Transforming Probabilities With Combinational Logic

Weikang Qian, Marc D. Riedel, Hongchao Zhou, and Jehoshua Bruck, *Fellow, IEEE*

***Abstract*—Schemes for probabilistic computation can exploit physical sources to generate random values in the form of bit streams. Generally, each source has a fixed bias and so provides bits with a specific probability of being one. If many different probability values are required, it can be expensive to generate all of these directly from physical sources. This paper demonstrates novel techniques for synthesizing combinational logic that transforms *source* probabilities into different *target* probabilities. We consider three scenarios in terms of whether the source probabilities are *specified* and whether they can be *duplicated*. In the case that the source probabilities are not specified and can be duplicated, we provide a specific choice, the set $\{0.4, 0.5\}$; we show how to synthesize logic that transforms probabilities from this set into arbitrary decimal probabilities. Further, we show that for any integer $n \geq 2$, there exists a single probability that can be transformed into arbitrary base-n fractional probabilities. In the case that the source probabilities are specified and cannot be duplicated, we provide two methods for synthesizing logic to transform them into target probabilities. In the case that the source probabilities are not specified, but once chosen cannot be duplicated, we provide an optimal choice.**

***Index Terms*—Logic synthesis, probabilistic logic, probabilistic signals, random bit streams, stochastic bit streams.**

I. Introduction and Background

Most digital circuits are designed to map *deterministic* inputs of zero and one to *deterministic* outputs of zero and one. An alternative paradigm is to design circuits that operate on *stochastic* bit streams. Each stream represents a real-valued number x ($0 \leq x \leq 1$) through a sequence of random bits that have probability x of being one and probability $1-x$ of being zero. Such circuits can be viewed as constructs that accept real-valued probabilities as inputs and compute real-valued probabilities as outputs.

Consider the example shown in Fig. 1. The bit streams A, B, and C represent the values 0.8, 0.5, and 0.4, respectively. Given that the input bit streams A and B are independent, an AND gate performs multiplication. Indeed, the probability of obtaining a one in the output bit stream is the product of the probabilities of obtaining a one in the input bit streams

$$\begin{aligned} c = P(C=1) &= P(A=1 \text{ and } B=1) \\ &= P(A=1)P(B=1) = a \cdot b. \end{aligned}$$

In prior work, we proposed a general method for synthesizing arbitrary *polynomial* functions through logical computation on stochastic bit streams [1]. An example is shown in Fig. 2. The circuit in the figure implements the polynomial $g(x) = \frac{1}{4} - \frac{1}{2}x + x^2$. The inputs consist of four independent stochastic bit streams X_1, X_2, Z_1, and Z_2. The streams X_1 and X_2 are *variable* inputs: both are set to one with the same probability x, but they are set independently. The streams Z_1 and Z_2 are *constant* inputs: they are set to one with probabilities $\frac{1}{4}$ and $\frac{3}{4}$, respectively. We can verify that the probability of obtaining a one in the output stochastic bit stream is $g(x)$

$$\begin{aligned} P(Y=1) &= P(Z_1=1)P(X_1=0)P(X_2=0) \\ &\quad + P(Z_2=1)P(X_1=1)P(X_2=1) \\ &= \frac{1}{4}(1-x)^2 + \frac{3}{4}x^2 \\ &= \frac{1}{4} - \frac{1}{2}x + x^2 = g(x). \end{aligned}$$

In this figure, with the input argument $x = \frac{1}{2}$, the output value is $g(x) = \frac{1}{4}$, as expected.

A premise for such designs is the availability of stochastic bit streams with the requisite probabilities. Such streams can either be generated from physical random sources or with pseudo-random constructs such as linear feedback shift registers. Fig. 3 illustrates the process. In each clock cycle, a random source generates a value R obeying a certain probability density function $f(R)$. A comparator compares the value R with a constant value C: it outputs a one if $R < C$ and a zero otherwise. The output of the comparator is a stream of random bits that have probability

$$p = \int_{-\infty}^{C} f(R)\,\mathrm{d}R \tag{1}$$

of being one.

Generating stochastic bit streams entails significant cost in terms of hardware resources. If the system employs pseudo-random number generators such as linear feedback shift registers, most of the cost is incurred in the pseudo-random source itself. The constant value can be generated relatively cheaply using a simple register [2].

If the system exploits a physical mechanism, the random source may be cheap but the constant value may be expensive

Manuscript received September 19, 2010; revised January 14, 2011; accepted March 8, 2011. Date of current version August 19, 2011. This work was supported by a grant from the Focus Center Research Program on Functional Engineered Nano-Architectonics, Semiconductor Research Corporation, under Contract 2003-NT-1107, and a CAREER Award, under Grant 0845650, from the National Science Foundation. This paper was recommended by Associate Editor I. Bahar.

W. Qian and M. D. Riedel are with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455 USA (e-mail: qianwk@umn.edu; mriedel@umn.edu).

H. Zhou and J. Bruck are with the Department of Electrical Engineering, California Institute of Technology, Pasadena, CA 91125 USA (e-mail: hzhou@caltech.edu; bruck@caltech.edu).

Digital Object Identifier 10.1109/TCAD.2011.2144630

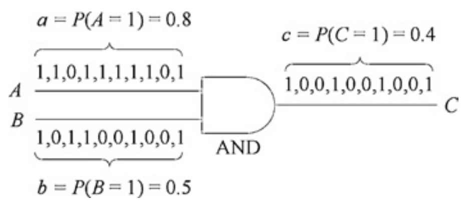


Fig. 1. AND gate multiplies the probabilities of obtaining a one in stochastic bit streams. Here, the probabilities of obtaining a one in the input streams are 0.8 and 0.5. The probability of obtaining a one in the output bit stream is $0.8 \times 0.5 = 0.4$.

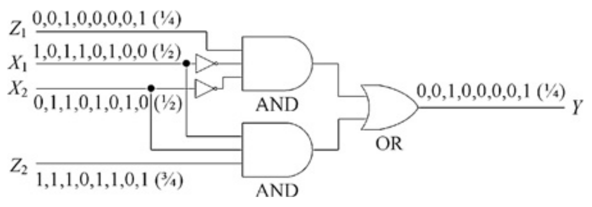


Fig. 2. Circuit implementing a polynomial $g(x) = \frac{1}{4} - \frac{1}{2}x + x^2$ on stochastic bit streams. The numbers in parentheses are the probabilities that the bits in the corresponding stream are one.

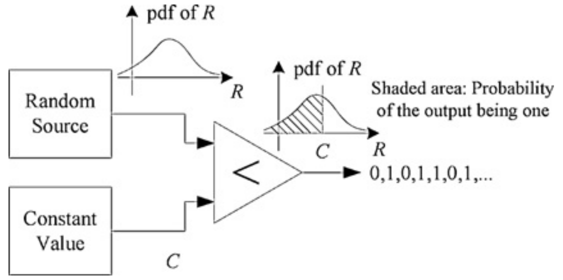


Fig. 3. Generating stochastic bit streams from random or pseudo-random sources.

to implement. For example, in [3], the authors described a scheme for exploiting the intrinsic thermal noise of nanoscale CMOS devices as the random source. This is inexpensive to do. However, in their approach, the constant value C corresponds to a supply voltage. Providing different supply voltages is comparatively expensive. If the application requires many stochastic bit streams with different probabilities, many constant values are required. The cost of generating these directly might be prohibitive.

This paper presents a synthesis strategy to mitigate this issue: we describe a method for synthesizing combinational logic to transform a set of stochastic bit streams representing a limited number of probabilities into stochastic bit streams representing other target probabilities.

It is convenient to treat stochastic bit streams mathematically as random Boolean variables. For what follows, we consider combinational logic that has random Boolean variables as inputs. When we say "a probability," we mean the probability of a random Boolean variable being one. When we say "a circuit," we mean a combinational circuit built with logic gates.

Example 1: Suppose that we have a set of source probabilities $S = \{0.4, 0.5\}$. As illustrated in Fig. 4, we can transform this set into new probabilities.

1) Given an input x with probability 0.4, an inverter will have an output z with probability 0.6 since

$$P(z = 1) = P(x = 0) = 1 - P(x = 1). \qquad (2)$$

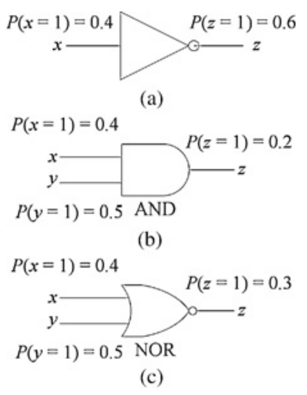


Fig. 4. Illustration of transforming a set of source probabilities into new probabilities with logic gates. (a) Inverter implementing $p_z = 1 - p_x$. (b) AND gate implementing $p_z = p_x \cdot p_y$. (c) NOR gate implementing $p_z = (1 - p_x) \cdot (1 - p_y)$.

2) Given inputs x and y with independent probabilities 0.4 and 0.5, an AND gate will have an output z with probability 0.2 since

$$\begin{aligned} P(z = 1) &= P(x = 1, y = 1) \\ &= P(x = 1)P(y = 1). \end{aligned} \qquad (3)$$

3) Given inputs x and y with independent probabilities 0.4 and 0.5, a NOR gate will have an output z with probability 0.3 since

$$\begin{aligned} P(z = 1) &= P(x = 0, y = 0) = P(x = 0)P(y = 0) \\ &= (1 - P(x = 1))(1 - P(y = 1)). \end{aligned}$$

Thus, using combinational logic, we obtain the set of probabilities $\{0.2, 0.3, 0.6\}$ from the set $\{0.4, 0.5\}$. □

Motivated by this example, we consider the problem of how to synthesize combinational logic to transform a set of source probabilities $S = \{p_1, p_2, \ldots, p_n\}$ into a target probability q. We assume that the probabilistic sources are all independent. We consider three scenarios.

1) *Scenario one:* Suppose that we are generating stochastic bit streams from physical random sources and that we have the flexibility to construct a number of constant value generators. This gives us the freedom to choose a set of source probabilities S. The cost of the random sources is negligible but the cost of generating the constant values for the comparators is considerable. Accordingly, we seek to minimize the cardinality of the set S. Note that we can produce multiple independent copies of each source probability in S cheaply, since each copy uses the same constant value. Thus, we assume that each probability in the set S can be used an arbitrary number of times. (We say that the probability can be *duplicated*.) The problem is to find a small set S and to demonstrate how to synthesize logic that transforms the values from this set into an arbitrary target probability q.
2) *Scenario two:* Suppose that we are given a collection of stochastic bit stream generators that produce a fixed set S of source probabilities. We cannot adjust the probabilities in S nor can we duplicate them; each source probability can only be used once. (Although we cannot

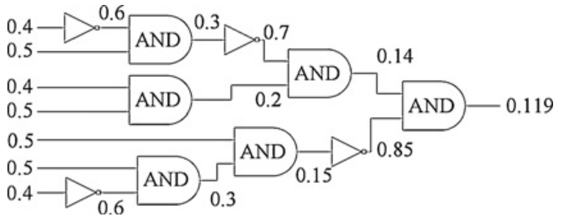


Fig. 5. Circuit synthesized by our algorithm to realize the decimal output probability 0.119 from the input probabilities 0.4 and 0.5.

duplicate the values, the set S can be a *multiset*, i.e., one that could contain multiple elements of the same value.) The problem is how to synthesize logic that has input probabilities taken from this fixed set S and produces an output probability q.

3) *Scenario three:* Suppose that we are generating stochastic bit streams with pseudo-random constructs such as linear feedback shift registers and we have full freedom to design the system. In this case, the cost of building the random sources is considerable. Suppose that we establish a budget of n random sources. Thus, the number of input stochastic bit streams is limited to n. Since it is relatively cheap to generate different constant values, we are able to choose n arbitrary source probabilities. The problem is to find a set S of n probabilities such that we can synthesize logic that transforms values from this set into an arbitrary probability q. Again, the set S can be a multiset. Since each probability in the source set S corresponds to an individual random source, each element of the set S can be used as an input probability at most once.

To summarize, we consider scenarios that differ in respect to:

1) whether the set S is specified or not;
2) whether the probabilities from S can be duplicated or not.

Specifically, in *scenario one*, the set S is not specified and the probabilities from S can be duplicated. In *scenario two*, the set S is specified and the probabilities from S cannot be duplicated. In *scenario three*, the set S is not specified and the probabilities from S cannot be duplicated.

Our contributions are as follows.

1) For *scenario one*, we demonstrate that a particular set consisting of only two elements, $S = \{0.4, 0.5\}$, can be transformed into arbitrary decimal probabilities. Further, we propose an algorithm based on fraction factorization to optimize the depth of the resulting circuit. Fig. 5 shows a circuit synthesized by our algorithm to realize the decimal output probability 0.119 from the input probabilities 0.4 and 0.5. The circuit consists of AND gates and inverters: each AND gate performs a multiplication of its inputs and each inverter performs a one-minus operation of its input.
2) Also for *scenario one*, we prove that for any given integer $n \geq 2$, there exists a set S consisting of a single element that can be transformed into arbitrary base-n fractional probabilities of the form $\frac{m}{n^d}$, where m and d are integers, satisfying that $d \geq 1$ and $0 \leq m \leq n^d$.
3) For *scenario two*, we solve the problem by transforming it into a linear 0-1 programming problem. Although approximate, the solution is optimal in terms of the difference between the target probability and the actual output probability.
4) Also for *scenario two*, we provide a greedy algorithm. Although the solution that it yields is not optimal, the difference between the target probability and the actual output probability is bounded. The algorithm runs very efficiently, yielding a solution in $O(n^2)$ time, where n is the cardinality of the set S.
5) For *scenario three*, we provide an optimal choice of the set S. Specifically, we first define a quality measure $H(S)$ for each choice S consisting of arbitrary probabilities. We prove that if the cardinality of S is n, then a lower bound on $H(S)$ is $\frac{1}{4(2^{2^n}-1)}$. Then we show that the set of source probabilities
$$S = \{p | p = \frac{2^{2^k}}{2^{2^k}+1}, k = 0, 1, \ldots, n-1\}$$
achieves the lower bound.

II. Related Work

The task of *analyzing* circuits operating on probabilistic inputs is well understood [4]. Aspects such as signal correlations of reconvergent paths must be taken into account. Algorithmic details for such analysis were first fleshed out by the testing community [5]. They have also found mainstream application for tasks such as timing and power analysis [6], [7].

The problem of synthesizing circuits to transform a given set of probabilities into a new set of probabilities appears in an early set of papers by Gill [8], [9]. He focused on synthesizing sequential state machines for this task.

Motivated by problems in neural computation, Jeavons *et al.* considered the problem of transforming stochastic binary sequences through what they call "local algorithms:" fixed functions applied to concurrent bits in different sequences [10]. This is equivalent to performing operations on stochastic bit streams with combinational logic, so in essence they were considering the same problem as we are. Their main result was a method for generating binary sequences with probability $\frac{m}{n^d}$ from a set of stochastic binary sequences with probabilities in the set $\{\frac{1}{n}, \frac{2}{n}, \ldots, \frac{n-1}{n}\}$. This is equivalent to our Theorem 2. In contrast to the work of Jeavons *et al.*, our primary focus is on minimizing the number of source probabilities needed to realize arbitrary base-n fractional probabilities.

The proponents of PCMOS discussed the problem of synthesizing combinational logic to transform probability values [11]. These authors suggested using a tree-based circuit to realize a set of target probabilities. This was positioned as future work; no details were given.

Wilhelm and Bruck proposed a general framework for synthesizing *switching circuits* to achieve a desired probability [12]. Switching circuits were originally discussed by Shannon [13]. These consist of relays that are either open or closed; the circuit computes a logical value of one if there exists a closed path through the circuit. Wilhelm and Bruck considered *stochastic* switching circuits, in which each switch has a certain probability of being open or closed. They proposed an algorithm that generates the requisite stochastic switching circuit to compute any *binary* probability.

Zhou and Bruck generalized Wilhelm and Bruck's work [14]. They considered the problem of synthesizing a stochastic switching circuit to realize an arbitrary base-n fractional probability $\frac{m}{n^d}$ from a probabilistic switch set $\{\frac{1}{n}, \frac{2}{n}, \ldots, \frac{n-1}{n}\}$. They showed that when n is a multiple of 2 or 3, such a realization is possible. However, for any prime number n greater than 3, there exists a base-n fractional probability that cannot be realized by any stochastic switching circuit.

In contrast to the work of Gill, to that of Wilhelm and Bruck, and to that of Zhou and Bruck, we consider combinational circuits: memoryless circuits consisting of logic gates. Our approach dovetails nicely with the circuit-level PCMOS constructs. It is orthogonal to the switch-based approach of Zhou and Bruck. Note that Zhou and Bruck assume that the probabilities in the given set S can be duplicated. We also consider the case where they cannot.

III. Scenario One: Set S Is Not Specified and the Elements Can Be Duplicated

In this scenario, we assume that the set S of probabilities is not specified. Once the set has been determined, each element of the set can be used as an input probability an arbitrary number of times. The inputs are all assumed to be independent. As discussed in the introduction, we seek a set S of small size.

A. Generating Decimal Probabilities

In this section, we consider the case where the target probabilities are represented as *decimal* numbers. The problem is to find a small set S of source probabilities that can be transformed into an arbitrary target decimal probability. We provide a set S consisting of two elements.

Theorem 1: With circuits consisting of fanin-two AND gates and inverters, we can transform the set of source probabilities $\{0.4, 0.5\}$ into an arbitrary decimal probability. □

Proof: First, we note that an inverter with a probabilistic input gives an output probability equal to one minus the input probability, as was shown in (2). An AND gate with two independent inputs performs a multiplication of the input probabilities, as was shown in (3). Thus, we need to prove: with the two operations $1-x$ and $x \cdot y$, we can transform the values from the set $\{0.4, 0.5\}$ into arbitrary decimal fractions. We prove this statement by induction on the number of digits n after the decimal point.

Base case:

1) $n = 0$. The values 0 and 1 correspond to deterministic inputs of zero and one, respectively.
2) $n = 1$. We can generate 0.1, 0.2, and 0.3 as follows:

$$\begin{aligned} 0.1 &= 0.4 \times 0.5 \times 0.5 \\ 0.2 &= 0.4 \times 0.5 \\ 0.3 &= (1-0.4) \times 0.5. \end{aligned}$$

Since we can generate the decimal fractions 0.1, 0.2, 0.3, and 0.4, we can generate 0.6, 0.7, 0.8, and 0.9 with an extra $1-x$ operation. Together with the source value 0.5, we can transform the pair of values 0.4 and 0.5 into any decimal fraction with one digit after the decimal point.

Inductive step:
Assume that the statement holds for all $m \le (n-1)$. Consider an arbitrary decimal fraction z with n digits after the decimal point. Let $u = 10^n \cdot z$. Here u is an integer.

Consider the following four cases.

1) The case where $0 \le z \le 0.2$.
   a) The integer u is divisible by 2. Let $w = 5z$. Then $0 \le w \le 1$ and $w = (u/2) \cdot 10^{-n+1}$, having at most $(n-1)$ digits after the decimal point. Thus, based on the induction hypothesis, we can generate w. It follows that z can be generated as $z = 0.4 \times 0.5 \times w$.
   b) The integer u is not divisible by 2 and $0 \le z \le 0.1$. Let $w = 10z$. Then $0 \le w \le 1$ and $w = u \cdot 10^{-n+1}$, having at most $(n-1)$ digits after the decimal point. Thus, based on the induction hypothesis, we can generate w. It follows that z can be generated as $z = 0.4 \times 0.5 \times 0.5 \times w$.
   c) The integer u is not divisible by 2 and $0.1 < z \le 0.2$. Let $w = 2 - 10z$. Then $0 \le w < 1$ and $w = 2 - u \cdot 10^{-n+1}$, having at most $(n-1)$ digits after the decimal point. Thus, based on the induction hypothesis, we can generate w. It follows that z can be generated as $z = (1 - 0.5 \times w) \times 0.4 \times 0.5$.
2) The case where $0.2 < z \le 0.4$.
   a) The integer u is divisible by 4. Let $w = 2.5z$. Then $0 < w \le 1$ and $w = (u/4) \cdot 10^{-n+1}$, having at most $(n-1)$ digits after the decimal point. Thus, based on the induction hypothesis, we can generate w. It follows that z can be generated as $z = 0.4 \times w$.
   b) The integer u is not divisible by 4 but is divisible by 2. Let $w = 2 - 5z$. Then $0 \le w < 1$ and $w = 2 - (u/2) \cdot 10^{-n+1}$, having at most $(n-1)$ digits after the decimal point. Thus, based on the induction hypothesis, we can generate w. It follows that z can be generated as $z = (1 - 0.5 \times w) \times 0.4$.
   c) The integer u is not divisible by 2 and $0.2 < u \le 0.3$. Let $w = 10z - 2$. Then $0 < w \le 1$ and $w = u \cdot 10^{-n+1} - 2$, having at most $(n-1)$ digits after the decimal point. Thus, based on the induction hypothesis, we can generate w. It follows that z can be generated as $z = (1 - (1 - 0.5 \times w) \times 0.5) \times 0.4$.
   d) The integer u is not divisible by 2 and $0.3 < u \le 0.4$. Let $w = 4 - 10z$. Then $0 \le w < 1$ and $w = 4 - u \cdot 10^{-n+1}$, having at most $(n-1)$ digits after the decimal point. Thus, based on the induction hypothesis, we can generate w. It follows that z can be generated as $z = (1 - 0.5 \times 0.5 \times w) \times 0.4$.
3) The case where $0.4 < z \le 0.5$. Let $w = 1 - 2z$. Then $0 \le w < 0.2$ and w falls into case 1. Thus, we can generate w. It follows that z can be generated as $z = 0.5 \times (1 - w)$.

Algorithm 1 Synthesize a circuit consisting of AND gates and inverters that transforms the probabilities from the set {0.4, 0.5} into a target decimal probability.

```
1: {Given an arbitrary decimal probability $0 \le z \le 1$.}
2: Initialize ckt;
3: while GetDigits($z$) > 1 do
4:     ($ckt, z$) ⇐ ReduceDigit($ckt, z$);
5: ckt ⇐ AddBaseCkt($ckt, z$); {Base case: $z$ has at most one
   digit after the decimal point.}
6: return ckt;
```

4) The case where $0.5 < z \le 1$. Let $w = 1 - z$. Then $0 \le w < 0.5$ and w falls into one of the above three cases. Thus, we can generate w. It follows that z can be generated as $z = 1 - w$.

For all of the above cases, we proved that we can transform the pair of values 0.4 and 0.5 into z with the two operations $1 - x$ and $x \cdot y$. Thus, we proved the statement for all $m \le n$. By induction, the statement holds for all integers n. □

Based on the proof above, we derive an algorithm to synthesize a circuit that transforms the probabilities from the set {0.4, 0.5} into an arbitrary decimal probability z. This is shown in Algorithm 1.

The function GetDigits(z) in Algorithm 1 returns the number of digits after the decimal point of z. The algorithm iterates until z has at most one digit after the decimal point. During each iteration, it calls the function ReduceDigit(ckt, z). This function, shown in Algorithm 2, converts z into a number w with one less digit after the decimal point than z. It is implemented based on the inductive step in the proof of Theorem 1. Finally, the algorithm calls the function AddBaseCkt(ckt, z) to add logic gates to realize a number z with at most one digit after the decimal point; this corresponds to the base case of the proof.

The function ReduceDigit(ckt, z) in Algorithm 2 builds the circuit from the output back to the inputs. During its construction, the circuit always has a single dangling input. Initially, the circuit is just a wire connecting an input to the output. The function AddInverter(ckt) attaches an inverter to the dangling input creating a new dangling input. The function AddAND(ckt, p) attaches a fanin-two AND gate to the dangling input; one of the AND gate's inputs is the new dangling input; the other is set to a random source of probability p. In Algorithm 2, Lines 3–4 correspond to Case 4 in the proof, Lines 5–7 correspond to Case 3, Lines 8–15 correspond to Case 1, and Lines 16–26 correspond to Case 2.

The area complexity of the synthesized circuit is linear in the number of digits after the target value's decimal point, since at most 3 AND gates and 3 inverters are needed to generate a value with n digits after the decimal point from a value with $(n-1)$ digits after the decimal point.[1] The number of AND gates in the synthesized circuit is at most $3n$.

Example 2: We show how to generate the probability value 0.757. Based on Algorithm 1, we can derive a sequence of operations that transform 0.757 to 0.7

[1]In Case 3, z is transformed into $w = 1 - 2z$ where w falls in Case 1(a). Thus, we actually need only 3 AND gates and 1 inverter for Case 3. For the other cases, it is not hard to see that we need at most 3 AND gates and 3 inverters.

Algorithm 2 ReduceDigit(ckt, z)

```
1: {Given a partial circuit ckt and an arbitrary decimal
   probability $0 \le z \le 1$.}
2: $n$ ⇐ GetDigits($z$);
3: if $z > 0.5$ then {Case 4}
4:     $z \Leftarrow 1 - z$; AddInverter($ckt$);
5: if $0.4 < z \le 0.5$ then {Case 3}
6:     $z \Leftarrow z/0.5$; AddAND($ckt$, 0.5);
7:     $z \Leftarrow 1 - z$; AddInverter($ckt$);
8: if $z \le 0.2$ then {Case 1}
9:     $z \Leftarrow z/0.4$; AddAND($ckt$, 0.4);
10:    $z \Leftarrow z/0.5$; AddAND($ckt$, 0.5);
11:    if GetDigits($z$) < $n$ then
12:        go to END;
13:    if $z > 0.5$ then
14:        $z \Leftarrow 1 - z$; AddInverter($ckt$);
15:    $z = z/0.5$; AddAND($ckt$, 0.5);
16: else {Case 2: $0.2 < z \le 0.4$}
17:    $z \Leftarrow z/0.4$; AddAND($ckt$, 0.4);
18:    if GetDigits($z$) < $n$ then
19:        go to END;
20:    $z \Leftarrow 1 - z$; AddInverter($ckt$);
21:    $z \Leftarrow z/0.5$; AddAND($ckt$, 0.5);
22:    if GetDigits($z$) < $n$ then
23:        go to END;
24:    if $z > 0.5$ then
25:        $z \Leftarrow 1 - z$; AddInverter($ckt$);
26:    $z = z/0.5$; AddAND($ckt$, 0.5);
27: END: return $ckt, z$;
```

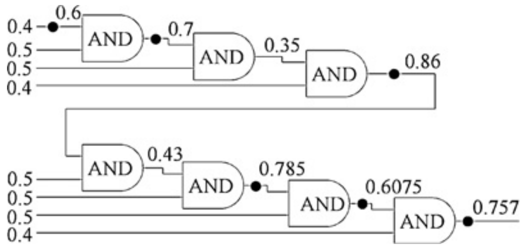


Fig. 6. Circuit transforming the set of source probabilities {0.4, 0.5} into a decimal output probability of 0.757.

$$\begin{aligned} 0.757 &\overset{1-}{\Rightarrow} 0.243 \overset{/0.4}{\Rightarrow} 0.6075 \overset{1-}{\Rightarrow} 0.3925 \overset{/0.5}{\Rightarrow} 0.785 \\ &\overset{1-}{\Rightarrow} 0.215 \overset{/0.5}{\Rightarrow} 0.43 \\ 0.43 &\overset{/0.5}{\Rightarrow} 0.86 \overset{1-}{\Rightarrow} 0.14 \overset{/0.4}{\Rightarrow} 0.35 \overset{/0.5}{\Rightarrow} 0.7. \end{aligned}$$

Since 0.7 can be realized as $0.7 = 1 - (1 - 0.4) \times 0.5$, we obtain the circuit shown in Fig. 6. (Note that here we use a black dot to represent an inverter.) □

Remarks.

1) One may question the usefulness of synthesizing a circuit that generates arbitrary *decimal* fractions. Wilhelm and Bruck proposed a scheme for synthesizing switching circuits that generate arbitrary *binary* probabilities [12]. By mapping every switch connected in series to an AND gate and every switch connected in parallel to an

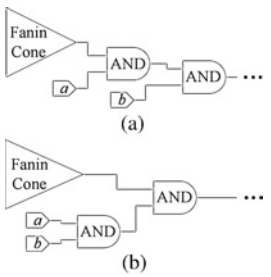


Fig. 7. Illustration of balancing to reduce the depth of the circuit. Here a and b are primary inputs. (a) Circuit before balancing. (b) Circuit after balancing.

OR gate, we can easily derive a combinational circuit that generates an arbitrary binary probability. Since any decimal fractional value can be approximated by a binary fractional value, we can build combinational circuits implementing decimal probabilities this way. However, the circuits synthesized by our procedure are less costly in terms of area.

To see this, consider a decimal fraction q with n digits. The circuit that Algorithm 1 synthesizes to generate q has at most $3n$ AND gates. For the approximation error of the binary fraction for q to be below $1/10^n$, the number of digits m of the binary fraction should be greater than $n \log_2 10$. In [12], it is proved that the minimal number of probabilistic switches needed to generate a binary fraction of m digits is m. Assuming that we build an equivalent combinational circuit consisting of AND gates and inverters, we need $m - 1$ AND gates to implement the binary fraction.[2] Thus, the combinational logic realizing the binary approximation needs more than $n \log_2 10 \approx 3.32n$ AND gates. This is more than the number of AND gates in the circuit synthesized by our procedure.

2) In many applications, we need to generate many different target probabilities. To make these target probabilities *independent*, we can generate each of them from a different collection of input probabilities. It is inexpensive to generate different collections of input probabilities taking values from the source set, since we can generate independent copies of each probability in the source set cheaply.

B. Reducing the Depth

The circuits produced by Algorithm 1 have a linear topology (i.e., each gate adds to the depth of the circuit). For practical purposes, we want circuits with shallower depth. In this section, we explore two kinds of optimizations for reducing the depth.

The first kind of optimization is at the logic level. The circuit synthesized by Algorithm 1 is composed of inverters and AND gates. We can reduce its depth by properly repositioning certain AND gates, as illustrated in Fig. 7. We refer to such optimization as *balancing*.

The second kind of optimization is at a higher level, based on the factorization of the decimal fraction. We use the following example to illustrate the basic idea.

[2]Of course, an OR gate can be converted into an AND gate with inverters at both the inputs and the output.

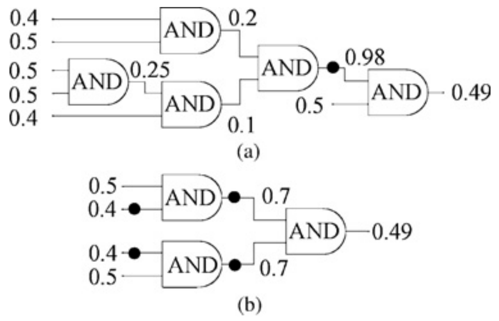


Fig. 8. Synthesizing combinational logic to generate the probability 0.49. (a) Circuit synthesized through Algorithm 1. (b) Circuit synthesized based on fraction factorization.

Example 3: Suppose we want to generate the decimal probability value 0.49.

Method based on Algorithm 1: We can derive the following transformation sequence:

$$0.49 \overset{/0.5}{\Rightarrow} 0.98 \overset{1-}{\Rightarrow} 0.02 \overset{/0.4}{\Rightarrow} 0.05 \overset{/0.5}{\Rightarrow} 0.1.$$

The synthesized circuit is shown in Fig. 8(a). Notice that the circuit is balanced; it has five AND gates and a depth of four.[3] Method based on factorization: Notice that 0.49 = 0.7 × 0.7. Thus, we can generate the probability 0.7 twice and feed these values into an AND gate. The synthesized circuit is shown in Fig. 8(b). Compared to the circuit in Fig. 8(a), both the number of AND gates and the depth of the circuit are reduced. □

Algorithm 3 shows the procedure that synthesizes the circuit based on the factorization of the decimal fraction. The factorization is actually carried out on the numerator. A crucial function is PairCmp(a_l, a_r, b_l, b_r), which compares the integer factor pair (a_l, a_r) with the pair (b_l, b_r) and returns a positive (negative) value if the pair (a_l, a_r) is better (worse) than the pair (b_l, b_r). Algorithm 4 shows how the function PairCmp(a_l, a_r, b_l, b_r) is implemented.

The quality of a factor pair (a_l, a_r) should reflect the depth of the circuit that generates the original probability based on that factorization. For this purpose, we define a function EstDepth(x) to estimate the depth of the circuit that generates the decimal fraction with a numerator x. If $1 \le x \le 9$, the corresponding fraction is $x/10$. EstDepth(x) is set as the depth of the circuit that generates the fraction $x/10$, which is

$$\text{EstDepth}(x) = \begin{cases} 0 & x = 4, 5, 6 \\ 1 & x = 2, 3, 7, 8 \\ 2 & x = 1, 9. \end{cases}$$

When $x \ge 10$, we use a simple heuristic to estimate the depth: we let EstDepth(x) = $\lceil \log_{10}(x) \rceil + 1$. The intuition behind this is that the depth of the circuit is a monotonically increasing function of the number of digits of x. The estimated depth of the circuit that generates the original fraction based on the factor pair (a_l, a_r) is

$$\max\{\text{EstDepth}(a_l), \text{EstDepth}(a_r)\} + 1. \quad (4)$$

[3]When counting depth, we ignore inverters.

Algorithm 3 ProbFactor(ckt, z)

```
1: {Given a partial circuit $ckt$ and an arbitrary decimal probability $0 \le z \le 1$.}
2: $n \Leftarrow$ GetDigits($z$);
3: if $n \le 1$ then
4:     $ckt \Leftarrow$ AddBaseCkt($ckt$, $z$);
5:     return $ckt$;
6: $u \Leftarrow 10^n z$; $(u_l, u_r) \Leftarrow (1, u)$; {$u$ is the numerator of the fraction $z$}
7: for each factor pair $(a, b)$ of $u$ do
8:     if PairCmp($u_l, u_r, a, b$) $< 0$ then
9:         $(u_l, u_r) \Leftarrow (a, b)$; {Choose the best factor pair for $z$}
10: $w \Leftarrow 10^n - u$; $(w_l, w_r) \Leftarrow (1, w)$;
11: for each factor pair $(a, b)$ of $w$ do
12:     if PairCmp($w_l, w_r, a, b$) $< 0$ then
13:         $(w_l, w_r) \Leftarrow (a, b)$; {Choose the best factor pair for $1 - z$}
14: if PairCmp($u_l, u_r, w_l, w_r$) $< 0$ then
15:     $(u_l, u_r) \Leftarrow (w_l, w_r)$; $z \Leftarrow w/10^n$;
16:     AddInverter($ckt$);
17: if IsTrivialPair($u_l, u_r$) then {$u_l = 1$ or $u_r = 1$}
18:     $(ckt, z) \Leftarrow$ ReduceDigit($ckt$, $z$);
19:     $ckt \Leftarrow$ ProbFactor($ckt$, $z$);
20:     return $ckt$;
21: $n_l \Leftarrow \lceil \log_{10}(u_l) \rceil$; $n_r \Leftarrow \lceil \log_{10}(u_r) \rceil$;
22: if $n_l + n_r > n$ then {Unable to factor $z$ into two decimal fractions in the unit interval}
23:     $(ckt, z) \Leftarrow$ ReduceDigit($ckt$, $z$);
24:     $ckt \Leftarrow$ ProbFactor($ckt$, $z$);
25:     return $ckt$;
26: $z_l \Leftarrow u_l/10^{n_l}$; $z_r \Leftarrow u_r/10^{n_r}$;
27: $ckt_l \Leftarrow$ ProbFactor($ckt_l$, $z_l$);
28: $ckt_r \Leftarrow$ ProbFactor($ckt_r$, $z_r$);
29: Connect the input of $ckt$ to an AND gate with two inputs as $ckt_l$ and $ckt_r$;
30: if $n_l + n_r < n$ then
31:     AddExtraLogic($ckt$, $n - n_l - n_r$);
32: return $ckt$;
```

The function PairCmp(a_l, a_r, b_l, b_r) essentially compares the quality of pair (a_l, a_r) and pair (b_l, b_r) based on (4). Further details are given in Algorithm 4.

In Algorithm 3, Lines 2–5 correspond to the trivial fractions. If the fraction z is non-trivial, Lines 6–9 choose the best factor pair (u_l, u_r) of u, where u is the numerator of the fraction z. Lines 10–13 choose the best factor pair (w_l, w_r) of w, where w is the numerator of the fraction $1 - z$. Finally, Lines 14–16 choose the better factor pair of (u_l, u_r) and (w_l, w_r). Here, we consider the factorization on both z and $1 - z$, since in some cases the latter might be better than the former. An example is $z = 0.37$. Note that $1 - z = 0.63 = 0.7 \times 0.9$; this has a better factor pair than z itself.

After obtaining the best factor pair, we check whether we can use it. Lines 17–20 check whether the factor pair (u_l, u_r) is trivial; a factor pair is considered trivial if $u_l = 1$ or $u_r = 1$. If the best factor pair is trivial, we call the function

Algorithm 4 PairCmp(a_l, a_r, b_l, b_r)

```
1: {Given two integer factor pairs $(a_l, a_r)$ and $(b_l, b_r)$}
2: $c_l \Leftarrow$ EstDepth($a_l$); $c_r \Leftarrow$ EstDepth($a_r$);
3: $d_l \Leftarrow$ EstDepth($b_l$); $d_r \Leftarrow$ EstDepth($b_r$);
4: Order($c_l, c_r$); {Order $c_l$ and $c_r$, so that $c_l \le c_r$}
5: Order($d_l, d_r$); {Order $d_l$ and $d_r$, so that $d_l \le d_r$}
6: if $c_r < d_r$ then {The circuit w.r.t. the first pair has smaller depth}
7:     return 1;
8: else if $c_r > d_r$ then {The circuit w.r.t. the first pair has larger depth}
9:     return -1;
10: else
11:     if $c_l < d_l$ then {The circuit w.r.t. the first pair has fewer ANDs}
12:         return 1;
13:     else if $c_l > d_l$ then {The circuit w.r.t. the first pair has more ANDs}
14:         return -1;
15:     else
16:         return 0;
```

ReduceDigit(ckt, z) in Algorithm 2 to transform z into a new value with one less digit after the decimal point. Then we perform factorization on the new value.

If the best factor pair is non-trivial, Lines 21–25 continue to check whether the factor pair can be transformed into two decimal fractions in the unit interval. Let n_l be the number of digits of the integer u_l and n_r be the number of digits of the integer u_r. If $n_l + n_r > n$, where n is the number of digits after the decimal point of z, then it is impossible to use the factor pair (u_l, u_r) to factorize z. For example, consider $z = 0.143$. Although we could factorize 143 as 11×13, we cannot use the factor pair (11, 13) since the factorization 0.11×1.3 and the factorization 1.1×0.13 both contain a fraction larger than 1; a probability value can never be larger than 1.

Finally, if it is possible to use the best factor pair, Lines 26–29 synthesize two circuits for fractions $u_l/10^{n_l}$ and $u_r/10^{n_r}$, respectively, and then combine these two circuits with an AND gate. Lines 30–31 check whether $n > n_l + n_r$. If this is the case, we have

$$z = u/10^n = u_l/10^{n_l} \cdot u_r/10^{n_r} \cdot 0.1^{n-n_l-n_r}.$$

We need to add an extra AND gate with one input probability as $0.1^{n-n_l-n_r}$ and the other input probability as $u_l/10^{n_l} \cdot u_r/10^{n_r}$. The extra logic is added through the function AddExtraLogic(ckt, m).

C. Empirical Validation

We empirically validate the effectiveness of the synthesis scheme that was presented in the previous section. For logic-level optimization, we use the "balance" command of the synthesis tool ABC [15]. We find that it is very effective in reducing the depth of tree-style circuits.[4]

[4]We find that the other synthesis commands of ABC such as "rewrite" do not affect the depth or the number of AND gates of a tree-style AND-inverter graph.

TABLE I

COMPARISON OF THE BASIC SYNTHESIS SCHEME, THE BASIC SYNTHESIS SCHEME WITH BALANCING, AND THE FACTORIZATION-BASED SYNTHESIS SCHEME WITH BALANCING

Number	Basic		Basic+Balance			Factor+Balance				
of Digits	#AND	Depth	#AND	Depth	Runtime	#AND	Depth	Runtime	#AND Imprv. (%)	Depth Imprv. (%)
n			a_1	d_1	(ms)	a_2	d_2	(ms)	$100(a_1 - a_2)/a_1$	$100(d_1 - d_2)/d_1$
2	3.67	3.67	3.67	2.98	0.22	3.22	2.62	0.22	12.1	11.9
3	6.54	6.54	6.54	4.54	0.46	5.91	3.97	0.66	9.65	12.5
4	9.47	9.47	9.47	6.04	1.13	8.57	4.86	1.34	9.45	19.4
5	12.43	12.43	12.43	7.52	0.77	11.28	5.60	0.94	9.21	25.6
6	15.40	15.40	15.40	9.01	1.09	13.96	6.17	1.48	9.36	31.5
7	18.39	18.39	18.39	10.50	0.91	16.66	6.72	1.28	9.42	35.9
8	21.38	21.38	21.38	11.99	0.89	19.34	7.16	1.35	9.55	40.3
9	24.37	24.37	24.37	13.49	0.75	22.05	7.62	1.34	9.54	43.6
10	27.37	27.37	27.37	14.98	1.09	24.74	7.98	2.41	9.61	46.7
11	30.36	30.36	30.36	16.49	0.92	27.44	8.36	2.93	9.61	49.3
12	33.35	33.35	33.35	17.98	0.73	30.13	8.66	4.13	9.65	51.8

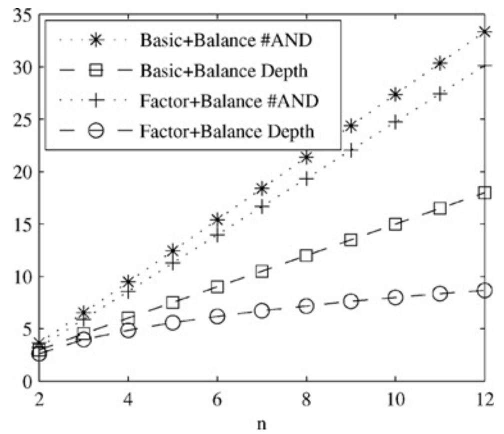


Fig. 9. Average number of AND gates and depth of the circuits versus n.

Table I compares the quality of the circuits generated by three different schemes. The first scheme, called "Basic," is based on Algorithm 1. It generates a linear-style circuit. The second scheme, called "Basic+Balance," combines Algorithm 1 and the logic-level balancing algorithm. The third scheme, called "Factor+Balance," combines Algorithm 3 and the logic-level balancing algorithm. We perform experiments on a set of target decimal probabilities that have n digits after the decimal point and average the results. Table I shows the results for n ranging from 2 to 12. When $n \leq 5$, we synthesize circuits for all possible decimal probabilities with n digits after the decimal point. When $n \geq 6$, we randomly choose 100 000 decimal probabilities with n digits after the decimal point as the synthesis targets. We show the average number of AND gates, the average depth, and the average CPU runtime.

From Table I, we can see that both the "Basic+Balance" and the "Factor+Balance" synthesis schemes have only millisecond-order CPU runtimes. Compared to the "Basic+Balance" scheme, the "Factor+Balance" scheme reduces the average number of AND gates by 10% and the average depth by more than 10%, for all n. The percentage of reduction of the average depth increases with increasing n. For $n = 12$, the average depth of the circuits is reduced by more than 50%.

In Fig. 9, we plot the average number of AND gates and the average depth of the circuits versus n for the "Basic+Balance" and "Factor+Balance" schemes. The figure shows that the "Factor+Balance" scheme is clearly superior. The average number of AND gates in the circuits synthesized by both schemes increases linearly with n. The average depth of the circuits synthesized by the "Basic+Balance" scheme also increases linearly with n. In contrast, the average depth of the circuits synthesized by the "Factor+Balance" scheme increases logarithmically with n.

D. Generating Base-n Fractional Probabilities

In Section III-A, we showed that there exists a pair of probabilities that can be transformed into an arbitrary decimal probability. In [16], we showed that we can further reduce the number of source probabilities down to one: there exists a real number $0 \leq r \leq 1$ that can be transformed into an arbitrary decimal probability with combinational logic. However, this number r is an irrational root of a polynomial. Here, we generalize this result. We show that for any integer $n \geq 2$, there exists a real number $0 \leq r \leq 1$ that can be transformed into an arbitrary base-n fractional probability $\frac{m}{n^d}$ with combinational logic.

First, we show that we can transform a set of probabilities $\{\frac{1}{n}, \frac{2}{n}, \ldots, \frac{n-1}{n}\}$ into an arbitrary base-n fractional probability $\frac{m}{n^d}$.

Theorem 2: Let $n \geq 2$ be an integer. For any integers $d \geq 1$ and $0 \leq m \leq n^d$, we can transform the set of probabilities $\{\frac{1}{n}, \frac{2}{n}, \ldots, \frac{n-1}{n}\}$ into a base-n fractional probability $\frac{m}{n^d}$ with a circuit having $2d - 1$ inputs. □

Proof: We prove the above claim by induction on d.

Base case: When $d = 1$, we can obtain each base-n fractional probability $\frac{m}{n}$ $(0 \leq m \leq n)$ directly from an input since the input probability set is $\{\frac{1}{n}, \ldots, \frac{n-1}{n}\}$ and the probabilities 0 and 1 correspond to deterministic values of zero and one, respectively.

Inductive step: Assume the claim holds for $d-1$. Now consider any integer $0 \leq m \leq n^d$. We can write m as $m = an^{d-1} + b$ with an integer $0 \leq a < n$ and an integer $0 \leq b \leq n^{d-1}$.

Consider a multiplexer with data input x_1 and x_2, selecting input s, and output y, as shown in Fig. 10. The Boolean function of the multiplexer is

$$y = (x_1 \wedge s) \vee (x_2 \wedge \neg s).^5$$

By the induction hypothesis, we can transform the set of probabilities $\{\frac{1}{n}, \frac{2}{n}, \ldots, \frac{n-1}{n}\}$ into the probability $\frac{b}{n^{d-1}}$ with a

[5]When discussing Boolean functions, we use $\wedge$, $\vee$, and $\neg$ to represent logical AND, OR, and negation, respectively. We adopt this convention since we use + and · to represent *arithmetic* addition and multiplication, respectively.

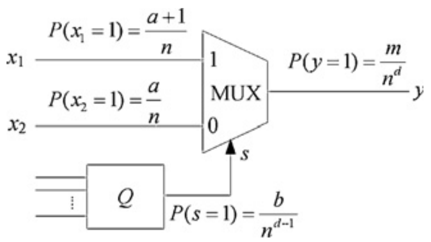


Fig. 10. Circuit generating the base-n fractional probability $\frac{m}{n^d}$, where m is written as $m = an^{d-1} + b$ with $0 \leq a < n$ and $0 \leq b \leq n^{n-1}$. The circuit Q in the figure generates the base-n fractional probability $\frac{b}{n^{d-1}}$.

circuit Q that has $2d - 3$ inputs. In order to generate the output probability $\frac{m}{n^d}$, we let the inputs x_1 and x_2 of the multiplexer have probability $\frac{a+1}{n}$ and $\frac{a}{n}$, respectively, and we connect the input s to the output of a circuit Q that generates the probability $\frac{b}{n^{d-1}}$, as shown in Fig. 10. Note that the inputs to x_1 and x_2 are either probabilistic inputs with a value from the set $\{\frac{1}{n}, \ldots, \frac{n-1}{n}\}$, or deterministic inputs of zero or one. With the primary inputs of the entire circuit being independent, all the inputs of the multiplexer are also independent. The probability that y is one is

$$\begin{aligned} P(y=1) &= P(x_1=1, s=1) + P(x_2=1, s=0) \\ &= P(x_1=1)P(s=1) + P(x_2=1)P(s=0) \\ &= \frac{a+1}{n}\frac{b}{n^{d-1}} + \frac{a}{n}\left(1 - \frac{b}{n^{d-1}}\right) \\ &= \frac{an^{d-1}+b}{n^d} = \frac{m}{n^d}. \end{aligned}$$

Therefore, we can transform the set of probabilities $\{\frac{1}{n}, \frac{2}{n}, \ldots, \frac{n-1}{n}\}$ into the probability $\frac{m}{n^d}$ with a circuit that has $2d - 3 + 2 = 2d - 1$ inputs. Thus, the claim holds for d. By induction, the claim holds for all $d \geq 1$. □

Remarks.

1) An equivalent result to Theorem 2 can be found in [10]. There it is couched in information theoretic language in terms of concurrent operations on random binary sequences.
2) Our proof of Theorem 2 is constructive. It shows that we can synthesize a chain of $d-1$ multiplexers to generate a base-n fractional probability $\frac{m}{n^d}$.
3) If some of the inputs to the chain of multiplexers are deterministic zeros or ones, we can further simplify the circuit. In such cases, the number of inputs of the entire circuit and the area of the circuit can be further reduced.

Next, we prove a theorem about the existence of a single real value that can be transformed into any value in a given set of rational probabilities through combinational logic.

Theorem 3: For any finite set of rational probabilities $R = \{p_1, p_2, \ldots, p_M\}$, there exists a real number $0 < r < 1$ that can be transformed into probabilities in the set R through combinational logic. □

Proof: We only need to prove that the statement is true under the condition that for all $1 \leq i \leq M$, $0 \leq p_i \leq 0.5$. In fact, given a general set of probabilities $R = \{p_1, p_2, \ldots, p_M\}$, we can derive a new set of probabilities $R^* = \{p_1^*, p_2^*, \ldots, p_M^*\}$, such that for all $1 \leq i \leq M$

$$p_i^* = \begin{cases} p_i, & \text{if } p_i \leq 0.5 \\ 1 - p_i, & \text{if } p_i > 0.5. \end{cases}$$

Then, for all $1 \leq i \leq M$, the element p_i^* of R^* satisfies that $0 \leq p_i^* \leq 0.5$. Once we prove that there exists a real number $0 < r < 1$ which can be transformed into any of the probabilities in the set R^*, then any probability in the original set R can also be generated from this value r: to generate $p_i = p_i^*$, we use the same circuit that generates the probability p_i^*; to generate $p_i = 1 - p_i^*$, we append an inverter to the output.

Therefore, we assume that for all $1 \leq i \leq M$, $0 \leq p_i \leq 0.5$. Further, without loss of generality, we can assume that $0 \leq p_1 < \cdots < p_M \leq 0.5$. Since probability 0 can be realized trivially by a deterministic value of zero, we assume that $p_1 > 0$. Since $p_1, \ldots, p_M$ are rational probabilities, there exist positive integers $a_1, \ldots, a_M$ and b such that for all $1 \leq i \leq M$, $p_i = \frac{a_i}{b}$. Since $0 < p_1 < \cdots < p_M \leq 0.5$, we have $0 < a_1 < \cdots < a_M \leq \frac{b}{2}$.

First, it is not hard to see that there exists a positive integer h such that $2^{h-1} > a_M h + 1$. For $k = 1, \ldots, h$, let $c_k = \left\lfloor \frac{\binom{h}{k}}{a_M} \right\rfloor$, where $\lfloor x \rfloor$ represents the largest integer less than or equal to x.

We will prove

$$a_M \sum_{k=1}^{h} c_k > 2^{h-1}. \tag{5}$$

In fact

$$\begin{aligned} 2^h - a_M \sum_{k=1}^{h} c_k &= \sum_{k=0}^{h} \binom{h}{k} - \sum_{k=1}^{h} \left\lfloor \frac{\binom{h}{k}}{a_M} \right\rfloor a_M \\ &= 1 + \sum_{k=1}^{h} \left(\frac{\binom{h}{k}}{a_M} - \left\lfloor \frac{\binom{h}{k}}{a_M} \right\rfloor \right) a_M. \end{aligned}$$

Since $x - \lfloor x \rfloor < 1$, we have

$$2^h - a_M \sum_{k=1}^{h} c_k < 1 + \sum_{k=1}^{h} a_M = a_M h + 1 < 2^{h-1}$$

or $a_M \sum_{k=1}^{h} c_k > 2^{h-1}$.

Now consider the polynomial $f(x) = \sum_{k=1}^{h} c_k x^k (1-x)^{h-k}$.

Note that $f(0) = 0$ and $f(0.5) = \frac{1}{2^h} \sum_{k=1}^{h} c_k$. Based on (5) and the fact that $a_M \leq \frac{b}{2}$, we have

$$f(0.5) > \frac{1}{2a_M} \geq \frac{1}{b}.$$

Thus, $f(0) = 0 < \frac{1}{b} < f(0.5)$. Based on the continuity of the polynomial f, there exists a real number $0 < r < 0.5 < 1$ such that $f(r) = \frac{1}{b}$.

For all $i = 1, \ldots, M$, set $l_{i,0} = 0$. For all $i = 1, \ldots, M$ and all $k = 1, 2, \ldots, h$, set $l_{i,k} = a_i c_k$. Since for all $k = 1, \ldots, h$,

c_k is an integer and $0 \le c_k \le \frac{\binom{h}{k}}{a_M}$, then for all $i = 1, \dots, M$ and all $k = 1, 2, \dots, h$, $l_{i,k}$ is an integer and $0 \le l_{i,k} = a_i c_k \le a_M c_k \le \binom{h}{k}$.

For $k = 0, 1, \dots, h$, let $A_k = \{(a_1, a_2, \dots, a_h) \in \{0, 1\}^h : \sum_{i=1}^{h} a_i = k\}$ (i.e., A_k consists of h-tuples over $\{0, 1\}$ having exactly k ones). For any $1 \le i \le M$, consider a circuit with h inputs realizing a Boolean function that takes exactly $l_{i,k}$ values 1 on each A_k ($k = 0, 1, \dots, h$). If we set all the input probabilities to be r, then the output probability is

$$\begin{aligned} p_o &= \sum_{k=0}^{h} l_{i,k} r^k (1-r)^{h-k} = \sum_{k=1}^{h} a_i c_k r^k (1-r)^{h-k} \\ &= a_i f(r) = \frac{a_i}{b}. \end{aligned}$$

Thus, we can transform r into any number in the set $\{p_1, \dots, p_M\}$ through combinational logic. □

Theorems 2 and 3 lead to the following corollary.

Corollary 1: Given an integer $n \ge 2$, there exists a real number $0 < r < 1$ which can be transformed into any base-n fractional probability $\frac{m}{n^d}$ (d and m are integers with $d \ge 1$ and $0 \le m \le n^d$) through combinational logic. □

Proof: Based on Theorem 3, there exists a real number $0 < r < 1$ which can be transformed into any probability in the set $\{\frac{1}{n}, \frac{2}{n}, \dots, \frac{n-1}{n}\}$. Further, based on Theorem 2, the statement in the corollary holds. □

IV. Scenario Two: Set S Is Specified and the Elements Cannot Be Duplicated

The problem considered in this scenario is: given a set $S = \{p_1, p_2, \dots, p_n\}$ and a target probability q, construct a circuit that, given inputs with probabilities from S, produces an output with probability q. Each element of S can be used as an input probability no more than once.

A. An Optimal Solution

In this section, we show an optimal solution to the problem based on linear 0-1 programming. With the assumption that the probabilities cannot be duplicated, we are building a circuit with n inputs, the ith input of which has probability p_i. (If a probability is not used, then the corresponding input is just a dummy.)

Our method is based on a truth table for n variables. Each row of the truth table is annotated with the probability that the corresponding input combination occurs. Assume that the n variables are $x_1, x_2, \dots, x_n$ and x_i has probability p_i. Then, the probability that the input combination $x_1 = a_1, x_2 = a_2, \dots, x_n = a_n$ ($a_i \in \{0, 1\}$, for $i = 1, \dots, n$) occurs is

$$P(x_1 = a_1, x_2 = a_2, \dots, x_n = a_n) = \prod_{i=1}^{n} P(x_i = a_i).$$

A truth table for a two-input XOR gate is shown in Table II. The fourth column is the probability that each input combination occurs. Here $P(x = 1) = p_x$ and $P(y = 1) = p_y$.

The output probability is the sum of the probabilities of input combinations that produce an output of one. Assume that the probability of the ith input combination, corresponding to minterm m_i, is r_i ($0 \le i \le 2^n - 1$) and that the output of the circuit corresponding to the ith input combination is z_i ($z_i \in \{0, 1\}, 0 \le i \le 2^n - 1$). Then, the output probability is

TABLE II
Truth Table for a Two-Input XOR Gate

x	y	z	Probability
0	0	0	$(1-p_x)(1-p_y)$
0	1	1	$(1-p_x)p_y$
1	0	1	$p_x(1-p_y)$
1	1	0	$p_x p_y$

$$p_o = \sum_{i=0}^{2^n-1} z_i r_i. \tag{6}$$

For the example in Table II, the output probability is

$$p_o = r_1 + r_2 = (1-p_x)p_y + p_x(1-p_y).$$

Thus, constructing a circuit with output probability q is equivalent to determining the z_i such that (6) evaluates to q. In the general case, depending on the values of p_i and q, it is possible that q cannot be exactly realized by any circuit. The problem then is to determine the z_i such that the difference between the value of (6) and q is minimized. We can formulate this as the following optimization problem:

$$\text{Find } z_i \text{ that minimizes } \left|\sum_{i=0}^{2^n-1} z_i r_i - q\right| \tag{7}$$

$$\text{such that } z_i \in \{0, 1\} \text{ for } i = 0, 1, \dots, 2^n - 1. \tag{8}$$

This optimization problem can be converted into a linear 0-1 programming problem that can be solved using standard techniques.

If the solution to the above optimization problem has $z_i = 1$, then the Boolean function should contain the minterm m_i; otherwise, it should not. A circuit implementing the solution can be readily synthesized.[6]

B. A Suboptimal Solution

The above solution is simple and optimal; it works well when n is small. However, when n is large, there are two difficulties with the implementation that might make it impractical. First, the solution is based on linear 0-1 programming, which is NP-hard. Therefore, the computational complexity will become significant. Second, if an application-specific integrated circuit is designed to implement the solution of the optimization problem, the circuit may need as many as $O(2^n)$ gates in the worst case. This may be too costly for large n.

In this section, we provide a greedy algorithm that yields suboptimal results. However, the difference between the output probability of the circuit that it synthesizes and the target probability q is bounded. The algorithm has good performance both in terms of its run-time and the size of the resulting circuit.

The idea of the greedy algorithm is that we construct a group of $n+2$ circuits $C_0, C_1, \dots, C_{n+1}$ such that the circuit C_k ($0 \le k \le n$) has k probabilistic inputs and one deterministic input

[6]In particular, a field-programmable gate array (FPGA) can be configured for the task. For an FPGA with n-input lookup tables, the ith configuration bit of the table would be set to z_i, for $i = 0, 1, \dots, 2^n - 1$.

of either zero or one and the circuit C_{n+1} has n probabilistic inputs and two deterministic inputs of either zero or one. For all $0 \le k \le n$, the circuit C_{k+1} is constructed from C_k by replacing one input of C_k with a two-input gate.

The circuit C_0 is constructed by connecting a single input x_0 directly to the output. The input x_0 is either a deterministic value of zero or one. Thus, the probability p_{i_0} of the input x_0 being a one is either 0 or 1. The choice of setting p_{i_0} to 0 or 1 depends on which one is closer to the value q. If $q < 1-q$, we set p_{i_0} to 0; otherwise, we set it to 1. In order for the circuit C_0 to realize the exact probability q, there is an ideal value $p^*_{i_0}$ that should replace the value p_{i_0}. It is not hard to see that $p^*_{i_0} = q$.

Now we assume that the Boolean function of the circuit C_k $(0 \le k \le n-1)$ is $f_k(x_0, x_1, \ldots, x_k)$ and the input probabilities are $P(x_0 = 1) = p_{i_0}, P(x_1 = 1) = p_{i_1}, \ldots, P(x_k = 1) = p_{i_k}$, where $p_{i_0} \in \{0, 1\}$ and $p_{i_1}, \ldots, p_{i_k} \in S$. Let $p^*_{i_k}$ be an ideal value such that if we replace p_{i_k} by $p^*_{i_k}$ and keep the remaining input probabilities unchanged then the output probability of C_k is exactly equal to q.

Our idea for constructing the circuit C_{k+1} is to replace the input x_k of the circuit C_k with a single gate with inputs x_k and x_{k+1}. Thus, the Boolean function of the circuit C_{k+1} is

$$f_{k+1}(x_0, \ldots, x_{k+1}) = f_k(x_0, \ldots, x_{k-1}, g_{k+1}(x_k, x_{k+1}))$$

where $g_{k+1}(x_k, x_{k+1})$ is a Boolean function on two variables. We keep the probabilities of the inputs $x_0, x_1, \ldots, x_k$ the same as those of the circuit C_k. We choose the probability of the input x_{k+1} from the remaining choices of the set S such that the output probability of the newly added single gate is the closest to $p^*_{i_k}$. Assume that the probability of the input x_{k+1} is $p_{i_{k+1}}$. In order to construct the circuit C_{k+2} in the same way, we also calculate an ideal probability $p^*_{i_{k+1}}$ such that if we replace $p_{i_{k+1}}$ by $p^*_{i_{k+1}}$ and keep the remaining input probabilities unchanged then the output probability of the circuit C_{k+1} is exactly equal to q.

To make things easy, we only consider AND gates and OR gates as the choices for the newly added gate. The choice depends on whether $p^*_{i_k} > p_{i_k}$. When $p^*_{i_k} > p_{i_k}$, we choose an OR gate to replace the input x_k of the circuit C_k. The first input of the OR gate connects to x_k and the second to x_{k+1} or to the negation of x_{k+1}. The probability of the input x_k is kept as p_{i_k}. The probability of the input x_{k+1} is chosen from the set $S \backslash \{p_{i_1}, \ldots, p_{i_k}\}$. Thus, the first input probability of the OR gate is p_{i_k} and the second is chosen from the set

$$S_{k+1} = \{p | p = p_j \text{ or } 1 - p_j, p_j \in S \backslash \{p_{i_1}, \ldots, p_{i_k}\}\}.$$

For an OR gate with two input probabilities a and b, its output probability is

$$a + b - ab = a + (1-a)b.$$

The second input probability of the OR gate is chosen as p in the set S_{k+1} such that the output probability of the OR gate $p_{i_k} + (1 - p_{i_k})p$ is the closest to $p^*_{i_k}$. Equivalently, p is the value in the set S_{k+1} that is the closest to the value $\dfrac{p^*_{i_k} - p_{i_k}}{1 - p_{i_k}}$. We have two cases for p.

1) The case where $p = p_{i_{k+1}}$, for some $p_{i_{k+1}} \in S \backslash \{p_{i_1}, \ldots, p_{i_k}\}$. We set the second input of the OR gate to be x_{k+1} and set its probability as $P(x_{k+1} = 1) = p_{i_{k+1}}$. The ideal value $p^*_{i_{k+1}}$ should set the output probability of the OR gate to be $p^*_{i_k}$, so it satisfies that

$$p_{i_k} + (1 - p_{i_k})p^*_{i_{k+1}} = p^*_{i_k} \tag{9}$$

or

$$p^*_{i_{k+1}} = \frac{p^*_{i_k} - p_{i_k}}{1 - p_{i_k}}.$$

2) The case where $p = 1 - p_{i_{k+1}}$, for some $p_{i_{k+1}} \in S \backslash \{p_{i_1}, \ldots, p_{i_k}\}$. We set the second input of the OR gate to be $\neg x_{k+1}$ and set its probability as $P(x_{k+1} = 1) = p_{i_{k+1}}$. The ideal value $p^*_{i_{k+1}}$ should set the output probability of the OR gate to be $p^*_{i_k}$, so it satisfies that

$$p_{i_k} + (1 - p_{i_k})(1 - p^*_{i_{k+1}}) = p^*_{i_k} \tag{10}$$

or

$$p^*_{i_{k+1}} = \frac{1 - p^*_{i_k}}{1 - p_{i_k}}.$$

When $p^*_{i_k} \le p_{i_k}$, we choose an AND gate to replace the input x_k of the circuit C_k. The first input of the AND gate connects to x_k and the second connects to x_{k+1} or to the negation of x_{k+1}. The probability of the input x_k is kept as p_{i_k}. The probability of the input x_{k+1} is chosen from the set $S \backslash \{p_{i_1}, \ldots, p_{i_k}\}$. Similar to the case where $p^*_{i_k} > p_{i_k}$, the second input probability of the AND gate is chosen as a value p in the set S_{k+1} such that the value $p \cdot p_{i_k}$ is the closest to $p^*_{i_k}$. Equivalently, p is the value in the set S_{k+1} that is the closest to the value $\dfrac{p^*_{i_k}}{p_{i_k}}$. We have two cases for p.

1) The case where $p = p_{i_{k+1}}$, for some $p_{i_{k+1}} \in S \backslash \{p_{i_1}, \ldots, p_{i_k}\}$. We set the second input of the AND gate to be x_{k+1} and set its probability as $P(x_{k+1} = 1) = p_{i_{k+1}}$. The ideal value $p^*_{i_{k+1}}$ satisfies $p^*_{i_{k+1}} = \dfrac{p^*_{i_k}}{p_{i_k}}$.
2) The case where $p = 1 - p_{i_{k+1}}$, for some $p_{i_{k+1}} \in S \backslash \{p_{i_1}, \ldots, p_{i_k}\}$. We set the second input of the AND gate to be $\neg x_{k+1}$ and set its probability as $P(x_{k+1} = 1) = p_{i_{k+1}}$. The ideal value $p^*_{i_{k+1}}$ satisfies

$$p^*_{i_{k+1}} = 1 - \frac{p^*_{i_k}}{p_{i_k}}.$$

Iteratively, using the procedure above, we can construct circuits $C_1, C_2, \ldots, C_n$. Finally, we construct a circuit C_{n+1}, which is built from C_n by replacing its input x_n with an OR gate or an AND gate with two inputs x_n and x_{n+1}. We keep the probabilities of the inputs $x_0, \ldots, x_n$ the same as those of the circuit C_n. The input x_{n+1} is set to a deterministic value of zero or one. Thus, the probability of the input x_{n+1} is either zero or one. The choice of either an OR gate or an AND gate depends on whether $p^*_{i_n} > p_{i_n}$. When $p^*_{i_n} > p_{i_n}$, we choose an OR gate. The ideal probability value for the input x_{n+1} is

$$p^*_{i_{n+1}} = \frac{p^*_{i_n} - p_{i_n}}{1 - p_{i_n}}. \tag{11}$$

When $p^*_{i_n} \leq p_{i_n}$, we choose an AND gate. The ideal probability value for the input x_{n+1} is

$$p^*_{i_{n+1}} = \frac{p^*_{i_n}}{p_{i_n}}. \tag{12}$$

The choice of setting the input x_{n+1} to a deterministic value of zero or one depends on which one is closer to the value $p^*_{i_{n+1}}$. If $|p^*_{i_{n+1}}| < |1 - p^*_{i_{n+1}}|$, then we set the input x_{n+1} to zero; otherwise, we set it to one.

There is no evidence to show that the difference between the output probability of the circuit and q decreases as the number of inputs increases. Thus, we choose the one with the smallest difference among the circuits $C_0, \ldots, C_{n+1}$ as the final construction. It is easy to see that this algorithm completes in $O(n^2)$ time. For all $1 \leq k \leq n+1$, the circuit C_k has k fanin-two gates. Thus, the final solution contains at most $n+1$ fanin-two logic gates.

The following theorem shows that the difference between the target probability q and the output probability of the circuit synthesized by the greedy algorithm is bounded.

Theorem 4: In *scenario two*, given a set $S = \{p_1, p_2, \ldots, p_n\}$ and a target probability q, let p be the output probability of the circuit constructed by the greedy algorithm. We have

$$|p - q| \leq \frac{1}{2}\prod_{k=1}^{n} \max\{p_k, 1 - p_k\}. \qquad \square$$

Due to space constraints, we omit the proof here.

Example 4: Given a set of input probabilities $S = \{0.4, 0.7, 0.8\}$ and a target probability $q = 0.63$, we show how to synthesize a circuit to generate the target probability based on the greedy algorithm.

For the circuit C_0, since $1-q < q$, we set its input x_0 to be a deterministic value of one, or, equivalently, $p_{i_0} = 1$. The circuit C_0 is shown in Fig. 11(a). The ideal value $p^*_{i_0} = q = 0.63$.

Since $p^*_{i_0} < p_{i_0}$, to construct the circuit C_1, we replace the input x_0 of the circuit C_0 by an AND gate. The probability of the first input of the AND gate is 1. The probability p of the second input of the AND gate is chosen from the set

$$S_1 = \{0.2, 0.3, 0.4, 0.6, 0.7, 0.8\}$$

so that it is the closest to the value $p^*_{i_0}/p_{i_0} = 0.63$. Thus, we choose $p = 0.6$. Notice that $p = 1 - 0.4$. We set the second input of the AND gate to be $\neg x_1$ and set its probability as $P(x_1 = 1) = p_{i_1} = 0.4$. The ideal value $p^*_{i_1} = 1 - p^*_{i_0}/p_{i_0} = 0.37$. The circuit C_1 is shown in Fig. 11(b). (Again, we use a black dot to represent an inverter.)

Iteratively, we can get circuit C_2, C_3, and C_4 as those shown in Fig. 11(c), (d), and (e), respectively. The ideal values $p^*_{i_2} = 0.925$, $p^*_{i_3} = 0.625$, and $p^*_{i_4} = 0.893$. The circuit whose output probability is the closest to the target probability 0.63 is the circuit C_3. Thus, we choose C_3 as the final construction. Since the input x_0 of C_3 is a deterministic value of one, we can further optimize C_3. The final result is shown in Fig. 11(f).

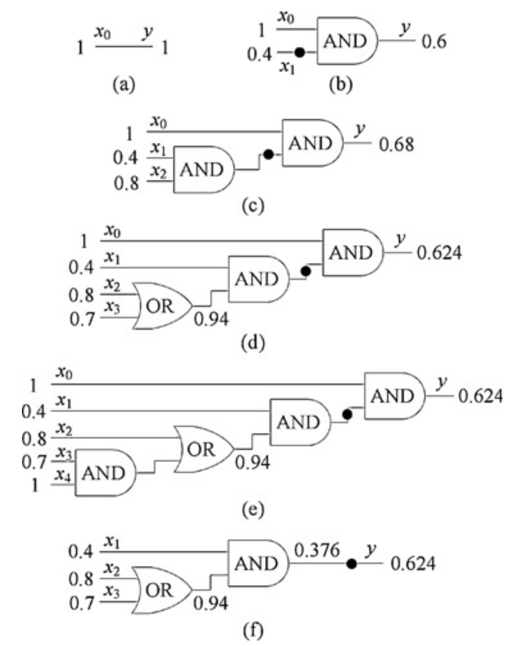


Fig. 11. Group of circuits synthesized by the greedy algorithm to generate the target probability $q = 0.63$ from the set of input probability $S = \{0.4, 0.7, 0.8\}$. The black dots in the figure represent inverters. (a) Circuit C_0. (b) Circuit C_1. (c) Circuit C_2. (d) Circuit C_3. (e) Circuit C_4. (f) Final construction.

V. Scenario Three: Set S Is Not Specified and the Elements Cannot Be Duplicated

In *scenario two*, when solving the optimization problem, the minimal difference $\left|\sum_{i=0}^{2^n-1} z_i r_i - q\right|$ is actually a function of q, which we denote as $h(q)$. That is

$$h(q) = \min_{\forall i, z_i \in \{0,1\}} \left|\sum_{i=0}^{2^n-1} z_i r_i - q\right|. \tag{13}$$

Assume that q is uniformly distributed on the unit interval. The mean of $h(q)$ for $q \in [0, 1]$ is solely determined by the set S. We can see that the smaller the mean is, the better the set S is for generating arbitrary probabilities. Thus, the mean of $h(q)$ is a good measure for the *quality* of S. We will denote it as $H(S)$. That is

$$H(S) = \int_0^1 h(q)\,\mathrm{d}q. \tag{14}$$

The problem considered in this scenario is: given an integer n, choose the n elements of the set S so that they produce a minimal $H(S)$.

Note that the only difference between *scenario two* and *scenario three* is that in *scenario three*, we are able to choose the elements of S. When constructing circuits, each element of S is still constrained to be used no more than once. As in *scenario two*, we are constructing a circuit with n inputs to realize each target probability. A circuit with n inputs has a truth table of 2^n rows. There are a total of 2^{2^n} different truth tables for n inputs. For a given assignment of input probabilities, we can get 2^{2^n} output probabilities.

Example 5: Consider the truth table shown in Table III. Here, we assume that $P(x = 1) = 4/5$ and $P(y = 1) = 2/3$. The

TABLE III
TRUTH TABLE FOR TWO VARIABLES

x	y	z	Probability
0	0	z_0	1/15
0	1	z_1	2/15
1	0	z_2	4/15
1	1	z_3	8/15

The output column $(z_0z_1z_2z_3)$ has a total of 16 different assignments.

corresponding probability of each input combination is given in the fourth column. For different assignments $(z_0z_1z_2z_3)$ of the output column, we obtain different output probabilities. For example, if $(z_0z_1z_2z_3) = (1010)$, then the output probability is $5/15$; if $(z_0z_1z_2z_3) = (1011)$, then the output probability is $13/15$. There are 16 different assignments for $(z_0z_1z_2z_3)$, so we can get 16 output probabilities. In this example, they are $0, 1/15, \dots, 14/15$ and 1. □

Let $N = 2^{2^n}$. For a set S with n elements, call the N possible probability values $b_1, b_2, \dots, b_N$ and assume that they are arranged in increasing order. That is $b_1 \le b_2 \le \cdots \le b_N$. Note that if the output column of the truth table consists of all zeros, the output probability is 0. If it consists of all ones, the output probability is 1. Thus, we have $b_1 = 0$ and $b_N = 1$.

The first question is: what is a lower bound for $H(S)$? We have the following theorem.

Theorem 5: A lower bound for $H(S)$ is $\dfrac{1}{4(N-1)}$. □

Proof: Note that for a q satisfying $b_i \le q \le \dfrac{b_i + b_{i+1}}{2}$, $h(q) = q - b_i$; for a q satisfying $\dfrac{b_i + b_{i+1}}{2} < q \le b_{i+1}$, $h(q) = b_{i+1} - q$. Thus

$$\begin{aligned} H(S) &= \int_0^1 h(q)\,\mathrm{d}q \\ &= \sum_{i=1}^{N-1}\left(\int_{b_i}^{\frac{b_i+b_{i+1}}{2}} (q - b_i)\,\mathrm{d}q + \int_{\frac{b_i+b_{i+1}}{2}}^{b_{i+1}} (b_{i+1} - q)\,\mathrm{d}q\right) \\ &= \frac{1}{4}\sum_{i=1}^{N-1}(b_{i+1} - b_i)^2. \end{aligned} \tag{15}$$

Let $c_i = b_{i+1} - b_i$, for $i = 1, \dots, N-1$. Since $\sum_{i=1}^{N-1} c_i = b_N - b_1 = 1$, by the Cauchy–Schwarz inequality, we have

$$H(S) = \frac{1}{4}\sum_{i=1}^{N-1} c_i^2 \ge \frac{1}{4(N-1)}\left(\sum_{i=1}^{N-1} c_i\right)^2 = \frac{1}{4(N-1)}.$$

□

The second question is: can this lower bound for $H(S)$ be achieved? We will show that the lower bound is achieved for the set

$$S = \{p \mid p = \frac{2^{2^k}}{2^{2^k}+1}, k = 0, 1, \dots, n-1\}. \tag{16}$$

In [16], we proved the following lemma.

Lemma 1: For a truth table on the inputs $x_1, \dots, x_n$ arranged in the order $x_n, \dots, x_1$, let

$$P(x_k = 1) = \frac{2^{2^{k-1}}}{2^{2^{k-1}}+1} \text{ for } k = 1, \dots, n.$$

The probability of the ith input combination $(0 \le i \le 2^n - 1)$ is $\dfrac{2^i}{2^{2^n}-1}$. □

Based on Lemma 1, we will show that the set S in (16) achieves the lower bound for $H(S)$.

Theorem 6: The set $S = \{p \mid p = \dfrac{2^{2^k}}{2^{2^k}+1}, k = 0, 1, \dots, n-1\}$ achieves the lower bound $\dfrac{1}{4(N-1)}$ for $H(S)$. □

Proof: By Lemma 1, for the given set S, the probability of the ith input combination $(0 \le i \le 2^n - 1)$ is $\dfrac{2^i}{2^{2^n}-1}$. Therefore, the set of $N = 2^{2^n}$ possible probabilities is

$$R = \{p \mid p = \sum_{i=0}^{2^n-1} z_i \frac{2^i}{2^{2^n}-1}, z_i \in \{0,1\}, \forall i = 0, \dots, 2^n - 1\}.$$

It is not hard to see that the N possible probabilities in increasing order are

$$b_0 = 0, b_1 = \frac{1}{N-1}, \dots, b_i = \frac{i}{N-1}, \dots, b_{N-1} = 1.$$

(Example 5 shows the situation for $n = 2$. We can see that with the set $S = \{2/3, 4/5\}$, we can get 16 possible probabilities: $0, 1/15, \dots, 14/15$ and 1.)

Thus, by (15), we have $H(S) = \dfrac{1}{4(N-1)}$. □

To summarize, if we have the freedom to choose n real numbers for the set S of source probabilities but each number can be used only once, the best choice is

$$S = \{p \mid p = \frac{2^{2^k}}{2^{2^k}+1}, k = 0, 1, \dots, n-1\}.$$

With the optimal set S, the truth table for a target probability q is easy to determine. First, round q to the closest fraction in the form of $\dfrac{i}{2^{2^n}-1}$. Suppose the closest fraction is $\dfrac{g(q)}{2^{2^n}-1}$. Then, the output of the ith row of the truth table is set as the ith least significant digit of the binary representation of $g(q)$. Again, a circuit implementing this solution can be readily synthesized.

VI. Conclusion

In this paper, we considered the problem of transforming a set of input probabilities into a target probability with combinational logic. The assumption that we make is that the input probabilities are exact and independent. For example, in synthesizing decimal output probabilities, we used multiple independent copies of the exact input probabilities 0.4 and 0.5. Of course, if we use physical sources to generate the input probabilities, there likely will be fluctuations. Also, the probabilistic inputs will likely be correlated. A future direction of research is how to design circuits that behave robustly in spite of these realities.

In addition to the three scenarios that we presented, there exists a fourth one that we have not considered: one in which the source probabilities are specified and can be duplicated. In this scenario, we would not expect to generate the target probability exactly. Thus, the problem is how to synthesize an area or delay optimal circuit whose output probability is a close approximation to the target value. We will address this problem in future work.

Acknowledgment

The authors would like to thank K. Bazargan and D. Lilja for their contributions. They were co-authors on a preliminary version of this paper [16].

References

[1] W. Qian and M. D. Riedel, "The synthesis of robust polynomial arithmetic with stochastic logic," in *Proc. Des. Autom. Conf.*, 2008, pp. 648–653.

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

[3] S. Cheemalavagu, P. Korkmaz, K. Palem, B. Akgul, and L. Chakrapani, "A probabilistic CMOS switch and its realization by exploiting noise," in *Proc. IFIP Int. Conf. VLSI*, 2005, pp. 535–541.

[4] K. P. Parker and E. J. McCluskey, "Probabilistic treatment of general combinational networks," *IEEE Trans. Comput.*, vol. 24, no. 6, pp. 668–670, Jun. 1975.

[5] J. Savir, G. Ditlow, and P. H. Bardell, "Random pattern testability," *IEEE Trans. Comput.*, vol. 33, no. 1, pp. 79–90, Jan. 1984.

[6] J.-J. Liou, K.-T. Cheng, S. Kundu, and A. Krstic, "Fast statistical timing analysis by probabilistic event propagation," in *Proc. Des. Autom. Conf.*, 2001, pp. 661–666.

[7] R. Marculescu, D. Marculescu, and M. Pedram, "Logic level power estimation considering spatiotemporal correlations," in *Proc. Int. Conf. Comput.-Aided Des.*, 1994, pp. 294–299.

[8] A. Gill, "Synthesis of probability transformers," *J. Franklin Instit.*, vol. 274, no. 1, pp. 1–19, 1962.

[9] A. Gill, "On a weight distribution problem, with application to the design of stochastic generators," *J. ACM*, vol. 10, no. 1, pp. 110–121, 1963.

[10] P. Jeavons, D. A. Cohen, and J. Shawe-Taylor, "Generating binary sequences for stochastic computing," *IEEE Trans. Inform. Theory*, vol. 40, no. 3, pp. 716–720, May 1994.

[11] L. Chakrapani, P. Korkmaz, B. Akgul, and K. Palem, "Probabilistic system-on-a-chip architecture," *ACM Trans. Design Autom. Electron. Syst.*, vol. 12, no. 3, pp. 1–28, 2007.

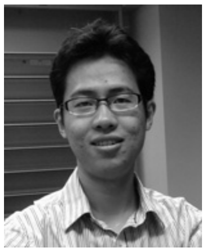
[12] D. Wilhelm and J. Bruck, "Stochastic switching circuit synthesis," in *Proc. Int. Symp. Inform. Theory*, 2008, pp. 1388–1392.

[13] C. E. Shannon, "The synthesis of two terminal switching circuits," *Bell Syst. Tech. J.*, vol. 28, no. 1, pp. 59–98, 1949.

[14] H. Zhou and J. Bruck, "On the expressibility of stochastic switching circuits," in *Proc. Int. Symp. Inform. Theory*, 2009, pp. 2061–2065.

[15] A. Mishchenko. (2007). *ABC: A System for Sequential Synthesis and Verification* [Online]. Available: http://www.eecs.berkeley.edu/alanmi/abc

[16] W. Qian, M. D. Riedel, K. Barzagan, and D. Lilja, "The synthesis of combinational logic to generate probabilities," in *Proc. Int. Conf. Comput.-Aided Des.*, 2009, pp. 367–374.



Weikang Qian received the B.Eng. degree in automation from Tsinghua University, Beijing, China, in 2006. He is currently pursuing the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis.

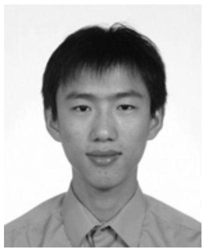
His current research interests include diverse fields such as computer-aided design of integrated circuits, circuit design for emerging technologies, and fault-tolerant computing.

Mr. Qian received the Doctoral Dissertation Fellowship at the University of Minnesota in recognition of his doctoral research.

Marc D. Riedel received the B.Eng. degree in electrical engineering with a minor in mathematics from McGill University, Montreal, QC, Canada, and the M.S. and Ph.D. degrees in electrical engineering from the California Institute of Technology (Caltech), Pasadena.

He has been an Assistant Professor of electrical and computer engineering with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, since 2006. He is a member of the Graduate Faculty in Biomedical Informatics and Computational Biology. From 2004 to 2005, he was a Lecturer in computation and neural systems with Caltech. He has held positions with Marconi Canada, Montreal, CAE Electronics, Montreal, Toshiba, Tokyo, Japan, and Fujitsu Research Laboratories, Singapore.

Dr. Riedel is a recipient of the NSF CAREER Award. His Ph.D. dissertation titled "Cyclic combinational circuits" received the Charles H. Wilts Prize for the Best Doctoral Research in Electrical Engineering from Caltech. His paper "The synthesis of cyclic combinational circuits" received the Best Paper Award at the Design Automation Conference.



Hongchao Zhou received the B.S. degree in mathematics and physics and the M.S. degree in control science and engineering from Tsinghua University, Beijing, China, in 2006 and 2008, respectively, and the M.S. degree in electrical engineering from the California Institute of Technology, Pasadena, in 2009. Currently, he is pursuing the Ph.D. degree in electrical engineering from the Department of Electrical Engineering, California Institute of Technology.

He was a Research Intern with the IBM China Research Laboratory, Beijing, from January 2006 to June 2008. His current research interests include information theory, stochastic systems, and networks.

Jehoshua Bruck (F'01) received the B.S. and M.S. degrees in electrical engineering from the Technion—Israel Institute of Technology, Haifa, Israel, in 1982 and 1985, respectively, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1989.

He is the Gordon and Betty Moore Professor of computation and neural systems and electrical engineering with the Department of Electrical Engineering, California Institute of Technology, Pasadena. His extensive industrial experience includes working with IBM Almaden Research Center, San Jose, CA, as well as cofounding and serving as the Chairman of Rainfinity, acquired by EMC, Hopkinton, MA, in 2005, and XtremIO, Cupertino, CA. His current research interests include information theory and systems and the theory computation in biological networks.

Dr. Bruck is a recipient of the Feynman Prize for Excellence in Teaching, the Sloan Research Fellowship, the National Science Foundation Young Investigator Award, the IBM Outstanding Innovation Award, and the IBM Outstanding Technical Achievement Award. His papers were recognized in journals and conferences, including winning the 2009 IEEE Communications Society Best Paper Award in Signal Processing and Coding for Data Storage for his paper on rank modulation for flash memories, the 2005 A. Schelkunoff Transactions Prize Paper Award from the IEEE Antennas and Propagation Society for his paper on signal propagation in wireless networks, and the 2003 Best Paper Award in the 2003 Design Automation Conference for his paper on cyclic combinational circuits.