AccALS: Accelerating Approximate Logic Synthesis by Selection of Multiple Local Approximate Changes

Xuan Wang[1], Sijun Tao[1], Jingjing Zhu[1], Yiyu Shi[2], and Weikang Qian[1,3]
[1]UM-SJTU Joint Inst. and [3]MoE Key Lab of AI, Shanghai Jiao Tong University, China; [2]University of Notre Dame, US
Emails: xuan.wang@sjtu.edu.cn, 603447229@sjtu.edu.cn, jingjingzhu@sjtu.edu.cn, yshi4@nd.edu, qianwk@sjtu.edu.cn

***Abstract*—Approximate computing is an energy-efficient computing paradigm for error-tolerant applications. To automatically synthesize approximate circuits, many iterative approximate logic synthesis (ALS) methods have been proposed. However, most of them do not consider applying multiple local approximate changes (LACs) in a single round, which can lead to a much shorter runtime. In this paper, we propose AccALS, a novel framework for Accelerating iterative ALS flows, based on simultaneous selection of multiple LACs in a single round. When selecting multiple LACs, there may exist conflicts among them. One important component of AccALS is a novel method to solve the conflicts. Another is an efficient measure for the mutual influence between two LACs. With its help, the problem of selecting multiple LACs is transformed into a maximum independent set problem to solve. The experimental results showed that compared to a state-of-the-art method, AccALS accelerates by up to $24.6\times$ with a negligible circuit quality loss.**

***Index Terms*—approximate computing, approximate logic synthesis, multiple local approximate changes**

I. Introduction

With the advent of the post-Moore era, it becomes increasingly difficult to further reduce the circuit cost [1]. Fortunately, there are many error-tolerant applications, such as image processing and machine learning. Under this circumstance, *approximate computing* [2]–[4] is proposed to design energy-efficient systems for these error-tolerant applications. It introduces slight imprecision in computation to reduce the area, delay, and power consumption of circuits.

An important area in approximate computing is *approximate logic synthesis (ALS)*, which aims to synthesize an approximate circuit with reduced hardware cost satisfying a given error constraint [5]. Recently, iterative ALS methods are popular due to their good synthesis quality [6]–[10]. They obtain the final approximate circuit by multiple rounds of *local approximate changes (LACs)*, where a LAC is a change to a local circuit that reduces the hardware cost, but may introduce some errors. For example, Fig. 1(a) shows a SASIMI LAC proposed in [7]. It replaces a signal by another almost identical existing signal v or v's negation. For the example accurate circuit shown in Fig. 1(a), the SASIMI LAC replaces signal e by signal c, leading to the approximate circuit shown in Fig. 1(b). Fig. 1(a) also shows an ALSRAC LAC proposed in [9]. It replaces a signal by a new function on some existing signals. For the example accurate circuit shown in Fig. 1(a), the ALSRAC LAC replaces signal f with the OR of signals b and c, leading to the approximate circuit shown in Fig. 1(c). For most iterative ALS methods [6]–[10], they are based on the *single-selection framework*. That is, in each round, all the candidate LACs for the current approximate circuit are first identified and evaluated, and then the single best LAC is selected and applied to the current approximate circuit to generate a new approximate circuit.

The runtime of the single-selection framework is long, since it just selects a single LAC in each round, albeit the time-consuming evaluation of all the candidate LACs. For example, it can take more than 20 hours to finish the synthesis on a large circuit [10]. Thus, it is imperative to accelerate it. For this purpose, the works [11], [12] propose methods to speed up the error evaluation of all the LACs. A few other works consider applying multiple LACs simultaneously in each round [13]–[15]. In [13], a method is proposed to select multiple LACs in the *AND-inverter graph (AIG)*, which is realized by solving a maximum flow problem. However, its primary target is reducing the circuit delay, not speeding up the ALS flow. The works [14], [15] use the evolutionary approach to select multiple LACs to obtain approximate circuits. The work [14] applies NSGA-II, a multi-objective genetic algorithm, to select multiple sub-functions from the approximate Boolean network. In [15], the *archived multi-objective simulated annealing (AMOSA)* heuristic is used to select multiple approximate cuts to generate approximate circuits, where the approximate cuts are obtained by the exact synthesis method.

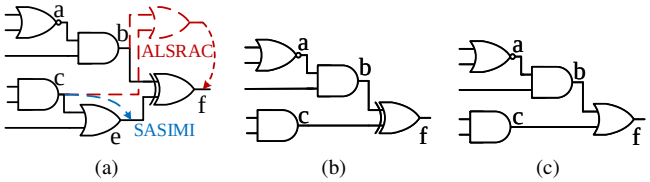


Fig. 1. LAC examples: (a) an accurate circuit; (b) an approximate circuit with the SASIMI LAC [7]; (c) an approximate circuit with the ALSRAC LAC [9].

Applying multiple LACs simultaneously in each round is a promising method for speeding up the ALS flow. However, despite the recent advance in this direction, some fundamental understanding of the relationship of multiple LACs is still lacking. First, the mutual influence of multiple LACs can affect the circuit error. Some LACs have positive influence, *i.e.*, they counteract with each other to mask the circuit error; some have negative influence, which amplifies the circuit error. Such a mutual influence is not well known. Second, some highly effective and widely used LACs are based on replacing a signal with some existing signals, such as SASIMI and ALSRAC LACs [7], [9] shown in Fig. 1(a). When applying multiple such LACs simultaneously, the application of a LAC can invalidate another, causing a *conflict*. The existing works either impose a rigid selection rule [13] or use a special type of LAC [14], [15] to avoid the conflicts. The general conflict problem is not well understood and solved.

In this work, we make a more systematic study on the relationship of multiple LACs, including their mutual influence and conflicts, and propose AccALS, a framework for Accelerating the iterative ALS flows by simultaneous selection of multiple LACs. The main contributions are as follows.

1. To formally characterize the mutual influence among multiple LACs, we introduce the notion of positive, negative, and independent LAC sets.
2. To select a maximal set of LACs with less mutual influence, we first define an index for measuring the mutual influence between two LACs, which is efficient to calculate. Then, with its help, we transform the selection problem into a *maximum independent set (MIS)* problem to solve.
3. As there exist potential LAC conflicts among the multiple selected LACs, we propose a novel method to solve the conflicts by building a LAC conflict graph.
4. Based on the above techniques, a general framework, AccALS, is proposed to simultaneously select multiple LACs with less mutual influence and positive mutual influence in a single round for the iterative ALS methods.

This work is supported by the National Key R&D Program of China under Grant 2021ZD0114701 and the National Natural Science Foundation of China under Grants T2293700 and T2293701. Corresponding author: Weikang Qian.

AccALS can be applied to any input distribution, any graph-based circuit representation, and any statistical error metric. As examples, this work considers three statistical error metrics, *error rate (ER)*, *normalized mean error distance (NMED)*, and *mean relative error distance (MRED)*. ER represents the probability that the circuit outputs are incorrect. NMED measures the average *error distance (ED)* normalized by the maximum output value, while MRED measures the average relative ED. The experimental results show that AccALS outperforms a state-of-the-art method in runtime by $6.3\times$ to $24.6\times$ with a negligible circuit quality loss.

II. Methodology

This section presents the proposed AccALS methodology.

A. Overview of AccALS

This section overviews the framework of AccALS. We first introduce some assumptions and notations used in this paper.

- Denote the current approximate circuit as $\mathcal{G}$. After applying multiple LACs to $\mathcal{G}$, the resulting circuit is denoted as $\mathcal{G}_{new}$. In this work, it is assumed that we only deal with the LACs whose affected local circuits have a single output. Many existing LACs satisfy this property, such as SASIMI [7] and ALSRAC [9] LACs. Note that the circuits AccALS handles are still multi-output.
- Denote the errors of $\mathcal{G}$ and $\mathcal{G}_{new}$ as e and e_{new}, respectively.
- Generally, a LAC applied on a node n means that node n is replaced by a new function on some *existing* nodes in $\mathcal{G}$, which are called *substitute nodes (SNs)*. The SNs of the LAC form a set S_n called *substitute node set (SNS)*. The node n itself is called the *target node (TN)* of the LAC. We denote the LAC as $L(S_n, n)$. For example, the SASIMI and the ALSRAC LACs in Fig. 1(a) are denoted as $L(\{c\}, e)$ and $L(\{b, c\}, f)$, respectively. In what follows, we draw LACs in an abstract way. Below shows an example.

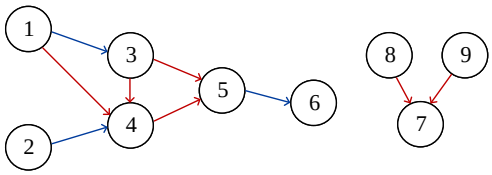


Fig. 2. A set of LACs represented in an abstract way.

Example 1. *Fig. 2 draws 6 different LACs in an abstract way. Each blue edge represents a LAC with only one SN. For example, the blue edge* $(2, 4)$ *represents the LAC* $L(\{2\}, 4)$ *with node* 2 *as its only SN and node* 4 *as its TN. Each pair of red edges pointing to the same node represents a LAC with two SNs. For example, the pair of red edges* $(1, 4)$ *and* $(3, 4)$ *pointing to node* 4 *represents the LAC* $(\{1, 3\}, 4)$*, where nodes* 1 *and* 3 *are its SNs, and node* 4 *is its TN.*

In an iterative ALS flow that selects a single LAC in each round, in order to decide which LAC to be chosen, it typically needs to evaluate the errors of all the candidate approximate circuits, each formed by applying a candidate LAC to the current circuit $\mathcal{G}$. A straightforward method to obtain the errors of all the candidate approximate circuits is to simulate all the circuits, but it is time-consuming. Thus, the state-of-the-art works [11], [12] propose methods to efficiently estimate the *error increase* due to a LAC ψ, which we denote as $\Delta E(\psi)$. Then, the error of the candidate approximate circuit (*i.e.*, the new approximate circuit $\mathcal{G}_{new}$) is estimated as $e_{est} = e + \Delta E(\psi)$, where e_{est} denotes the estimated error.

In order to speed up the iterative ALS flow by selecting multiple LACs in each round, the same task of *efficiently* evaluating the errors of all the possible candidate approximate circuits exists, where each circuit is obtained by applying a set of candidate LACs to the current approximate circuit. For this purpose, a straightforward idea is to extend the above estimator for a single LAC to the following one:

$$e_{est} = e + \sum_{\psi \in L} \Delta E(\psi), \tag{1}$$

where L is the set of applied LACs. However, the above estimator only holds when the LACs in set L do not affect each other. Unfortunately, this is not always true in reality: Different LACs may influence each other, causing a large difference between the estimated error e_{est} and the actual error e_{new}. In the following, we introduce a classification on a set L of candidate LACs based on the difference between e_{est} and e_{new}. Note that the classification is based on a *tolerance parameter* σ, which is a relatively small non-negative value.

- Case 1: $e_{est} - e_{new} > \sigma$. In this case, the actual error e_{new} is smaller than the estimated one, e_{est}, which means that the LACs in set L counteract with each other to mask some error. We call the LAC set a *positive LAC set*.
- Case 2: $|e_{est} - e_{new}| \leq \sigma$. In this case, the LACs in set L have less mutual influence. We call the LAC set an *independent LAC set*.
- Case 3: $e_{est} - e_{new} < -\sigma$. In this case, the influence of the LACs in set L amplifies the circuit error. We call the LAC set a *negative LAC set*.

The basic idea of AccALS is to choose a good independent LAC set L_{indp} and a good positive LAC set L_{rand} from the candidate LAC sets in each round. Then, we evaluate the effects of applying the LACs in L_{indp} and L_{rand} to the current approximate circuit, and choose the LAC set with better performance as the final choice.

Algorithm 1: The proposed framework of AccALS.

Input: an original circuit $\mathcal{G}_{org}$ and an error bound e_b.
Output: an approximate circuit $\mathcal{G}$ with error $e \leq e_b$.

```
1  G_new ← G_org; e ← 0;
2  while e ≤ e_b do
3      G ← G_new; G_c1 ← G; G_c2 ← G;
4      L_top ← ObtainTopSet(e, e_b);
5      {L_sol, N_sol} ← FindSolveLACConf(L_top);
6      L_indp ← SelectIndpLACs(L_sol, G, N_sol, e, e_b);
7      L_rand ← SelectRandomLACs(L_sol);
8      apply LACs in L_indp to G_c1 to obtain G_new1, and calculate the
         accurate error e_new1 between G_new1 and G_org;
9      apply LACs in L_rand to G_c2 to obtain G_new2, and calculate the
         accurate error e_new2 between G_new2 and G_org;
10     if e_new1 < e_new2 or (e_new1 = e_new2 and |L_indp| ≥ |L_rand|) then
11         G_new ← G_new1; e ← e_new1
12     else G_new ← G_new2; e ← e_new2 ;
13 return G;
```

The main procedure of AccALS is shown in Algorithm 1. Its inputs are an original circuit $\mathcal{G}_{org}$ and an error bound e_b, while its output is an approximate circuit $\mathcal{G}$ with error no more than e_b. Line 1 initializes the new circuit $\mathcal{G}_{new}$ as the original circuit $\mathcal{G}_{org}$ and the actual error e of $\mathcal{G}_{new}$ as 0. In each round, Line 3 sets the current approximate circuit $\mathcal{G}$ as $\mathcal{G}_{new}$, and makes two copies $\mathcal{G}_{c1}$ and $\mathcal{G}_{c2}$ of the current circuit $\mathcal{G}$. To reduce the search space of the selection of multiple LACs, we only consider a set L_{top} of top LACs with the smallest error increases. Line 4 calls the function *ObtainTopSet* to obtain this set. Its details will be described in Section II-B. The following steps of AccALS only focus on the LACs in L_{top}.

Different from an iterative ALS flow that selects a single LAC in each round, when we select multiple LACs in a round, there is another issue: sometimes, the selected LACs cannot be applied simultaneously. Below shows an example.

Example 2. *Suppose that in a round, we select the LACs* $L(\{2\},4)$ *and* $L(\{1,3\},4)$ *shown in Fig. 2. However, they cannot be applied simultaneously, since they have the same TN, and each node in the circuit can only be applied with one LAC in each round.*

When two LACs cannot be applied simultaneously, we say that they are *in conflict*. Thus, another task to handle is to find and solve these LAC conflicts. It is done by the function *FindSolveLACConf* at Line 5, which obtains the conflict-free LAC set L_{sol} from L_{top} together with its corresponding TN set N_{sol}. Its details will be described in Section II-C.

Line 6 calls the function *SelectIndpLACs* to select a subset of LACs from the conflict-free LAC set L_{sol} to form a good independent LAC set L_{indp}. It is based on an efficient index for measuring the mutual influence between two LACs and the formulation of an MIS problem. The details will be described in Section II-D. Line 7 calls the function *SelectRandomLACs* to randomly select some LACs from L_{sol} to form another LAC set L_{rand}. However, the LAC set L_{rand} may be positive, independent, or negative.

Lines 8–12 further decide the better one between set L_{indp} and set L_{rand} and apply the LACs in the set to the current approximate circuit finally. Specifically, Line 8 applies all the LACs in L_{indp} to $\mathcal{G}_{c1}$ to generate the new approximate circuit $\mathcal{G}_{new1}$, and calculates the actual error e_{new1} between $\mathcal{G}_{new1}$ and $\mathcal{G}_{org}$. Same as Line 8, Line 9 calculates the actual error e_{new2} after applying all the LACs in L_{rand} to $\mathcal{G}_{c2}$. Note that although we evaluate the actual errors of two new approximate circuits, its runtime is relatively negligible compared to the runtime of the other steps in the entire algorithm. Then, Lines 10–11 choose to apply the LACs in L_{indp}, and set the final new approximate circuit $\mathcal{G}_{new}$ as $\mathcal{G}_{new1}$ and the error e as e_{new1} if

- the error e_{new1} is smaller than e_{new2}, or
- the error e_{new1} equals e_{new2}, and the number of LACs in L_{indp} is no less than the number of LACs in L_{rand}, indicating that applying the LACs in L_{indp} is likely to have a larger area reduction without increasing error.

Otherwise, we apply the randomly selected LACs in L_{rand} (see Line 12). We remark that the above criterion of selecting the LAC set takes the error as the primary concern and the area reduction as the secondary concern. We choose this criterion in this work as it leads to good synthesis quality according to an existing work selecting a single LAC in each iteration [9]. When the error e is no more than the error bound e_b, the loop continues (see Line 2). Otherwise, it terminates, and Line 13 returns the latest approximate circuit $\mathcal{G}$.

B. Obtaining Top LAC Set

This section presents the details of the function *ObtainTopSet*, which obtains a set L_{top} of top LACs with the smallest error increases $\Delta E(\psi)$. First, we identify the candidate LAC set L_{cand} from the current approximate circuit $\mathcal{G}$. For each LAC ψ in L_{cand}, we estimate its error increase $\Delta E(\psi)$ by the SEALS method [12]. Then, we need to determine the size of set L_{top}, denoted as r_{top}. We set it as

$$r_{top} = \left(\frac{e_b - e}{e_b}\right) \max(r_{ref}, r_{min}), \tag{2}$$

where r_{ref} is a reference top LAC number and $r_{min} \geq 1$ is the number of LACs with the *minimum* error increase in L_{cand}. It should be noted that sometimes, r_{min} is larger than r_{ref}. In this case, to fully search the LAC space, we consider all the LACs with the minimum error increase, which explains the term $\max(r_{ref}, r_{min})$ in Eq. (2). We also found that as the circuit error e is closer to the error bound e_b, the error increases of the candidate LACs usually are larger. In this case, it is not preferable to select many LACs with larger error increases, as it reduces the total iteration number of the ALS flow significantly, leading to a synthesis quality drop. Therefore, we reduce r_{top} as e gets closer to e_b, which is achieved by the term $\frac{e_b - e}{e_b}$ in Eq. (2). For the special case where r_{top} calculated by Eq. (2) is either less than 1 or larger than the total number of LACs in L_{cand}, we set it to 1 or $|L_{cand}|$, respectively.

C. Finding and Solving LAC Conflicts

This section presents the details of the function *FindSolveLACConf* to obtain the conflict-free LAC set L_{sol} from the set L_{top} together with its corresponding TN set N_{sol}. The LAC conflicts can be divided into two types:

- Type 1: The LACs with the same TN. These LACs are in conflict, since each node can only be applied with one LAC in each round. An example is given by Example 2.
- Type 2: Two LACs such that an SN of one LAC is the TN of the other. For example, in Fig. 2, node 3 is the TN of the LAC $L(\{1\},3)$. However, node 3 is also an SN of the LAC $L(\{1,3\},4)$. In this case, we cannot apply these two LACs simultaneously, since applying the LAC $L(\{1\},3)$ removes node 3 and hence, the LAC $L(\{1,3\},4)$ no more exists.

To find and solve LAC conflicts in the set L_{top}, first we define a LAC conflict graph.

Definition 1. *A* ***LAC conflict graph*** *is a weighted undirected graph with each node representing a LAC in set* L_{top}*. The weight of each node is the error increase of its corresponding LAC. For any two nodes, they are connected by an edge if their corresponding LACs have a Type-1 or Type-2 conflict.*

Example 3. *Fig. 3 shows the LAC conflict graph for the set of LACs shown in Fig. 2. We represent the 6 LACs in Fig. 2, i.e.,* $L(\{1\},3)$*,* $L(\{1,3\},4)$*,* $L(\{2\},4)$*,* $L(\{3,4\},5)$*,* $L(\{5\},6)$*, and* $L(\{8,9\},7)$ *as nodes* T_1*,* T_2*,* T_3*,* T_4*,* T_5*, and* T_6 *in the LAC conflict graph shown in Fig. 3, respectively. The value near each node is its weight, i.e., the error increase of its corresponding LAC with ER as the error metric. Since LACs* $L(\{1\},3)$ *and* $L(\{1,3\},4)$ *have a Type-2 conflict, their corresponding nodes* T_1 *and* T_2 *are connected by an edge. Since* $L(\{1,3\},4)$ *and* $L(\{2\},4)$ *have a Type-1 conflict, their corresponding nodes* T_2 *and* T_3 *are also connected by an edge. The other edges in Fig. 3 are added similarly.*

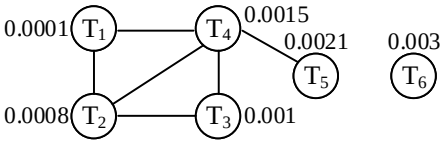


Fig. 3. An example of LAC conflict graph.

The detailed procedure of the function *FindSolveLACConf* can be divided into the following two steps.

1) Step 1: Given the top LAC set L_{top}, construct its corresponding LAC conflict graph G_{conf}.

2) Step 2: Find a set of nodes in G_{conf} without mutual connection, *e.g.*, the set $\{T_1, T_3, T_5\}$ in Fig. 3. By the definition of a LAC conflict graph, if a set of nodes in G_{conf} has no mutual connection, then the corresponding set of LACs has no conflict. Among the various choices, we want to find one such that the sum of the error increases of the selected LACs is as small as possible for the same reason described in Section II-B. Besides, we also want to obtain as many LACs without conflicts as possible, which leaves more options for the final selection of multiple LACs. Thus, we aim at finding a set S_{sel} of nodes in G_{conf} without mutual connection such that the sum of their weights is as small as possible, while the node number is as large as possible.

We propose a heuristic method to solve the above problem. It initializes the set S_{sel} of selected nodes as empty. Then, it traverses the nodes in G_{conf} in the ascending order of their weights. For each

node being visited, if it does not connect to any existing node in S_{sel}, it is added into S_{sel}. Otherwise, it is skipped. Finally, for each node in set S_{sel}, its corresponding LAC is added into set L_{sol}, and the TN of the LAC is added into set N_{sol}.

Example 4. *Consider the LAC conflict graph G_{conf} in Fig. 3. The list of nodes in G_{conf} in the ascending order of their weights is T_1, T_2, T_3, T_4, T_5, T_6. First, T_1 is added into the set S_{sel}. Since node T_2 connects to T_1, it is skipped. For node T_3, since it does not connect to T_1, the only node in S_{sel}, it is added into set S_{sel}. For the remaining nodes in G_{conf}, we perform a similar procedure. Finally, $S_{sel} = \{T_1, T_3, T_5, T_6\}$. Thus, the LAC set L_{sol} contains $L(\{1\}, 3)$, $L(\{2\}, 4)$, $L(\{5\}, 6)$, and $L(\{8, 9\}, 7)$, and its corresponding TN set N_{sol} contains TNs 3, 4, 6, and 7.*

D. Obtaining Independent LAC Set

This section presents the details of the function *SelectIndpLACs*, which selects a subset of LACs from L_{sol} to form a good independent LAC set L_{indp}. Note that after resolving conflicts, each LAC in set L_{sol} has a unique TN. Thus, we convert this problem to finding a set of nodes N_{indp} from N_{sol} so that the LACs applied to these nodes are more likely to form an independent LAC set. For this purpose, we first define an index measuring the likelihood that the LACs applied to a pair of nodes in N_{sol} form a *dependent* LAC set (*i.e.*, a positive or negative LAC set) in Section II-D1. With the help of the index, Section II-D2 shows how to obtain the set N_{indp}. Then, Section II-D3 shows how to obtain a good independent LAC set based on N_{indp}.

1) Index for Measuring the Likelihood of Two LACs to Form a Dependent LAC Set: Given two nodes n_i and n_j in a circuit, suppose that n_j is before n_i in a topological order. We want to efficiently measure the likelihood that the LACs applied to n_i and n_j form a dependent LAC set. For this purpose, we define a circuit structure-based index p_{ji} by distinguishing two cases:

1. The case where there is a path from node n_j to node n_i. Let $d(n_j, n_i)$ be the length of the shortest path from n_j to n_i. Intuitively, the larger $d(n_j, n_i)$ is, the less likely that the LACs applied to nodes n_j and n_i form a dependent LAC set. Thus, the index p_{ji} is set as $\frac{1}{d(n_j, n_i)}$.
2. The case where there is no path from node n_j to node n_i. In this case, we focus on the *transitive fanouts (TFOs)* of nodes n_j and n_i. Intuitively, the smaller the intersection of the TFOs of nodes n_i and n_j, the less likely that the LACs applied to nodes n_j and n_i form a dependent LAC set. Thus, the index p_{ji} is set as $\frac{|F(n_j) \cap F(n_i)|}{|F(n_j)|}$, where $F(n_i)$ and $F(n_j)$ are the TFOs of node n_i and node n_j, respectively.

2) Obtaining Set N_{indp}: This section shows how we obtain the set N_{indp} from the set N_{sol} using the index defined above.

We first build an undirected graph G_{sol} with the set of nodes as N_{sol}. For each pair of nodes n_i and n_j in G_{sol}, they are connected by an edge in G_{sol} if and only if their index p_{ji} is larger than a given bound t_b. Thus, if two nodes are connected, it is more likely that the LACs applied to them form a dependent LAC set; otherwise, it is less likely. Now, in order to get the set N_{indp} satisfying that the LACs applied to the nodes in N_{indp} are more likely to form a good independent LAC set, we try to pick a maximum number of nodes in N_{sol} such that they are not pairwise connected in the graph G_{sol}. This is just an MIS problem. Thus, by solving such a problem on the graph G_{sol}, we obtain the set N_{indp}. In our implementation, an open-source tool KaMIS [16] is applied to solve the MIS problem.

3) Selecting Independent LAC Set: Given the conflict-free LAC set L_{sol} and the set N_{indp}, we choose the LACs from L_{sol} such that their TNs are in set N_{indp}, which form the potential independent LAC set L_{pote}. If we apply all the LACs in L_{pote} to the current approximate circuit, it may cause a rapid circuit error increase, reducing the total iteration number of the ALS flow and degrading the circuit quality. Thus, conservatively, we only select the top LACs with the smallest error increases from L_{pote} to form the final independent LAC set L_{indp}.

Then, a key question is how to determine the size of L_{indp}. Let it be r_{indp}. To decide r_{indp}, we also introduce a parameter r_{sel} as a reference selected LAC number. We first count the number of LACs with their error increases no more than 0 in L_{pote}, *i.e.*, r_{neg}. If r_{neg} is no smaller than r_{sel}, we set r_{indp} as r_{neg}, and apply all the r_{neg} LACs with error increases no more than 0 to the circuit, which is likely to reduce the circuit area without increasing the circuit error. Otherwise, to avoid increasing the circuit error too rapidly, we set an error limit as λe_b, where λ is a parameter and e_b is the given error bound. Then, among the first r_{sel} LACs in L_{pote} with the smallest error increases, we select the *maximum* number of them such that their total estimated error e_{est} calculated by Eq. (1) is no more than λe_b, which can be done by checking the first r_{sel} LACs in the ascending order of their error increases. For the special case where the LAC with the minimum error increase causes e_{est} exceeding λe_b, we just select the one with the minimum error increase.

E. Improvement Techniques

To avoid synthesis quality drop, we further propose two improvement techniques. The basic idea is to combine the selection of multiple LACs in a round with the selection of a single LAC in a round together. For the following two cases, we only select the best LAC with the minimum error increase in a single round.

1. The error e_{new} of the new approximate circuit $\mathcal{G}_{new}$ is larger than $l_e e_b$, where $l_e \in [0, 1]$ is a parameter close to 1. In this case, the error increases of the candidate LACs for $\mathcal{G}_{new}$ usually are large. If multiple LACs are applied simultaneously, the ALS flow may stop earlier, degrading the synthesis quality.
2. At the end of each round, first, the estimated error e_{est} is calculated according to Eq. (1). Then, we compare the the actual error e_{new} and the estimated error e_{est} to determine whether the selected LAC set is negative. For $e_{new} = 0$, it can be shown that $e_{est} \geq 0$, and hence, the LAC set cannot be negative. For $e_{new} > 0$, we measure the relative error difference β between e_{new} and e_{est}, *i.e.*, $\beta = \frac{e_{new} - e_{est}}{e_{new}}$. If β is larger than a parameter $l_d \in [0, 1]$, it means that the selected multiple LACs form a negative LAC set. In this case, we revert to the approximate circuit obtained in the last round, and only the best LAC with the minimum error increase is selected.

III. Experimental Results

This section shows the experimental results of AccALS. We conduct all the experiments on a 12-core/24-thread Intel Xeon Gold 6146 processor running at 3.2GHz with 144GB RAM, and a single thread is used in our experiments. AccALS can deal with any input distribution and any statistical error metric. In our experiments, the inputs are set as uniformly distributed. Table I lists the benchmark circuits used in our experiments along with their node numbers in their AIG representations, areas, and delays. The circuits include some ISCAS [17], small arithmetic, large EPFL arithmetic [18], and LGSynt91 circuits [19]. They are fully optimized by ABC using the script "*strash*; *resyn2*×10; *amap*". The area and delay of each circuit are normalized to the area and delay of the INV_X1 gate in the MCNC library [19], respectively. For the ISCAS, small arithmetic, and LGSynt91 circuits, to mitigate randomness in the experiments, each experiment is conducted three times, and their average results are reported. Due to the long runtime of EPFL arithmetic circuits, each experiment is conducted only once. The bound t_b in Section II-D2, the parameter λ in Section II-D3, and the parameters l_e and l_d in Section II-E are set as 0.5, 0.9, 0.9, and 0.3, respectively. The reference top LAC number r_{ref} in Section II-B and the reference selected LAC number r_{sel} in Section II-D3 affect the number of selected LACs in each round. Intuitively, given an error bound, the

larger the size of a circuit, the larger the total number of LACs that need to be applied to generate the final approximate circuit. Thus, for circuits with larger sizes, more LACs can be applied in each round to speed up the ALS flow. Through extensive experiments, for the circuits with AIG node number less than 600, we set $r_{ref} = 100$ and $r_{sel} = 20$. For the circuits with AIG node number between 600 and 4999, we set $r_{ref} = 200$ and $r_{sel} = 40$. For the circuits with AIG node number no less than 5000, we set $r_{ref} = 400$ and $r_{sel} = 80$. To evaluate the synthesis quality of approximate circuits, we use *area ratio*, *delay ratio*, and *area-delay-product (ADP) ratio* (*i.e.*, the area/delay/ADP of the approximate circuit over that of the original one). In addition, synthesis runtime is reported.

Table I. Benchmarks used in our experiments. #Nd: number of nodes.

ISCAS & Small arithmetic				EPFL arithmetic				LGSynt91			
Ckt	#Nd	Area	Delay	Ckt	#Nd	Area	Delay	Ckt	#Nd	Area	Delay
alu4	1428	2798	12.7	div	23667	47081	3420.0	alu2	361	622	46.8
c1908	363	758	37.3	log2	38540	64914	541.2	apex6	607	1091	20.3
c3540	915	1604	55.0	sin	7044	12169	254.8	frg2	695	1195	16.6
c880	316	585	24.9	sqrt	21951	44512	3568.1	term1	149	262	23
cla32	420	958	38.5	square	20032	37927	400.8				
ksa32	507	1128	17.8								
mtp8	515	1069	37.8								
rca32	312	666	16.1								
wal8	462	1081	45.3								

A. Statistical Analysis of AccALS

In each round of the AccALS framework, we will obtain two LAC sets, the independent LAC set L_{indp} and the set L_{rand} with random LACs, and select the better one (see Lines 8–12 of Algorithm 1). In this section, we analyze the ratio of the rounds in which the LAC set L_{indp} is selected. We call it L_{indp} *ratio* for short. We test on the last 5 circuits listed in column 1 of Table I (*i.e.*, all the small arithmetic circuits) under three statistical error metrics, *i.e.*, ER, NMED, and MRED. The thresholds of ER, NMED, and MRED are 5%, 0.19531%, and 0.19531%, respectively. Fig. 4 shows the L_{indp} ratio of AccALS for the small arithmetic circuits under three error metrics. For *mtp8*, *rac32*, and *wal8* under ER constraint, the L_{indp} ratio is larger than 0.95. In addition, the average L_{indp} ratios of all the three error metrics are larger than 0.7. This shows that our sophisticatedly-constructed independent LAC set is usually better than the randomly-constructed LAC set.

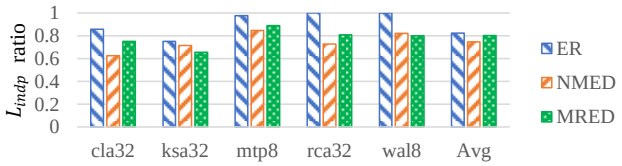


Fig. 4. The L_{indp} ratio of AccALS for the small arithmetic circuits under three error metrics.

B. Comparison with SEALS Method

In this section, we compare AccALS with a state-of-the-art single selection-based ALS method, SEALS [12], which speeds up the iterative ALS flows based on a sensitivity metric. The type of LAC used in both AccALS and SEALS is ALSRAC [9]. Same as SEALS, we use the MCNC library [19] as the technology library.

1) Experiments on Small ISCAS and Arithmetic Circuits: This section compares AccALS with SEALS on the small ISCAS and arithmetic circuits listed in column 1 of Table I. Three statistical error metrics are considered, *i.e.*, ER, NMED, and MRED.

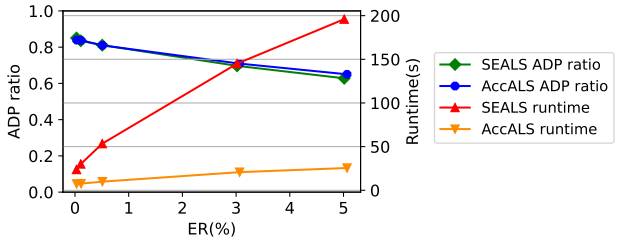


Fig. 5. The average ADP ratio and runtime of the small ISCAS and arithmetic circuits under 5 ER thresholds for AccALS and SEALS.

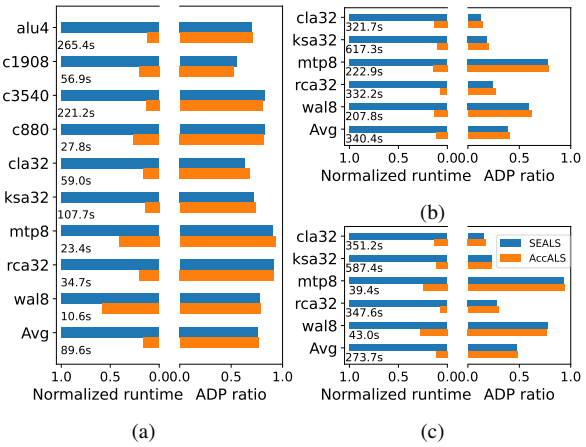


Fig. 6. Comparison between AccALS and SEALS on the small ISCAS and arithmetic circuits: (a) under ER constraint; (b) under NMED constraint; (c) under MRED constraint. The value on each row is the runtime of SEALS.

a) Performance under ER Constraint: We compare AccALS with SEALS under 5 ER thresholds, 0.03%, 0.1%, 0.5%, 3%, and 5%, for all the ISCAS and arithmetic circuits listed in column 1 of Table I. Fig. 5 plots the average ADP ratio and runtime of these circuits versus the ER threshold, where the results for each ER threshold are the average values over all the tested circuits. For both AccALS and SEALS, as ER increases, the average ADP ratio of the circuits decreases, and the average runtime increases. In addition, the average ADP ratios of AccALS are close to those of SEALS under all the ER thresholds. As ER increases, the speedup ratio of AccALS over SEALS gradually increases (up to 7.7× under 5% ER threshold). This demonstrates the effectiveness of AccALS in speeding up the ALS flow under large error thresholds.

Fig. 6(a) further details the average normalized runtime and the ADP ratio of AccALS and SEALS for each circuit. The results of each circuit are the average values under the above 5 ER thresholds. For *c1908*, *c3540*, and *c880*, the approximate circuits synthesized by AccALS have smaller ADP than SEALS. On average, AccALS synthesizes approximate circuits with a slightly (*i.e.*, 0.67%) larger ADP than SEALS, while being 6.3× faster. This shows the effectiveness of AccALS in speeding up the iterative ALS flows.

b) Performance under NMED Constraint: Since NMED is an error metric for arithmetic circuits, we only test on the last 5 circuits listed in column 1 of Table I. The results of each circuit are the average values under 4 NMED thresholds (0.00153%, 0.00610%, 0.02441%, 0.19531%). Fig. 6(b) shows the average normalized runtime and the ADP ratio of AccALS and SEALS under NMED constraint. On average, AccALS synthesizes approximate circuits with a slightly (*i.e.*, 1.74%) larger ADP than SEALS, while being 8.8× faster.

c) Performance under MRED Constraint: Same as NMED, MRED is an error metric for arithmetic circuits. Thus, we use the same circuits and error thresholds as those used in Section III-B1b to compare AccALS with SEALS. Fig. 6(c) shows the average normalized runtime and the ADP ratio of AccALS and SEALS under MRED constraint. On average, AccALS synthesizes approximate circuits with a slightly (*i.e.*, 0.86%) larger ADP than SEALS, while being 8.5× faster.

2) Experiments on Large EPFL Circuits: To show the scalability of AccALS, this section compares AccALS with SEALS under ER constraint for those large EPFL circuits in column 5 of Table I. The ER threshold is set as 0.1%. Table II lists the area and delay ratios and the runtime of AccALS and SEALS. We highlight the data in **bold** when AccALS outperforms SEALS. From Table II, the approximate circuits of *div* and *sqrt* obtained by AccALS have smaller areas than those obtained by SEALS. On average, AccALS synthesizes approximate circuits with a negligible area increase (*i.e.*, 0.26%) and delay increase (*i.e.*, 1.16%) compared to SEALS. For these EPFL circuits, the runtime of SEALS is extremely long. In particular, the runtime of SEALS for *div* is 59.1 hours. By using AccALS, the runtime reduces to 2.4 hours, which is a significant improvement. Additionally, it is notable that by using AccALS, the area and delay of *div* both reduce. On average, AccALS is 24.6× faster than SEALS. Compared to the experiments on the small circuits in Section III-B1, AccALS shows more speedup on large circuits with a negligible circuit quality loss, which demonstrates the scalability of AccALS.

Table II. Comparison between AccALS and SEALS on the large EPFL arithmetic circuits under the ER threshold of 0.1%.

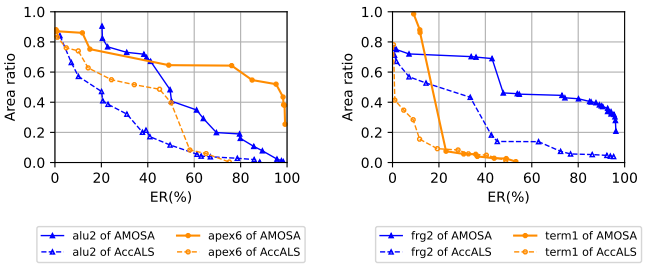
circuit	area ratio		delay ratio		time (s)	
	AccALS	SEALS	AccALS	SEALS	AccALS	SEALS
div	**26.93%**	27.18%	**27.22%**	28.66%	**8703**	212924
log2	90.60%	90.27%	**94.38%**	94.62%	**4535**	126404
sin	96.58%	95.28%	96.70%	89.91%	**589**	10369
sqrt	**78.30%**	78.39%	79.56%	79.55%	**5556**	133557
square	99.78%	99.76%	98.95%	98.23%	**617**	8026
Avg	78.44%	78.18%	79.36%	78.20%	**4000**	98256

C. Comparison with AMOSA Method

This section further compares AccALS with a recent evolutionary-based ALS method that selects multiple LACs in a round [15]. We call it *AMOSA* for short, since it applies the AMOSA heuristic to select multiple LACs. We consider ER constraint. The type of LAC used in AccALS is ALSRAC [9]. Same as AMOSA, we use the 45nm NanGate standard-cell library [20] as the technology library. The LGSynt91 circuits listed in column 9 of Table I are selected for evaluation, since they are also used in [15]. The results of AMOSA are taken from [15]. For each circuit, we apply AccALS to iteratively generate the approximate circuits with the ER bound as the maximum ER of the AMOSA design.

For the hardware cost, same as AccALS, AMOSA focuses on reducing circuit area instead of delay. Thus, we compare the area of AccALS with that of AMOSA. Fig. 7 plots the area ratio-ER curves of the approximate designs synthesized by AccALS and AMOSA for the LGSynt91 circuits. For all the circuits, AccALS always reduces more area than AMOSA, except for *term1* under the ERs larger than 20%. Especially, for *alu2*, *apex6*, and *term1*, the area ratios of AccALS are up to 50% smaller than those of AMOSA. This shows that AccALS outperforms AMOSA in circuit area optimization.

Table III lists the synthesis time for AccALS and AMOSA for a single run. Note that the results of AMOSA are taken from [15], and the experiments in [15] are conducted on a processor better than ours. From Table III, the runtime of AccALS is always smaller than that of AMOSA. On average, AccALS is 13× faster than AMOSA.



(a) alu2 and apex6 (b) frg2 and term1

Fig. 7. The area ratio of AccALS and AMOSA for the 4 LGSynt91 circuits under ER constraint.

Table III. The runtime (s) for the LGSynt91 circuits.

method	alu2	apex6	frg2	term1	average
AMOSA	960	360	1380	3120	1455
AccALS	16.91	158.69	263.52	5.77	111.23

IV. Conclusion

In this work, we propose AccALS, accelerating the iterative ALS flows by selecting multiple LACs in a single round. First, an efficient technique is designed to solve the LAC conflicts. Then, an index is proposed to efficiently measure the mutual influence between two LACs. With its help, an MIS problem is formulated and solved to select a maximal set of LAC with less mutual influence. AccALS outperforms a state-of-the-art method in runtime with a negligible circuit quality loss.

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 *ETS*, 2013, pp. 1–6.

[3] Q. Xu *et al.*, "Approximate computing: A survey," *IEEE Des. Test.*, vol. 33, no. 1, pp. 8–22, 2016.

[4] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, 62:1–62:33, 2016.

[5] I. Scarabottolo *et al.*, "Approximate logic synthesis: A survey," *Proc. IEEE*, vol. 108, no. 12, pp. 2195–2213, 2020.

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

[7] S. Venkataramani *et al.*, "Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits," in *DATE*, 2013, pp. 1367–1372.

[8] Y. Wu and W. Qian, "An efficient method for multi-level approximate logic synthesis under error rate constraint," in *DAC*, 2016, pp. 1–6.

[9] C. Meng *et al.*, "ALSRAC: Approximate logic synthesis by resubstitution with approximate care set," in *DAC*, 2020, pp. 1–6.

[10] J. Ma *et al.*, "Approximate logic synthesis using Boolean matrix factorization," *IEEE TCAD*, vol. 41, no. 1, pp. 15–28, 2021.

[11] S. Su *et al.*, "VECBEE: A versatile efficiency-accuracy configurable batch error estimation method for greedy approximate logic synthesis," *IEEE TCAD*, 2022.

[12] C. Meng *et al.*, "SEALS: Sensitivity-driven efficient approximate logic synthesis," in *DAC*, 2022, pp. 439–444.

[13] Z. Zhou *et al.*, "DALS: Delay-driven approximate logic synthesis," in *ICCAD*, 2018, pp. 1–7.

[14] J. Echavarria *et al.*, "Approximate logic synthesis of very large Boolean networks," in *DATE*, 2021, pp. 1552–1557.

[15] M. Barbareschi *et al.*, "A catalog-based AIG-rewriting approach to the design of approximate components," *IEEE Trans. Emerg. Topics Comput.*, 2022.

[16] S. Lamm *et al.*, "Finding near-optimal independent sets at scale," in *ALENEX*, 2016, pp. 138–150.

[17] M. Hansen *et al.*, "Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering," *IEEE DTC*, vol. 16, no. 3, pp. 72–80, 1999.

[18] EPFL, *The EPFL combinational benchmark suite*, https://lsi.epfl.ch/page-102566-en-html/benchmarks/, 2021.

[19] S. Yang, "Logic synthesis and optimization benchmarks," Microelectronics Center of North Carolina, Tech. Rep., 1991.

[20] Nangate, Inc., *Nangate 45nm open cell library*, https://si2.org/open-cell-library/, 2022.