity is controllable by the factorization degree. In addition, ALSRAC [9] uses approximate care sets and Monte Carlo simulation to replace sub-circuits with approximate sub-circuits by computing irredundant sums-of-products.

Our proposed approach is classified as a netlist transformation technique, where the heuristic works on the gate-level representation of the Boolean circuit and performs circuit simplification using mutual information. There exists some works that utilize mutual information to learn Boolean circuits [10], [11]; however, these techniques synthesize the circuit using a labeled dataset without any knowledge of the circuit structure.

III. Preliminaries

A. Circuit Terminology

A logical circuit f can be represented as a Directed Acyclic Graph (DAG) $f = G(N, E)$, where each node $n \in N$ represents a logical gate or a primary input/output, and $e(a, b) \in E$ is an edge connecting node a to b. The primary inputs (PIs) and the primary outputs (POs) are a subset of circuit's nodes and have zero fanin and fanouts, respectively.

A cone $c = (r_c, N_c, E_c)$, with root $r_c \in N$, $N_c \subseteq N$ nodes, and $E_c \subseteq E$ edges is a subgraph of f where all nodes in the cone are a transitive fanin (TFI) of node r_c. A fanout-free cone c with $L_c \subseteq N_c$ leaves is a subgraph of the circuit such that

$$\forall a, b \in N_c \backslash L_c \quad e(a, b) \in E \Rightarrow e(a, b) \in E_c, \tag{1}$$

where a leaf in a fanout-free cone is defined as a node that has zero fanin (primary input), or has at least one fanin from outside the cone, or has an edge to the outside of the cone. An example of a fanout-free cone is depicted in Fig. 1. In the rest of the paper, cone and fanout-free cone are used interchangeably.

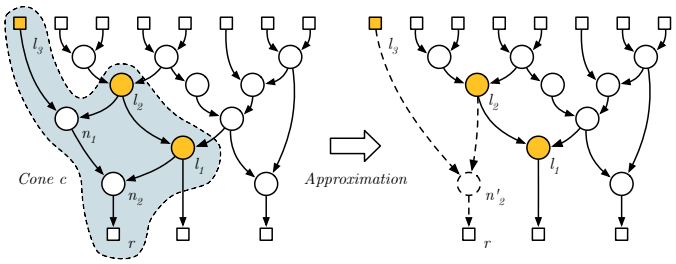


Fig. 1. An example shows a fanout-free cone c, where $N_c = \{n_1, n_2, l_1, l_2, l_3\}$, $L_c = \{l_1, l_2, l_3\}$, and $r_c = r$.

B. Entropy and Mutual Information

In information theory, the entropy of a random variable X with values distributed over $\mathcal{X}$, is the average level of information or "uncertainty" about the variable's possible outcomes and is given by

$$H(X) = -\sum_{x \in \mathcal{X}} P(x) \log P(x). \tag{2}$$

where $P(x)$ is the marginal probability of x.

For two random variables X and Y with values over the space $\mathcal{X} \times \mathcal{Y}$, the mutual information $I(X; Y)$ measures the average reduction in uncertainty about X that results from learning Y; or vice versa [12]. This can be expressed as follows:

$$I(X; Y) = H(X) - H(X|Y), \tag{3}$$

where $H(X)$ is the entropy of X and $H(X|Y)$ is the conditional entropy of X given Y. The conditional entropy measures the average uncertainty that remains about X when Y is known:

$$H(X|Y) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x, y) \log \left(\frac{P(x, y)}{P(y)} \right), \tag{4}$$

where $P(x, y)$ is the joint probability of x and y, and $P(y)$ is the marginal probability of y.

Normalized mutual information: Let $I(X; Y)$ denote the mutual information between random variables X and Y, the normalized mutual information to the entropy of X is denoted as $I_N(X; Y)$ [13] and is given by

$$I_N(X; Y) = \frac{I(X; Y)}{H(X)} \in [0, 1]. \tag{5}$$

IV. Problem Definition

Given a sequence of training input values $X = (x_i)$, where x_i denotes the i-th sample drawn from possible values $\mathcal{X} \subseteq \mathbb{B}^n$, and an exact Boolean circuit implementing function $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{Y} \subseteq \mathbb{B}^m$ are the corresponding output values

of the circuit, and a maximum error tolerance $E_{\max}$, the aim is to find an approximate circuit f' such that

$$\begin{aligned} \arg\min_{f'} \quad & \text{Area}(f') \\ \text{s.t.} \quad & \mathrm{E}\big(f'(X), f(X)\big) \leq E_{\max} \end{aligned} \tag{6}$$

where $\mathrm{E}(\cdot)$ is an arbitrary error metric between exact and approximate circuit.

V. Methodology

In this section, we discuss the methodology of MIDAS, which is an iterative algorithm to approximate combinational circuits. Considering an input circuit, MIDAS partitions the circuit into sub-circuits, finds a set of Pareto-efficient sub-circuits, approximates each sub-circuit independently, and finally commits the best approximation. This continues as long as approximation error is less than or equal to the maximum error constraint or algorithm cannot approximate any sub-circuit anymore.

A. Approximate Logic Synthesis from Examples

This section devotes to the proposed technique for approximate logic synthesis of a single output Boolean function from examples which is used in the MIDAS flow. We call this method *LSEm*, which is inspired by a technique proposed by Boroumand *et al.* [11].

The proposed algorithm is illustrated in Algorithm 1. The input to the algorithm is a set of *training examples* $X \rightarrow Y$, where $X \in \mathbb{B}^n$ are the input patterns and $Y \in \mathbb{B}$ are the corresponding binary labels, and *approximation factor* $\varepsilon \in [0, 1]$. The algorithm iteratively builds the Boolean function f by increasing the normalized mutual information between f and Y until reaching ε. Choosing ε controls the amount of approximation; for example $\varepsilon = 1$ associates with no approximation, while $\varepsilon = 0$ associates with highest possible approximation, *i.e.* connecting an input (or a constant value) directly to the output.

Algorithm 1: LSEm algorithm.

```
   Inputs     : Primary Inputs (PI), Training examples (X, Y)
   Output     : Approximate circuit f
   Parameters: Approximation factor ε
1  Add all primary inputs to the candidate node list C
2  f ← ∅
3  do
4      Simulate members of C with X
5      Find active node list (A)
6      F ← ∅, sup = 1
7      while F = ∅ and sup < sup_max do
8          sup++
9          for i in range(1, |A|) do
10             Find Boolean Function f_i(a_i, ..., a_{i+sup-1})
11             if f_i is found then Add f_i to F
12         end
13     end
14     Append F members to C
15     f ← argmax_{f'∈C} I_N(f'; Y)
16 while I_N(f; Y) < ε and |A| > 1
17 return Minimize(f)                          // Using ABC
```

The algorithm keeps two lists: *candidate node list* C, which is initially populated with primary inputs, and the sorted *active node list* $A = \{a_1, \ldots, a_n\}$. In (Line 5), the algorithm constructs $A \subseteq C$, such that $|A|$ is minimized and

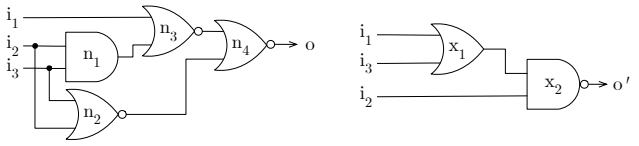
$$\begin{aligned} \arg\max_{A \subseteq C} \quad & I_N(A; Y) \\ \text{s.t.} \quad & I_N(A; Y) \geq \varepsilon, \end{aligned} \tag{7}$$

where A is sorted based on the increasing value of mutual information; the first node in the list has the highest mutual information to the output, and the second node in the list has the highest mutual information if combined with the first node and so on.

In the next step, the algorithm tries to find $|A| - \texttt{sup} - 1$ Boolean functions F, where $\texttt{sup}$ is initialized with 2 and denotes the number inputs to each Boolean function. For each $a_i \in A$, we find the functionality of a single output Boolean function $f_i(S)$, $S = \{a_i, a_{i+1}, \ldots, a_{i+\texttt{sup}-1}\}$ and $|S| = \texttt{sup}$, by enumerating all possible Boolean functions $(2^{2^{\texttt{sup}}})$, such that $I_N\big(A[1:i]; Y\big) < I_N\big(A[1:i-1] \cup f_i; Y\big)$ and mutual information $I_N\big(A[1 : i-1] \cup f_i; Y\big)$ is maximized. For scalability, we use the same technique that has been utilized in [11] to learn the functionality of Boolean functions where $\texttt{sup} \geq 4$.

In case of *success*, all successful $f_i \in F$ are added to the candidate list C, meaning there exists at least one Boolean function that increases the mutual information. If no Boolean function is found, the algorithm tries again by increasing the support size ($\texttt{sup}$) until reaching $\texttt{sup}_{\texttt{max}}$. The loop repeats while $|A| > 1$ and no member of C has a normalized mutual information bigger equal than ε with the output. In case of breaking the loop, the member with the highest normalized mutual information to the output is the solution of the algorithm. For further logic minimization, the approximate circuit is given to an exact logic minimizer (Line 17).

Example 1. *Given an exact circuit, shown in Fig. 2(a), a uniform distributed training example set, and approximation factor $\varepsilon = 0.7$, the algorithm starts by initializing the candidate node list C with primary inputs. In the first iteration, $A = \{i_2, i_1, i_3\}$ with $I_N(i_2; Y) = 0.19$, $I_N(i_2, i_1; Y) = 0.49$, and $I_N(i_2, i_1, i_3; Y) = 1.0$. The algorithm finds $x_1 = \overline{i_2 i_1}$, where $I_N(x_1; y) = 0.31 > 0.19$ and $x_2 = \overline{i_1 i_3}$, where $I_N(i_1, x_2; y) = 0.65 > 0.49$, and adds them to the candidate node list. In the second iteration, $A = \{x_1, i_3, i_1\}$, meaning that x_1 has the highest mutual information to the output. The algorithm finds $x_3 = x_1 \overline{i_3}$, where $I_N(x_1; Y) < I_N(x_3; Y)$. In the last iteration, $A = \{x_3, i_1\}$, and the algorithm finds the final Boolean function $x_4 = x_3 \overline{i_1}$. The output circuit is $f = \overline{(\overline{i_1 i_2}\ \overline{i_3})} i_1$ which simplifies to $\overline{(i_2 + i_3)} i_1$, shown in Fig. 2(b). The approximate circuit only produces one incorrect output (when $i_1 = 0, i_2 = 1, i_3 = 1$) out of all possible input combinations.*



(a) Original Circuit (b) Approximate Circuit

Fig. 2. Example demonstrating an approximate logic synthesis generated by LSEm under approximation factor $\varepsilon = 0.7$ versus the original circuit.

B. Approximate Space Exploration

The flow of the proposed MIDAS algorithm is shown in Algorithm 2. The inputs to the algorithm are circuit f with $|N|$ nodes, training samples X, error metric E, and maximum error tolerance E_{max}. Parameter d is the maximum cone depth, and the output is circuit f' approximating f.

In the first step, the algorithm finds list C, which consists of all possible fanout-free cones of the circuit with a maximum depth of d. The maximum depth of a cone defines as the maximum number of nodes in the critical path of a cone. In the next step, the algorithm calculates the sensitivity and area of each cone in C. Each fanout-free cone $c = (r_c, L_c, N_c, E_c)$ is associated with an area, which can be interpreted as the size of the cone, denoted as $A(c) = \text{Area}(N_c \backslash L_c)$, *i.e.* the maximum possible reduction of area of a circuit if such cone is approximated. Sensitivity of a cone with root r_c defines as the average amount of error, negating the output of r_c in the graph of the circuit, will introduce to the output. Sensitivity of cone c denotes as $S(c)$, and can be easily calculated by simulating the aforementioned transformation of the circuit over the set of training examples.

After finding C, where $|C| = |N|$, we measure the area and the sensitivity for each member of C. The algorithm then selects the Pareto-efficient cones $C' \subseteq C$, by minimizing the cost function below (Line 6):

$$\min([A^{-1}(c), S(c)]), c \in C.$$

These Pareto-efficient cones exhibits the best possible choices for approximation to reduce highest area while introducing lowest possible error to the output. Therefore, each selected cone c is approximated using the method introduced in Section V-A, and is denoted as c'. The aim is to build a Boolean circuit that approximates the cone's root using information available in the cone's leaves. and outputs a single bit (Line 10). This information is derived from the distribution of r_c and L_c which is captured by a simulation over training samples. The LSEm algorithm requires a value for *approximation factor* ε, which can be selected based on the sensitivity of each cone by grouping sensitivity values into multiple bins to assign ε. There is a linear correlation between sensitivity and approximation factor, meaning that the higher the sensitivity, the higher ε is required, and vice versa.

For each Pareto-efficient cone $c \in C'$, the algorithm generates an independent instance of the current approximate circuit, denoted as f'', and substitutes c with its approximate version c', and subsequently calculates the error between f and f''. If area has been reduced and error is less equal E_{max}, f'' is a candidate and appended to the list F. In the case of not satisfying the error condition, the algorithm tries this process again by increasing ε by $\alpha > 1$, *i.e.* decreasing the amount of approximation, or decreasing error while increasing area. This continues as long as the error is larger than E_{max}, or $\varepsilon \leq 1$, meaning that no more area can be reduced. In this case, the algorithm removes the corresponding cone from C and continues (Line 15).

In Lines 18–21, the algorithm finds the best approximation candidate that introduces the minimum error among all approximation candidates (F). After each successful substitution, we need to update the cone list C and re-simulate the affected nodes and re-calculate their sensitivity (Line 20). Given F is an empty set, none of the Pareto-efficient points are beneficial, meaning that no instance satisfies neither the area nor the error constraint. In this case, algorithm removes all current points from C and tries the next set of Pareto-efficient points. This loop continues finding approximate substitutions while $C \neq \emptyset$; otherwise, the algorithm returns the best current instance of approximate circuit f' as the final solution.

C. Complexity of MIDAS

The bottleneck of the algorithm is in Line 10, where mutual information is calculated. Using a hash table implementation, the calculation run-time of mutual information grows linearly to the size of the training example set $|X|$. In each iteration of the algorithm, we aim to remove at least one node from N, and find at most $|C'| = |N|$ Pareto-efficient cones, so the complexity of the algorithm in worst-case is $O(|N|^2|X|)$.

Algorithm 2: MIDAS flow.

Inputs : Circuit f, maximum error tolerance E_{max}, Simulation inputs $X = (x_i)$
Output : Approximate circuit f'
Parameters: Maximum cone depth d

1 $f' \leftarrow f$, Simulate nodes in circuit f' with X
2 Sensitivity check nodes in f'
3 Find all cones C with maximum depth d
4 Sort C based on sensitivity
5 **while** $C \neq \emptyset$ **do**
6 Find Pareto-efficient cones $C' \subseteq C$
7 **for** each $c \in C'$ **do**
8 Calculate ε based on sensitivity of c
9 **while** TRUE **do**
10 $c' \leftarrow \text{LSEm}(r_c, L_c, \varepsilon)$
11 $f'' \leftarrow f'$, and substitute c with c' in f''
12 **if** $E(f, f'') \leq E_{\text{max}}$ **then**
13 Append f'' to F, and **break**
14 **else** $\varepsilon \leftarrow \varepsilon \times \alpha$
15 **if** $\varepsilon > 1$ **then** Remove c from C, and **break**
16 **end**
17 **end**
18 **if** $F \neq \emptyset$ **then**
19 $f' \leftarrow$ Best candidate in F with the lowest error
20 Update C and sensitivity of affected nodes of f'
21 **else** Remove all C' from C
22 **end**
23 **return** f'

VI. Experimental Results

A. Experiment Setup

To evaluate the performance of our proposed method, we assess MIDAS on a conventional set of logic synthesis benchmarks from the combinational multi-level circuits labelled as 'logic' in ISCAS'85 [14], and a set of arithmetic circuit benchmarks, including adders (`rca32`, `cla32`, and `ksa32`) and multipliers (`wal8` and `mtp8`). Table I illustrates the characteristics of both sets, including the number of input/outputs, area, and delay of the original exact designs. For our experiments, the parameters of MIDAS are chosen as $d = 5$ and $\alpha = 0.2$. For each benchmark, a set of training samples and test samples are generated using a uniformly distributed random generator, where the training samples are used for logic simplification process of MIDAS and test samples are used to evaluate the accuracy performance. The resource utilization reported in this paper is estimated based on a network of LUTs, where the area is proportional to the number of LUTs and the delay is proportional to the length of the longest LUT-path. To make a fair comparison, all output circuits are re-synthesized and technology mapped into 2-input LUTs using the ABC logic synthesis tool [15].

B. Error Metrics

We use *error rate* (ER) and *mean relative error distance* (MRED) as the error metrics [16] to quantify the accuracy performance of the approximate circuits. For each benchmark, we evaluate the error on a different input sample set, *i.e.* test sample set, than training sample set which is used during approximate synthesis to demonstrate the accuracy performance of the algorithm for generalizing beyond seen examples. Comparing the original exact circuit and the approximate circuit, the ER is the fraction of incorrect outputs out of the total number of input samples, and MRED is the average relative absolute error distance between approximate output and exact output across samples over total number of samples.

C. Training Set Size Effect

The effectiveness of the proposed algorithm is dependent on using a representative training data set. Indeed, for each distinct training set, the generated circuit may be different; therefore, to demonstrate the effect of training sample size on the test error and area, we ran multiple experiments for different training set sizes and a fixed test set (1M uniformly distributed random test samples), and we report the results in Fig. 3 for one ISCAS benchmark (`c1908`). The x-axis is relative to the size of the truth table and scaled logarithmic. For each point, we calculate the 95% confidence interval for the mean, shown as error bars. For both test error and area, the confidence interval becomes smaller as we increase the size of training examples, and the curve approaches a horizontal asymptote. The first figure demonstrates the generalization aspect of our proposed approach, which confirms that as we increase the size of the training set, the test error approaches the training error and error bar size decreases. For the second figure, as we increase the training size, the approximated circuit becomes more area-expensive to target the same maximum training error threshold (ER $\leq 0.01\%$). This figure also shows the variance of area decreases as we increase the training size.

TABLE I
Characteristics of benchmarks.

Suite	Benchmark	I/O	Area	Delay
Logical	`c880`	60/26	1048	26
	`c1908`	33/25	900	21
	`c7552`	207/108	4088	40
	`c3540`	50/22	3348	32
	`c2670`	233/140	1796	17
	`c5315`	178/123	4336	23
Arithmetic	`rca32`	64/33	924	30
	`cla32`	64/33	1408	30
	`wal8`	16/16	1316	28
	`ksa32`	64/33	1448	19
	`mtp8`	16/16	1484	33

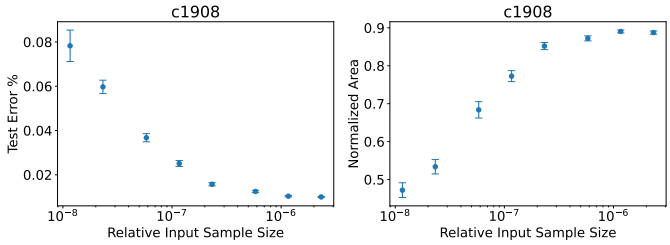


Fig. 3. The effect of training size on the test error and area for a fixed maximum error threshold ER $\leq 0.01\%$. The number of experiments for each point is 100, and a uniform sampling is used for training and test set samples.

D. Approximate Logic Synthesis Results

In this section, we are presenting the experimental results for approximate logic synthesis of circuits under both ER and MRED error metrics for logical and arithmetic circuits, respectively. We compared our work with ALSRAC [9] which has shown improvement over recent and former state-of-the-art approximate logic synthesis techniques.

1) Approximation under ER constraint: In this experiment, we approximated the selected logical benchmarks from ISCAS benchmark suite using MIDAS and ALSRAC under ER constraint. Table II demonstrates the normalized area/delay of approximate circuits to the area/delay of the original circuits, under various maximum error thresholds ER $= \{0.1\%, 0.5\%, 1\%, 3\%, 5\%\}$. As we can see, in most cases and under a same error threshold, MIDAS is capable of generating circuits that have lower area comparing to ALSRAC. In addition, for lower error rates (*e.g.* $\leq 1\%$), the gap between MIDAS and ALSRAC is larger. The reason is that MIDAS uses mutual information and a large number of training samples to perform simplification, whereas ALSRAC decides upon relatively small random input patterns which definitely increases the probability of getting stuck in local minima when error threshold is low. In some cases, e.g.

TABLE II
COMPARISON OF MIDAS AND ALSARC UNDER VARIOUS ER CONSTRAINTS. AREA AND DELAY ARE NORMALIZED TO THE EXACT CIRCUITS.

Benchmark	MIDAS										ALSARC									
	ER=0.10%		ER=0.50%		ER=1%		ER=3%		ER=5%		ER=0.10%		ER=0.50%		ER=1%		ER=3%		ER=5%	
	Area	Delay	Area	Delay	Area	Delay	Area	Delay	Area	Delay	Area	Delay	Area	Delay	Area	Delay	Area	Delay	Area	Delay
c880	93.51%	100.00%	93.51%	100.00%	93.51%	100.00%	88.93%	88.46%	87.40%	92.31%	91.22%	100.00%	88.93%	100.00%	85.88%	100.00%	84.35%	100.00%	83.21%	96.15%
c1908	95.56%	100.00%	91.56%	100.00%	88.00%	95.24%	61.33%	90.48%	30.67%	33.33%	100.00%	100.00%	100.00%	100.00%	95.11%	100.00%	55.56%	71.43%	36.00%	33.33%
c7552	86.89%	97.50%	84.54%	95.00%	84.05%	97.50%	84.05%	97.50%	83.37%	95.00%	98.78%	95.00%	98.78%	95.00%	97.90%	95.00%	95.25%	95.00%	95.16%	95.00%
c3540	94.62%	93.75%	93.55%	96.88%	91.28%	96.88%	82.68%	96.88%	69.65%	96.88%	100.00%	100.00%	100.00%	100.00%	98.69%	100.00%	90.44%	96.88%	81.60%	96.88%
c2670	74.16%	117.65%	73.50%	117.65%	73.05%	100.00%	72.16%	100.00%	70.82%	88.24%	100.00%	100.00%	76.50%	100.00%	74.05%	94.12%	71.16%	88.24%	68.37%	76.47%
c5315	99.45%	91.30%	99.54%	91.30%	98.99%	91.30%	98.52%	91.30%	98.43%	86.96%	101.15%	91.30%	101.15%	91.30%	100.78%	91.30%	98.02%	91.30%	98.48%	91.30%
Average	90.70%	100.03%	89.37%	100.14%	88.15%	96.82%	81.28%	94.10%	73.39%	82.12%	98.72%	97.72%	94.25%	97.72%	92.07%	96.74%	82.46%	90.47%	77.14%	81.52%

TABLE III
COMPARISON BETWEEN MIDAS AND ALSARC FOR ARITHMETIC BENCHMARKS UNDER MRED METRIC.

Benchmark	Normalized Area		Normalized Delay	
	MIDAS	ALSARC	MIDAS	ALSARC
rca32	36.80%	26.84%	70.00%	46.67%
cla32	29.55%	18.75%	66.67%	40.00%
wal8	95.44%	94.53%	96.43%	100.00%
ksa32	21.82%	17.68%	94.74%	68.42%
mtp8	97.04%	97.84%	100.00%	96.97%
Average	56.13%	51.13%	85.57%	70.41%

c2670, the circuit generated by MIDAS has higher delay compared to the exact circuit, which is due to ABC logic optimization that prioritizes area reduction over delay.

2) Approximation under MRED constraint: For arithmetic circuits, we evaluate our approach using MRED, which takes the weight of the output bits into the account. Table III illustrates the results for the maximum error constraint $E_{\max} = 0.01\%$. This table shows that MIDAS has been successfully simplified the circuits under MRED; however, in comparison to ALSRAC, MIDAS performed almost similarly for multipliers, wal8 and mtp8, and slightly less effective for the other benchmarks. The reason is that MIDAS performs better where the circuit consists of larger fanout-free cones, and due to the structure of arithmetic circuits, there is a less chance of having large fanout-free cones to perform the simplification process effectively. To support this argument, we count the average cone size of all benchmarks; comparing with logical benchmarks, on average, arithmetic benchmarks have lower number of fanout-free cones ($\approx 0.5\times$).

VII. CONCLUSION

This work proposes an approximate logic synthesis methodology that uses mutual information to steer the exploration of the circuit approximation search space. The evidence from this study implies that for both logical and arithmetic benchmarks and under ER and MRED metrics, MIDAS is able to synthesize approximate circuits effectively. In comparison to the baseline, MIDAS performed significantly better under ER metric, specifically for lower maximum error thresholds. However, under MRED, it performs less effective comparing to the baseline which is due to the intrinsic structure of arithmetic circuits. Future work includes optimizing the algorithm for circuits that has a limited number of large fanout-free cones and using word-level metrics to approximate arithmetic circuit.

REFERENCES

[1] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, pp. 1–33, 2016.

[2] I. Scarabottolo, G. Ansaloni, G. A. Constantinides, L. Pozzi, and S. Reda, "Approximate logic synthesis: A survey," *Proceedings of the IEEE*, 2020.

[3] N. Petra, D. De Caro, V. Garofalo, E. Napoli, and A. G. Strollo, "Truncated binary multipliers with variable correction and minimum mean square error," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 6, pp. 1312–1325, 2010.

[4] J. Schlachter, V. Camus, K. V. Palem, and C. Enz, "Design and applications of approximate circuits by gate-level pruning," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 5, pp. 1694–1702, 2017.

[5] I. Scarabottolo, G. Ansaloni, and L. Pozzi, "Circuit carving: A methodology for the design of approximate hardware," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 545–550.

[6] G. Liu and Z. Zhang, "Statistically certified approximate logic synthesis," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 344–351.

[7] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2013, pp. 1367–1372.

[8] S. Hashemi, H. Tann, and S. Reda, "BLASYS: approximate logic synthesis using boolean matrix factorization," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[9] C. Meng, W. Qian, and A. Mishchenko, "ALSRAC: Approximate logic synthesis by resubstitution with approximate care set," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[10] A. L. Oliveira and A. Sangiovanni-Vincentelli, "Learning complex boolean functions: Algorithms and applications," in *Advances in Neural Information Processing Systems*, 1994, pp. 911–918.

[11] S. Boroumand, C.-S. Bouganis, and G. A. Constantinides, "Learning boolean circuits from examples for approximate logic synthesis," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 2021, pp. 524–529.

[12] D. J. C. MacKay, *Information Theory, Inference & Learning Algorithms*. USA: Cambridge University Press, 2002.

[13] T. O. Kvålseth, "On normalized mutual information: measure derivations and properties," *Entropy*, vol. 19, no. 11, p. 631, 2017.

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

[15] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 24–40.

[16] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *IEEE Transactions on computers*, vol. 62, no. 9, pp. 1760–1771, 2012.