Approximate Logic Synthesis by Genetic Algorithm with an Error Rate Guarantee

Chun-Ting Lee, Yi-Ting Li, Yung-Chih Chen, and Chun-Yao Wang

ABSTRACT

Approximate computing is an emerging design technique for error-tolerant applications, which may improve circuit area, delay, or power consumption by trading off a circuit's correctness. In this paper, we propose a novel approximate logic synthesis approach based on genetic algorithm targeting at depth minimization with an error rate guarantee. We conduct experiments on a set of IWLS 2005 and MCNC benchmarks. The experimental results demonstrate that the depth can be reduced by up to 50%, and 22% on average under a 5% error rate constraint. As compared with the state-of-the-art method, our approach can achieve an average of 159% more depth reduction under the same 5% error rate constraint.

ACM Reference Format:
Chun-Ting Lee, Yi-Ting Li, Yung-Chih Chen, and Chun-Yao Wang. 2023. Approximate Logic Synthesis by Genetic Algorithm with an Error Rate Guarantee . In *28th Asia and South Pacific Design Automation Conference (ASPDAC '23), January 16–19, 2023, Tokyo, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3566097.3567890

1 INTRODUCTION

As the size of transistors goes into nano-scale, energy efficiency has become a major challenge in VLSI design. To deal with this problem, designers propose to minimize the circuit size while preserving its functionality as much as possible. Meanwhile, many applications used today, such as image processing and machine learning, exhibit the error-tolerant property. With this trend, approximate computing [4] has been proposed as a novel energy-efficient design paradigm recently. Approximate computing sacrifices the accuracy to achieve a smaller area, delay, or power consumption of the resultant circuits. When the introduced errors are carefully managed, the quality of a circuit is nearly unaffected, while the area, depth and power consumption can be reduced significantly.

Many previous works [1][8][9][10][11][12][13][14][17][18][20][21][23] have demonstrated the effectiveness of approximate computing, including manual design approximation and approximate logic synthesis. Manual design approximation focuses on arithmetic circuits, such as adders [8][12][23] and multipliers [9][10][13]. Approximate logic synthesis aims to synthesize an approximate circuit satisfying the given error constraints [1][11][14][17][18][20][21].

Due to the larger design space exploration of approximate logic synthesis, there exists a growing number of works in this field. In [11], Lai *et al.* proposed an approximate logic synthesis method for threshold logic circuits. In [20], Venkataramani *et al.* proposed to identify signal pairs in a circuit that have higher probabilities to be the same value and substitute one with the other. In [17], Su *et al.* proposed a batch error estimation method based on Monte Carlo simulation to derive better approximate circuits. Recently, to improve efficiency, Meng *et al.* proposed an efficient simulation-based approximate resubstitution with the approximate care set to produce approximate circuits [14]. Tam *et al.* proposed an efficient node merging approach, which merged two nodes with the similar functionality [18]. However, these previous works only selected a local approximate change in each round and focused on area minimization. Circuit delay is another important issue in logic circuits. If we reduce the circuit depth, the timing performance will be improved. Thus, in this work, we propose an evolutionary approach – genetic algorithm, for approximate logic synthesis targeting at searching global approximation and having depth reduction under a given error rate constraint, which is the first work for depth minimization.

Our approach consists of four phases. In the first phase, we simplify the original circuit by replacing nodes with constant values (0 or 1) according to their functional similarities to 0 or 1. In the second phase, we partition the simplified circuit into many subcircuits with a subcircuit size bound. Next, we apply the designed genetic algorithm on each subcircuit to produce a set of subcircuit candidates that have smaller depth and area but with similar functionalities. In the last phase, we combine the candidates into a complete approximate circuit satisfying the required error rate constraint. Experimental results show that the proposed approach reduces much more circuit depth than the state-of-the-art [18] under the same error rate constraint.

The main contributions of this work are twofold:

1) We propose an evolutionary approach to approximate circuits based on genetic algorithm, which is the *first work* targeting at depth minimization of approximate logic synthesis.
2) The experimental results demonstrate that our approach achieves an average of 159% more depth reduction as compared with the state-of-the-art under a 5% error rate constraint.

This work is supported in part by the National Science and Technology Council of Taiwan under MOST 109-2221-E-007-082-MY2, MOST 109-2221-E-155-047-MY2, MOST 110-2224-E-007-007, MOST 111-2218-E-007-010, MOST 111-2221-E-007-121, and MOST 111-2221-E-011-137-MY3.

C.-T. Lee, Y.-T. Li, and C.-Y. Wang are with the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 300044, R.O.C. Y.-C. Chen is with the Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan 106335, R.O.C.

ASPDAC '23, January 16–19, 2023, Tokyo, Japan
© 2023 Association for Computing Machinery.
ACM ISBN 978-1-4503-9783-4/23/01…$15.00
https://doi.org/10.1145/3566097.3567890

2 PRELIMINARIES

2.1 Error Metrics

To evaluate the error resulted from an approximate circuit, several error metrics have been used, such as error magnitude, error distance, and error rate. Error magnitude [16] refers to the maximal numerical deviation of the outputs in an approximate circuit. Error distance [12] refers to the arithmetic distance between the outputs of an approximate circuit and the original one with the same input pattern. Error rate [2][8][9][11][18] refers to the ratio of the number of input patterns that produce incorrect outputs in an approximate circuit to the total number of input patterns. In this work, we adopt the error rate as the error metric since it is the most commonly used error metric and more appropriate to our work.

2.2 And-Inverter-Graphs

The circuits this work deals with are represented in And-Inverter-Graphs (AIGs) [15]. In an AIG, each node is either a primary input (PI) or a two-input AND gate, and an edge may contain a dot indicating an inverter. Different primitive gates represented in an AIG are shown in Fig. 1. XOR and XNOR gates contain three AND gates, while the other gates contain only one AND gate.

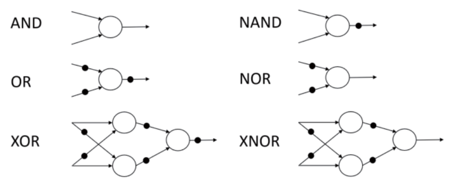


Figure 1: The gate expression of primitive gates in an AIG.

2.3 Genetic Algorithm

Genetic Algorithm [5] is an evolutionary method inspired by biological evolutions. It emulates the biological evolution process to search the global optimal solution. Genetic operations, such as crossover and mutation, are used to generate an offspring in each generation. The crossover operates on two individuals and generates an offspring that inherits their traits. The mutation increases diversities by introducing new solutions. The generated offsprings are evaluated by a fitness function. If the fitness value of an offspring is high, it will be selected as a parent, which will be selected to generate offsprings in the next generation with a higher probability. Thus, the basis of genetic algorithm is an evolutionary process that evolves solutions for achieving the global optimum.

There exist some previous works that use genetic algorithms in the optimization of logic circuits [6][19]. The work [6] proposed to use genetic algorithms to minimize circuit area while preserving the circuit's functionality. In approximate computing, the work [19] proposed to use Cartesian Genetic Programming, a special form of genetic algorithm, to produce an approximate circuit. However, the benchmarks they targeted are arithmetic circuits and small circuits with only a few gates. For scalability, our goal is to target at circuits with different sizes, and relax the accuracy constraint to produce approximate circuits.

3 PROPOSED APPROACH

In this section, we present the proposed approach, which comprises four phases, including *node to constant*, *circuit partitioning*, *genetic algorithm design*, and *subcircuit selection and combination*. Then, we show the overall flow of the proposed approach.

3.1 Node to Constant

The *node to constant* phase aims at replacing nodes with constant values (0 or 1). There may exist some nodes in a circuit that have similar functionality to constant values (0 or 1). If these nodes are replaced by 0 or 1, the circuit can be simplified with a very small error. Thus, we set a probability bound p, which is a user-specified parameter, to examine if a node value has a higher probability of being 0 than p. To determine the functional similarity of a node to 0 or 1, we randomly simulate r patterns to estimate the 0's probability of each node in the circuit. If a node has the probability of $\frac{|0|}{r} \geq p$, we replace the node with a constant 0, and remove the nodes in the maximum fanout-free cone (MFFC) of the replaced node, where $|0|$ is the number of patterns causing a node to be 0 after simulating r random patterns. If a node has the probability of $\frac{|0|}{r} \leq 1 - p$, we replace the node with a constant 1, and remove the nodes in the MFFC of the replaced node. Note that removing the nodes in the MFFC may reduce the depth and area simultaneously.

Since there may exist many nodes in the circuit that can be replaced by 0 or 1, it is important to determine the node order for examination. Thus, we first compute the size of MFFC of each node in the circuit, and examine the nodes by their MFFC sizes in a descending order. To evaluate the accuracy of the simplified circuit, we use another set of r random patterns to estimate the error rate of the simplified circuit after every replacement. If the error rate of the simplified circuit is less than 0.5ε, we continue examining nodes in this phase, where ε is the given error rate constraint of the resultant approximate circuit. If the remaining nodes in the simplified circuit cannot be replaced by 0 or 1, or the error rate of the simplified circuit exceeds the error rate constraint, this phase is terminated and the simplified circuit is returned.

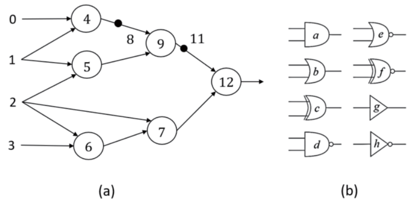


Figure 2: An example for labelling nodes in an original subcircuit with primitive gates. (a) An original subcircuit represented in AIG. (b) The used primitive gates.

3.2 Circuit Partitioning

After having the simplified circuit from the first phase, we perform the *circuit partitioning*, which aims at dividing the simplified circuit into many subcircuits with a subcircuit size bound. This is because the time spent on evaluating a circuit's error rate decreases as the size of a circuit shrinks. Kernighan-Lin (KL) algorithm [7] and Fiduccia-Mattheyses (FM) algorithm [3] are widely used partitioning methods in logic circuits. Both algorithms are two-way partitioning methods aiming for minimizing the cut size between two subcircuits. However, FM algorithm has a lower computation complexity. Thus, we adopt FM algorithm as our partitioning engine in this work.

The basis of FM algorithm is to move one node with the maximum gain from one side to the other side at a time. We compute the gain of node i, i.e., g_i, in the circuit by EQ (1).

$$g_i = cut\ size\ before\ move - cut\ size\ after\ move \tag{1}$$

When a node i is moved across the cut, it will be locked. Then, we update the gains of the other nodes connected to the node i. The movement continues until all the nodes have been locked, and we compute the largest partial sum of the gains for actual movement. This process is repeated until the largest partial sum no more than 0, which means that there is no room for minimizing the cut size. Since the FM algorithm is a two-way partitioning algorithm, we iteratively conduct the algorithm to divide the circuit into a number of subcircuits satisfying the given conditions of $|PI| \leq I$ and $|node| \leq N$ simultaneously, where $|PI|$ is the number of inputs in a subcircuit, $|node|$ is the number of nodes in a subcircuit, and I, N are user-specified constraints.

3.3 Genetic Algorithm Design

After obtaining subcircuits by circuit partitioning, we apply the proposed genetic algorithm on each subcircuit for approximation. As mentioned in Section 2, genetic algorithm is an evolutionary method that evolves solutions in every generation to achieve the global optimum. Since a chromosome is an individual subcircuit represented by a string of genes in genetic algorithm, we first label each node in the *simplified circuit* obtained in the first phase from the PIs to the POs in the topological ordering with a unique integer, and label each primitive gate with an alphabet as shown in Fig. 2. The example in Fig. 2(a) is a subcircuit with labelling on each node. But we assume that the node with ID 10 is in another subcircuit. The reason that we use the topological ordering to label the nodes in the simplified circuit is to avoid the cyclic structures in the resultant approximate circuit after combination. Circuits with cyclic structures are uncommon and not allowed in our work. Since the structures of subcircuits may be varied substantially in this phase, the order that the ID of a node larger than the IDs of the nodes within its transitive fanin (TFI) cone is strictly followed.

Fig. 3 shows an example of a chromosome. In this chromosome, there are eight sets of genes corresponding to the subcircuit in Fig.

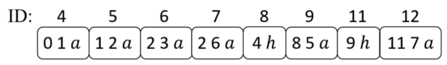


Figure 3: The chromosome with eight sets of genes that represents the original subcircuit in Fig. 2(a).

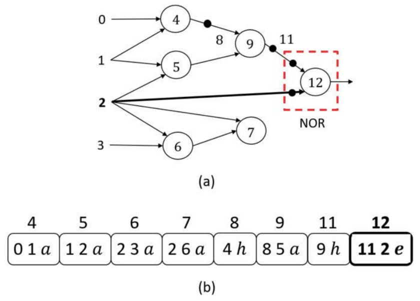


Figure 4: One of the generated initial populations about the subcircuit of Fig. 2(a). (a) An initial population expressed in circuit form. (b) An initial population expressed in chromosome form.

2(a). A set of genes represents either an internal node or a PO in the subcircuit. For example, the first set of genes (0, 1, a) represents the node with ID 4 in Fig. 2(a), where 0 and 1 refer to the fanins of the node 4, and a refers to the gate type AND in Fig. 2(b). Since a set of genes in a chromosome contains the necessary information about the corresponding node in a subcircuit, we can transform the chromosome into a unique subcircuit after performing genetic operations.

The initial populations with respect to an original subcircuit are generated as follows. To generate a number of subcircuit candidates with similar functionalities and smaller depths than the original subcircuit, we remove only one internal AND node and replace it with one of its fanin nodes at a time. Meanwhile, for each PO in the transitive fanout (TFO) cone of the removed node, we change its gate type to every other primitive gate.

We use an example to demonstrate the process of the initial population generation. One of the generated initial populations about the subcircuit of Fig. 2(a) is shown in Fig. 4. In Fig. 4(a), the node 7 is removed and replaced by its fanin node of ID 2. Moreover, the gate type of the PO (labelled with ID 12) is changed to NOR gate. The chromosome in Fig. 4(b) reflects this change, i.e., the last set of genes (11, 7, a) is changed to (11, 2, e). The total number of generated initial populations about the subcircuit of Fig. 2(a) is $5 \times 2 \times 6 = 60$ since there are five internal AND nodes with two fanin nodes, and six primitive two-input gates, $a \sim f$. Note that the number of the initial populations about a subcircuit depends on the numbers of AND nodes and POs in the subcircuit. That is, different subcircuits may have different numbers of the initial populations.

The fitness function, which is constituted by the error rate and the area of a subcircuit, is used to evaluate the quality of a subcircuit candidate after performing genetic operations as shown in EQ (2).

$$Fitness = \frac{1}{W_E \cdot ErrorRate + W_A \cdot AreaRatio} \tag{2}$$

In EQ (2), $ErrorRate$ is the ratio of the number of simulation patterns with incorrect outputs to the total number of simulation patterns, and $AreaRatio$ refers to the ratio of the number of AND gates in the subcircuit candidate to the number of AND gates in the original subcircuit. The parameters W_E and W_A in EQ (2) are used to adjust the weights of $ErrorRate$ and $AreaRatio$, and are determined by users. The higher fitness value of a subcircuit candidate represents the better quality.

Note that the reason that we consider the area instead of the depth in the fitness fuction is as follows. The depth reduction on a subcircuit does not imply the depth reduction on the approximate circuit after combination since the maximum path of a subcircuit does not always overlap with the maximum path of the complete approximate circuit. On the other hand, the operations we design in the genetic algorithm mainly change the fanins of the nodes. If more nodes are removed in each subcircuit, we have higher chances to obtain the approximate circuit with a smaller depth.

Table 1: An example for computing the error rate of the subcircuit candidate in Fig. 4(a).

Simplified Circuit			Original Subcircuit	Subcircuit Candidate
0123	Number	Number	Node 12	Node 12
0000	9132	91	0	0
0001	558	6	0	0
0010	451	5	0	0
0011	2519	**25**	**1**	**0**
0100	35546	355	0	0
0101	3104	31	0	0
0110	8690	87	0	0
0111	982	10	0	0
1000	20018	200	0	0
1001	5705	57	0	0
1010	9550	96	0	0
1011	3745	**37**	**1**	**0**
1100	0	0	–	–
1101	0	0	–	–
1110	0	0	–	–
1111	0	0	–	–

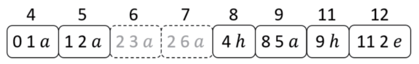
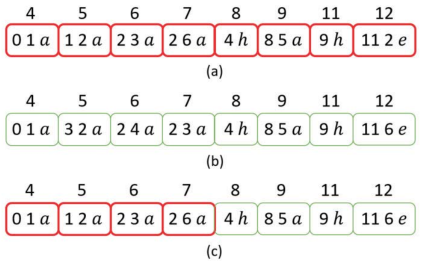


Figure 5: An example for decoding the chromosome in Fig. 4(b) to obtain the area.

The error rate computation in this phase focuses on the subcircuits obtained after partitioning. When we simulate the originally simplified circuit, the number of values 0, 1 appearing on the inputs of a subcircuit is probably unbalanced due to don't cares in the originally simplified circuit. However, if we apply exhaustive patterns at the inputs of a subcircuit, the input patterns that seldom appear or even never appear in the simulation of the originally simplified circuit will be still used to compute the error rate of the subcircuit, which distorts the error rate computation. To avoid the difference between the computed error rate and the real error rate of a subcircuit, we randomly simulate r patterns on the simplified circuit and record the values at the inputs of a subcircuit. Then, $0.01r$ input patterns on a subcircuit are extracted as the input patterns for computing the error rate of the subcircuit candidates since it is not necessary to simulate a large number of patterns on a subcircuit.

Table 1 shows an example about the simulation on the subcircuit candidate of Fig. 4(a). First, we simulate 100,000 random patterns on the originally simplified circuit for recording the values appearing at the inputs of the subcircuit of Fig. 2(a). In Table 1, Column 2 lists the distribution of the numbers of different patterns that appears at the inputs (0, 1, 2, 3) of the subcircuit after simulation, which is quite unbalanced. Then, we extract 1% of these 100,000 patterns, i.e., 1,000, as the input patterns for subcircuit simulation as listed in Column 3. Columns 4 and 5 list the output values of the node 12 after simulating the original subcircuit in Fig. 2(a) and the subcircuit candidate in Fig. 4(a), respectively, with these 1,000 patterns. According to Table 1, the simulation patterns 0011 and 1011 produce incorrect outputs in the subcircuit candidate. Since the total number of patterns 0011 and 1011 is (25 + 37) = 62, the error rate of this subcircuit candidate is (25 + 37) / 1000 = 0.062.

On the other hand, to compute the area of a subcircuit candidate, we decode the corresponding chromosome as follows: Since a set of genes in the chromosome contains the information about fanin nodes and the gate type of a node, we can traverse the chromosome from the genes of the POs to the genes of the PIs to obtain active nodes, which represent the area of the subcircuit candidate. For example, we traverse the chromosome in Fig. 4(b) to identify the active nodes of the subcircuit candidate as shown in Fig. 5. The nodes 6 and 7 will be excluded since they are not in the TFI cone of



4 5 6 7 8 9 11 12
0 1 a | 1 2 a | 2 3 a | 2 6 a | 4 h | 8 5 a | 9 h | 11 2 e
(a)
4 5 6 7 8 9 11 12
0 1 a | 3 2 a | 2 4 a | 2 3 a | 4 h | 8 5 a | 9 h | 11 6 e
(b)
4 5 6 7 8 9 11 12
0 1 a | 1 2 a | 2 3 a | 2 6 a | 4 h | 8 5 a | 9 h | 11 6 e
(c)

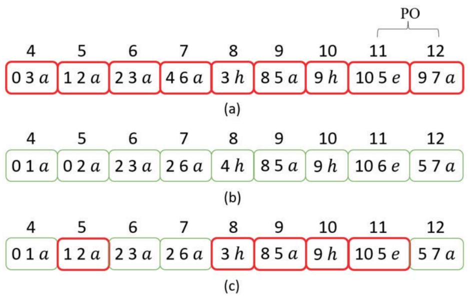
Figure 6: An example of the crossover on two single-PO parents. (a) The first parent. (b) The second parent. (c) The generated offspring.

the output node 12. As a result, the obtained area (in terms of gate count) of the subcircuit candidate is 4, where the inverters (nodes 8 and 11) are ignored in calculating the area in an AIG.

The fitness values of all the generated subcircuit candidates are computed by the fitness function in EQ (2). We select k fitter candidates with higher fitness values as parents in the n^{th} generation, where k is a user-specified parameter. Next, the selected parents will generate offsprings as the population in the $(n+1)^{th}$ generation by the genetic operations of crossover and mutation. In the following paragraphs, we will discuss the designs of crossover and mutation operations used in this work.

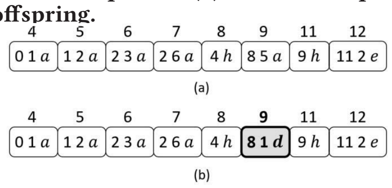
The crossover is the main genetic operation, which uses two parents to generate a new offspring inheriting a part of parents' genes. Since the number of POs and the IDs of nodes in all the parents about a subcircuit are identical, we separate the parents into two categories for the crossover, i.e., the parents with a single PO and the parents with multiple POs. For the single-PO parents, the crossover is performed on two parents by inheriting the left half of a parent's chromosome and the right half of the other parent's chromosome to form an offspring. Fig. 6 demonstrates an example of the crossover operation on the single-PO parents. The chromosomes in Figs. 6(a) and 6(b) represent the first parent and the second one, and both of them have a single PO, node 12. The offspring is generated by inheriting the first four gene sets (0, 1, a), (1, 2, a), (2, 3, a), (2, 6, a) from the first parent, and the last four gene sets (4, h), (8, 5, a), (9, h), (11, 6, e) from the second parent as shown in Fig. 6(c).

On the other hand, for the multiple-PO parents, the crossover is performed as follows. Since the parents have been simulated, we can compute each PO's error rate in the two parents. For each PO with the same ID in the two parents, we select the PO with better quality from the two parents. Then, the selected PO and the nodes in its TFI cone are inherited to generate an offspring. For the PO selection, we give the first priority to the error rate of a PO and the second priority to the area of a PO's TFI cone. That is, when we examine the POs with the same ID in the two parents, we select the PO with a smaller error rate from the two parents first. However, if a PO has the same error rate in both parents, we select the PO with fewer nodes in its TFI cone from the two parents for depth and area saving. For the other situation, i.e., a PO that is with the same error rate and the same number of nodes in its TFI cone in both parents, we randomly select one of the two POs from the two parents. Nonetheless, when we find that all the POs of the generated offspring are from only one parent after selecting the last PO, we randomly select the PO in the other parent for avoiding the situation that an offspring is identical to a parent. After all the POs have been selected from the two parents, the selected POs and the nodes in their TFI cones are inherited to form an offspring. However, the number of nodes in the offspring may not be identical to that in a parent. For each missing node, it can be randomly inherited from one of the parents. Note that a node in the offspring will not inherit repeatedly even it is in all the TFI cones of different POs.



PO
4 5 6 7 8 9 10 11 12
0 3 a | 1 2 a | 2 3 a | 4 6 a | 3 h | 8 5 a | 9 h | 10 5 e | 9 7 a
(a)
4 5 6 7 8 9 10 11 12
0 1 a | 0 2 a | 2 3 a | 2 6 a | 4 h | 8 5 a | 9 h | 10 6 e | 5 7 a
(b)
4 5 6 7 8 9 10 11 12
0 1 a | 1 2 a | 2 3 a | 2 6 a | 3 h | 8 5 a | 9 h | 10 5 e | 5 7 a
(c)

Figure 7: An example of the crossover on two multiple-PO parents. (a) The first parent. (b) The second parent. (c) The generated offspring.



4 5 6 7 8 9 11 12
0 1 a | 1 2 a | 2 3 a | 2 6 a | 4 h | 8 5 a | 9 h | 11 2 e
(a)
4 5 6 7 8 9 11 12
0 1 a | 1 2 a | 2 3 a | 2 6 a | 4 h | 8 1 d | 9 h | 11 2 e
(b)

Figure 8: An example for mutation on an individual. (a) The original individual. (b) The mutated individual.

Fig. 7 demonstrates an example of the crossover on the multiple-PO parents. The chromosome in Figs. 7(a) and 7(b) represent the first parent and the second one, and both of them have two POs, nodes 11 and 12. Suppose that the node 11 has a smaller error rate in the first parent than that in the second parent, and the node 12 has the same error rate in both parents. First, the node 11 in the first parent is selected. Then, we compute the size of the nodes in the node 12's TFI cone of the two parents. The node 12's TFI cone in the first parent has 6 nodes, i.e., the nodes 4, 5, 6, 7, 8, and 9, while the node 12's TFI cone in the second parent has 3 nodes, i.e., the nodes 5, 6, and 7. Thus, the node 12 in the second parent is selected. After the selection, the node 11 in the first parent and the nodes in its TFI cone are inherited, i.e., the gene sets (10, 5, e), (9, h), (8, 5, a), (3, h), (1, 2, a) of the first parent. For the second PO, the node 12 in the second parent and the nodes in its TFI cone are inherited as well, i.e., the gene sets (5, 7, a), (2, 6, a), (2, 3, a) of the second parent. Next, since the node 4 is not inherited from the two parents, we randomly inherit the node 4 from one of the two parents, e.g., the gene set (0, 1, a) of the second parent, to form a complete chromosome. The new offspring is shown in Fig. 7(c).

The mutation is the other genetic operation in this work, which aims to increase diversities of populations by changing some genes in the chromosome of an individual. The mutation operation of an individual is designed as follows. An active node, which is identified by decoding the chromosome for computing the area in an individual, is randomly selected and its gene is modified to be a mutation. We consider to mutate the active nodes only since they are the nodes that may affect the functionality. We randomly change the gate type of the selected node to the other types with the same number of fanins, and replace one fanin node by another active node with a smaller ID than the selected node for avoiding the cyclic structures.

We use an example to demonstrate the mutation in Fig. 8. The chromosome in Fig. 8(a) is an individual to be mutated. Assume that the node 9 is selected for mutation. The gate type of the node 9 is changed from AND gate to NAND gate, i.e., a to d, and the node 9's fanin node 5 is replaced by the node 1, as shown in Fig. 8(b). If we decode the chromosome of Fig. 8(b) for computing its area, we will find that its area becomes smaller than that of the original individual. This is because the node 9's fanin node 5 is replaced by a PI, the node 1. Note that the node 5 cannot be replaced by the nodes with IDs larger than the node 9 for avoiding the cyclic structures.

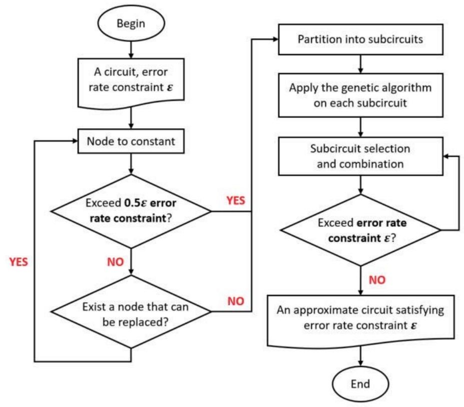

Begin
A circuit, error rate constraint ε
Node to constant
Exceed 0.5ε error rate constraint?
YES
NO
YES
Exist a node that can be replaced?
NO
Partition into subcircuits
Apply the genetic algorithm on each subcircuit
Subcircuit selection and combination
Exceed error rate constraint ε?
YES
NO
An approximate circuit satisfying error rate constraint ε
End

Figure 9: The overall flow of the proposed approach.

For the initial population of a subcircuit, we select k individuals with higher fitness values as parents to participate the crossover and mutation process. λ_c and λ_m individuals are generated in each generation by crossover and mutation operations, respectively. Then these new individuals will be evaluated by the fitness function. In the next generation, k individuals with higher fitness values are selected as the parents with a higher probability. The process is repeated for G generations, and the subcircuit candidates with higher fitness values are recorded as good approximate subcircuits for the succeeding selection and combination process.

3.4 Subcircuit Selection and Combination

After having the approximate subcircuits with higher fitness values for each subcircuit, we select the best approximate subcircuit for each subcircuit to form a complete approximate circuit of the original circuit. We also evaluate the error rate of the complete approximate circuit in this phase using r random patterns. If the error rate of the complete approximate circuit exceeds the given error rate constraint ε, we replace some subcircuits by other approximate subcircuits that have smaller error rates. However, if there is no approximate subcircuit with a smaller error rate than that of the subcircuit to be replaced, we select other subcircuits for the replacement. Here comes another question. Which subcircuits should be replaced for the error rate improvement? We choose subcircuits for the replacement based on the fitness values in an ascending order, i.e., replacing an approximate subcircuit with a smaller fitness value first. To elevate the efficiency of the estimation, we replace five approximate subcircuits simultaneously if the error rate is larger than 1.2ε; otherwise, we replace one approximate subcircuit at a time. After the replacement, we obtain a new complete approximate circuit, and we estimate its error rate again. The process will be iteratively conducted until the error rate of the complete approximate circuit no more than ε. Then, an approximate circuit satisfying the error rate constraint is returned.

3.5 Overall Flow

The overall flow of the proposed approach is shown in Fig. 9. The given inputs are an original circuit and an error rate constraint ε. The output is an approximate circuit satisfying the error rate constraint. First, we simplify the original circuit by replacing nodes with constant values in the *node to constant* phase. If the error rate of the simplified circuit exceeds 0.5ε or none of the remaining nodes can be replaced, we partition the simplified circuit into many subcircuits. Next, we apply the designed genetic algorithm on each subcircuit to produce approximate subcircuits. After having the approximate subcircuits, we combine them and calculate the error rate of the complete approximate circuit. If the error rate of the complete approximate circuit exceeds ε, some approximate subcircuits are replaced by other approximate subcircuits with smaller error rates; otherwise, the approximate circuit is returned. Note that our approach is not only suitable on AIGs or other circuits with two-input gates only, it is also applicable to general circuits.

4 EXPERIMENTAL RESULTS

We implemented the proposed approach in C++ language. The experiments were conducted on an Intel Xeon E5-2650V2 2.60 GHz CentOS 6.10 platform with 256GBytes memory. The benchmarks are from IWLS2005 [25] and MCNC [22]. Each benchmark was initially transformed into the AIG format by ABC [24].

The user-specified parameters of our approach are heuristically set as follows. The number of random patterns r for the simulation is set to 100,000. The probability bound p in the *node to constant* phase is set to 98%. The maximum numbers of inputs and nodes in a subcircuit are limited to $I = 10$ and $N = 50$, respectively, in the *circuit partitioning* phase. In one generation of genetic algorithm, the parent size k is set to 10, λ_c and λ_m are set to 10 and 50, respectively. The number of generations G is 10. For the parameters in the fitness function, the weight W_A of $AreaRatio$ is set to 1, and the weight W_E of $ErrorRate$ is set to $5 + \lfloor \frac{|subcircuit|}{50} \rfloor$. The reasons about this setting in fitness function are as follows. First, the larger the fitness value is, the smaller the term $(W_E \cdot ErrorRate + W_A \cdot AreaRatio)$ is. Also, we prefer not to have the subcircuit with a larger error rate. Hence, we enlarge the parameter W_E to reflect the penalty of having a subcircuit with a larger error rate in the fitness function. That is, we heuristically set W_A to 1 and W_E to 5 for the case that the number of subcircuits is fewer than 50. As the number of subcircuits increases 50, the W_E adds 1 such that each subcircuit tolerates a smaller error rate for meeting the error rate constraint of the complete approximate circuit.

Since our approach involves randomness in some procedures, such as the random simulation and genetic operations, the experiments on each benchmark were conducted for three times and the averaged result is reported.

We conducted two experiments. In the first experiment, we compare our results against that of the state-of-the-art [18] under a 5% error rate constraint, [18] is proposed for area reduction though. The program of [18] was released by the authors. We used the same set of random patterns for the error rate evaluation. The result comparison is summarized in Table 2. Columns 1 ∼ 5 list the information of the benchmarks including names, the numbers of PIs and POs, the number of nodes, depths, and the number of paths with this depth value. Columns 6 ∼ 10 list the experimental results of our approach, including the percentages of the area reduction (AR), depth reduction (DR), the number of paths for the depth, error rate (E), and the required CPU time measured in second. Columns 11 ∼ 14 list the corresponding results of [18]. For example, the benchmark *c5315* has two paths with the maximum depth of 35. Our approach approximated the circuit having 22.86% depth reduction, 14.91% area reduction, and 3.49% error rate in 110.96 seconds, while [18] spent 35.39 seconds to obtain the approximate circuit with a longer depth, only 6.15% area reduction, and 4.87% error rate.

According to Table 2, our approach resulted in 159% more depth reduction on average as compared to [18]. The error rates of all the benchmarks are within 5% in both approaches. The CPU time overhead of our approach is only 59.09 seconds on average. This result indicates that our genetic algorithm can achieve more depth reduction using the designed fitness function. Moreover, the positive side effect of our approach is the area reduction. As we can see, our approach also achieves 9% more area reduction than [18] on average. However, for the benchmarks that do not have depth reduction and have multiple paths for this depth, our approach can also reduce the number of paths having this depth value.

In the second experiment, we demonstrate the effectiveness of our work under a 10% error rate constraint by comparing the depth reduction between our approach and the state-of-the-art [18] as shown in Table 3. According to Table 3, our approach saves more

Table 2: The comparison of experimental results between our approach and the state-of-the-art [18] under a 5% error rate constraint.

Benchmark					Ours					State-of-the-art [18]			
Name	|PI|/|PO|	|Node|	Depth	|Path|	AR(%)	DR(%)	|Path|	E(%)	Time (s)	AR(%)	DR(%)	E(%)	Time (s)
misex	25/18	91	6	4	64.84	0.00	1	4.70	3.90	58.24	0.00	4.75	3.81
c880	60/26	323	25	1	21.67	0.00	1	4.77	22.34	17.34	0.00	5.00	10.15
chkn	29/7	344	21	1	80.36	42.86	1	4.85	9.18	80.81	38.10	4.35	10.37
c1908	33/25	412	32	1	63.11	50.00	4	4.00	22.47	60.92	28.13	3.37	9.86
c2670	233/140	694	20	1	39.48	15.00	1	4.61	42.55	22.19	0.00	3.17	18.61
simple_spi	148/144	815	10	2	23.31	10.00	1	4.73	71.90	20.86	0.00	4.02	36.95
c3540	50/22	941	42	1	9.67	16.67	1	4.96	75.94	11.58	-4.76	4.99	32.98
dalu	75/16	1067	23	1	20.52	4.35	1	4.82	73.60	33.83	21.74	4.62	44.97
cps	24/109	1244	20	1	71.38	50.00	1	4.41	81.22	68.65	20.00	3.42	47.92
c5315	**178/123**	**1415**	**35**	**2**	**14.91**	**22.86**	**2**	**3.49**	**110.96**	**6.15**	**-2.86**	**4.87**	**35.39**
c7552	207/108	1537	60	1	21.34	41.67	2	3.75	130.62	7.29	-1.67	4.11	38.51
alu4	14/8	1601	24	1	46.35	45.83	1	4.93	112.95	44.28	0.00	4.80	97.77
s15850	611/684	2752	34	2	30.96	32.35	1	4.97	272.62	32.34	29.41	4.84	191.94
des_area	368/192	4391	27	10	10.09	0.00	8	4.88	438.88	5.26	0.00	4.37	232.04
s38417	1664/1742	8147	25	1	22.30	0.00	1	4.94	808.07	25.56	0.00	4.18	579.59
Average					**36.02**	**22.11**	—	—	**151.81**	**33.02**	**8.54**	—	**92.72**
Ratio					**1.09**	**2.59**	—	—	—	**1**	**1**	—	—

Table 3: The comparison of experimental results between our approach and the state-of-the-art [18] under a 10% error rate constraint.

Benchmark					Ours					State-of-the-art [18]			
Name	|PI|/|PO|	|Node|	Depth	|Path|	AR(%)	DR(%)	|Path|	E(%)	Time (s)	AR(%)	DR(%)	E(%)	Time (s)
misex	25/18	91	6	4	73.63	33.33	1	9.05	4.67	71.43	0.00	9.45	3.13
c880	60/26	323	25	1	24.77	4.00	1	9.68	21.46	21.05	0.00	9.73	15.26
chkn	29/7	344	21	1	83.72	42.86	1	9.75	11.46	89.53	57.14	8.62	14.54
c1908	33/25	412	32	1	63.59	65.63	1	8.00	33.49	62.86	65.63	8.78	13.69
c2670	233/140	694	20	1	41.07	15.00	1	8.92	43.50	35.45	0.00	9.58	30.88
simple_spi	148/144	815	10	2	26.31	10.00	1	8.57	82.61	25.15	0.00	8.02	40.68
c3540	50/22	941	42	1	12.43	21.43	1	8.93	77.79	17.22	0.00	9.31	65.04
dalu	75/16	1067	23	1	34.58	13.04	2	8.62	66.70	46.30	21.74	9.41	80.24
cps	24/109	1244	20	1	73.31	50.00	1	9.25	67.52	69.77	20.00	8.31	52.27
c5315	178/123	1415	35	2	18.19	25.71	2	9.03	107.55	8.48	-2.86	8.29	40.74
c7552	207/108	1537	60	1	31.03	43.33	2	7.05	125.05	11.39	-1.67	9.94	48.30
alu4	14/8	1601	24	1	56.03	45.83	1	9.82	173.12	52.34	4.17	9.90	165.01
s15850	611/684	2752	34	2	37.65	32.35	1	9.34	318.62	35.32	29.41	9.81	272.11
des_area	368/192	4391	27	10	10.86	0.00	4	9.87	417.48	8.75	0.00	9.67	314.71
s38417	1664/1742	8147	25	1	24.49	24.00	6	9.58	818.41	26.70	0.00	10.00	773.20
Average					**40.78**	**28.43**	—	—	**157.96**	**38.78**	**12.90**	—	**128.65**
Ratio					**1.05**	**2.20**	—	—	—	**1**	**1**	—	—

depth than that for 5% error rate constraint, and achieves 120% more depth reduction than [18]. The CPU time overhead is only 29.31 seconds on average as compare to the state-of-the-art. Moreover, our approach also achieves more area reduction than [18] under a 10% error rate constraint. On the other hand, as we can see, the CPU time of our approach under two different error rate constraints is similar, while the CPU time of [18] increases as the error rate constraint relaxes. The reason behind this is that the time spent in our approach is on performing the genetic algorithm, and its runtime is mainly determined by the number of generations. In addition, since we partition a circuit into subcircuits, the CPU time is expected to be a linear growth as a circuit size increases. Thus, our approach is more scalable than [18] for different error rate constraints and larger circuits.

5 CONCLUSION

In this paper, we propose an evolutionary approach based on genetic algorithm to synthesize approximate circuits with a smaller depth and an error rate guarantee. The main ideas include replacing nodes to constant values, partitioning the simplified circuit, and approximating subcircuits by genetic operations. The experimental results show that our approach achieves much more depth reduction and area reduction as compared with the state-of-the-art.

REFERENCES

[1] M. Barbareschi *et al.*, "A Catalog-based AIG-Rewriting Approach to the Design of Approximate Components," *IEEE Trans. Emerg. Topics Comput.*, 2022.

[2] J. Echavarria *et al.*, "Probabilistic Error Propagation through Approximated Boolean Networks," *Proc. DAC*, 2020, pp. 1-6.

[3] C. M. Fiduccia *et al.*, "A Linear-Time Heuristic for Improving Network Partitions," *Proc. DAC*, 1982, pp. 175-181.

[4] J. Han *et al.*, "Approximate Computing: An Emerging Paradigm for Energy-Efficient Design," *Proc. ETS*, 2013, pp. 1-6.

[5] J. H. Holland, "Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence," *MIT Press*, 1992.

[6] S. Karakatic *et al.*, "Optimization of Combinational Logic Circuits with Genetic Programming," *Elektronika ir Elektrotechnika*, 2013.

[7] B. W. Kernighan *et al.*, "An Efficient Heuristic Procedure for Partitioning Graphs," *The Bell System Technical Journal*, 1970, pp. 291-307.

[8] Y. Kim *et al.*, "An Energy Efficient Approximate Adder with Carry Skip for Error Resilient Neuromorphic VLSI Systems," *Proc. ICCAD*, 2013, pp. 130-137.

[9] P. Kulkarni *et al.*, "Trading Accuracy for Power with an Underdesigned Multiplier Architecture," *Proc. VLSID*, 2011, pp. 346-351.

[10] K. Y. Kyaw *et al.*, "Low-Power High-Speed Multiplier for Error-Tolerant Application," *Proc. EDSSC*, 2010, pp. 1-4.

[11] Y.-A Lai *et al.*, "Efficient Synthesis of Approximate Threshold Logic Circuits with an Error Rate Guarantee," *Proc. DATE*, 2018, pp. 773-778.

[12] J. Liang *et al.*, "New Metrics for the Reliability of Approximate and Probabilistic Adders," *IEEE Trans. Comput.*, 2013, pp. 1760-1771.

[13] C. Liu *et al.*, "A Low-Power, High-Performance Approximate Multiplier with Configurable Partial Error Recovery," *Proc. DATE*, 2014, pp. 1-4.

[14] C. Meng *et al.*, "ALSRAC: Approximate Logic Synthesis by Resubstitution with Approximate Care Set," *Proc. DAC*, 2020, pp. 1-6.

[15] L. Hellerman, "A Catalog of Three-Variable Or-Invert and And-Invert Logical Circuits," *IEEE Trans. Electron. Comput.*, 1963, pp. 198-223.

[16] I. Scarabottolo *et al.*, "Partition and Propagate: an Error Derivation Algorithm for the Design of Approximate Circuits," *Proc. DAC*, 2019, pp. 1-6.

[17] S. Su *et al.*, "Efficient Batch Statistical Error Estimation for Iterative Multi-level Approximate Logic Synthesis," *Proc. DAC*, 2018, pp. 1-6.

[18] K. S. Tam *et al.*, "An Efficient Approximate Node Merging with an Error Rate Guarantee," *Proc. ASP-DAC*, 2021, pp. 266-271.

[19] Z. Vasicek *et al.*, "Evolutionary Approach to Approximate Digital Circuits Design," *IEEE Trans. Evol. Comput.*, 2015, pp. 432-444.

[20] S. Venkataramani *et al.*, "Substitute-and-Simplify: A Unified Design Paradigm for Approximate and Quality Configurable Circuits," *Proc. DATE*, 2013, pp. 1367-1372.

[21] A. Wendler *et al.*, "A fast BDD Minimization Framework for Approximate Computing," *Proc. DATE*, 2020, pp. 1372-1377.

[22] S. Yang, "Logic Synthesis and Optimization Benchmarks," Microelectronics Center of North Carolina, Tech. Rep., 1991.

[23] N. Zhu *et al.*, "An Enhanced Low-Power High-Speed Adder for Error-Tolerant Application," *Proc. ISIC*, 2009, pp. 69-72.

[24] Berkeley Logic Synthesis and Verification Group. *ABC: A System for Sequential Synthesis and Verification* [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/abc

[25] http://iwls.org/iwls2005/benchmarks.html