Quantified Satisfiability-based Simultaneous Selection of Multiple Local Approximate Changes under Maximum Error Bound

Chenfei Lou[1], Weihua Xiao[1], and Weikang Qian[1,2]
[1]University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University, Shanghai, China
[2]MoE Key Laboratory of Artificial Intelligence, Shanghai Jiao Tong University, Shanghai, China
Emails: {cf_lou, 019370910014, qianwk}@sjtu.edu.cn

2022 IEEE International Symposium on Circuits and Systems (ISCAS) | 978-1-6654-8485-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISCAS48785.2022.9937529

***Abstract*—Approximate computing is an emerging low-power design technique for error-tolerant applications. One key enabling technique for approximate circuit design is approximate logic synthesis (ALS). Many ALS methods are based on a scheme that iteratively selects one single local approximate change (LAC) in each round until the error bound is reached. However, this scheme fails to consider the joint effect of multiple LACs whose induced errors may counteract with each other when applied simultaneously. In this work, we propose a method to select multiple LAC candidates in a single round under a given bound on maximum error distance (MaxED). It first builds a miter by adding a multiplexer into the network for each LAC candidate in the network. Then, a quantified satisfiability problem is formulated on the miter and solved to obtain a maximal set of LACs that can be applied simultaneously. The experimental results show that under the normalized MaxED bound of 1%, our method reduces the circuit area by up to 39%, which is 20% higher than the area reduction achieved by a baseline method that iteratively selects one single LAC per round.**

***Index Terms*—approximate computing, approximate logic synthesis, satisfiability, quantified satisfiability, maximum error distance**

I. Introduction

The scaling in CMOS transistor size [1] has greatly increased the power density of VLSI circuits. Meanwhile, some complex circuits still occupy a large area. Thus, it is desirable to decrease both the power consumption and the area of a circuit. It is noteworthy that exact correctness is not always required for many applications like data mining, machine learning, and image processing [2]. Under such circumstances, *approximate computing*, an emerging technique that exploits the error tolerance in some applications to decrease the area and power consumption, is attracting more attention [3].

One key enabling technique for approximate circuit design is *approximate logic synthesis (ALS)* [4]–[16], which aims at automatically optimizing a circuit under given error constraints. Many ALS methods are based on iterative applications of *local approximate changes (LACs)* [4]–[9]. In these methods, many LACs are identified in each round, then a figure-of-merit (FoM) is evaluated for each of them, and finally the one with the highest FoM is chosen and applied. The iteration repeats until the accumulated error reaches the error threshold. Such ALS schemes, though simple to implement, have one obvious shortcoming: each round only selects one LAC to be applied, which fails to exploit the opportunity that applying multiple LACs simultaneously can lead to a better approximation, since errors from different LACs can possibly counteract with each other.

In this work, for error specification as the *maximum error distance (MaxED)*, we propose a new ALS method to overcome the above shortcoming. It can take into account an arbitrary number of LAC candidates in a single round, and select a maximal subset of those candidates, such that by applying them, the resulting MaxED is strictly below a given error bound. With this method, we achieve simultaneous selection of multiple LACs by considering their joint effect. The proposed method first builds a miter by adding a multiplexer (MUX) into the network for each LAC candidate in the network. Then, a *quantified satisfiability (QSAT)* problem is formulated on the miter and solved to yield a maximal set of LACs that are simultaneously applicable. According to the experimental results, our method reduces the circuit area by up to 39% for the normalized MaxED bound of 1%, which is 20% higher than the area reduction achieved by a baseline method that iteratively selects one single LAC per round.

The rest of the paper is organized as follows. Section II provides the preliminaries. Section III presents the proposed method. Section IV shows the experimental results. Section V concludes the paper.

This work is supported by the National Key R&D Program of China under Grant 2020YFB2205501. Corresponding author: Weikang Qian.

II. Preliminaries

A. Terminologies on Circuits and Boolean Functions

A combinational circuit is usually represented by a *directed acyclic graph (DAG)*. *Primary inputs (PIs)* and *primary outputs (POs)* of a circuit are the nodes in the DAG with zero in-degree and out-degree, respectively. The *level of a node* is the length of the longest possible path from a PI to that node. If a node A is the direct input of another node B, A is called a *fanin* of B, and B is called a *fanout* of A. If there exists a directed path from a node C to another node D, then C is called a *transitive fanin (TFI)* of D. The *TFI cone* of a node N, denoted as *TFI*(N), is the set of all TFIs of N and N itself. The *maximum fanout-free cone (MFFC)* of a node E, denoted by *MFFC*(E), is the maximum set of nodes such that $E \in$ *MFFC*(E), and for all $F \neq E$, $F \in$ *MFFC*(E) if and only if all the fanouts of F are also in *MFFC*(E) [17].

A *literal* is either a variable or its negation. A Boolean expression is called a *conjunctive normal form (CNF)* if it is a sum of products of literals. For instance, the Boolean expression $(c+\bar{a}+\bar{b})(\bar{c}+a)(\bar{c}+b)$ is a CNF.

B. Satisfiability and Quantified Satisfiability

Satisfiability (SAT) problem is an important problem in computer science, which can be stated as follows: given a Boolean function $F(x_1, \ldots, x_n)$, tell whether there exists an assignment on $x_1, \ldots, x_n$ such that F yields 1 under the assignment. If yes, return *SAT* together with a satisfying assignment; otherwise, return *unSAT*. SAT technique has been widely used in electronic design automation [18]. Such an application typically requires a transformation from a circuit to a CNF, which can be realized by Tseitin transformation [19].

An extension of SAT, *quantified satisfiability (QSAT)*, aims at solving whether a *prenex CNF (PCNF)* is satisfiable [20]. A PCNF can be expressed as $\psi = \Pi.\Phi$, where Φ is a CNF expression, and $\Pi = Q_1X_1 \cdots Q_nX_n$ consists of multiple quantifiers Q_i. Each quantifier Q_i is either an existential quantifier $\exists$ or a universal quantifier $\forall$, and binds to a set of variables X_i. The mutual intersection of the sets X_i is empty, while their union contains all the variables that appear in CNF Φ. Without loss of generality, we only consider the case where universal and existential quantifiers appear alternately in a PCNF. For instance, the following expression is a PCNF:

$$\psi = \exists a\ \forall b, c\ \exists d\ (a + c)(a + \bar{b} + d).$$

For a PCNF $\psi = Q_1X_1 \cdots Q_nX_n\Phi(X_1 \cup \cdots \cup X_n)$, its solution is meaningful only if the outermost quantifier Q_1 is an existential

quantifier $\exists$. In this case, the solution for this PCNF is an assignment for the variables in X_1 such that under this assignment, the PCNF is satisfiable.

To solve QSAT problems, Lonsing and Egly proposed a method called QCDCL [20]. The corresponding C library *depqbf* is made open-source, which is used in our work.

III. Proposed Method

Our method builds a miter with all the LAC candidates and then formulates and solves QSAT problems on the miter to determine a maximal subset of LACs that can be applied simultaneously. The procedure in one iteration can be summarized as the 4 steps below:

1) Obtain all the LAC candidates in the current network.
2) Copy the network and add a MUX for each LAC candidate, whose selection signal controls whether the LAC candidate is applied. The obtained network is called a *MUXed network*.
3) Build a miter between the original network and the MUXed network, and use QSAT and binary search to determine a maximal set of LACs that can be applied simultaneously under the error bound.
4) Apply the set of LACs determined by the previous step to generate an approximate network.

After the above 4 steps, we obtain an approximate network. As the network changes, it brings new opportunities to extract new LACs that can further reduce the area. Thus, the above iteration of 4 steps is repeated until there is no more improvement to the network area.

In the following, we elaborate the above 4 steps in Sections III-A–III-D, respectively. Note that we use a special type of LAC called SASIMI LAC [5] as an example to show how our method works. However, our method also works for other types of LACs.

A. Obtaining SASIMI LAC Candidates

We apply SASIMI method to select LAC candidates [5]. SASIMI identifies pairs of nodes in a network that have a high probability to give equal outputs under the same assignments of PIs [5]. Then, for each identified pair, we can substitute one node in the pair by the other to reduce the circuit area, while the error induced by the substitution is expected to be small due to the similarity of the outputs of the two nodes. In each node pair, the node at a lower level is called the *substituting signal (SS)*, while the other is called the *target signal (TS)*. Note that we require *SS* to have a lower level because it avoids forming a combinational loop after *TS* is substituted by *SS*. Once the substitution is applied, the nodes in the set $(MFFC(TS) - TFI(SS))$ can be deleted, leading to an area save of $Area\,(MFFC(TS) - TFI(SS))$. The original SASIMI method selects the *TS*-*SS* pair with the highest FoM in each iteration. In our case, we modify it so that a set of *TS*-*SS* pairs are selected. The *TS* nodes in these pairs range over all the nodes in the network, and each corresponding *SS* node is the one that yields the highest FoM with its *TS* node. Algorithm 1 shows the procedure to obtain the set of *TS*-*SS* pairs, or the *SASIMI LAC candidates*. As it shows, *TS* node is traversed through all the nodes in the network (Line 3), while in each iteration where *TS* node is fixed, one *SS* node with the highest FoM is identified (Lines 5–11). Then, this pair of *TS* and *SS* nodes is added into the candidate LAC set (Line 12).

In our implementation, we use a batch error estimation technique [21] to accelerate the estimation of P_{diff}.

B. Building MUXed Network for All the LAC Candidates

After obtaining the set of LACs, we regard them only as candidates and do not apply them (i.e., substitute each TS by the corresponding SS) immediately. Instead, for each LAC, we insert a MUX to control whether the LAC is applied by the following procedure:

Algorithm 1: Obtain SASIMI LAC candidates.

input : a given circuit *Ckt*.
output: the set *LAC_set* containing all the SASIMI LAC candidates from *Ckt*.

```
1  LAC_set = ∅;
2  Signals = a list of all signals in Ckt in a topological order;
3  foreach TS in Signals do
4      max_FoM = 0; SS_best = null;
5      foreach SS in Signals do
6          if SS.Level > TS.Level then continue;
7          P_diff = Probability of TS ≠ SS;
8          if P_diff > 0.5 then SS = ¬SS;
9          FoM = (Area(MFFC(TS) − TFI(SS)) − P_diff) / max(P_diff, 1 − P_diff);
10         if FoM > max_FoM then
11             SS_best = SS; max_FoM = FoM;
12     LAC_set.append(<SS_best, TS>);
13 return LAC_set;
```

1) Connect *TS* and *SS* to the 0-input and the 1-input of a MUX, respectively.
2) Add a new PI and connect it to the selection input of the MUX.
3) Connect all the original fanouts of *TS* to the output of the MUX.
4) Delete all the connections between *TS* and its original fanouts.

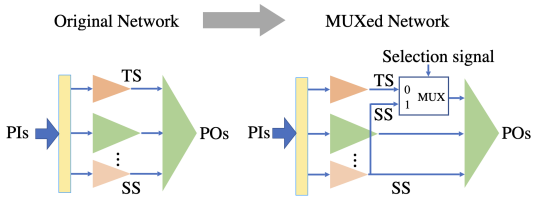

Fig. 1. Building a MUXed network based on the LAC candidates.

Fig. 1 shows the difference before and after inserting a MUX. As illustrated, when the selection signal is set to 0, the signal *TS* is still fed to all of its original fanouts, so the local function is unchanged. However, when the selection signal is set to 1, the signal *SS* becomes the fanin for all the original fanouts of *TS*, which is equivalent to substituting *TS* by *SS*. To sum up, whether a SASIMI LAC is applied is simply controlled by the corresponding selection signal.

This operation is applied to all the candidate LACs. The obtained network is called a *MUXed network*. Suppose that there are m LACs in total. Then, the MUXed network has m additional PIs, each determining the application of a LAC. These m PIs are called *LAC controlling PIs (LCPIs)*. By controlling LCPIs, we can model the circuit function resulting from any combination of the LACs.

C. Finding a Maximal LAC Set

With the obtained MUXed network, we now find a maximal set of LAC candidates that can be applied simultaneously, while still satisfying the error bound. To achieve this goal, we first build a miter between the original network and the MUXed network (see Section III-C1) and then add a sorting network to control the number of applied LACs (see Section III-C2). Finally, we solve for a maximal set of LACs using binary search algorithm and QSAT technique on the miter (see Section III-C3).

1) Error Distance Miter and QSAT Formulation: we build an *error distance miter*, as shown in Fig. 2. It consists of the original network, the MUXed network, a subtractor that calculates the absolute difference between the outputs of the original network and the MUXed network, and a comparator to determine whether the difference is smaller than a user-specified threshold T.

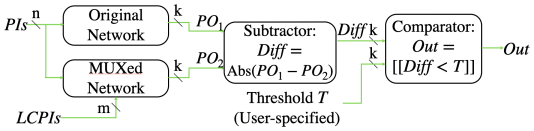

Fig. 2. The error distance miter.

Then, we apply Tseitin transformation to get the CNF corresponding to the error distance miter. Let the consequent CNF be Φ. To ensure that the error distance is always smaller than the threshold, we need to find an assignment of the LCPIs such that under all assignments of PIs, Φ is always SAT.

Note that the CNF Φ contains a set of variables other than the PIs and the LCPIs, which we denote as V. Thus, the problem we face is exactly a QSAT problem of the following PCNF

$$\exists \mathit{LCPI}\ \forall \mathit{PI}\ \exists V\ \Phi.$$

If the QSAT problem is unSAT, then there is no combination of LACs that can be applied to meet the error threshold. Otherwise, each solution of the QSAT problem gives an assignment of LCPIs that corresponds to a valid approximation to the original network. For example, suppose $m = 3$ and a solution is $\mathit{LCPI}_1 = 1$, $\mathit{LCPI}_2 = 0$, and $\mathit{LCPI}_3 = 1$, Then, a corresponding approximate circuit can be obtained by applying LAC_1 and LAC_3 but ignoring LAC_2. We define *satisfying LAC set* as a set of LACs that can be applied to meet the error threshold. For the example above, the corresponding satisfying LAC set is $\{\mathit{LAC}_1, \mathit{LAC}_3\}$.

2) Expanded Error Distance Miter with a Sorting Network: Generally, there exist multiple solutions for QSAT, including a trivial one with all the LCPI variables assigned to 0 (i.e., no LAC applied). Therefore, we need to design a mechanism to find the best solution. Here, we define the best solution as the assignment where the number of 1s among the LCPI variables is maximized, since a good first-order measure for the reduced area caused by the approximation is the number of applied LACs.

To achieve the goal, we add a sorting network to the error distance miter to sort the LCPIs based on their values. The sorting network is a component that takes a number of input bits, and outputs them in a sorted order (i.e., from the smallest to the largest) [22]. The following equation shows an example of the input and the output of an 8-bit sorting network:

$$00101001 \longrightarrow 00000111.$$

As can be seen, if we can find a value i such that the i^{th} output is 1 but the $(i-1)^{\text{th}}$ output is 0, then the number of 1s among the input bits is $(t-i+1)$, where t is the number of input bits. Readers interested in the details on sorting networks can refer to [22].

By resorting to the sorting network, we further build an *expanded error distance miter*, as shown in Fig. 3, to help find the maximum number of 1s among the LCPI variables. It is built upon the error distance miter shown in Fig. 2. We further add an m-bit sorting network and feed the m LCPIs to the inputs of the sorting network, so that the m outputs of the sorting network $s_1, \ldots, s_m$ are the sorted version of the LCPIs. We denote Ntk_i $(1 \le i \le m)$ as the network obtained by adding an AND gate over the output *Out* of the original miter and the signal s_i. We denote Φ_i as the CNF corresponding to the network Ntk_i. Given this setup, if the PCNF $\exists \mathit{LCPI}\ \forall \mathit{PI}\ \exists V\ \Phi_i$ is SAT, then there exists a satisfying LAC set with at least $(m-i+1)$ LACs.

3) Binary Search for Finding a Maximal Satisfying LAC Set: With the help of the expanded error distance miter, we can find the maximum number of 1s in the LCPIs, which leads to a maximal satisfying LAC set. The idea is to find the least index i_0 such that

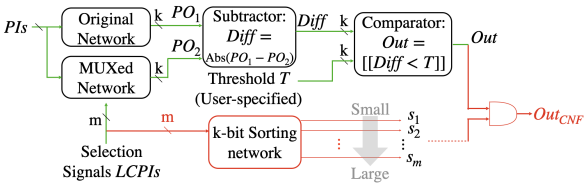

Fig. 3. The expanded error distance miter.

the PCNF $\exists \mathit{LCPI}\ \forall \mathit{PI}\ \exists V\ \Phi_{i_0}$ is SAT. To efficiently find the least index, we apply a binary search algorithm, as shown in Algorithm 2. In each round, we maintain a search interval $[l, r]$, and solve the QSAT problem on PCNF $\exists \mathit{LCPI}\ \forall \mathit{PI}\ \exists V\ \Phi_i$, where $i = \lceil \frac{l+r}{2} \rceil$ is the midpoint of the current search interval $[l, r]$. Then, we update the search interval based on the QSAT result. If the PCNF is unSAT, it means $i_0 > i$, so the new search interval is $[i+1, r]$. Otherwise, $i_0 \le i$, and the new search interval is set to $[l, i-1]$. Note that the updating mechanism guarantees that all the indices smaller than l (larger than r) lead to unSAT (SAT). The search stops when $l > r$. At this point, $(r+1)$ has become the least index i_0 that makes the PCNF $\exists \mathit{LCPI}\ \forall \mathit{PI}\ \exists V\ \Phi_{i_0}$ SAT.

Algorithm 2: Find a maximal satisfying LAC set.

Input: the PCNFs of the expanded error distance miters $\{\exists \mathit{LCPI}\ \forall \mathit{PI}\ \exists V\ \Phi_k\}_{k=1}^{m}$.
Output: the assignment *assignment* for a PCNF that corresponds to a maximal satisfying LAC set.

```
l = 1; r = m;
while l ≤ r do
    i = ⌈(l+r)/2⌉; pcnf = ∃LCPI ∀PI ∃V Φ_i;
    (result, assignment) = QSATsolve(pcnf);
    if result == unSAT then l = i + 1;
    else r = i − 1;
(result, assignment) = QSATsolve(∃LCPI ∀PI ∃V Φ_{r+1});
return assignment;
```

D. LAC Application

This section shows how to apply the LACs in the obtained maximal satisfying set to the original network to obtain the approximate network. It consists of two steps: 1) Connect the output of the *SS* to the original fanouts of *TS*; 2) Delete the redundant nodes in the set $(\mathit{MFFC}(\mathit{TS}) - \mathit{TFI}(\mathit{SS}))$.

One straightforward but problematic method is to directly execute the above two steps for each LAC one by one, i.e., delete all the redundant nodes of the current LAC before the next one is considered. The problem of the method is that a redundant node for the current LAC may be the *TS* node or *SS* node for a following LAC that has not been considered yet, and if it is deleted, the LAC afterwards may become invalid and cannot be applied anymore. We call the problem the *immediate-deletion* problem. Fig. 4 shows an example of the problem, where LAC_1 aims at substituting node 3 by node 7, while LAC_2 aims at substituting node 2 by node 4. However, node 4, which is among the redundant nodes for LAC_1, is the *SS* node for LAC_2. Therefore, after LAC_1 is applied, LAC_2 becomes invalid and cannot be applied anymore.

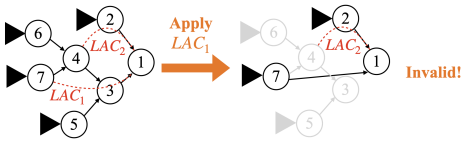

Fig. 4. An illustration of the immediate-deletion problem.

To handle the above problem, we propose a *post-deletion method* to apply all the LACs. It first performs step 1 only when traversing through all the LACs in the satisfying LAC set, and after all those LACs have been traversed, it deletes all the redundant nodes all together. In this context, a node is regarded as redundant when either it has no fanouts or all of its fanouts are redundant. Fig. 5 illustrates an example, where node 2 and node 3 are to be substituted by node 4 and node 7, respectively. We first perform step 1 to all the LACs in the satisfying LAC set, which results in an intermediate network shown in the middle of Fig. 5. Then, we delete the redundant nodes, which, by definition, are nodes 2, 3, and 5. We obtain the final approximate network as shown in the right of Fig. 5.

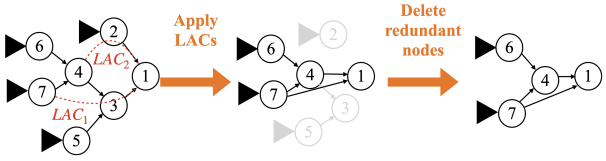

Fig. 5. An illustration of the proposed post-deletion method.

IV. Experimental Results

This section presents the experimental results on our method. To show the advantage of multiple selection, we also implement a *baseline method* that selects and applies only one SASIMI LAC per round. In each round of it, we first use the same method described in Section III-A to select a set of SASIMI LAC candidates. Then, we sort the LAC candidates in descending order of the area reduction that they can bring. After that, we apply the first LAC candidate that strictly satisfies the MaxED constraint. The above operations repeat until in one round all the LAC candidates fail to satisfy the error constraint. We implement both our method and the baseline method in C++. We use ABC [23] to conduct logic optimization and technology mapping with the MCNC standard cell library [24]. We test both methods on 5 different arithmetic circuits, where the first two are 8-bit adders, the third one is a 4-bit multiplier, and the last two are taken from the EPFL combinational benchmark suite [25]. Their information is listed in Table I. Since different circuits have different bit width for the outputs, we use the *normalized maximum error distance (NMaxED)* as the error metric, which is defined as:

$$NMaxED = \frac{\max\ |Output_{approx} - Output_{exact}|}{2^{Output\_Bit\_Width}}.$$

Table I. Arithmetic benchmarks.

Name	I/O	Gate Count	Level	Initial Area
RCA_8	17/9	56	18	145
CLA_8	17/9	64	19	198
MUL_4	8/8	85	16	321
int2float	11/7	183	10	398
priority	128/8	453	17	1179

We measured the area reduction percentage of the approximate circuits over the input exact circuit under different NMaxED thresholds, and the results are shown in Fig. 6, where the results by our method are shown in solid lines and labeled with letter "m" to denote "multiple selection", while those by the baseline method are shown in dashed lines and labeled with letter "s" to denote "single selection". Note that when the areas of the circuits decrease, their delays do not increase, since SASIMI method guarantees that the selected LACs never increase the length of the critical path.

As we can see, for all the circuits and under all the error thresholds, our method outperforms the baseline method. For our method, when the error threshold is relatively loose (i.e., 12.5%), the area reductions of all the circuits are more than 60% of the original area. Specially, the area reduction for the circuit `int2float` is as high as 80.2%. In comparison, only two circuits' area reductions achieved by the baseline method are over 60.0%, with the largest area reduction as 69.8%, which is 10.4% lower than our method. For the error threshold of 25%, the area reductions of all the circuits achieved by our method are all higher than 65%, with 4 of them higher than 85%, while for the baseline method, all circuits' area reductions are lower than 80.0%. Under the error bound of 1% and 5%, the area reductions achieved by our method are up to 39% and 65%, respectively, which are 20% and 8% higher than the largest area reductions achieved by the baseline method.

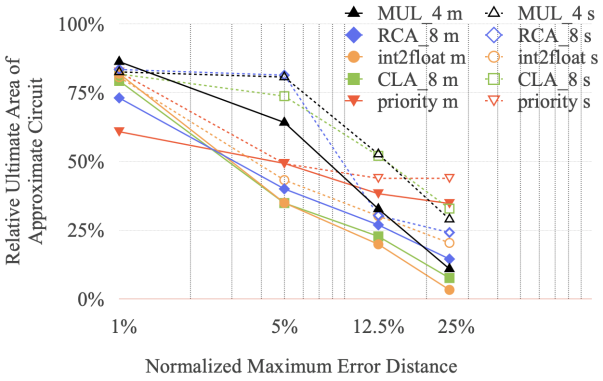

Fig. 6. Relative area of the approximate circuits *vs.* normalized maximum error distance.

Fig. 7 shows the runtime of our method and the baseline method. We can see that our method sometimes takes longer time than the baseline method. This can be attributed to the fact that our method needs to solve QSAT problems with 3 nested quantifiers, while the baseline method only needs to solve QSAT problems with 2 nested quantifiers.

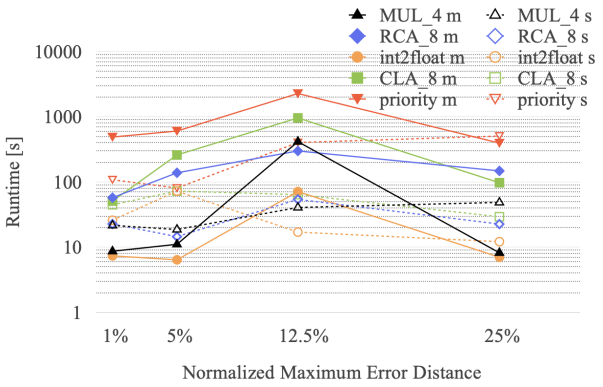

Fig. 7. Runtime *vs.* normalized maximum error distance.

V. Conclusion

In this work, we propose a novel ALS method that can select multiple LAC candidates in a single iteration under the MaxED constraint. It is based on a QSAT formulation on a proposed miter. The experimental results show that our method reduces the circuit area by up to 39% under the normalized MaxED bound of 1%. Meanwhile, our method still has space for further improvement, e.g., its scalability. Techniques like partition and error propagation [26] can be exploited to improve the scalability for our method, which is left for future research.

REFERENCES

[1] M. M. Waldrop, "The chips are down for Moore's law," *Nature*, vol. 530, no. 7589, pp. 144–147, 2016.

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

[3] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design & Test*, vol. 33, no. 1, pp. 8–22, 2016.

[4] D. Shin and S. K. Gupta, "A new circuit simplification method for error tolerant applications," in *Design, Automation and Test in Europe*, 2011, pp. 1–6.

[5] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits," in *Design, Automation and Test in Europe*, 2013, pp. 1367–1372.

[6] A. Chandrasekharan, M. Soeken, D. Große, and R. Drechsler, "Approximation-aware rewriting of AIGs for error tolerant applications," in *International Conference on Computer-Aided Design*, 2016, pp. 83:1–83:8.

[7] S. Froehlich, D. Große, and R. Drechsler, "Approximate hardware generation using symbolic computer algebra employing Grobner basis," in *Design, Automation and Test in Europe*, 2018, pp. 889–892.

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

[9] C. Meng, W. Qian, and A. Mishchenko, "ALSRAC: Approximate logic synthesis by resubstitution with approximate care set," in *Design Automation Conference*, 2020, pp. 187:1–187:6.

[10] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "SALSA: Systematic logic synthesis of approximate circuits," in *Design Automation Conference*, 2012, pp. 796–801.

[11] J. Miao, A. Gerstlauer, and M. Orshansky, "Multi-level approximate logic synthesis under general error constraints," in *International Conference on Computer-Aided Design*, 2014, pp. 504–510.

[12] Z. Vasicek and L. Sekanina, "Evolutionary approach to approximate digital circuits design," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 432–444, 2015.

[13] Y. Wu and W. Qian, "An efficient method for multi-level approximate logic synthesis under error rate constraint," in *Design Automation Conference*, 2016, pp. 128:1–128:6.

[14] G. Liu and Z. Zhang, "Statistically certified approximate logic synthesis," in *International Conference on Computer-Aided Design*, 2017, pp. 344–351.

[15] Z. Zhou, Y. Yao *et al.*, "DALS: Delay-driven approximate logic synthesis," in *International Conference on Computer-Aided Design*, 2018, pp. 86:1–86:7.

[16] J. Castro-Godínez, H. Barrantes-García, M. Shafique, and J. Henkel, "AxLS: A framework for approximate logic synthesis based on netlist transformations," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 8, pp. 2845–2849, 2021.

[17] J. Cong and Y. Ding, "Combinational logic synthesis for LUT based field programmable gate arrays," *ACM Transactions on Design Automation of Electronic Systems*, vol. 1, no. 2, pp. 145–204, 1996.

[18] J. P. Marques-Silva and K. A. Sakallah, "Boolean satisfiability in electronic design automation," in *Design Automation Conference*, 2000, pp. 675–680.

[19] G. S. Tseitin, *On the complexity of derivation in propositional calculus*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1968, pp. 466–483.

[20] F. Lonsing and U. Egly, "DepQBF 6.0: A search-based QBF solver beyond traditional QCDCL," in *Automated Deduction*, 2017, pp. 371–384.

[21] S. Su, Y. Wu, and W. Qian, "Efficient batch statistical error estimation for iterative multi-level approximate logic synthesis," in *Design Automation Conference*, 2018, pp. 54:1–54:6.

[22] K. E. Batcher, "Sorting networks and their applications," in *American Federation of Information Processing Societies*, 1968, pp. 307–314.

[23] A. Mishchenko *et al.*, "ABC: a system for sequential synthesis and verification, release 90703," accessed Feb., 2021. [Online]. Available: http://people.eecs.berkeley.edu/∼alanmi/abc/

[24] A. de Geus, "Logic synthesis and optimization benchmarks for the 1986 design automation conference," in *Design Automation Conference*, 1986, pp. 78–78.

[25] A. T. Calvino *et al.*, "The EPFL combinational benchmark suite," accessed May, 2021. [Online]. Available: https://github.com/lsils/benchmarks

[26] I. Scarabottolo, G. Ansaloni, G. A. Constantinides, and L. Pozzi, "A formal framework for maximum error estimation in approximate logic synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–15, 2021.