SEALS: Sensitivity-driven Efficient Approximate Logic Synthesis

Chang Meng¹, Xuan Wang¹, Jiajun Sun¹, Sijun Tao¹, Wei Wu², Zhihang Wu², Leibin Ni², Xiaolong Shen², Junfeng Zhao², and Weikang Qian¹,³

¹UM-SJTU Joint Inst. and ³MoE Key Lab of AI, Shanghai Jiao Tong University, China; ²Huawei Technologies Co., China

ABSTRACT

Approximate computing is an emerging computing paradigm to design energy-efficient systems. Many greedy approximate logic synthesis (ALS) methods have been proposed to automatically synthesize approximate circuits. They typically need to consider all local approximate changes (LACs) in each iteration of the ALS flow to select the best one, which is time-consuming. In this paper, we propose SEALS, a Sensitivity-driven Efficient ALS method to speed up an greedy ALS flow. SEALS centers around a newly proposed concept called sensitivity, which enables a fast and accurate error estimation method and an efficient method to filter out unpromising LACs. SEALS can handle any statistical error metric. The experimental results show that it outperforms a state-of-the-art ALS method in runtime by 12× to 15× without reducing circuit quality.

KEYWORDS

approximate computing, approximate logic synthesis, partial difference, sensitivity, error estimation

1 INTRODUCTION

In the post-Moore era, how to sustain the energy-efficiency improvement of computing systems becomes a great challenge. Meanwhile, many error-tolerant applications are widely used today, including image processing, data mining, and machine learning. Under this circumstance, approximate computing, an emerging computing paradigm, was proposed to design energy-efficient systems [1, 2]. Its basic idea is to deliberately introduce slight imprecision in computation to improve circuit area, delay, and power consumption.

One topic of approximate computing is approximate logic synthesis (ALS), which aims to find an optimal approximate circuit for a target circuit under a relaxed error bound [3]. Various ALS methods have been proposed [4–13]. Among them, one popular category is the greedy ALS methods due to their good synthesis quality [4, 5, 8, 12]. They iteratively choose the best local approximate change (LAC) from the candidate LACs and apply it to the circuit. In [4], by setting a wire as a constant 0 or 1, the circuit is simplified under error rate (ER) or error distance (ED) constraint. In [5], an ALS method called SASIMI is proposed, which replaces a signal in the circuit by another almost identical signal or its negation to synthesize an approximate circuit. It can deal with ER or mean ED (MED) constraint. In [8], a greedy ALS method is proposed under ER constraint, which is based on Boolean expression simplification of the nodes in a Boolean network. In [12], an ALS approach called ALSRAC is proposed to handle ER or ED constraint. It performs approximate resubstitution to obtain approximate circuits. For all these greedy ALS methods, in each of their iterations, all the candidate LACs in the circuit need to be evaluated, before the best one is selected and applied to the circuit. Thus, if many LACs exist, it takes a long runtime to evaluate all of them.

Notably, all these greedy ALS methods need to evaluate the errors of all the LACs in a circuit in each iteration. It is shown in [14] that an accurate error estimation is important to boost their synthesis quality. Traditionally, the simulation-based method is used for accurate error estimation, which applies each LAC to the current approximate circuit and simulates the new circuit to obtain the errors. However, this method is time-consuming. Thus, more and more attention has been paid to fast error estimation of approximate circuits used within an ALS flow [14–16]. The work [15] propagates the signal distributions of cascaded sub-circuits to estimate the bit error rate of the final output. However, the ED metric is not considered, and the reconvergent paths influence its estimation accuracy. In [16], an error propagation model is applied to each sub-circuit to obtain the maximum ED of the final output. Nevertheless, it cannot handle common statistical error metrics such as ER and MED. In [14], a batch error estimation method based on Monte Carlo (MC) simulation and local change propagation is proposed to ensure both the accuracy and efficiency of error estimation. However, as shown in [14], it still takes much time to perform error estimation even for small approximate circuits under ER constraint. Thus, how to improve the speed of a greedy ALS flow while maintaining the error estimation accuracy is still a challenge.

In this paper, to address the aforementioned challenge, we propose SEALS, a Sensitivity-driven Efficient ALS method to speed up the greedy ALS flow. SEALS can be applied to any input distribution, any graph-based circuit representation, and any statistical error metric, such as ER and MED.

The main contributions of our work are as follows.

1. We show that a common operation in a greedy ALS flow, i.e., calculating the error increase by a LAC, can be reduced to calculating a newly proposed metric called sensitivity, which essentially captures how sensitive the error change is with respect to the change of a node value.
2. We further demonstrate that the sensitivities of all the nodes in a circuit can be efficiently obtained, thus enabling an efficient and accurate error estimation. The method to obtain sensitivities is based on calculating another newly proposed metric called partial difference (PD). We show that the PDs of all the nodes in a circuit can be calculated efficiently in a recursive way.
3. We propose the entire SEALS flow, which applies sensitivities to calculate the error increases of LACs. It also applies an efficient LAC filtering method for acceleration. The method filters out the nodes with high sensitivities and the LACs with large error increases, thus greatly reducing the number of candidate LACs that need to be accurately evaluated in each iteration.

The experimental results show that compared with a state-of-the-art method, SEALS accelerates by 12x to 15x under three different error metrics over a wide range of circuits. The rest of the paper is organized as follows. Section 2 elaborates our method to calculate error increase, which is a key component in SEALS. Section 3 presents the SEALS methodology. Section 4 shows the experimental results. Section 5 concludes the paper.

2 CALCULATING ERROR INCREASE

This section presents a key step in SEALS, which is to calculate the error increase of a given circuit due to an applied LAC.

2.1 Error Increase Calculation and Sensitivity

This section shows that the error increase calculation can be reduced to calculating a newly defined metric called sensitivity. We first describe some assumptions and notations used in this paper.

- In order to flexibly handle any input distribution and any large circuit, the error estimation is based on MC simulation [14]. Assume that \( m \) random input patterns are sampled from the given input distribution. We refer to the \( i \)-th \( (1 \leq i \leq m) \) random input pattern as input pattern \( i \).
- Denote the original input circuit as \( G_{\text{org}} \) and the current approximate circuit (i.e., the given circuit) as \( G \).
- Denote the set of nodes in \( G \) as \( N \) and the set of POs of \( G \) as \( O \).
- Suppose that a LAC \( \psi \) is applied to \( G \) and the resulting new approximate circuit is \( G_{\text{new}} \).
- Suppose that the LAC \( \psi \) affects a local circuit \( K \) in the current circuit \( G \) \( (K \subseteq G) \) and the resulting new local circuit is \( K_{\text{new}} \) \( (K_{\text{new}} \subseteq G_{\text{new}}) \). One assumption in our work is that we only consider LACs such that the affected local circuits have a single output. We note that many existing LACs satisfy this property, such as those proposed in [4, 5, 8, 12].
- Denote the single output node of \( K \) and \( K_{\text{new}} \) as \( n \) and \( n_{\text{new}} \), respectively. Denote their values under input pattern \( i \) as \( V_n(i) \) and \( V_{n_{\text{new}}}(i) \), respectively.
- Denote the binary numbers represented by the POs of \( G_{\text{org}}, G, \) and \( G_{\text{new}} \) under input pattern \( i \) as \( Y_{\text{org}}(i), Y(i), \) and \( Y_{\text{new}}(i), \) respectively.
- In our proposed method, we simulate circuits with the \( m \) random input patterns to obtain some values to be used later. Specifically, we simulate the original circuit \( G_{\text{org}} \) to obtain an output value vector \( Y_{\text{org}} = (Y_{\text{org}}(1), \ldots, Y_{\text{org}}(m)) \). We also simulate the current approximate circuit \( G \) to obtain a node value matrix \( M \). Each entry in \( M \), denoted as \( M[i,n] \) \((1 \leq i \leq m, n \in N)\), represents the value of node \( n \) under input pattern \( i \).
- Denote the error between two values \( v_1 \) and \( v_2 \) as \( f(v_1, v_2) \).

With the MC simulation, the error of \( G \) can be calculated as \[ \frac{1}{m} \sum_{i=1}^{m} f(Y(i), Y_{\text{org}}(i)) \] and that of \( G_{\text{new}} \) can be calculated similarly. Note that with a proper choice of \( f \), various error metrics can be calculated. For example, if \( f(v_1, v_2) \) returns 1 if \( v_1 \neq v_2 \) and 0 otherwise, then ER is calculated.

For the LAC \( \psi \), we want to calculate the error increase \( \Delta E(\psi) \) between \( G_{\text{new}} \) and \( G \), which equals the error of \( G_{\text{new}} \) minus that of \( G \). With the MC simulation, it can be calculated as follows:

\[ \Delta E(\psi) = \frac{1}{m} \sum_{i=1}^{m} (f(Y_{\text{org}}(i), Y_{\text{org}}(i)) - f(Y(i), Y_{\text{org}}(i))). \] (1)

To calculate \( \Delta E \) by Eq. (1), it reduces to calculating the expression \( e_1 = f(Y_{\text{new}}(i), Y_{\text{org}}(i)) - f(Y(i), Y_{\text{org}}(i)) \), which is essentially the error increase under input pattern \( i \). Next, we prove that \( e_1 = e_2 \), with \( e_2 \) as follows:

\[ e_2 = (V_{\text{new}}(i) - V_n(i)) \times \left( f(\hat{Y}(i), V_n = 1) - f(\hat{Y}(i), V_n = 0), Y_{\text{org}}(i)) \right), \] (2)

where \( \hat{Y}(i, V_n = u) (u \in \{0, 1\}) \) is the binary number represented by the POs of \( G \) under input pattern \( i \), if we set the value of \( n \) as \( u \).

It can be proved by considering the following 3 cases:

1. The case where \( V_n(i) = V_{n_{\text{new}}}(i) \). In this case, the LAC \( \psi \) does not change the output value of the local circuit under input pattern \( i \), and hence, does not affect the POs. Thus, the error increase \( e_1 \) is 0. Since \( V_n(i) = V_{n_{\text{new}}}(i) \), by Eq. (2), we also have \( e_2 = 0 = e_1 \).

2. The case where \( V_n(i) = 0 \) and \( V_{n_{\text{new}}}(i) = 1 \). In this case, the LAC \( \psi \) causes a 0 to 1 change on the output \( n \) of the local circuit under input pattern \( i \). The binary number represented by the POs of the current approximate circuit changes from \( \hat{Y}(i, V_n = 0) = \hat{Y}(i, V_n = 1) \), and hence, the error increase \( e_1 \) should be \( f(\hat{Y}(i, V_n = 1), Y_{\text{org}}(i)) - f(\hat{Y}(i, V_n = 0), Y_{\text{org}}(i)) \), which equals \( e_2 \).

3. The case where \( V_n(i) = 1 \) and \( V_{n_{\text{new}}}(i) = 0 \). It is symmetric to case 2 and the claim can be proved similarly.

An important component in Eq. (2) is \[ \Delta L[i,n] = f(\hat{Y}(i, V_n = 1), Y_{\text{org}}(i)) - f(\hat{Y}(i, V_n = 0), Y_{\text{org}}(i)). \] (3)

It represents the error increase by changing the value of \( n \) from 0 to 1. Intuitively, if \( \Delta L[i,n] \) is larger, then the value change on node \( n \) will cause a larger change on error, and hence, the error is more sensitive to the node \( n \). Thus, we refer to \( \Delta L[i,n] \) as error sensitivity, or sensitivity in short. The values of \( \Delta L[i,n] \) for all the \( m \) random input patterns and all the nodes in \( G \) form a sensitivity matrix (SM) \( \Delta L \). We will show how to obtain the SM in Section 2.2.

Based on the above discussion, the error increase \( \Delta E(\psi) \) by a LAC can be obtained through the SM by the function \( \text{ErrorEstimate} \) shown in Algorithm 1. The inputs of Algorithm 1 are a LAC \( \psi \), the current approximate circuit \( G \), the node value matrix \( M \) of \( G \) obtained from the MC simulation, and the SM \( \Delta L \). The output is the estimated error increase \( \Delta E(\psi) \) by the LAC \( \psi \).

**Algorithm 1:** The function \( \text{ErrorEstimate}(\psi, G, M, \Delta L) \) to estimate the error increase by a LAC.

**Input:** a LAC \( \psi \), a circuit \( G \), a node value matrix \( M \) of size \( m \times |N| \), and a SM \( \Delta L \).

**Output:** estimated error increase \( \Delta E(\psi) \).

1. \((n, n_{\text{new}}) \leftarrow \) outputs of local circuits \( K \) and \( K_{\text{new}} \) affected by \( \psi \);
2. Obtain the signal value vector \( V_n \) of \( n \) from \( M \);
3. Obtain the signal value vector \( V_{n_{\text{new}}} \) of \( n_{\text{new}} \) by simulating \( K_{\text{new}} \);
4. \( \Delta E(\psi) \leftarrow 0; \)
5. for \( i \) from 1 to \( m \) do
6. \( \Delta E(\psi) \leftarrow \Delta E(\psi) + (V_{n_{\text{new}}}(i) - V_n(i)) \cdot \Delta L[i,n]; \)
7. return \( \Delta E(\psi) \).
In Algorithm 1, Line 1 identifies the output nodes \( n \) and \( n_{\text{new}} \) of the local circuits \( \mathcal{K} \) and \( \mathcal{K}_{\text{new}} \) affected by the LAC \( \psi \), respectively. We use two signal value vectors \( \mathbf{V}_n \) and \( \mathbf{V}_{n_{\text{new}}} \) to store the values of the nodes \( n \) and \( n_{\text{new}} \) under all the \( m \) random input patterns, respectively. Line 2 obtains \( \mathbf{V}_n \) from the node value matrix \( \mathbf{M} \). Line 3 obtains \( \mathbf{V}_{n_{\text{new}}} \) by simulating the local circuit \( \mathcal{K}_{\text{new}} \) with inputs taken from \( \mathbf{M} \). Since \( \mathcal{K}_{\text{new}} \) is usually small, the simulation process only costs little time. Then, Lines 4–6 accumulate the error increase by the LAC \( \psi \) based on Eqs. (2) and (3).

2.2 Obtaining Sensitivity

We show how to obtain the SM \( \Delta \mathbf{L} \) in this section. By Eq. (3), we need to obtain \( \hat{Y}(i, V_n = 1) \), \( \hat{Y}(i, V_n = 0) \), and \( Y_{\text{org}}(i) \) to calculate \( \Delta L[i, n] \). As we stated in Section 2.1, \( Y_{\text{org}}(i) \) is obtained by the MC simulation. Thus, we focus on obtaining \( \hat{Y}(i, V_n = 1) \) and \( \hat{Y}(i, V_n = 0) \). Next, we show how to obtain them by first considering the case where \( V_n(i) = 1 \). Since \( V_n(i) = 1 \), by definition, we have

\[
\hat{Y}(i, V_n = 1) = Y(i).
\]

With the node value matrix \( \mathbf{M} \) available, we can get the values of all the POs of \( \mathcal{G} \) and then get the value of \( Y(i) \). To obtain \( \hat{Y}(i, V_n = 0) \), a direct method is to simulate \( \mathcal{G} \) again by flipping the value of \( n \) under input pattern \( i \), which is time-consuming. Instead, we propose to get \( \hat{Y}(i, V_n = 0) \) first, which is called PD. We use a two-dimensional PD matrix (PDM) \( \Delta \mathbf{L} \) to store the PD for each node \( n \) \( \in \mathcal{N} \) under each random input pattern. Each entry in \( \Delta \mathbf{L} \) is denoted as:

\[
\Delta \mathbf{Y}[i, n] = \hat{Y}(i, V_n = 1) - \hat{Y}(i, V_n = 0).
\]

Different from Boolean difference, which indicates whether a change of an internal node causes a change on POs or not, PD gives the exact magnitude change of the binary number represented by the POs with respect to an internal node change. We will show an efficient method to get the PDM in Section 2.3. Now, we assume that the PDM is obtained. Then, by Eqs. (4) and (5), we can get

\[
\hat{Y}(i, V_n = 0) = \hat{Y}(i, V_n = 1) - \Delta \mathbf{Y}[i, n] = Y(i) - \Delta \mathbf{Y}[i, n].
\]

Then, by Eqs. (3), (4), (5), and (6), we can get the sensitivity

\[
\Delta \mathbf{L}[i, n] = f(Y(i), Y_{\text{org}}(i)) - f(Y(i) - \Delta \mathbf{Y}[i, n], Y_{\text{org}}(i)).
\]

For the other case where \( V_n(i) = 0 \), we can similarly get

\[
\Delta \mathbf{L}[i, n] = f(Y(i) + \Delta \mathbf{Y}[i, n], Y_{\text{org}}(i)) - f(Y(i), Y_{\text{org}}(i)).
\]

Based on the above discussion, we present the procedure GetSen-Matrix to get the SM \( \Delta \mathbf{L} \) in Algorithm 2. The inputs are the current approximate circuit \( \mathcal{G} \), the node value matrix \( \mathbf{M} \) of \( \mathcal{G} \), and the output value vector \( \hat{Y}_{\text{org}} \) of the original circuit \( \mathcal{G}_{\text{org}} \). The output is the SM \( \Delta \mathbf{L} \). In Algorithm 2, Line 1 applies the method to be described in Section 2.3 to calculate the PDM \( \Delta \mathbf{Y} \) of \( \mathcal{G} \) based on the matrix \( \mathbf{M} \). Then, for each \( 1 \leq i \leq m \), Line 3 extracts the value \( Y_{\text{org}}(i) \) from \( \hat{Y}_{\text{org}} \), and calculates the approximate output value \( \hat{Y}(i) \) from \( \mathbf{M} \). Then, for each node \( n \) in \( \mathcal{G} \), if \( M[i, n] = 1 \), which means that \( V_n(i) = 1 \), Line 5 calculates \( \Delta \mathbf{L}[i, n] \) by Eq. (7). Otherwise, Line 6 computes \( \Delta \mathbf{L}[i, n] \) by Eq. (8).

2.3 Obtaining Partial Difference Matrix

This section presents our proposed efficient method to obtain the PDM. It obtains the PD for each node \( n \) \( \in \mathcal{N} \) in a reverse topological order of the circuit. First, the PD for each PO is obtained. Denote the \( k \)-th \((0 \leq k < |\mathcal{O}|)\) node in the PO set \( \mathcal{O} \) of \( \mathcal{G} \) as \( o_k \),

\[
\Delta W_{o_k}[i, n] = W_{o_k}(i, V_n = 1) - W_{o_k}(i, V_n = 0),
\]

which has a weight of \( 2^k \) in the binary representation. By Eq. (5) and the definition of \( \hat{Y}(i, V_n = u) \), we have

\[
\Delta Y[i, o_k] = \hat{Y}(i, V_{o_k} = 1) - \hat{Y}(i, V_{o_k} = 0) = 2^k.
\]

Next, we present our method for calculating the PD for each node \( n \) \( \in \mathcal{N} \). We begin with some definitions.

- In a circuit, if there exists a path from node \( A \) to \( B \), then \( B \) is a transitive fanout (TFO) of \( A \).
- A cut of a node \( n \) in a circuit is a set of nodes in the circuit satisfying that 1) each path from \( n \) to a PO passes at least one node in the set and 2) no node in the set can be removed without violating condition 1.
- The disjoint cut \( S_n \) for a node \( n \) is a cut not equal to \( \{n\} \) but closest to \( n \) satisfying that the TFOs of all nodes in the cut have no mutual intersection.

To obtain the disjoint cut for a node \( n \), a node set \( S_n \) is initialized as the direct fanouts of \( n \). If the TFOs of some nodes in \( S_n \) intersect, then the node \( c \) \( \in S_n \) with the smallest topological order is removed from \( S_n \), and \( c \)'s fanouts are added into \( S_n \). We repeat this step until the TFOs of all nodes in \( S_n \) have no mutual intersection.

**Example 1.** Consider an example circuit in Fig. 1. A topological order of it is \( I_0, I_1, I_2, I_3, I_4, a, b, c, d, o_0, o_1 \). We want to get the disjoint cut \( S_{a_0} \) for node \( a_0 \). \( S_{a_0} \) is initialized as the direct fanouts of \( a_0 \), i.e., \( S_{a_0} = \{b, c\} \). However, the TFOs of \( b \) and \( c \) intersect at node \( o_0 \). Since \( b \) precedes \( c \) in the topological order, we replace \( b \) by its direct fanout \( a_0 \) and \( S_{a_0} \) is updated to \( S_{a_0} = \{c, o_0\} \). This step is repeated until the TFOs of all nodes in \( S_{a_0} \) have no intersection. Finally, we obtain the disjoint cut for node \( a_0 \) as \( S_{a_0} = \{d, o_0\} \). For node \( d \), we can obtain its disjoint cut as \( S_d = \{o_1\} \) in a similar way.

![Figure 1: An example circuit.](image-url)
where $W_c(i, V_n = u)$ ($u \in \{0, 1\}$) is the value of node $c$ under input pattern $i$ if we set the value of $n$ as $u$.

Our method to calculate the PD of a node $n \in N \setminus O$ is based on the theorem below.

**Theorem 1.** For a node $n \in N \setminus O$, we have

$$\Delta Y[i, n] = \sum_{c \in S_n} \Delta Y[i, c] \Delta W_c[i, n],$$

where $S_n$ is the disjoint cut of $n$.

**Proof.** In the proof, we focus on one specific input pattern, so the input pattern index $i$ is omitted for simplicity.

Suppose that $S_n = \{c_1, \ldots, c_r\}$ and for $1 \leq j \leq r$, the set of POs that are TPOs of $c_j$ is $O_j$. By the definition of disjoint cut, the sets $O_1, \ldots, O_r$ are mutually disjoint. Let $\Omega = \bigcup_{j=1}^{r} O_j$. Clearly, $\Omega$ is the set of POs that are TPOs of $n$.

By the definition of $\bar{Y}(V_n = u)$ and $W_c(V_n = u)$, we have

$$\bar{Y}(V_n = u) = \sum_{c \in \Omega} \bar{Y}(V_n = 1) - \bar{Y}(V_n = 0) = \sum_{c \in \Omega} \Delta W_c[c].$$

For any $o_k \in \Omega \setminus \Omega$, since $o_k$ is not a TFO of $n$, we have $W_{o_k}(V_n = 1) = W_{o_k}(V_n = 0)$. Combining this fact with Eqs. (5), (10), and (12), we have

$$\Delta Y[n] = \bar{Y}(V_n = 0) - \bar{Y}(V_n = 0) = \sum_{o_k \in \Omega} \Delta W_{o_k}[n].$$

Similarly, we can get that for any $c_j \in S_n$,

$$\Delta Y[c_j] = \sum_{o_k \in O_j} \Delta W_{o_k}[c_j].$$

Now, we prove that for any $o_k \in \Omega$,

$$\Delta W_{o_k}[c_j] \Delta W_{o_k}[n] = \Delta W_{o_k}[n].$$

We distinguish among 3 possible cases of $\Delta W_{o_k}[c_j]$.

1. $\Delta W_{o_k}[c_j] = 0$. By Eq. (10), we have $W_{c_j}(V_n = 1) = W_{c_j}(V_n = 0)$, which means that $c_j$ is a constant independent of the value of $n$. Since $c_j$ is a node in the disjoint cut $S_n$, all paths from $n$ to $o_k$ pass through $c_j$. Thus, $o_k$ is also a constant independent of the value of $n$, i.e., $W_{o_k}(V_n = 1) = W_{o_k}(V_n = 0)$. Thus, by Eq. (10), $\Delta W_{o_k}[n] = 0$.

2. $\Delta W_{o_k}[c_j] = 1$. By Eq. (10), we have $W_{c_j}(V_n = 1) = 1$ and $W_{c_j}(V_n = 0) = 0$. Thus, the value of node $c_j$ equals that of node $n$. Since all paths above, all paths from $n$ to $o_k$ pass through $c_j$, we have that for any $u \in \{0, 1\}$, $W_{o_k}(V_n = u) = W_{o_k}(V_c = u)$. By Eq. (10), we have $\Delta W_{o_k}[n] = \Delta W_{o_k}[c_j] = \Delta W_{o_k}[c_j] \Delta W_{o_k}[n].$

3. $\Delta W_{o_k}[c_j] = -1$. This case is symmetric to case 2 and the claim can be proved similarly.

By Eqs. (13), (14), and (15), we finally conclude that

$$\Delta Y[n] = \sum_{o_k \in \Omega} \Delta W_{o_k}[n] = \sum_{c_j \in S_n} \sum_{o_k \in O_j} \Delta W_{o_k}[c_j] \Delta W_{o_k}[n].$$

By Theorem 1, we can simply calculate the PDs of all nodes $n \in N \setminus O$ in a reverse topological order of the circuit based on Eq. (11), starting from $\Delta Y[i, o_k]$’s for $o_k \in O$, which are obtained by Eq. (9). In order to apply Eq. (11), we also need to obtain the local PD for each node $c$ in the disjoint cut $S_n$ with respect to $n$. By Eq. (10), we need to get $W_c(i, V_n = 1)$ and $W_c(i, V_n = 0)$. We first extract one of them from the node value matrix $M$ based on the actual value of $n$, and then get the other by flipping the value of the node $n$ and simulating the local circuit between $n$ and $c$. Note that the local circuit is usually small, so the simulation process takes little time.

Below shows an example of getting PDs by Eq. (11).

**Example 2.** Consider an example circuit in Fig. 1. Assume that input pattern $i$ is $01010$. The value of each wire is shown above the wire under this input pattern. We want to calculate PD $\Delta Y[i, a]$ for node $a$. From Example 1, the disjoint cut of node $a$ is $S_a = \{d, o_1\}$ and that of $d$ is $S_d = \{a_1\}$. By Eq. (9), $\Delta Y[i, o_1] = 2^1 = 2$ and $\Delta Y[i, a_1] = 2^0 = 1$. By Eq. (10), $\Delta W_{o_1}(i, d) = W_{o_1}(i, V_d = 1) - W_{o_1}(i, V_d = 0) = 1, \Delta W_d(i, a) = 0, \Delta W_{o_1}(i, a) = 1$. Thus, by Eq. (11), we have $\Delta Y[i, d] = \Delta Y[i, a_1] \Delta W_{o_1}[i, d] = 2$ and

$$\Delta Y[i, a] = \Delta Y[i, d] \Delta W_d[i, a] + \Delta Y[i, o_1] \Delta W_{o_1}[i, a] = 1.$$

### 3 SEALS METHODOLOGY

This section presents the proposed SEALS method to speed up the iterative greedy ALS flow. The procedure of SEALS is shown in Algorithm 3. The inputs are an original circuit $G_{org}$, an error bound $e_b$, a node number ratio $l_n$, and a LAC number ratio $l_l$. The output is an approximate circuit $G$ with error no more than $e_b$. The algorithm uses two kinds of MC simulations: 1) MC simulation with a small number of input patterns for quickly filtering out some promising LACs (Lines 4–12); 2) MC simulation with a large number of input patterns for accurate error estimation (Lines 13–18).

**Algorithm 3:** The proposed flow of SEALS.

**Input:** an original circuit $G_{org}$, an error bound $e_b$, a node number ratio $l_n$, and a LAC number ratio $l_l$.

**Output:** an approximate circuit $G$ with error $e \leq e_b$.

1. $G_{new} \leftarrow G_{org}$; $e \leftarrow 0$;
2. while $e \leq e_b$ do
3. $G \leftarrow G_{new}$;
4. generate a small set $Z_1$ of $J$ random input patterns;
5. simulate $G_{org}$ with $Z_1$ to obtain the output value vector $\bar{Y}_{org}$;
6. simulate $G$ with $Z_1$ to obtain the node value matrix $M_1$;
7. $M_1 \leftarrow$ GetSenMatrix($G$, $M_1$, $\bar{Y}_{org}$);
8. qualified node set $S_n \leftarrow$ NodeFilter($G$, $l_n$, $l_l$);
9. identify candidate LAC set $S_{new}$ from $S_n$;
10. $\Delta E_l[\cdot] \leftarrow$ ErrorEstimate($\cdot$, $G$, $M_1$, $\bar{Y}_{org}$);
11. $\Delta E_l[\cdot] \leftarrow$ LACFilter($S_{new}$, $Z_1$, $l_l$);
12. generate a large set $Z_2$ of $K$ random input patterns;
13. simulate $G_{org}$ with $Z_2$ to obtain the output value vector $\bar{Y}_{org}$;
14. simulate $G$ with $Z_2$ to obtain the node value matrix $M_2$;
15. $M_2 \leftarrow$ GetSenMatrix($G$, $M_2$, $\bar{Y}_{org}$);
16. $\Delta E_l[\cdot] \leftarrow$ ErrorEstimate($\cdot$, $G$, $M_2$, $\bar{L}_2$);
17. $\Delta E_l[\cdot] \leftarrow$ ErrorEstimate($\cdot$, $G$, $M_2$, $\bar{L}_2$);
18. choose the LAC $y^*$ with the smallest $\Delta E_l[y^*]$ and then apply it to a copy of $G$ to obtain $G_{new}$;
19. calculate the accurate error $e$ between $G_{new}$ and $G_{org}$;
20. return $G$;

In Algorithm 3, Line 1 initializes the new circuit $G_{new}$ as the original circuit $G_{org}$, and sets the actual error $e$ of $G_{new}$ as 0. In the main loop, Line 3 sets the current approximate circuit $G$ as $G_{new}$. Line 4 generates a small number of $J$ random input patterns by sampling a given input distribution. Lines 5–6 apply them to
simulate the circuits $G_{org}$ and $G$ to obtain the output value vector $\tilde{Y}_{org}$ and the node value matrix $M_1$, respectively.

Based on the MC simulation with a small number of $J$ input patterns, Lines 7–8 first filter out some unpromising nodes with high sensitivities to obtain the qualified node set $S_N$. Then, for each node in $S_N$, Lines 9–12 filter out some unpromising LACs with large error increases to obtain the qualified LAC set $S_G$. Specifically, Line 7 first calls the function GetSenMatrix shown in Algorithm 2 to obtain the SM $\Delta L_1$ for the current approximate circuit $G$ based on $M_1$ and $\tilde{Y}_{org}$. According to the SM $\Delta L_1$ and the node number ratio $I_n$, Line 8 calls the function NodeFilter to filter some unpromising nodes to obtain the qualified node set $S_N$ for $G$. At this moment, without knowing the actual LACs, this function just calculates an upper bound on the error increase by any LAC $\psi_n$ applied to a node $n$. Specifically, by Eqs. (1), (2), and (3), we have the error increase of $\psi_n$, $\Delta E(\psi_n) \leq \sum_{i=1}^{I_n} |\Delta L_1[i, n]|$. We call the upper bound on $\Delta E(\psi_n)$ the node sensitivity of $n$. Thus, this function first calculates the node sensitivity of each node $n$ in $G$ as $\sum_{i=1}^{I_n} |\Delta L_1[i, n]|$. Then, it picks $I_n|N|$ nodes with the lowest node sensitivities to form the qualified node set $S_N$. Then, Line 9 identifies the candidate LAC set $S_D$ only on the nodes in the set $S_N$. Note that the actual type of LAC can be any existing one such as those proposed in [4, 5, 8, 12], as long as its affected local circuits have a single output. Then, for each candidate LAC $\psi$ in $S_D$, Line 11 calls the function ErrorEstimate shown in Algorithm 1 to estimate the error increase $\Delta E_1[\psi]$ based on $M_1$ and the SM $\Delta L_1$. Then, Line 12 calls the function LACFilter to obtain the qualified LAC set $S_G$ based on the candidate LAC set $S_D$, the estimated error increase $\Delta E_1$, and the LAC number ratio $I_r$. This function picks $I_r|S_D|$ LACs in $S_D$ with the smallest error increases to construct the qualified node set $S_G$.

After obtaining the qualified LAC set $S_G$, Lines 13–18 perform a round of more accurate estimate of the error increases of the LACs in $S_G$. It is similar to Lines 4–7 and 10–11 and still based on the SM. The main difference is that a large number of $K$ random input patterns are used (see Line 13). Then, Line 19 chooses the LAC $\psi^*$ with the smallest error increase and then applies it to a copy of $G$ to obtain the new approximate circuit $G_{new}$. Finally, Line 20 calculates the actual error $e$ between $G_{new}$ and $G_{org}$. When the error $e$ is no more than the error bound $e_\theta$, the loop will continue (see Line 2). Otherwise, Line 21 returns the latest approximate circuit $G$.

4 EXPERIMENTAL RESULTS

This section shows the experimental results of SEALS. All the experiments are conducted on a 16-core AMD 5950X processor running at 2.2GHz with 64GB RAM. The small number $J$, the large number $K$, the node number ratio $I_n$, and the LAC number ratio $I_r$ in Algorithm 3 are set as $2^{10}$, $2^{17}$, 0.5, and 0.1, respectively. The primary inputs are assumed to be uniformly distributed, although SEALS can deal with any input distribution. We consider three representative statistical error metrics, i.e., ER, normalized MED (NMED) and mean relative ED (MRED) [12].

The benchmarks used in our experiments include some ISCAS [17] and arithmetic circuits, which are listed in Table 1 along with their areas and delays. The area and delay of each circuit are normalized to the area and delay of the inv_X1 gate in the MCNC library [18], respectively. The area ratio (i.e., the area of the approximate circuit over that of the accurate one) and the delay ratio (i.e., the delay of the approximate circuit over that of the accurate one) are used to evaluate the quality of approximate circuits. In addition, the runtimes of the algorithms are reported. To take randomness into account, each experiment is repeated three times and the average results are reported.

We compare SEALS with a state-of-the-art method, called Su’s method [14], which applies batch error estimation method to improve the error estimation accuracy and speed up the greedy ALS flow. Su’s method is also based on MC simulation and we set its number of random input patterns as $10^3$, which is a little smaller than the value $K$ in SEALS. The type of LAC used in both SEALS (see Line 9 of Algorithm 3) and Su’s method is the one from [5], which replaces a signal in a circuit with another signal or its negation. To select the best LAC in each iteration, Su’s method uses the same criterion as [5]. Since Su’s method is relatively slow, for a fair comparison, we also add a LAC filter to it. Su’s method does not calculate sensitivity values, so the same filters as SEALS are not added to it. Instead, we add a filter that filters out each LAC such that the pair of signals involved in the LAC do not share a common dependent primary input. Note that the pair of signals involved in such a LAC are very likely to be quite different, and hence, the LAC is not a promising choice.

4.1 Performance under ER Constraint

In this section, we compare SEALS with Su’s method under ER constraint for all the circuits in Table 1. Note that Su’s method uses a three-dimensional 0-1 matrix to indicate whether an internal approximation error will influence the value of each PO in the circuit. Thus, under ER constraint, bit-parallel operation can be applied to accelerate it, which realizes concurrent computation for different rounds of logic simulation. Therefore, we implement two versions of Su’s method: 1) without bit-parallel operation (w/o BP), which is the same as the implementation in [14]; 2) with bit-parallel operation (w/ BP), which is the enhanced version of Su’s method.

Table 2 lists the average area and delay ratios and the average runtime of SEALS and Su’s method under ER constraint. For Su’s methods w/o BP and w/ BP, the area and delay of an approximate circuit synthesized by them are identical. Thus, Table 2 lists only one column for the average area/delay ratio of Su’s method (named Su’s). The values for each circuit are the average results under 7 ER thresholds (0.1%, 0.3%, 0.5%, 0.8%, 1%, 3%, 5%). We highlight the data in bold when SEALS performs better than Su’s method. Similar notations are applied in the following tables. From Table 2, for nearly half of the benchmarks, SEALS synthesizes approximate circuits with smaller area and shorter delay than Su’s method. All the circuits except alu4 synthesized by SEALS have either smaller areas or shorter delays, where at least one quantity is largely improved and the other is slightly worse. In terms of runtime, SEALS is 14x faster than Su’s method w/o BP on average. Since SEALS uses SM that stores binary radix numbers, the bit parallel operation cannot

<p>| Table 1: Benchmarks used in our experiments. |</p>
<table>
<thead>
<tr>
<th>Circuit Area Delay</th>
<th>Circuit Area Delay</th>
<th>Circuit Area Delay</th>
<th>Circuit Area Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>alu4</td>
<td>2798</td>
<td>12.7</td>
<td>c880</td>
</tr>
<tr>
<td>c1908</td>
<td>758</td>
<td>37.3</td>
<td>cl32a</td>
</tr>
<tr>
<td>c3540</td>
<td>1604</td>
<td>55.0</td>
<td>ksa32</td>
</tr>
</tbody>
</table>
be applied to it under ER constraint. However, no significant runtime difference exists between SEALS and Su’s method w/ BP.

Table 2: Comparison of SEALS and Su’s methods w/o BP and w/ BP under ER constraint.

<table>
<thead>
<tr>
<th>circuit</th>
<th>Aver. area ratio</th>
<th>Aver. delay ratio</th>
<th>Average time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Su</td>
<td>SEALS</td>
<td>Su</td>
</tr>
<tr>
<td>alu4</td>
<td>76.92%</td>
<td>80.09%</td>
<td>100.00%</td>
</tr>
<tr>
<td>c1908</td>
<td>81.47%</td>
<td>79.40%</td>
<td>90.46%</td>
</tr>
<tr>
<td>c3540</td>
<td>94.26%</td>
<td>94.04%</td>
<td>100.00%</td>
</tr>
<tr>
<td>c880</td>
<td>94.82%</td>
<td>96.41%</td>
<td>97.93%</td>
</tr>
<tr>
<td>cla32</td>
<td>86.65%</td>
<td>72.38%</td>
<td>78.18%</td>
</tr>
<tr>
<td>ksa32</td>
<td>88.16%</td>
<td>85.49%</td>
<td>95.75%</td>
</tr>
<tr>
<td>mtp8</td>
<td>96.11%</td>
<td>97.82%</td>
<td>98.94%</td>
</tr>
<tr>
<td>rca32</td>
<td>98.03%</td>
<td>96.44%</td>
<td>99.38%</td>
</tr>
<tr>
<td>wal8</td>
<td>96.15%</td>
<td>89.35%</td>
<td>93.82%</td>
</tr>
<tr>
<td>Avg</td>
<td>90.29%</td>
<td>87.94%</td>
<td>94.94%</td>
</tr>
</tbody>
</table>

4.2 Performance under NMED Constraint

This section compares SEALS with Su’s method under NMED constraint. Since NMED is an error metric for arithmetic circuits, only such circuits in Table 1 are selected. Note that when calculating the error metrics related to ED, such as NMED and MRED, we need to use the binary number represented by the POs to calculate the final error magnitude. Thus, bit parallel operation is not applicable for these error metrics for both SEALS and Su’s method.

Table 3 lists the average area and delay ratios and the average runtime of SEALS and Su’s method under NMED constraint. The values for each benchmark are the average results under 8 NMED thresholds (0.00153%, 0.00305%, 0.00610%, 0.01221%, 0.02441%, 0.04883%, 0.09766%, 0.19531%). From Table 3, all the approximate circuits obtained by SEALS except mtp8 have smaller areas than those obtained by Su’s method. In an average sense, SEALS synthesizes approximate circuits with both smaller area and delay than Su’s method. Furthermore, SEALS dramatically reduces the runtime over Su’ method for all the circuits. On average, SEALS is 12× faster than Su’s method.

Table 3: Comparison of SEALS and Su’s method under NMED constraint.

<table>
<thead>
<tr>
<th>circuit</th>
<th>Average area ratio</th>
<th>Average delay ratio</th>
<th>Average time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Su</td>
<td>SEALS</td>
<td>Su</td>
</tr>
<tr>
<td>cla32</td>
<td>25.22%</td>
<td>22.05%</td>
<td>51.07%</td>
</tr>
<tr>
<td>ksa32</td>
<td>27.35%</td>
<td>24.59%</td>
<td>79.85%</td>
</tr>
<tr>
<td>mtp8</td>
<td>82.45%</td>
<td>83.55%</td>
<td>97.62%</td>
</tr>
<tr>
<td>rca32</td>
<td>29.11%</td>
<td>28.47%</td>
<td>98.52%</td>
</tr>
<tr>
<td>wal8</td>
<td>77.69%</td>
<td>66.57%</td>
<td>89.18%</td>
</tr>
<tr>
<td>Avg</td>
<td>48.36%</td>
<td>45.05%</td>
<td>83.25%</td>
</tr>
</tbody>
</table>

4.3 Performance under MRED Constraint

This section compares SEALS with Su’s method under MRED constraint. Same as NMED, since MRED is an error metric for arithmetic circuits, we only choose such circuits in Table 1. Table 4 lists the average area and delay ratios and the average runtime of SEALS and Su’s method under MRED constraint. The values for each benchmark are the average results under 8 MRED thresholds same as those used in Section 4.2. From Table 4, all the approximate circuits obtained by SEALS except mtp8 (resp. cla32) have smaller areas (resp. delays) than those by Su’s method. Compared with Su’s method, the runtime of SEALS is dramatically reduced. On average, SEALS is 15× faster than Su’s method.

Table 4: Comparison of SEALS and Su’s method under MRED constraint.

<table>
<thead>
<tr>
<th>circuit</th>
<th>Average area ratio</th>
<th>Average delay ratio</th>
<th>Average time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Su</td>
<td>SEALS</td>
<td>Su</td>
</tr>
<tr>
<td>cla32</td>
<td>28.29%</td>
<td>25.93%</td>
<td>53.80%</td>
</tr>
<tr>
<td>ksa32</td>
<td>29.58%</td>
<td>28.66%</td>
<td>81.81%</td>
</tr>
<tr>
<td>mtp8</td>
<td>95.98%</td>
<td>97.03%</td>
<td>98.94%</td>
</tr>
<tr>
<td>rca32</td>
<td>33.20%</td>
<td>31.87%</td>
<td>97.59%</td>
</tr>
<tr>
<td>wal8</td>
<td>94.95%</td>
<td>84.04%</td>
<td>93.60%</td>
</tr>
<tr>
<td>Avg</td>
<td>56.40%</td>
<td>53.72%</td>
<td>85.15%</td>
</tr>
</tbody>
</table>

5 CONCLUSION

In this work, we propose SEALS, a sensitivity-driven efficient ALS method to speed up a greedy ALS flow. Based on the newly proposed metric called sensitivity, we propose a fast and accurate error estimation method and an efficient LAC filtering technique to filter out some unpromising LACs. SEALS outperforms a state-of-the-art ALS method in runtime without reducing circuit quality.

REFERENCES