Advanced Ordering Search for Multi-level Approximate Logic Synthesis

Chang Meng[†], Paul Weng[†], Sanbao Su[†], Weikang Qian[†‡]
[†]University of Michigan-Shanghai Jiao Tong University Joint Institute
and [‡]MoE Key Lab of Artificial Intelligence,
Shanghai Jiao Tong University, Shanghai, China
{changmeng, paul.weng, gawaine, qianwk}@sjtu.edu.cn

ABSTRACT

Approximate computing, an emerging design paradigm for error-tolerant applications, has received considerable interest recently. **Approximate logic synthesis (ALS)** is an automatic process to generate approximate circuits. Many existing ALS methods are implemented in an iterative greedy way. For each iteration, they choose a **local approximate change (LAC)** with the highest score. Since greedy approaches have the shortcoming of easily getting into local minima, we apply two advanced search algorithms, beam search and Monte Carlo tree search, to determine a better ordering of LACs. Each algorithm achieves an order in which the area of approximate designs is improved.

KEYWORDS

approximate logic synthesis, approximate computing, logic synthesis, Monte Carlo tree search, beam search

1 INTRODUCTION

Approximate computing is an emerging design paradigm for digital systems. As energy consumption of digital systems grows rapidly, energy efficiency has become an essential concern in their design [1]. Fortunately, many applications are error-resilient, such as image processing, data mining, and machine learning. Thus, they allow to introduce some errors into the overall function. If the errors are introduced properly, the application-level quality is still maintained, but the area, delay, and power consumption of the circuits could be dramatically reduced. Therefore, approximate computing has received considerable attention recently.

This work focuses on **approximate logic synthesis** (**ALS**) for multi-level circuits. ALS automatically generates inexact circuits of better quality from the original accurate circuits under the constraints on some error metrics. Typical quality measures include circuit area and delay, while typical error metrics include error rate and error magnitude. To derive an approximate design, many existing works iteratively apply **local approximate changes** (**LACs**) to the circuit. Some representative LACs include simply replacing a gate by a constant 0/1 [2] and substituting a signal by another one with similar functionality [3].

Many existing ALS methods are iterative. In each iteration, they determine which LAC should be applied greedily [2–6]. Specifically, a score is calculated for each valid LAC based on its local quality improvement (such as area improvement) and induced error. Then, the one with the highest score is selected. The iteration terminates when a given error limit is reached.

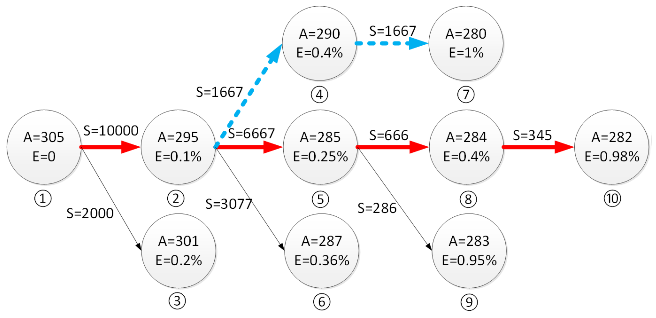

Figure 1: A greedy ALS method is stuck into a local optimum. A node represents a circuit and an edge represents a local approximate change (LAC). A **and** E **denote the area and error rate of the corresponding circuit, respectively.** S **is the score of an LAC.**

However, a greedy ALS method has its natural drawback of getting stuck into a local optimum. Fig. 1 shows an example for this. Assume that the error rate threshold is 1%. The target is to synthesize a circuit with the minimum area and error rate no more than 1%. Node ① is the input accurate circuit, which has an area of 305 and no error (i.e., $A = 305, E = 0$). Each edge corresponds to an LAC and its score S is put near the edge. If the method always chooses an LAC with the highest score and modifies the circuit with the choice each time, it will move along the red path and generate an inexact design with area of 282 and error rate of 0.98%. Unfortunately, the best order leading to the best solution should be ①→②→④→⑦. Thus, a greedy ALS method does not guarantee to produce an approximate circuit with the best quality.

To overcome this issue, we propose to utilize two advanced search algorithms to find a better ordering of applying LACs. The first search algorithm we apply is beam search. It is an extension of the basic greedy method by keeping track of multiple top-scored LACs. The second is **Monte Carlo tree search** (**MCTS**). It is a method for finding optimal decisions by taking random samples in the search space. It has been successfully applied to many planning problems [7–9]. Our experimental results showed that these algorithms are effective in finding a better ordering than the basic greedy search and thus, improve the quality of the final approximate circuits.

The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 presents the preliminaries and the basic ideas. Sections 4 and 5 describe the proposed ALS methods with beam search and Monte Carlo tree search, respectively. Section 6 shows the experimental results. Section 7 concludes our work.

2 RELATED WORK

Many ALS methods have been proposed previously. Most of them applied a greedy iterative method to find the sequence of LACs to apply to the circuit. They differ by the types of LACs used and the scoring mechanisms. Shin and Gupta proposed LACs that replace a gate by a constant 0/1 [2]. The score of an LAC is evaluated as the ratio of area reduction over an error measure called rate-significance value. Venkataramani *et al.* proposed LACs that replace a signal by another with close functionality [3]. To determine which LAC to choose at each step, they designed a comprehensive score on area, delay, and error. Su *et al.* used the same LACs as Venkataramani *et al.* [10], but they used a different quality evaluation of candidate LACs, which is defined as the reduced area over the increased error rate. They proposed a method to efficiently and accurately obtain the error rate and showed that it could help improve the synthesis quality compared to [3]. Wu and Qian proposed an ALS method working on Boolean network representation of a circuit. Their proposed LACs remove some literals from the Boolean expression of each node in the network [4]. The ratio of the literal reduction over the estimated error rate is used to assess each candidate LAC. Chandrasekharan *et al.* proposed to get inexact circuit by approximately rewriting AND-inverter graph. Their proposed LACs rewrite the cut on the critical path [5]. The criterion is to choose the cut with the minimum size. Yao *et al.* proposed LACs that approximate a maximal fanout-free cone in the circuit by a tree of gates obtained through approximate disjoint bi-decomposition [6]. They used the same score as the one used in [10]. Hashemi *et al.* proposed LACs that approximate the truth table of multi-output subcircuits through Boolean matrix factorization [11]. They assessed the quality of an LAC by the error rate.

There also exist other non-greedy ALS methods. Venkataramani *et al.* proposed to transform the ALS problem into the traditional logic synthesis problem by encoding the error magnitude as a function [12]. Then, they could exploit external don't cares to improve the circuit. Liu and Zhang proposed a statistically certified ALS framework using the techniques from stochastic optimization [13]. In each iteration, the proposed method randomly selects a proper LAC and accepts it with a certain probability.

Our work is different from the above-mentioned approaches in terms of how we search for a good ordering of LACs. We use advanced search algorithms to find a good sequence of LACs from a global perspective.

3 PRELIMINARIES AND BASIC IDEAS

3.1 Error Metric

Our method can handle any error metric, such as error rate, average error magnitude, and maximum error magnitude. However, in the following, we just use **error rate (ER)** for illustration. Let the set of input vectors of the circuit be $\{\mathbf{x}_1, \ldots, \mathbf{x}_\mathrm{M}\}$. Assume that $\mathbf{x}_i (1 \leq i \leq M)$ occurs with a probability p_i. Let $\hat{\mathbf{y}}_i$ and $\mathbf{y}_i$ be the approximate and accurate output vectors for $\mathbf{x}_i$, respectively. ER is calculated as the probability that the outputs are incorrect.

Given that the number of input vectors is exponential to the input size of a circuit, it is impractical to calculate the exact ER for a large circuit. Therefore, in our experiments, we derive the ER by logic simulation using a sufficiently large number of random input stimuli.

3.2 Local Approximate Change

We use the LACs proposed in [3] to illustrate our method, although it is not only limited to this type of LACs. These LACs replace one signal in the circuit, called **target signal (TS)**, by another, called **substitution signal (SS)**. By doing so, the gates in the netlist only contributing to generate the TS can be removed and the circuit area is reduced. In the approximate computing context, we do not require TS and SS to be exactly identical in their functions. In our approach, a valid LAC in a gate netlist must satisfy:

- TS is a gate.
- SS can be a gate or its negation, a primary input (PI) signal or its negation, or a constant 0/1.
- SS has no larger arrival time than TS. This guarantees that the final approximate circuit has no larger delay than the original accurate one.
- After substituting TS with SS, the ER conforms to the constraint.

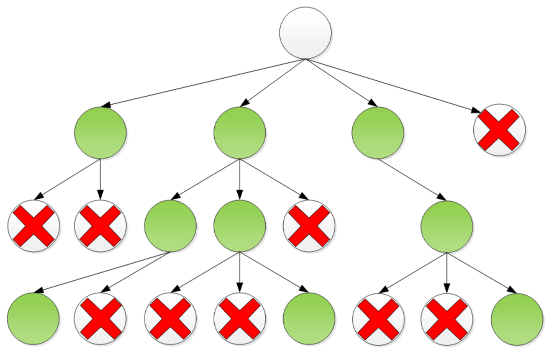

Figure 2: An illustration of beam search with $K = 3$.

3.3 Problem Formulation and Basic Ideas

Given an accurate gate netlist and an ER constraint, we want to find a good ordering to apply LACs, with which the ALS method could achieve an approximate circuit with smaller area. This problem can be formulated as a state-space search problem where states represent the original circuit and all approximate circuits obtained from it by successive applications of LACs. A good solution satisfying the ER constraint can be found by searching a corresponding state-space search tree. Fig. 1 shows an example of the search tree for our problem. In such a tree, each node represents a gate netlist, and the actions of each node are valid LACs. The successors of a node represent all the circuits obtained by applying valid LACs on it. The root node is the original netlist and a leaf node of the tree, called **goal state**, is a circuit that cannot be further approximated without increasing its ER above the limit. Our target is to achieve a goal state with the minimum area while satisfying the ER constraint.

The main challenge of our problem is the huge search space. On the one hand, by the type of LACs we use here, the number of valid LACs at each node is quadratic to the number of signals in the circuit, since a TS can be any gate and an SS can be any signal with no larger arrival time. Consequently, the number of branches from each node is extremely large. On the other hand, even if the branching factor were a small constant, the size of the search tree would grow exponentially with its depth. As many LACs may be applied before reaching the final ER limit, the depth of the search tree may be large. Therefore, it is infeasible to perform basic search algorithms such as breadth-first search and depth-first search.

To solve this challenge, we turn to advanced state-graph search algorithms. By focusing on promising candidates, these search algorithms tend to explore earlier the branches possibly containing the best solution in the search tree. We apply two advanced search algorithms in this work. The first one is beam search, which is an extension of the basic greedy search. The second is Monte Carlo tree search.

Algorithm 1: The BS-ALS algorithm.

```
Input  : the original netlist $C_{ori}$, the error rate threshold $T_{er}$,
         and the branching factor $K$;
Output : an approximate netlist $C_{ax}$;
1  list $L \leftarrow \{C_{ori}\}$, set of candidate approximate circuits $S \leftarrow \emptyset$;
2  while $L \neq \emptyset$ do
3      if $|L| > K$ then
4          keep $K$ best circuits with the smallest error rates and
             remove others from $L$;
5      new list $L_{new} \leftarrow \emptyset$;
6      for each circuit $C$ in $L$ do
7          if $C$ has no valid LACs then
8              $S \leftarrow S \cup C$;
9          else
10             for each valid LAC $l_v$ of $C$ do
11                 apply $l_v$ on $C$ and get new circuit $C_{new}$;
12                 $L_{new} \leftarrow L_{new} \cup C_{new}$;
13     $L \leftarrow L_{new}$;
14 find the circuit $C_{ax}$ with minimum area from $S$;
15 return $C_{ax}$;
```

4 ALS BY BEAM SEARCH

Beam search is a straightforward way to extend the basic greedy search. The basic greedy search only keeps one most promising state at each level of the search tree and expands it to reach to the states at the next level. Beam search extends this by always keeping K rather than just one most promising states at each level and expanding them to reach all new states [14]. Fig. 2 illustrates a beam search with $K = 3$. At each level, all states expanded from the last level are checked and only 3 states (i.e., the green nodes) with the highest scores are kept. Then, only these 3 states are expanded by their actions to reach the states at the next level.

The basic idea behind beam search is that it maintains K parallel "search threads", among which useful information is passed. In essence, the states that generate the best successors will inform the others that the optimal solution is more likely located in the corresponding branch. Thus, it quickly abandons unpromising searches and focuses at the promising choices.

We combine Beam Search with the ALS method and design the **BS-ALS** algorithm. The detailed algorithm is shown in Alg. 1. It takes the original circuit C_{ori} as input. We maintain a list L to store the best K states. Initially, L only contains the original circuit C_{ori} (Alg. 1, Line 1). Meanwhile, a set S of candidate approximate circuits is used to record the possible solutions. In each iteration, the algorithm first resizes the list L and keeps at most K best circuits in L (Alg. 1, Lines 3–4). In our implementation, we use ER as the criterion. Then, we create an empty list L_{new} and iterate over all circuits in L. For a circuit C in L, if it has no valid LACs, it is a goal state and it will be added into the candidate set S (Alg. 1, Lines 7–8). Note that by our definition, a valid LAC should ensure that the ER of the updated circuit does not exceed the ER threshold. Thus, it is possible for a circuit to have no valid LACs. Otherwise, we will obtain all circuits generated by applying the valid LACs to circuit C and add them into the list L_{new} (Alg. 1, Lines 10–12). After all circuits in L are checked, L is updated by the list L_{new} (Alg. 1, Line 13). The entire loop terminates until L is empty. This happens when all circuits in the last iteration have no valid LACs. Then, the circuit with the minimum area in set S is returned.

Algorithm 2: The MCTS-ALS algorithm.

```
Input  : the original netlist $C_{ori}$, the error rate threshold $T_{er}$,
         and the runtime limit $T$;
Output : an approximate netlist $C_{ax}$
// $Area(C)$ is the area of the circuit $C$
1 create root node $R$ with the original netlist $C_{ori}$;
2 final approximate circuit $C_{ax} \leftarrow C_{ori}$;
3 while runtime is less than $T$ do
4     $V \leftarrow$ Selection ();
5     new expanded node $N_e \leftarrow$ Expansion ($V$);
6     (new approx. circuit $C_{new}$, reward $\Delta$) $\leftarrow$ Playout ($N_e$);
7     Backpropagation ($N_e, \Delta$);
8     if $Area(C_{ax}) > Area(C_{new})$ then $C_{ax} \leftarrow C_{new}$ ;
9 return $C_{ax}$;
```

5 ALS BY MONTE CARLO TREE SEARCH

Monte Carlo tree search (MCTS) is an emerging search method that exploits randomness to efficiently explore large search trees, while focusing its effort on the most promising parts of the tree search [15]. It has been shown to be an essential component for building an expert-level computer player for the game of Go [16], but it has also revealed to be efficient in other domains. In particular, planning or ordering is also a domain in which MCTS-based techniques are widely used [7, 8, 17]. MCTS enjoys several advantages compared to traditional search algorithms. Notably, MCTS is:

- Asymmetric. MCTS expands the search tree asymmetrically, which is suitable for a large search space. It visits more frequently nodes that are likely to reach good goal states, and therefore focuses on exploring the promising parts of the tree.
- Anytime. MCTS can be terminated at any time and return the best solution found so far.

We combine Monte Carlo Tree Search with the ALS method and design the **MCTS-ALS** algorithm, aiming at finding a good ordering of LACs and getting an approximate circuit with higher quality improvement. The overall algorithm is shown in Alg. 2 and some supporting functions are shown in Alg. 3. The algorithm is based on the most popular version of MCTS, the **upper confidence bound for trees (UCT)** algorithm [15]. We integrate the domain knowledge of ALS into it.

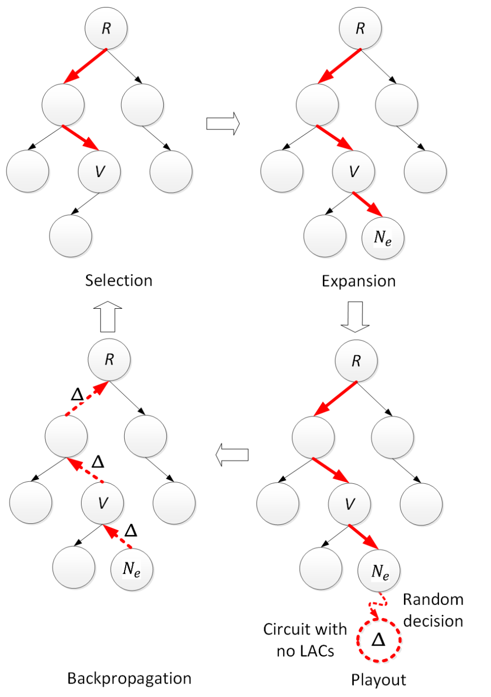

Figure 3: The steps of Monte Carlo tree search (MCTS).

The algorithm first creates the root node R of the search tree, which corresponds to the original netlist C_{ori} (Alg. 2, Line 1). Like any MCTS algorithm, it iteratively expands the search tree by repeating four basic steps—selection, expansion, playout, and backpropagation (Alg. 2, Lines 4–7)—until a computational budget is reached. For simplicity, we use a total runtime limit T (Alg. 2, Line 3). Unlike standard MCTS, we keep track of the best goal state (i.e., approximate circuit) C_{ax} found so far (Alg. 2, Line 8), which is returned at the end. We can therefore get an approximate design at any time, and its quality increases with more runtime.

An illustration of the four basic steps, which are described in details next, is shown in Fig. 3. Conceptually, the selection step chooses a promising node V in the current search tree to further expand during the expansion step, which adds a new node N_e by applying a new LAC (if any) to V. The playout step applies a random sequence of LACs to N_e, which provides a noisy evaluation Δ to it. This information is backpropagated to N_e and its predecessors in the backpropagation step.

Some of these steps call a supporting function *GetValidLACs*, which returns a set of valid LACs for a circuit. It will be described in detail later.

Selection. This step (Alg. 3, Lines 1–7) traverses the current search tree to find a most promising node to expand. It

Algorithm 3: The supporting functions used in the MCTS-ALS algorithm.

```
// $Ckt(V)$ is the circuit of node $V$
Function Selection():
    node $V \leftarrow$ root node $R$;
    candidate LAC set $S_{LAC} \leftarrow$ GetValidLACs($Ckt(V), T_{er}$);
    while $S_{LAC} \neq \emptyset$ and all LACs in $S_{LAC}$ have been explored do
        $V \leftarrow \arg\max_{V_j \in V's\ children} \frac{Q(V_j)}{N(V_j)} + \beta\sqrt{\frac{2\ln N(V)}{N(V_j)}}$;
        $S_{LAC} \leftarrow$ GetValidLACs($Ckt(V), T_{er}$);
    return $V$;
Function Expansion($V$):
    $S_{LAC} \leftarrow$ GetValidLACs($Ckt(V), T_{er}$);
    if $S_{LAC} = \emptyset$ then return $V$ ;
    randomly choose one unexplored $LAC \in S_{LAC}$;
    add a new child $N_e$ to $V$;
    apply $LAC$ on $Ckt(V)$ and get $Ckt(N_e)$;
    return $N_e$;
Function Playout($N_e$):
    $C_{new} \leftarrow Ckt(N_e)$;
    $S_{LAC} \leftarrow$ GetValidLACs($C_{new}, T_{er}$);
    while $S_{LAC} \neq \emptyset$ do
        choose $LAC \in S_{LAC}$ randomly;
        simplify $C_{new}$ with $LAC$;
        $S_{LAC} \leftarrow$ GetValidLACs($C_{new}, T_{er}$);
    simplify $C_{new}$ with a traditional logic synthesis tool;
    reward $\Delta \leftarrow 1 - Area(C_{new}) / Area(C_{ori})$;
    return ($C_{new}, \Delta$);
Function Backpropagation($V, \Delta$):
    while $V$ is not null do
        $Q(V) \leftarrow Q(V) + \Delta$;
        $N(V) \leftarrow N(V) + 1$;
        $V \leftarrow$ parent of $V$;
```

starts from root R and iteratively selects child nodes until a candidate node V is reached. A **candidate node** is either a goal state or an expandable node, which is a node that still has some LACs that have not been tried yet.

For a non-candidate node V, the selection rule for the next child node V_j consists in maximizing the following expression (Alg. 3, Line 5):

$$\frac{Q(V_j)}{N(V_j)} + \beta\sqrt{\frac{2\ln N(V)}{N(V_j)}}, \tag{1}$$

where $Q(V_j)$ is the sum of rewards Δ received in node V_j, $N(V_j)$ (resp. $N(V)$) counts how many times node V_j (resp. V) has been explored, and β is a constant parameter. The reward provides a noisy evaluation of the quality of a sequence of actions. In the general case, an MCTS algorithm aims at finding a sequence of actions that maximizes the reward. In the context of ALS, we want to choose a sequence of LACs to apply to maximize the final area saving of the approximate circuit over the original circuit. Thus, we define the reward Δ as the **area reduction ratio (ARR)**:

$$\Delta = ARR \triangleq 1 - \frac{Area(C_{new})}{Area(C_{ori})}, \tag{2}$$

where $Area(C_{new})$ is the area of the new approximate circuit derived from playout (see the "playout" paragraph later for detail) and $Area(C_{ori})$ is the area of the original circuit.

The values of Q and N are accumulated through the backpropagation step (see the "backpropagation" paragraph later for detail). Empirically, β is set to $\sqrt{2}$ when the reward Δ is limited within the range $[0, 1]$ [18], which is the case for our choice of Δ.

Eq. (1) computes a high-probability upper confidence bound on the estimation of the true value of a node. It was analyzed in [19] as a selection rule in MCTS algorithms. It balances the exploitation of the action currently believed to be optimal with the exploration of other actions that may turn out to be superior in the long run [15]. More specifically, the first term of Eq. (1), $Q(V_j)/N(V_j)$, gives an estimated expectation of the reward for choosing node V_j. A potential good ordering of LACs corresponds to a path from the root to a goal state in which the nodes have high expectations of rewards. Thus, the first term encourages the exploitation of promising LACs. However, when $N(V_j)$ is small, which means that node V_j has only been explored for a few times, the estimation $Q(V_j)/N(V_j)$ may be far from the true expectation. Thus, the second term, which is larger if $N(V_j)$ is small, encourages the selection of under-explored nodes.

Expansion. This step (Alg. 3, Lines 8–14) is an essential operation to grow the search tree from candidate node V obtained in the selection step if V is expandable. If V is a goal state (i.e., no valid LACs), V itself is returned. Otherwise, an unexplored valid LAC of candidate V is chosen and applied to generate a new approximate circuit, i.e., a new node N_e is added into the current tree as a child of V. Finally, the node N_e is returned (Alg. 3, Lines 11–14).

Playout. This step (Alg. 3, Lines 15–24) is a Monte Carlo process. It starts from the new node N_e and iteratively applies valid LACs at random to reach an approximate circuit with no more valid LACs, which corresponds to a goal state (Alg. 3, Lines 18–21). Since a sequence of introduced LACs may induce some redundancies in the circuit, the final circuit generated by playout is optimized by a traditional logic synthesis tool (Alg. 3, Line 22). This could further reduce the area, while not increasing the ER. Finally, the final approximate circuit and the reward Δ calculated by Eq. (2) are returned.

Backpropagation. This step (Alg. 3, Lines 25–29) updates two pieces of vital information in each node along the path from the newly expanded node N_e back to root R. They are the total reward Q and the total visited time N. For each visited node, total reward Q increases by a reward Δ returned from the playout step and total visited time N is incremented by 1. These two values are used in Eq. (1) to guide the selection step in future rounds.

Speed-up by Limiting the Number of Valid LACs. The selection, expansion, and playout steps all call function *GetValidLACs* that returns a set of valid LACs to consider for a circuit under an ER threshold. However, as mentioned in Section 3.3, the number of valid LACs is quadratic to the number of signals in the circuit. If we consider all valid LACs for each node, then the tree would grow too fast, which would lead a to prohibitive amount of computation. Thus, to make our algorithm practical, we propose to only select a subset of valid LACs: after obtaining all valid LACs, we only keep the top B of them that introduce the smallest ERs to the circuit as the LAC candidates. In this way, we reduce the number of unpromising LACs and hence, avoid the unnecessary exploration of unpromising tree branches with higher ERs.

Table 1: Benchmark information

Circuit	Area	Delay	# PIs/POs
C432	309	21.9	36/7
C499	792	15.3	41/32
C880	629	16.4	60/26
C1908	747	24	33/25
C2670	1374	15.7	233/140
C3540	1915	28.7	50/22
C5315	2408	30.2	178/123
C7552	3328	25.2	207/108

6 EXPERIMENTAL RESULTS

In this section, we present the experimental results of our proposed methods based on advanced search algorithms. The algorithms were implemented in C++ and tested on a desktop computer with an eight-core 3.6GHz Xeon CPU, operating on Ubuntu 16.04.

We applied the LACs mentioned in Section 3 to simplify the circuit. We chose ER as the error metric. It was measured by performing logic simulation. We assumed that all primary input vectors are uniformly distributed. For each algorithm, we randomly generated 100,000 input vectors at the beginning and applied them in each logic simulation for measuring the ER throughout the entire process of approximate circuit generation. However, when we obtained the ER of the final approximate circuit, we used 1,000,000 different input vectors to measure the ER more accurately. Note that due to the randomness, if the final measured ER exceeds the threshold, we undid the last applied LAC and repeated this process until we reached a circuit with the final measured ER smaller than the threshold. The **area reduction ratio** (**ARR**), defined as the ratio of the reduced area over the original area (see Eq. (2)), was used to evaluate the quality of inexact designs. Although circuit delay is not the focus here, all produced approximate designs do not have delay increase over their original designs thanks to our proper choices of LACs.

We used the ISCAS85 benchmark suite and mapped those benchmarks with MCNC generic standard cell library [20]. The delay, area, and number of primary inputs/outputs of each benchmark is listed in Table 1. The traditional logic synthesis and technology mapping were performed by the logic synthesis tool ABC [21].

6.1 Comparison of Approximate Circuit Quality for Different Methods

We compared the performance of the basic greedy method with our proposed approaches, BS-ALS and MCTS-ALS. The greedy method used for comparison here is a state-of-the-art method described in [10]. It scores an LAC by the ratio of the area reduction over the increased ER. In the BS-ALS algorithm, we kept $K = 10$ best candidates for each level of the search tree. In the MCTS-ALS algorithm, the top $B = 100$ LACs with lower ERs were kept.

The results of the three methods are shown in Table 2. We list ARRs for 4 ER thresholds (0.5%, 1%, 3%, 5%) for each benchmark. "Greedy", "BS", and "MCTS" in the table correspond to the greedy, BS-ALS, and MCTS-ALS methods, respectively. We highlight the best quality improvement at a certain ER constraint for a benchmark in bold.

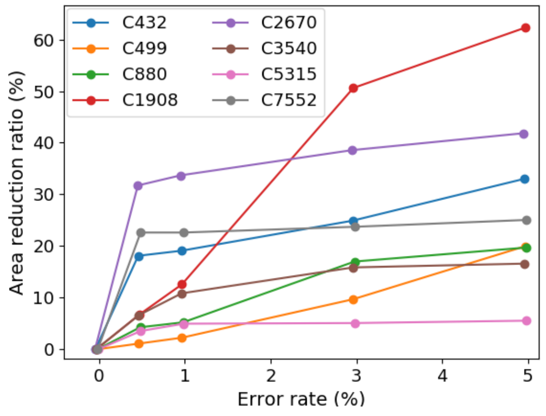

Figure 4: Area reduction ratio v.s. error rate for MCTS-ALS.

We can see that for most cases, BS-ALS reduces more area than the greedy method. On average, BS-ALS behaves slightly better and improves the ARR by 0.43%, 0.63%, 0.82%,

Table 2: Area reduction ratios (ARRs) of three different ALS methods.

Circuit	ARR at $ER = 0.5\%$			ARR at $ER = 1\%$			ARR at $ER = 3\%$			ARR at $ER = 5\%$			Average ARR		
	Greedy	BS	MCTS	Greedy	BS	MCTS	Greedy	BS	MCTS	Greedy	BS	MCTS	Greedy	BS	MCTS
C432	**18.12%**	**18.12%**	**18.12%**	18.12%	18.77%	**19.09%**	18.12%	21.36%	**24.92%**	20.06%	**33.01%**	**33.01%**	18.61%	22.82%	**23.79%**
C499	**1.14%**	**1.14%**	**1.14%**	**2.27%**	**2.27%**	**2.27%**	9.09%	8.84%	**9.72%**	14.27%	13.76%	**19.95%**	6.69%	6.50%	**8.27%**
C880	**4.29%**	**4.29%**	**4.29%**	4.61%	4.61%	**5.25%**	6.68%	6.84%	**17.01%**	18.28%	18.28%	**19.71%**	8.47%	8.51%	**11.57%**
C1908	3.21%	3.48%	**6.69%**	7.63%	7.90%	**12.58%**	40.96%	42.97%	**50.60%**	61.98%	**62.25%**	**62.25%**	28.45%	29.51%	**33.03%**
C2670	27.87%	28.31%	**31.73%**	28.82%	30.71%	**33.70%**	32.53%	32.61%	**38.57%**	40.90%	40.03%	**41.85%**	32.53%	32.91%	**36.46%**
C3540	3.29%	3.29%	**6.63%**	3.92%	3.97%	**10.86%**	8.36%	8.30%	**15.87%**	14.10%	14.20%	**16.61%**	7.42%	7.44%	**12.49%**
C5315	2.41%	**3.57%**	**3.57%**	3.53%	**4.98%**	**4.98%**	4.44%	4.98%	**5.11%**	5.15%	5.15%	**5.56%**	3.88%	4.67%	**4.81%**
C7552	13.64%	15.20%	**22.63%**	14.63%	15.32%	**22.63%**	15.75%	16.56%	**23.74%**	16.26%	17.67%	**25.06%**	15.07%	16.19%	**23.51%**
Average	9.25%	9.68%	**11.85%**	10.44%	11.07%	**13.92%**	16.99%	17.81%	**23.19%**	23.88%	25.54%	**28.00%**	15.14%	16.02%	**19.24%**

and 1.66% for the four ER thresholds over the greedy method. Furthermore, with the help of MCTS, we can find even better orderings of LACs and further improve the quality of approximate circuits. Indeed, MCTS-ALS performs best for all cases in the table as all the values in the "MCTS" columns are highlighted in bold. On average, it improves ARR by 2.6%, 3.48%, 6.2%, and 4.12% under the four ER thresholds over the greedy method. In particular, MCTS-ALS further reduces the area by 12.95% over the greedy method for circuit C432 when $ER = 5\%$. Fig. 4 plots the relationship between ER and ARR for all the benchmarks using MCTS-ALS. We can see that it could reduce 15%–60% area for most benchmarks under 5% ER threshold.

It deserves a mention that BS-ALS sometimes generates worse approximate designs than the greedy method (e.g., circuit C499 under ER threshold of 5%). Since beam search is an extension of the basic greedy search, it also cannot guarantee to find the best solution in the search space due to falling into a local minimum. Even if beam search keeps more than one most promising state, it is possible to make bad choices at some steps, resulting in missing an optimal goal state eventually. That is why BS-ALS sometimes does not beat the basic greedy method.

6.2 Comparison of Runtime for Different Methods

With regard to the runtime to obtain the results in Table 2, the greedy method takes 16 minutes on average, while our BS-ALS method consumes about 10× time compared with the greedy method. This is reasonable since in the experiments, we maintained $K = 10$ parallel "search threads". For the MCTS-ALS method, the given runtime limit T was set as 24 hours and it generates an approximate circuit as good as possible.

Since the runtime mentioned above is not equivalent for different ordering methods, we further explore the tradeoff between runtime and ARR for different ALS methods. To illustrate how much time the MCTS-ALS method consumes to reach the same approxiamte circuit quality as the greedy method, we terminate MCTS-ALS flow as soon as it finds an approximate design with ARR larger than or equal to that of the greedy flow. We recorded the terminating time T_M and compare it with the time T_G of the greedy method in Table 3. We also list the ratio $\frac{T_M}{T_G}$ in the table. We considered three different ERs of 1%, 3%, and 5%. On average, in order to get the same quality, MCTS-ALS method requires 4× runtime compared to the greedy method. Although the runtime of MCTS is longer than the greedy method, it has the advantage of continuously improving the quality as runtime increases, which will be discussed in the following section. It is beneficial when the quality is the primary concern.

Table 3: Runtime of the greedy and the MCTS-ALS methods for reaching the same area improvement. T_G and T_M denotes the time in seconds of the greedy and the MCTS-ALS methods, respectively.

Circuit	$ER = 1\%$			$ER = 3\%$			$ER = 5\%$			Average
	T_G	T_M	$\frac{T_M}{T_G}$	T_G	T_M	$\frac{T_M}{T_G}$	T_G	T_M	$\frac{T_M}{T_G}$	$\frac{T_M}{T_G}$
C432	9	10	1.1	15	120	8.0	17	101	5.9	5.0
C499	6	11	1.8	27	53	2.0	36	35	1.0	1.6
C880	17	18	1.1	21	21	1.0	32	226	7.0	3.0
C1908	76	151	2.0	211	639	3.0	213	998	4.7	3.2
C2670	266	264	1.0	354	368	1.0	438	4326	9.9	4.0
C3540	469	3802	8.1	699	3492	5.0	1513	1510	1.0	4.7
C5315	971	2900	3.0	1329	1300	1.0	2117	4266	2.0	2.0
C7552	2821	28000	9.9	2999	18993	6.3	3101	28912	9.3	8.5
Average	579.4	4394.5	3.5	706.9	3121.5	3.4	933.4	5387.5	8.2	4.0

6.3 Quality Configurable ALS Flow with MCTS

MCTS can be stopped at any time and return the best ordering of LACs, which corresponds to an approximate circuit with the highest quality (such as ARR) so far. The quality of an approximate circuit gradually improves as iteration (i.e., a loop of selection, expansion, playout, and backpropagation) number increases. Fig. 5 shows the relationship between ARR and iteration time of MCTS using circuit C1908 and ER threshold of 3% as an example. The staircase-like curve in blue plots ARRs versus iteration times for MCTS-ALS, while the horizontal line in orange, drawn as a reference, is the ARR of the approximate circuit generated by the greedy method. We can see that the quality produced by MCTS-ALS grows in a staircase-like way, since the current best area is updated at the end of each loop, as shown in Alg. 2. These two curves intersect at the **third** iteration of the MCTS-ALS method. This means after three loops of MCTS, it has already found an LAC ordering that produces an approximate circuit with the same quality (i.e., 40.9% area reduction) as the greedy method.

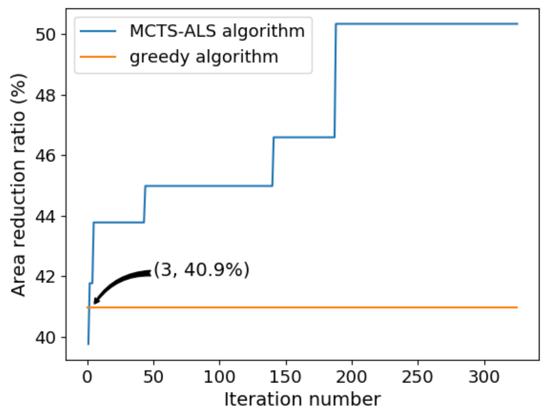

Figure 5: Circuit quality improves with iteration number for the MCTS-ALS method. The results were obtained on circuit C1908 under ER threshold of 3%.

Consequently, we could flexibly use the MCTS-ALS method to generate inexact designs with different qualities in accordance with the user requirement. In case of a strict demand on quality improvement from the user, MCTS-ALS could be given more computational budget to produce a better approximate design. In contrast, if the user does not care much about the quality, MCTS-ALS could produce a satisfying design in shorter time.

7 CONCLUSION

In this paper, we proposed to utilize advanced search methods to determine a good ordering of applying the local approximate changes for multi-level approximate logic synthesis. We proposed two new ALS methods, BS-ALS and MCTS-ALS, based on beam search and Monte Carlo tree search, respectively. They show improvement over the basic greedy search method. The experimental results showed that MCTS-ALS performs best. It has the attractive feature of continuously improving the design quality as more runtime is allowed. As a future work, we will consider how to further speed-up the MCTS-ALS algorithm.

ACKNOWLEDGMENTS

This work is supported by National Natural Science Foundation of China (NSFC) under Grant No. 61574089 and 61872238.

REFERENCES

[1] Jie Han and Michael Orshansky. Approximate computing: an emerging paradigm for energy-efficient design. In *European Test Symposium*, pages 1–6, 2013.
[2] Doochul Shin and Sandeep K Gupta. A new circuit simplification method for error tolerant applications. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 1–6, 2011.
[3] Swagath Venkataramani et al. Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 1367–1372, 2013.
[4] Yi Wu and Weikang Qian. An efficient method for multi-level approximate logic synthesis under error rate constraint. In *Design Automation Conference*, pages 128:1–128:6, 2016.
[5] Arun Chandrasekharan et al. Approximation-aware rewriting of aigs for error tolerant applications. In *International Conference on Computer-Aided Design*, page 83, 2016.
[6] Yue Yao et al. Approximate disjoint bi-decomposition and its application to approximate logic synthesis. In *International Conference on Computer Design*, pages 517–524, 2017.
[7] Radha-Krishna Balla and Alan Fern. UCT for tactical assault planning in real-time strategy games. In *International Joint Conference on Artificial Intelligence*, pages 40–45, 2009.
[8] Ronald Bjarnason et al. Lower bounding klondike solitaire with monte-carlo planning. In *International Conference on Automated Planning and Scheduling*, 2009.
[9] Christopher R Mansley et al. Sample-based planning for continuous action markov decision processes. In *International Conference on Automated Planning and Scheduling*, 2011.
[10] Sanbao Su et al. Efficient batch statistical error estimation for iterative multi-level approximate logic synthesis. In *Design Automation Conference*, pages 54:1–54:6, 2018.
[11] Soheil Hashemi et al. BLASYS: Approximate logic synthesis using boolean matrix factorization. In *Design Automation Conference*, pages 55:1–55:6, 2018.
[12] Swagath Venkataramani et al. SALSA: systematic logic synthesis of approximate circuits. In *Design Automation Conference*, pages 796–801, 2012.
[13] Gai Liu and Zhiru Zhang. Statistically certified approximate logic synthesis. In *International Conference on Computer-Aided Design*, pages 344–351, 2017.
[14] Stuart Russell and Peter Norvig. *Artificial intelligence: a Modern approach*. Prentice Hall, 2009.
[15] Cameron B Browne et al. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.
[16] David Silver et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.
[17] John Asmuth and Michael L Littman. Learning is planning: near bayes-optimal reinforcement learning via monte-carlo tree search. *arXiv preprint arXiv:1202.3699*, 2012.
[18] Levente Kocsis et al. Improved monte-carlo search. *Univ. Tartu, Estonia, Tech. Rep*, 1, 2006.
[19] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European Conference on Machine Learning*, pages 282–293, 2006.
[20] Saeyang Yang. Logic synthesis and optimization benchmarks. Technical report, Microelectronics Center of North Carolina, 1991.
[21] Alan Mishchenko et al. ABC: a system for sequential synthesis and verification, release 80916. http://people.eecs.berkeley.edu/~alanmi/abc/. Accessed November 19, 2018.