Optimizing Multi-level Combinational Circuits for Generating Random Bits

Chen Wang and Weikang Qian
University of Michigan-SJTU Joint Institute
Shanghai Jiao Tong University
Shanghai, China
{wangchen_2011, qianwk}@sjtu.edu.cn

***Abstract*—Random bits are an important construct in many applications, such as hardware-based implementation of probabilistic algorithms and weighted random testing. One approach in generating random bits with required probabilities is to synthesize combinational circuits that transform a set of source probabilities into target probabilities. In [1], the authors proposed a greedy algorithm that synthesizes circuits in the form of a gate chain to approximate target probabilities. However, since this approach only considers circuits of such a special form, the resulting circuits are not satisfactory both in terms of the approximation error and the circuit depth. In this paper, we propose a new algorithm to synthesize combinational circuits for generating random bits. Compared to the previous one, our approach greatly enlarges the search space. Also, we apply a linear property of probabilistic logic computation and an iterative local search method to increase the efficiency of our algorithm. Experimental results comparing the approximation errors and the depths of the circuits synthesized by our method to those of the circuits synthesized by the previous approach demonstrate the superiority of our method.**

I. Introduction

A random bit, also known as a Bernoulli random variable, is a special discrete random variable which takes value 1 with probability p and value 0 with probability $1-p$. Random bits are an important construct in many applications, such as hardware-based implementation of probabilistic algorithms, weighted random testing, and digital circuits computing on stochastic bit streams.

Researchers proposed to implement those Monte Carlo sampling-based algorithms on hardware to accelerate them [2]–[4]. Probabilistic bits play an important role in the sampling procedure. Weighted random testing is one technique used in built-in self-tests (BIST) [5]. By assigning random bits with different probabilities to different inputs, we can achieve a high fault coverage with a moderate number of random test patterns [6]. This requires the generation of random bits with specific probabilities of being 1. Probabilistic bits are also an important ingredient of a computational paradigm called logical computation on stochastic bit streams [7], [8]. In this paradigm, data is encoded by a *random* bit stream through the probability of ones in the stream and is processed by traditional digital circuits. The paradigm shift offers advantages including simple hardware design and fault tolerance [9].

Random bits with required probabilities can be produced using pseudorandom number generators (PRNGs). However, PRNGs essentially produce deterministic sequences which could fail some statistical tests. Furthermore, they require a large amount of hardware cost. An alternative approach is to use truly random sources, which are physical sources with high entropy such as thermal noise in circuits, radioactive decay, and brownian motion [10], [11]. However, it is hard to control their probabilities. Therefore, a post-processing unit is usually needed. One post-processing mechanism relies on applying a digital circuit to transform a set of "source" probabilities into the "target" probabilities [1], [12], [13]. Wilhelm and Bruck proposed a general framework for synthesizing *switching* circuits to achieve a desired probability [12]. They proposed a method to transform independent random bits with probability 0.5 to be 1 into an arbitrary binary probability of the form $\frac{m}{2^d}$. Zhou and Bruck extended the previous work and proposed an approach to generate an arbitrary probability of the form $\frac{m}{n^d}$ from a set of source probabilities $\{\frac{1}{n}, \frac{2}{n}, \ldots, \frac{n-1}{n}\}$ [13]. These two works are based on a technology which could provide many *independent* random bits with the same probabilities.

However, because the truly random sources are hard to control, a more realistic situation is that all of the available random sources provide different probabilities. The collection of source random bits forms a set of source probabilities $S = \{p_1, p_2, \ldots, p_n\}$, where each p_i corresponds to one random bit. The task is to properly choose some source random bits and design a digital circuit that takes these random bits as inputs and produces an output random bit with a target probability q^*. For example, given a source probability set $\{0.4, 0.5, 0.7\}$, if we want to generate a random bit with probability 0.2 of being 1, we can use an AND gate and choose its two input probabilities as 0.4 and 0.5, since the output probability of an AND gate to be 1 equals the product of its two input probabilities. Note that since each source random bit maps to one probability value in the set S, therefore, each probability in S can be used **at most once**. This type of random bit generation problem can be formulated as:

> Given a set of source probabilities $S = \{p_1, p_2, \ldots, p_n\}$ and a target probability q^*, design a circuit that takes random inputs with probabilities chosen from S and produces a random output with probability q^*. Each element in S can be used as an input probability at most once.

In [1], the authors considered the above problem and proposed a method that synthesizes a combinational circuit to produce a required probability. As they observed, given a finite number of source probabilities, only a finite number of output probabilities can be realized *exactly*. Thus, in general, we can only find a solution that generates an output probability very close to the target q^*. Therefore, the above design problem can be recast as an optimization problem, where we want to find an optimal combinational circuit together with its input probabilities so that its output probability is the closest one to q^* among all the possibilities. However, given the exponential number of combinations of source probabilities and the exponential number of combinational circuits, this problem is a hard combinatorial optimization problem. In [1], the authors proposed a greedy algorithm that could find a suboptimal solution to this problem. However, as we will show in Section II, their algorithm only explores a small portion of the entire solution space. As a result,

the circuits synthesized by their method are not satisfactory both in terms of the approximation error and the circuit depth.

In this work, we propose a new solution to synthesize *combinational circuits* to transform a set of source probabilities into target probabilities. Our algorithm explores a much larger solution space than the previous one. Experimental results demonstrate that our algorithm greatly improves the approximation error and the depth of the circuits for generating target probabilities.

The remainder of this paper is organized as follows. Section II gives some preliminaries and reviews the synthesis algorithm proposed in [1]. Section III discusses our new algorithm and the key techniques we propose. Section IV presents experimental results comparing the performance of our algorithm to that of the algorithm in [1]. Section V concludes the paper.

II. Preliminaries and Related Work

Since any combinational circuit can be implemented with inverters, AND gates, and OR gates, we focus on circuits built with these three basic types of gates. By default, if we say "a probability", we mean the probability of a random bit being 1.

Assuming that the input random bits are independent, the relationships between the input probabilities and the output probability for an inverter, an AND gate, and an OR gate are listed below.

1) For an inverter, if its input probability is p, then its output probability is $1 - p$.
2) For an AND gate, if its input probabilities are p and q, then its output probability is
$$p \cdot q. \tag{1}$$
3) For an OR gate, if its input probabilities are p and q, then its output probability is
$$p + q - p \cdot q. \tag{2}$$

An algorithm for synthesizing a combinational circuit to generate the required probability is proposed in [1]. The idea of that algorithm is to apply a greedy strategy to incrementally construct an "optimal" circuit. Specifically, given a set of n source probabilities, the algorithm will construct n candidate circuits $C_1, \ldots, C_n$ *in sequence* and select the one with the smallest approximation error. The circuit C_k $(k = 1, \ldots, n)$ has k inputs $x_1, \ldots, x_k$ and their probabilities are $p_{i_1}, \ldots, p_{i_k} \in S$. The circuit C_{k+1} is built from the circuit C_k by replacing its last input x_k with a two-input gate. The two input probabilities of the gate added are p_{i_k} and $p_{i_{k+1}}$, where $p_{i_{k+1}}$ is chosen from the remaining probabilities in the set S. In constructing the circuit C_{k+1} from the circuit C_k, all the other parts of the circuit C_k including the input probabilities $p_{i_1}, \ldots, p_{i_{k-1}}$ are kept the same. Thus, the circuit C_{k+1} is an incremental design from C_k, where we only need to determine the type of the gate to add and the value of $p_{i_{k+1}}$. These two design choices are determined so that they let the output of the circuit C_{k+1} be the closest to q^* among all the possible choices. If necessary, an inverter can be inserted between $p_{i_{k+1}}$ and the input of the gate added.

However, this algorithm has two main shortcomings. First, since the algorithm always adds a new gate and connects it to the *last* input of the previous candidate circuit, all the circuits constructed are in the form of a gate chain. Since the algorithm only considers circuits of this special form, it searches a relatively small portion of the entire circuit space. Thus, the resulting output probability may not be one of the closest approximations to the target probability. Second, since the generated circuit is in the form of a gate chain, it has a large depth and hence a long delay.

III. A New Algorithm for Synthesizing Random Bits

In this section, we propose a new algorithm to address the two major shortcomings of the previous approach. The idea is to enlarge the search space so that we are able to reach an arbitrary circuit in the solution space. This exposes us to better solutions and also decreases the depth of the circuit. However, general circuits could contain reconvergent paths, which make the probabilistic analysis and synthesis difficult. Therefore, taking computational efficiency into consideration, we actually enlarge the search space to all the circuits that are in the form of a gate tree.

A. Strategy for Searching the Optimal Tree-Style Circuits

Our algorithm adopts the incremental constructing idea of the algorithm proposed in [1]. Given a set of n source probabilities, we will sequentially construct n candidate circuits $C_1, \ldots, C_n$ and then choose the one whose output probability is the closest to q^*. The circuit C_{k+1} is incrementally designed from the circuit C_k by replacing one input of C_k with a two-input gate. Different from the previous algorithm, our algorithm will consider all the inputs of C_k for the replacement and choose the best location to place the new gate. This result shows that the candidate circuits can be in the form of a gate tree. Algorithm 1 presents the major flow of our algorithm.

Algorithm 1 The proposed algorithm.

```
{Given a set of source probability S and a target probability q*.}
n ⇐ |S|;
Construct the base-case circuit C_1;
for k = 1 to n − 1 do
    for j = 1 to k do
        Construct circuit C_(k+1)j by an optimal replacement of the j-th
        input of C_k with a two-input gate;
    end for
    C_{k+1} ⇐ ChooseBest(C_(k+1)1, ..., C_(k+1)k);
    {ChooseBest returns the circuit whose output probability is the
    closest to q* among all the circuits in its argument list.}
end for
C ⇐ ChooseBest(C_1, ..., C_n);
return C;
```

Our procedure starts from the "base-case" circuit C_1, which has a single input. Its output is either connected to its input x_1 directly or through an inverter. Its input probability p_{i_1} is chosen from the set S. Thus, the output probability is one of the values in an augmented set
$$S' = \{p_1, \ldots, p_n, 1 - p_1, \ldots, 1 - p_n\}.$$
We apply a greedy strategy in constructing C_1: we choose the output probability as the closest one in S' to q^*.

The circuit C_k $(k = 2, \ldots, n)$ has k inputs $x_1, \ldots, x_k$ and their probabilities are $p_{i_1}, \ldots, p_{i_k} \in S$. The circuit C_{k+1} $(k = 1, \ldots, n-1)$ is incrementally constructed from the circuit C_k by replacing one of the inputs of C_k with a two-input gate. In order to get the optimal circuit C_{k+1} whose output probability is the closest to q^*, we consider every input of C_k for the replacement. We obtain the best replacement for each input first. Then we compare all these best replacements for all the inputs and choose the best one among them as the circuit C_{k+1}. One basic step in our algorithm is to obtain the best replacement for a specific input.

In our algorithm, we replace a specific input x_j $(1 \le j \le k)$ of C_k with either an AND gate or an OR gate. The remaining parts of the circuit C_k including the input probabilities are not changed. We want to choose the replacing gate type and its input probability

values to let the output of the modified circuit be the closest to q^*. We achieve this by two steps:

1) Compute the "ideal" probability value $p^*_{i_j}$ for the input x_j, so that if it replaces the input probability p_{i_j} of the circuit C_k, the output probability of C_k is exactly the target probability q^*.
2) Obtain the best choice for the replacing gate type and its two input probabilities so that the output probability of the gate is the closest to the ideal probability $p^*_{i_j}$.

We discuss these two steps in the following two subsections.

B. Obtain the Ideal Probability Value for an Input

We compute the ideal probability value $p^*_{i_j}$ for an input x_j by applying a linear property of probabilistic computation with combinational circuits. As demonstrated in [14], the output probability of a combinational circuit is a multivariate polynomial on its input probabilities and the degree for each input probability variable is at most one. Mathematically, for a combinational circuit with n input $x_1, \ldots, x_n$ and an output y, suppose its input probabilities are $P(x_i = 1) = r_i$ for $i = 1, \ldots, n$ and its output probability is $P(y = 1) = r_y$. Then the relation between r_y and $r_1, \ldots, r_n$ can be characterized by a multivariate polynomial $r_y = F(r_1, \ldots, r_n)$. The degree of each variable r_i in this polynomial is at most one.

We apply the above property on the circuit C_k. Suppose $P(x_j = 1) = r_j$ and $P(y = 1) = r_y$. Then r_y can be represented as

$$r_y = a \cdot r_j + b, \tag{3}$$

where a and b are constants determined by the remaining input probabilities of C_k. From the previous iteration where the circuit C_k is constructed, we can get the output probability of the circuit C_k when the probability of x_j is p_{i_j}. Suppose it is q. Then from Equation (3), we have

$$q = a \cdot p_{i_j} + b. \tag{4}$$

Now we set the input probability of x_j either as a 0 or as a 1: if $p_{i_j} \geq 0.5$, we set it as 0; otherwise, we set it as 1. Denote this input probability as p_{const}. We keep the other input probabilities the same and compute the output probability. Suppose it is q_{const}. Then from Equation (3), we have another equation

$$q_{const} = a \cdot p_{const} + b. \tag{5}$$

By solving the system of Equations (4) and (5), we can obtain the constants a and b. Further, based on the definition of *ideal* probability, we have

$$q^* = a \cdot p^*_{i_j} + b.$$

We can obtain $p^*_{i_j}$ as

$$p^*_{i_j} = \frac{q^* - b}{a}.$$

The following example illustrates how we obtain the ideal probability for an input.

Example 1

Consider the circuit shown in Fig. 1. Assume that the target probability $q^* = 0.59$. *We want to get the ideal value* p^* *for the input with the probability* 0.07.

With the probability for that input being 0.07, *the actual output probability is* 0.58776. *Since* $0.07 < 0.5$, *we replace* 0.07 *with* 1 *and reevaluate the output probability. We find that the output probability is* 0. *Now we have the following system of equations:*

$$0.07 \cdot a + b = 0.58776$$
$$1 \cdot a + b = 0$$

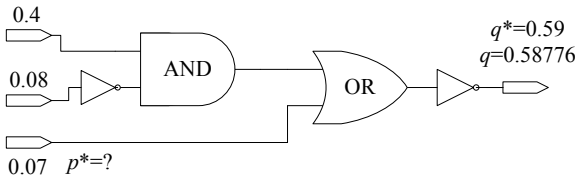


Fig. 1: An example for calculating the ideal probability p^*. In this example, we want to obtain p^* for the lowest input.

Solving it, we obtain the coefficients $a = -0.632$ *and* $b = 0.632$. *Then* p^* *at this input is calculated as* $p^* = \frac{q^* - b}{a} = 0.06646$. □

Remark: For a circuit in the form of a gate tree, since there is no reconvergent path, we can also obtain the ideal probability for an input by propagating an "ideal" output probability for each gate along the path from the output of the circuit to this primary input. For the circuit in Example 1, in order to obtain the target probability $q^* = 0.59$, we require the output probability of the OR gate to be 0.41. Since the top input probability of the OR gate is 0.368, we can obtain that the ideal probability for the lowest input should be 0.06646, by solving the equation $0.368 + p^* - 0.368 \cdot p^* = 0.41$. However, the approach based on the linear property is more powerful than the reverse-propagation approach, because it also works for general circuits which could have reconvergent paths. With this approach, we greatly enhance the efficiency of our algorithm, since the calculation of ideal probability values is carried out frequently in our procedure.

C. Obtain the Best Local Replacement

Due to the linear property of probabilistic computation, the choice for the replacing gate type and its input probability values which lets the output of the modified circuit be the closest to q^*, corresponds to a choice that lets the output probability of the **replacing gate** be the closest to $p^*_{i_j}$. In the algorithm proposed in [1], when a good replacement is sought, one input probability of the replacing gate is still chosen as p_{i_j}. The other input probability $p_{i_{k+1}}$ is chosen from the remaining probabilities in the set S, i.e., from the set $S \backslash \{p_{i_1}, \ldots, p_{i_k}\}$. This allows two degrees of freedom in searching the best choice, i.e., the gate type and one input probability.

In our algorithm, in order to get a better solution, we relax the restriction that the first input probability should be fixed. Thus, we allow three degrees of freedom. The gate type is either an AND or an OR gate.[1] The two input probabilities are chosen from the set $S' = S \backslash \{p_{i_1}, \ldots, p_{i_{j-1}}, p_{i_{j+1}}, \ldots, p_{i_k}\}$. We also allow to insert an inverter between a primary input and an input of the gate.

One way to obtain the optimal choice is to enumerate all combinations of the two input probabilities from the set S' and the gate type and choose the best one. The complexity is $O(|S'|^2)$.

Here we propose an iterative method to obtain a good choice. Although this procedure cannot guarantee to obtain the best choice, it is fast and in many cases returns a choice close to the optimal one.

The idea of the iterative method is to repeat a basic procedure (described below) until the difference between the output probability of the replacing gate and the ideal probability $p^*_{i_j}$ stops decreasing in comparison with the previous iteration.

Since an inverter can be used, we augment the candidate source probability set by adding all the values which can be represented as 1 minus one value in the original candidate set. In other words, given an original candidate set R, we augment the set to $R' = \{p | p = r \text{ or } 1 - r, r \in R\}$.

[1]Our approach can also be adapted to allow other two-input gates, such as XOR and XNOR gate.

The basic procedure is to fix one input probability and then determine the best choice of the gate type and the other input probability. In the next iteration, the newly determined probability is fixed and the previously fixed probability is relaxed. Correspondingly, the newly determined probability and 1 minus that value are removed from the augmented candidate probability set, while the previously fixed probability and 1 minus that value are put back into the candidate set. The iterative method starts with the fixed input probability being p_{i_j}, which is the probability for the input x_j of the circuit C_k.

The basic procedure is implemented in a similar way to a method described in [1]. Suppose that for the current iteration, the value of the fixed probability is p. We distinguish two cases:

1) The case where $p < p^*_{i_j}$. Then we choose the gate type as an OR gate. The reason is that we want to increase the probability for the input x_j of the circuit C_k; based on Equation (2), the output probability of an OR gate is larger than either of its two input probabilities. With the gate type fixed, we can determine the *ideal* probability p^* for the other input of the OR gate by solving Equation (2). We obtain

$$p^* = \frac{p^*_{i_j} - p}{1 - p}. \tag{6}$$

The best choice for the other input probability of the OR gate is the value closest to p^* in the augmented candidate probability set R'.

2) The case where $p \geq p^*_{i_j}$. Then we choose the gate type as an AND gate. The ideal probability p^* for the other input of the AND gate is determined by solving Equation (1), which gives

$$p^* = \frac{p^*_{i_j}}{p}. \tag{7}$$

Similarly, the best choice for the other input probability of the AND gate is the value closest to p^* in the augmented candidate probability set R'.

The following example illustrates how the iterative method works.

Example 2

Suppose that we have a set of candidate probability $S = \{0.08, 0.63, 0.4, 0.07\}$. *We want to choose the type of a gate and two input probabilities for the gate so that the output probability of the gate is close to a value* $p^*_{i_j} = 0.6344$. *We apply the iterative method to get the solution. Suppose that the initial fixed input probability is* 0.63.

In the first iteration, since $0.63 < p^*_{i_j}$, *an OR gate is chosen. By Equation* (6), *the ideal probability* p^* *for the other input is* 0.0119. *The augmented candidate set is* $R' = \{0.08, 0.4, 0.07, 0.92, 0.6, 0.93\}$. *The closest value to* p^* *in* R' *is* 0.07, *which is chosen as the other input probability. The output probability of the OR gate is* $p = 0.6559$ *and the error is* 0.0215.

In the second iteration, the fixed input probability is 0.07 *and the augmented candidate probability set is* $R' = \{0.08, 0.63, 0.4, 0.92, 0.37, 0.6\}$. *Since* $0.07 < p^*_{i_j}$, *the new gate should be OR. By Equation* (6), *the ideal probability* p^* *for the other input is* 0.607. *The closest value to* p^* *in* R' *is* 0.6, *which is chosen as the other input probability. The output probability of the gate is* $p = 0.628$ *and the error is* 0.0064.

In the third iteration, we fixed the input probability 0.6. *The augmented candidate probability set is* $R' = \{0.08, 0.63, 0.07, 0.92, 0.37, 0.93\}$. *Applying the basic procedure, we choose an OR gate and choose the other input probability as* 0.08. *The output probability of the gate is* $p = 0.632$ *and the error is* 0.0024.

In the fourth iteration, we fixed the input probability 0.08. *As a result of the basic procedure, we choose an OR gate and choose the other input probability as* 0.6. *The output probability of the gate is* $p = 0.632$ *and the error is* 0.0024. *At this point, the error stops decreasing and the iteration terminates. We obtain the final solution: the gate is an OR gate, the first input probability is* 0.08, *the second input probability is* 0.4, *and an inverter is inserted between the second input and the input of the OR.* □

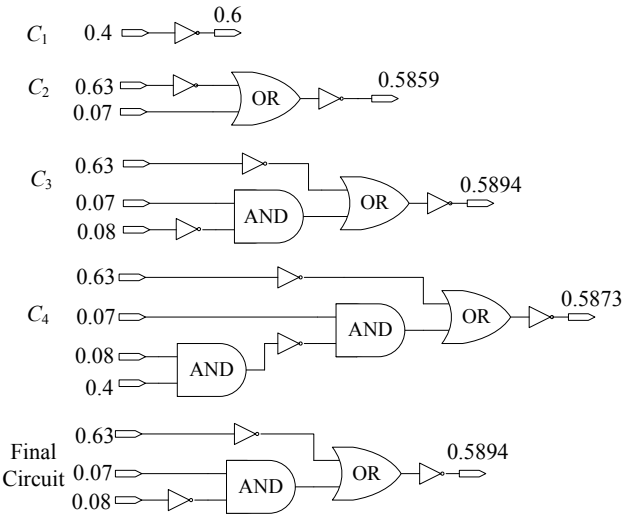


Fig. 2: The sequence of the circuits synthesized by our algorithm in solving the random bit generation problem with the source probability set $S = \{0.08, 0.63, 0.4, 0.07\}$ and the target probability $q^* = 0.59$. The final circuit is the one with the smallest output error among $C_1, \ldots, C_4$.

We conclude this section with the following example which demonstrates the major steps of our proposed algorithm. The sequence of the circuits synthesized by our algorithm together with the final circuit are shown in Fig. 2.

Example 3

Suppose that the set of source probabilities is $S = \{0.08, 0.63, 0.4, 0.07\}$ *and the target probability is* $q^* = 0.59$. *Our algorithm includes the following major steps.*

1) *Construct the circuit* C_1, *which is the base-case circuit. The augmented source probability set is* $S' = \{0.08, 0.63, 0.4, 0.07, 0.92, 0.37, 0.6, 0.93\}$. *The closest value to* q^* *in* S' *is* 0.6. *Therefore,* 0.4 *is selected as the input probability and an inverter is inserted. The circuit* C_1 *is shown in Fig. 2.*
2) *Construct the circuit* C_2. *Since* C_1 *has only one input, we only need to consider one location for the gate replacement. By calling the iterative method to find the optimal local replacement for that input, we obtain the optimal choice as shown in Fig. 2.*
3) *Construct the circuit* C_3. *By calling the iterative method to find the optimal local replacement for the top input of* C_2, *we obtain the closest output probability as* 0.5878. *If the bottom input of* C_2 *is replaced, the closest output probability is* 0.5894. *Comparing these two optimal choices, the replacement at the bottom input is better. The resulting circuit* C_3 *is shown in Fig. 2.*
4) *Construct the circuit* C_4. *The optimal replacements for the top input, the middle input, and the bottom input of* C_3 *lead to output probabilities* 0.5819, 0.6057, *and* 0.5873, *respectively. The closest one to* q^* *is* 0.5873. *Thus, we choose the bottom input for replacement. The resulting circuit* C_4 *is shown in Fig. 2.*
5) *Obtain the final circuit by choosing the one with the smallest output error among the circuit* $C_1, \ldots, C_4$. *In this example, it is* C_3. □

IV. Experimental Results

We perform two sets of experiments comparing the performance of our proposed algorithm ("our method") to the algorithm proposed in [1] ("previous method").

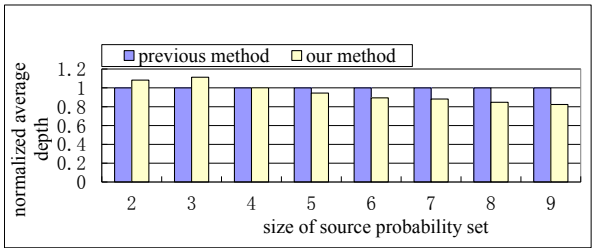
We randomly synthesize probability generation problems as our test cases. The sizes of the source probability sets range from 2 to 9. For a specific set size, we randomly generate 800 test cases whose source probability set is of that size. For each test case, its source probabilities and target probability are randomly sampled from the open unit interval $(0, 1)$. Each test case also satisfies the condition that when the previous method is applied to it, the output error $|q - q^*| \geq 0.001 \cdot q^*$.

In the first set of experiments, we aim at synthesizing a circuit with output probability q such that $|q - q^*|$ is the minimal. We compare the performance of our method with the previous method using our synthetic test cases. Table I lists the average depth, the average number of gates, the average relative output error, and the average product of the depth and the relative error of the circuits synthesized by the two methods. When counting the depth and the gate number, we ignore inverters. It is because inverters have less impact on area, delay and power consumption than AND and OR gates. The relative output error is calculated as $\frac{|q-q^*|}{q^*}$. The average product of the depth and the relative error is obtained by first multiplying the depth and the relative error of each circuit and then averaging over all the products. These metrics are averaged over all the problems with the same size of source probability set. We also plot and compare the average depths, the average relative output errors, and the average products of depth and error of the circuits synthesized by the two methods in Fig. 3. We use the results of the circuits generated by the previous method as baselines. From the figure, we can see that our method produces circuits with smaller depths and smaller output errors. Furthermore, as the set size increases, the relative output error of the circuit synthesized by our method decreases dramatically. Since the aim in this set of experiment is to synthesize a circuit with output error as small as possible, more gates may be needed for the circuit synthesized by our method than by the previous method.

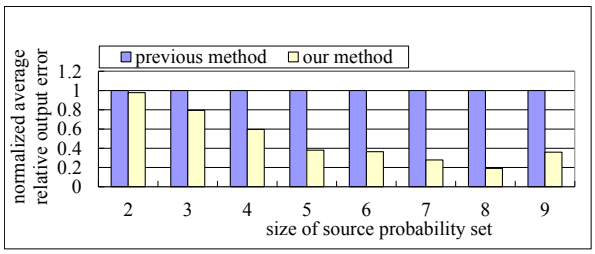
TABLE I: Comparison of the circuits synthesized by our method with the ones synthesized by the previous method.

$\|S\|$	previous method				our method			
	depth	#gates	error (%)	(depth ×error ×100)	depth	#gates	error (%)	(depth ×error ×100)
2	0.59	0.59	18.22	9.38	0.64	0.64	17.83	9.66
3	1.33	1.33	7.61	9.24	1.48	1.48	6.04	8.69
4	2.31	2.31	3.04	6.54	2.31	2.42	1.81	4.01
5	3.16	3.16	1.31	4.05	2.98	3.21	0.50	1.45
6	3.98	3.98	0.66	2.65	3.56	3.94	0.24	0.81
7	4.81	4.81	0.42	2.00	4.24	4.81	0.12	0.37
8	5.67	5.67	0.31	1.71	4.80	5.65	0.058	0.22
9	6.63	6.63	0.26	1.65	5.46	6.51	0.092	0.22

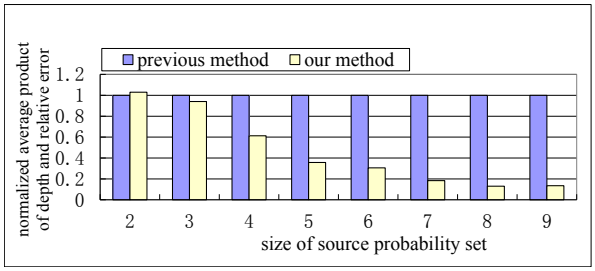
Observing that in some realistic situations, the source probabilities may also be subject to some randomness or errors due to some physical uncertainties. Therefore, we can only anticipate output probabilities that fall in some range around the target probabilities. We model this situation as another optimization problem: Besides a source probability set S and a target probability q^*, we are given an error tolerance ratio e. Our aim is to synthesize a circuit of minimal depth or gate number, whose output probability q satisfies $|q - q^*| \leq e \cdot q^*$. We consider such a minimization problem in our second set of experiments.



(a) Comparison of the average depth.



(b) Comparison of the average relative output error.



(c) Comparison of the average product of the depth and the relative output error.

Fig. 3: Comparison of three metrics of the circuits synthesized by our method to those of the circuits synthesized by the previous method.

In order to solve the new optimization problem, we only need to slightly modify our algorithm. We check each circuit synthesized in sequence to see whether its output probability error is below the threshold $e \cdot q^*$. Once we find the error is below the threshold, we immediately terminate our procedure. The last circuit is the one with the minimal depth or gate number. If none of the intermediate circuits generated in our procedure has its output error below the threshold, then we return the circuit with the smallest output error. The algorithm proposed in [1] can also be modified similarly to get a solution to the new optimization problem.

We choose $e = 0.01$ and reuse the test cases in the first set of experiments. We apply both a modification of our method and a modification of the previous method on these test cases. Table II lists the results comparing the two methods. The column titled "valid cases(%)" shows the percentage of the test cases with a specific source probability set size for which a method could find a circuit with output probability error below the threshold. As shown in the table, applying our method is more likely to find a circuit with output probability error below the threshold than using the previous method. For all the circuits with output probability error below the threshold, we average their depths and their gate numbers. The results for the two methods are shown in Table II, from which we can conclude that the circuits synthesized by our method have both smaller gate numbers and smaller depths than those synthesized by the previous method.

We also plot and compare the average depths of the circuits synthesized by the two methods in Fig. 4. We use the depths of the

circuits generated by the previous method as baselines. As shown in the figure, the relative depth of the circuit synthesized by our method decreases as the size of the source set increases.

TABLE II: Comparison of our method with the previous method in optimizing the circuit depth and gate number subject to an output error tolerance.

$\|S\|$	previous method			our method		
	valid cases(%)	#gates	depth	valid cases(%)	#gates	depth
2	5.75	1.00	1.00	6.38	1.00	1.00
3	20.88	1.52	1.52	29.25	1.47	1.47
4	45.88	2.11	2.11	66.75	2.03	1.97
5	67.75	2.51	2.51	88.50	2.24	2.14
6	83.63	3.07	3.07	96.13	2.44	2.29
7	91.25	3.57	3.57	99.25	2.59	2.41
8	96.13	4.37	4.37	99.50	2.85	2.58
9	97.25	4.79	4.79	98.50	2.93	2.60

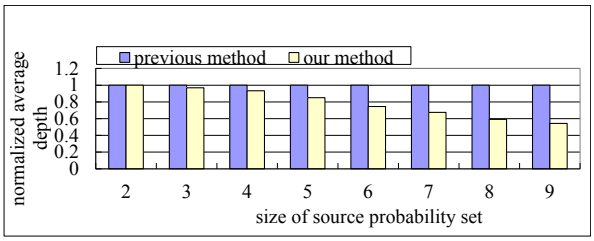


Fig. 4: Comparison of the depths of the circuits synthesized by our method to those of the circuits synthesized by the previous method in optimizing the circuit depth subject to an output error tolerance.

V. Conclusion and Future Work

In this paper, we propose a new algorithm for synthesizing combinational circuits to transform source probabilities into target probabilities. We apply a linear property of probabilistic logic computation and an iterative local search method to increase the efficiency of our algorithm. Compared to the method proposed in [1], our approach greatly enlarge the search space. Therefore, the circuits synthesized by our method have much smaller depths and output errors. However, our algorithm is still a greedy algorithm: We obtain a new candidate circuit through a local optimal change to the previous candidate circuit; other portion of the previous circuit remains the same. Therefore, the final solution largely depends on the first few candidate circuits.In our future work, we will study how to design and apply some global optimization techniques to find a better solution to the random bit generation problem.

The circuit synthesized in this paper only has one output. If we want to obtain multiple independent outputs, we require the inputs to be independent as well. It is because the outputs from the same set of input sources may have correlation. However, in some cases we need multiple output random bits that have correlation. We will study how to design an optimal circuit that generates the outputs with the required correlation in our future work.

References

[1] W. Qian, M. D. Riedel, H. Zhou, and J. Bruck, "Transforming probabilities with combinational logic," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 9, pp. 1279–1292, 2011.

[2] N. Asadi, T. Meng, and W. Wong, "Reconfigurable computing for learning Bayesian networks," in *International Symposium on Field Programmable Gate Arrays (FPGA)*, 2008, pp. 203–211.

[3] N. Asadi *et al.*, "Paralearn: a massively parallel, scalable system for learning interaction networks on FPGAs," in *International Conference on Supercomputing*, 2010, pp. 83–94.

[4] V. Mansinghka, "Natively probabilistic computation," Ph.D. dissertation, Massachusetts Institute of Technology, 2009.

[5] J. Hartmann and G. Kemntiz, "How to do weighted random testing for BIST," in *International Conference on Computer-Aided Design*, 1993, pp. 568–571.

[6] F. Muradali, V. K. Agarwal, and B. Nadeau-Dostie, "A new procedure for weighted random built-in self-test," in *International Test Conference*, 1990, pp. 660–669.

[7] B. Gaines, "Stochastic computing systems," in *Advances in Information Systems Science*. Plenum, 1969, vol. 2, ch. 2, pp. 37–172.

[8] B. Brown and H. Card, "Stochastic neural computation I: Computational elements," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 891–905, 2001.

[9] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE Transactions on Computers*, vol. 60, no. 1, pp. 93–105, 2011.

[10] B. Sunar, W. Martin, and D. Stinson, "A provably secure true random number generator with built-in tolerance to active attacks," *IEEE Transactions on Computers*, vol. 56, no. 1, pp. 109–119, 2007.

[11] B. Barak, R. Shaltiel, and E. Tromer, "True random number generators secure in a changing environment," in *International Workshop on Cryptographic Hardware and Embedded Systems*, 2003, pp. 166–180.

[12] D. Wilhelm and J. Bruck, "Stochastic switching circuit synthesis," in *International Symposium on Information Theory*, 2008, pp. 1388–1392.

[13] H. Zhou and J. Bruck, "On the expressibility of stochastic switching circuits," in *International Symposium on Information Theory*, 2009, pp. 2061–2065.

[14] W. Qian and M. D. Riedel, "The synthesis of robust polynomial arithmetic with stochastic logic," in *Design Automation Conference*, 2008, pp. 648–653.