

Two-Level Logic Synthesis for Probabilistic Computation*

Weikang Qian and Marc D. Riedel
 Department of Electrical and Computer Engineering,
 University of Minnesota, Twin Cities
 {qianx030, mriedel}@umn.edu

ABSTRACT

A classical problem in logic synthesis is to find a sum-of-product (SOP) Boolean expression with the minimum number of product terms that implements a Boolean function. In this paper, we focus on a related yet different synthesis problem targeted toward probabilistic computation. In our scenario, we want to synthesize a Boolean function that covers an exact number of minterms, say m minterms. This is the only requirement; which m minterms are covered does not matter. The objective is to find a SOP Boolean expression with the minimum number of product terms with this property. Solving this problem provides a solution to the problem of generating arbitrary probability values from unbiased input probabilities values of 0.5 with combinational logic.

We first show a method for constructing a set of product terms to cover exactly m minterms; this gives an upper bound. Then, as our major contribution, we propose a method for deriving a lower bound. We show that the problem of finding such a lower bound can be converted into an optimization problem which we call the *optimal subtraction problem*. We solve it with dynamic programming. Experiments on benchmarks show that for some numbers m , the lower bound meets the upper bound, indicating that the solution corresponding to the upper bound is optimal. For some other numbers m , the gap between the lower bound and the upper bound is small, indicating that the solution corresponding to the upper bound is nearly optimal.

1. INTRODUCTION

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 digital circuits that operate on *random* Boolean bit streams. Such circuits transform input probabilities, encoded by random bit streams, into output probabilities, also encoded by random bit streams. For example, consider the AND gate shown in Figure 1. It has two inputs that are one with independent probability 0.5. It has an output that is one with probability 0.25. Thus, we can say that the gate *generates* the probability value 0.25 from two copies of the probability value 0.5.

We call the paradigm of digital gates operating on random bit streams *probabilistic computation*. It has numerous application areas. For example, in built-in self-test (BIST), probabilistic computation is used to generate weighted random testing patterns [1]. In hardware implementations of probabilistic algorithms, it is used to generate probabilistic seed values from random sources [2].

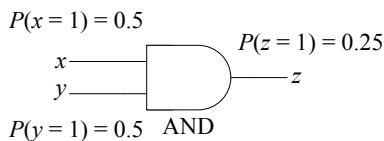


Figure 1: An AND gate with two inputs that are one with independent probability 0.5. Its output is one with probability 0.25.

In prior work, we described a method for synthesizing combinational logic to generate arbitrary decimal probabilities from a pair of probabilities, 0.4 and 0.5 [3].

In this work, we consider the same problem: synthesizing combinational logic to generate arbitrary probability values. However, we assume that we are only given access to *unbiased* input probabilities from an external source: the source provides *independent* copies of the probability value 0.5.

Given combinational logic with a single output and n inputs, if all inputs independently have probability 0.5 of being one, then each input combination has probability $\frac{1}{2^n}$ of occurring. If the combinational circuit contains exactly m minterms, then the probability that the output is one is $\frac{m}{2^n}$. Thus, the set of probabilities that can be realized by this model is $\{s | s = \frac{k}{2^n}, k = 0, 1, \dots, 2^n\}$.

Example 1

Table 1 shows a truth table for a Boolean function

$$y = x_0x_1 \vee x_2.^1$$

The last column shows the probability of each input combination occurring. If each input variable has probability of 0.5 being one, each input combination has probability of 1/8 of occurring. Since the Boolean function contains 5 minterms, the probability of y being one is 5/8. \square

Table 1: A truth table for the Boolean function $y = x_0x_1 \vee x_2$. The last column shows the probability of each input combination occurring, under the assumption that each input variable has probability 0.5 of being one.

x_0	x_1	x_2	y	Probability
0	0	0	0	1/8
0	0	1	1	1/8
0	1	0	0	1/8
0	1	1	1	1/8
1	0	0	0	1/8
1	0	1	1	1/8
1	1	0	1	1/8
1	1	1	1	1/8

We consider the synthesis problem of implementing a probability $\frac{m}{2^n}$, where $0 \leq m \leq 2^n$ is an arbitrary given integer. To implement the probability $\frac{m}{2^n}$, we can simply choose m input combinations and set their output values to be one. However, there are many ways to choose the m input combinations out of a total of 2^n input combinations; different choices may result in vastly different complexity of implementation. This motivates a new and interesting problem in logic synthesis:

Problem: what is the optimal way to synthesize logic that covers exactly m minterms if the choice which m minterms are covered does not matter?

The complexity of logic circuit depends on its implementation. In this work, we focus on two-level implementation of logic circuit [4]. Since two-level logic synthesis plays an important role

¹When discussing Boolean functions, we will use \vee to represent logical OR. We adopt this convention since later we will use $+$ to represent *arithmetic* addition.

*This work is supported by the grant No. 2003-NT-1107 from the SRC Focus Center Research Program on Functional Engineered Nano-Architectonics (FENA) and by the NSF CAREER award grant No. 0845650.

in multilevel logic synthesis [5], we believe that first understanding the two-level version of the synthesis problem for probabilistic computation will facilitate future research in attacking the multi-level version.

Minimizing the area of the two-level implementation is equivalent to minimizing the number of product terms of the sum-of-product (SOP) representation of a Boolean function [6]. Thus, the problem can be formulated as:

Given the number of variables n for a Boolean function and an integer $0 \leq m \leq 2^n$, find a SOP Boolean expression with the minimum number of product terms that contains exactly m minterms.

Example 2

Suppose that we want to synthesize a 4-variable Boolean function that contains 7 minterms. This is equivalent to filling in the Karnaugh map of 4 variables with exactly 7 ones. Figure 2 shows two different ways to fill one in. The optimal SOP Boolean expression for the function shown in Figure 2(a) is $\bar{x}_0\bar{x}_2 \vee \bar{x}_0x_3 \vee x_1x_2x_3$. It contains three product terms, whereas the optimal SOP Boolean expression for the function shown in Figure 2(b) is $\bar{x}_0\bar{x}_2 \vee x_1x_3$. It contains two product terms.

We can see that different choices for filling in 7 ones in the Karnaugh map can lead to optimal SOP Boolean expressions with different numbers of product terms. \square

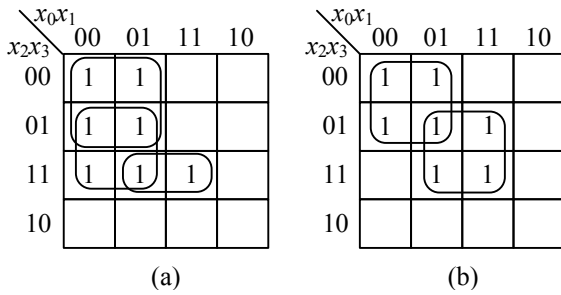


Figure 2: The Karnaugh maps of two different Boolean functions both containing 7 minterms: (a) The optimal SOP expression is $\bar{x}_0\bar{x}_2 \vee \bar{x}_0x_3 \vee x_1x_2x_3$, which contains three product terms. (b) The optimal SOP expression is $\bar{x}_0\bar{x}_2 \vee x_1x_3$, which contains two product terms.

The problem of finding the minimum number of product terms to cover exactly m minterms is a very hard combinatorial optimization problem. As the first step towards solving it, we formulate upper and lower bounds. We first show a method to construct a set of product terms to cover a given number of minterms. It gives an upper bound. Then, as the major contribution, we propose a lower bound on the minimum number of product terms to cover exactly m minterms. Such a lower bound is useful because:

1. A lower bound is a good place to start the search for an exact optimal solution to such a hard combinatorial optimization problem. (Consider NOVA [7], a procedure for finding optimal encodings for finite state machines: it starts the search for the optimal encoding from a sophisticated lower bound on the length of the encoding.)
2. For a hard combinatorial optimization problem, if an exact optimal solution cannot be obtained, then we hope for a good suboptimal solution. A lower bound lets us measure how good a suboptimal solution is.

Indeed, as revealed by the experimental results on benchmarks, for some numbers m , our proposed lower bound equal the upper bound, indicating that the solution corresponding to the upper bound is optimal. For some other numbers m , the upper bound is slightly larger than the lower bound, indicating that the solutions are suboptimal, but still quite good.

The paper is organized as follows. In Section 2, we introduce some preliminaries and show the upper bound. In Section 3, we derive a strong lower bound. In fact, we show that the lower bound problem can be reduced to what we call the *optimal subtraction problem*. We provide a dynamic-programming-based algorithm to solve this problem. The proposed algorithm has time complexity $O(d^2)$, where d is the number of bits in the binary representation of m . Experimental results on benchmarks are given in Section 4.

2. PRELIMINARIES

We focus on sum-of-product expression of Boolean functions and we adopt the same definition as given in [8]. The set of n variables of a Boolean function is denoted as x_0, \dots, x_{n-1} . For a variable x , x and \bar{x} are referred to as *literals*. A *Boolean product*, also known as *cube*, denoted by c , is a conjunction of literals such that x and \bar{x} do not appear simultaneously. A *minterm* is a cube in which each of the n variables appear once, in either its complemented or uncomplemented form. A sum-of-product Boolean expression is a disjunction of a set of cubes. If two cubes satisfy that $c_1c_2 = 0$, we say that the cube c_1 and c_2 are *disjoint*.

Definition 1

Define $V(f)$ to be the number of minterms contained in a Boolean function f . \square

If a cube c contains k literals ($0 \leq k \leq n$), then the number of minterms contained in the cube is $V(c) = 2^{n-k}$. Note that when a cube contains 0 literals, it is a special cube $c = 1$, which contains all minterms in the entire Boolean space. There is another special cube called *empty cube*, which is $c = 0$. The number of minterms contained in an empty cube is $V(c) = 0$. Thus, the number of minterms contained in a cube is in the set $S = \{s | s = 0 \text{ or } s = 2^k, k = 0, 1, \dots, n\}$.

We use η to denote the minimum number of cubes to cover exactly m minterms. The following theorem gives a method to construct a set of cubes to cover a given number of minterms, which also indicates an upper bound on η .

Theorem 1

If the binary representation of $0 \leq m \leq 2^n$ contains λ ones, i.e., if $m = \sum_{i=0}^n m_i 2^i$ such that $\sum_{i=0}^n m_i = \lambda$, then we can cover exactly m minterms with λ cubes. \square

PROOF. In the case that $m = 0$ or $m = 2^n$, the theorem is obviously true.

Now we consider $0 < m < 2^n$. Suppose that the λ ones in the binary representation of m are $m_{i_0} = m_{i_1} = \dots = m_{i_{\lambda-1}} = 1$, where $0 \leq i_0 < i_1 < \dots < i_{\lambda-1} \leq n-1$. Define $i_\lambda = n$. Then, we can construct λ cubes $c_0, \dots, c_{\lambda-1}$ such that c_j consists of literals $x_{i_j}, \dots, x_{i_{(j+1)}-1}$ and literals $\bar{x}_{i_{(j+1)}}, \dots, \bar{x}_{n-1}$. It is not hard to see that $V(c_j) = 2^{i_j}$. Further, for any i and j such that $0 \leq i < j \leq \lambda-1$, $c_i c_j = 0$, which means that $c_0, \dots, c_{\lambda-1}$ are pairwise disjoint. Thus the number of minterms covered by the λ cubes is $\sum_{j=0}^{\lambda-1} V(c_j) = \sum_{j=0}^{\lambda-1} 2^{i_j} = m$. Therefore, we can cover exactly m minterms with λ cubes. \square

Example 3

Suppose that we want to synthesize a set of cubes on 4 variables to cover 11 minterms. Since the binary representation of 11 is $(1011)_2$, thus, based on the proof of Theorem 1, we can derive a set of 3 cubes to cover 11 minterms, which are $c_0 = x_0\bar{x}_1\bar{x}_2\bar{x}_3$, $c_1 = x_1x_2\bar{x}_3$ and $c_2 = x_3$. \square

Theorem 1 gives an upper bound on the minimum number of cubes covering exactly m minterms, i.e., $\eta \leq \lambda$.

In Example 2, we want to find a set of cubes to cover 7 minterms. Since the binary representation of 7 has 3 ones, we are able to cover 7 minterms with 3 cubes. However, as shown in Figure 2(b), we can do better by using only 2 cubes to cover 7 minterms. Exploring the

reason behind this reveals the basic idea to optimize the number of cubes to cover a given number of minterms: we can potentially use a set of non-disjoint cubes to reduce the number of cubes. The basic principle we use is the inclusion-exclusion principle:

Lemma 1

Given λ cubes $c_0, \dots, c_{\lambda-1}$, the number of minterms cover by the λ cubes is

$$V\left(\bigvee_{i=0}^{\lambda-1} c_i\right) = \sum_{i=0}^{\lambda-1} V(c_i) - \sum_{\substack{i,j: \\ 0 \leq i < j \leq \lambda-1}} V(c_i c_j) + \sum_{\substack{i,j,k: \\ 0 \leq i < j < k \leq \lambda-1}} V(c_i c_j c_k) - \dots + (-1)^{\lambda-1} V\left(\prod_{i=0}^{\lambda-1} c_i\right). \quad (1)$$

□

To make the representation compact in the following sections, we make the following definitions.

Definition 2

An integer $\Gamma \geq 0$ is represented as a binary number, or a binary sequence, $(\gamma_{\lambda-1} \dots \gamma_0)_2$, if $\Gamma = \sum_{i=0}^{\lambda-1} \gamma_i 2^i$, where $\gamma_i \in \{0, 1\}$. Given two integers i and j such that $0 \leq j \leq i \leq \lambda - 1$, we use $\Gamma[i, j]$ to denote a subsequence $(\gamma_i \dots \gamma_j)_2$ of the binary number of Γ . For two binary sequences a and b , we use $[a : b]$ to denote the binary sequence of a followed by b . □

For example, given $a = (1001)_2$ and $b = (1010)_2$, $[a : b] = (10011010)_2$.

Definition 3

Given a cube c and $\gamma \in \{0, 1\}$, define

$$c^\gamma = \begin{cases} 1, & \text{if } \gamma = 0 \\ c, & \text{if } \gamma = 1 \end{cases}$$

Given λ cubes $c_0, \dots, c_{\lambda-1}$ and an integer $\Gamma = (\gamma_{\lambda-1} \dots \gamma_0)_2$, define C^Γ to be the intersection of a subset of cubes c_i with $\gamma_i = 1$, i.e., $C^\Gamma = \prod_{i=0}^{\lambda-1} c_i^{\gamma_i}$. □

Definition 4

Given an integer $\Gamma = (\gamma_{\lambda-1} \dots \gamma_0)_2$, define $B(\Gamma)$ to be the number of ones in the binary representation of Γ , i.e., $B(\Gamma) = \sum_{i=0}^{\lambda-1} \gamma_i$. □

With the above definitions, Equation (1) can be rewritten as

$$V\left(\bigvee_{i=0}^{\lambda-1} c_i\right) = \sum_{\Gamma=1}^{2^\lambda-1} (-1)^{B(\Gamma)-1} V\left(C^\Gamma\right). \quad (2)$$

3. A LOWER BOUND ON THE MINIMUM NUMBER OF CUBES

In previous section, we showed an upper bound on the minimum number of cubes η to cover a given number of minterms. In this section, we show a *lower* bound on the minimum number of cubes. It is related to the following problem, which we call the *optimal subtraction problem*:

Given two integers $m \geq 0$ and $t \geq 0$, find two integers $a \geq 0$ and $b \geq 0$ to minimize the number of ones in the binary representation of a , $B(a)$, subject to the constraints that $m = a - b$ and that the number of ones in the binary representation of b is no larger than t , i.e., $B(b) \leq t$.

In what follows, we will first show how a necessary condition on the minimum number of cubes to cover m minterms leads to the

optimal subtraction problem. Then, we will show some mathematical properties of the optimal subtraction problem and demonstrate a dynamic-programming-based solution to that problem. Finally, we will show how to achieve a lower bound on the minimum number of cubes to cover m minterms by solving the optimal subtraction problem.

3.1 The Origin of Optimal Subtraction Problem

Suppose that we can find a minimum of η cubes $c_0, \dots, c_{\eta-1}$ to cover exactly m minterms. Then, based on Equation (2), we have

$$m = \sum_{\Gamma=1}^{2^\eta-1} (-1)^{B(\Gamma)-1} V\left(C^\Gamma\right)$$

We can split the sum expression in the above equation into two sum expressions a and b such that a contains all terms with a plus sign and b contains all terms with a minus sign, i.e.,

$$a = \sum_{\substack{1 \leq \Gamma \leq 2^\eta-1: \\ B(\Gamma) \text{ is odd}}} V\left(C^\Gamma\right), \quad b = \sum_{\substack{1 \leq \Gamma \leq 2^\eta-1: \\ B(\Gamma) \text{ is even}}} V\left(C^\Gamma\right). \quad (3)$$

Then, $a, b \geq 0$ and $m = a - b$.

It is not hard to see that a contains $2^{\eta-1}$ terms and b contains $2^{\eta-1} - 1$ terms, where each term $V\left(C^\Gamma\right)$ is from the set $S = \{s | s = 0 \text{ or } s = 2^k, k = 0, 1, \dots, \eta\}$. Let $N_a = 2^{\eta-1}$ and $N_b = 2^{\eta-1} - 1$. Next we will show $N_a \geq B(a)$ and $N_b \geq B(b)$. First, we give a general theorem:

Theorem 2

Given N integers $0 \leq j_0 \leq j_1 \leq \dots \leq j_{N-1}$, let $z = \sum_{k=0}^{N-1} 2^{j_k}$. Then, $B(z) \leq N$. □

PROOF (SKETCH). Suppose that $B(z) = M$. Since the binary representation of z contains M ones, z can be represented as $z = \sum_{k=0}^{M-1} 2^{i_k}$, where $0 \leq i_0 < i_1 < \dots < i_{M-1}$. We could show that there exist $M + 1$ number r_0, r_1, \dots, r_M , such that

$$-1 = r_0 < r_1 < \dots < r_{M-1} < r_M = N - 1 \quad (4)$$

and for $l = 0, \dots, M - 1$, $\sum_{k=r_l+1}^{r_{l+1}} 2^{j_k} = 2^{i_l}$.

Therefore, from Equation (4), we have

$$N = r_M - r_0 = \sum_{l=1}^M (r_l - r_{l-1}) \geq M, \text{ or } B(z) = M \leq N. \quad \square$$

Suppose that the N_a elements $V\left(C^{\Gamma^i}\right)$ of the sum expression (3) for a are arranged as

$$V\left(C^{\Gamma^1}\right) = \dots = V\left(C^{\Gamma^r}\right) = 0 < V\left(C^{\Gamma^{r+1}}\right) \leq \dots \leq V\left(C^{\Gamma^{N_a}}\right),$$

where $r \geq 0$ and for $r + 1 \leq i \leq N_a$,

$$V\left(C^{\Gamma^i}\right) \in S^* = \{s | s = 2^k, k = 0, 1, \dots, \eta\}.$$

Therefore, there exist $N_a - r$ integers $0 \leq j_0 \leq \dots \leq j_{N_a-r-1}$, such that $a = \sum_{k=0}^{N_a-r-1} 2^{j_k}$. Applying Theorem 2, we conclude that $B(a) \leq N_a - r \leq N_a$. Similarly, we have $B(b) \leq N_b$.

Therefore, we get a necessary condition on the minimum number of cubes to cover exactly m minterms.

Corollary 1

If the minimum number of cubes to cover exactly m minterms is η , then there exist two integers $a, b \geq 0$ such that $m = a - b$ and $B(a) \leq 2^{\eta-1}$, $B(b) \leq 2^{\eta-1} - 1$. □

The necessary condition can be verified by posing it as the optimal subtraction problem shown at the beginning of this section, with $t = 2^{\eta-1} - 1$. If the minimum value of $B(a)$ of the optimal subtraction problem is greater than $2^{\eta-1}$, then the minimum number of cubes to cover m minterms is greater than η .

3.2 Some Mathematical Properties of the Optimal Subtraction Problem

In this section, we show some properties of the optimal subtraction problem. We use these to derive a dynamic-programming-based solution to the optimal subtraction problem in Section 3.3.

We assume that $m = (m_d \dots m_0)_2$ with $m_d = 0$ and that the optimal solution is $a^* = (a_d^* \dots a_0^*)_2$ and $b^* = (b_d^* \dots b_0^*)_2$. Since a^* and b^* form a feasible solution to the optimal subtraction problem, we have $a^* = m + b^*$ and $B(b^*) = \sum_{i=0}^d b_i^* \leq t$.

Suppose that a subsequence $m[k+l-1, k]$ of the binary representation of m is a sequence of ones, i.e., $m[k+l-1, k] = (1 \dots 1)_2$. If the binary addition $m[k-1, 0] + b^*[k-1, 0]$ generates a carry 1, then $b^*[k+l-1, k]$ must be in the form of $(0 \dots 0)_2$. On the other hand, if the binary addition $m[k-1, 0] + b^*[k-1, 0]$ generates a carry 0, then $b^*[k+l-1, k]$ must be in either the form of $(0 \dots 00)_2$ or the form of $(0 \dots 01)_2$. Mathematically, it is stated by the following two lemmas.

Lemma 2

If for some $1 \leq l \leq d+1$ and $0 \leq k \leq d+1-l$, such that $m[k+l-1, k] = (1 \dots 1)_2$ and

$$m[k-1, 0] + b^*[k-1, 0] = 2^k + a^*[k-1, 0],$$

then $b^*[k+l-1, k] = (0 \dots 0)_2$. \square

PROOF (SKETCH). By contraposition, we assume that the optimal solution satisfies that $\sum_{i=k}^{k+l-1} b_i^* > 0$. Then, we have

$$a^* = 2^{k+l} a^*[d, k+l] + 2^k b^*[k+l-1, k] + a^*[k-1, 0]. \quad (5)$$

Now consider $b' = 2^{k+l} b^*[d, k+l] + b^*[k-1, 0]$. Then, we have

$$a' = m + b' = 2^{k+l} a^*[d, k+l] + a^*[k-1, 0]. \quad (6)$$

Since we assume that $\sum_{i=k}^{k+l-1} b_i^* > 0$, we have

$$B(b') = \sum_{i=0}^{k-1} b_i^* + \sum_{i=k+l}^d b_i^* < B(b^*) \leq t.$$

Further, by Equation (5) and (6),

$$B(a') = \sum_{i=0}^{k-1} a_i^* + \sum_{i=k+l}^d a_i^* < B(a^*) = \sum_{i=0}^{k-1} a_i^* + \sum_{i=k}^{k+l-1} b_i^* + \sum_{i=k+l}^d a_i^*.$$

Therefore, a^* and b^* are not the optimal solution, which contradicts the assumption. Therefore, the lemma holds. \square

Lemma 3

If for some $1 \leq l \leq d+1$ and $0 \leq k \leq d+1-l$, such that $m[k+l-1, k] = (1 \dots 1)_2$ and

$$m[k-1, 0] + b^*[k-1, 0] = a^*[k-1, 0],$$

then $b^*[k+l-1, k+1] = (0 \dots 0)_2$ and $b_k^* = 0$ or 1. \square

PROOF. The proof is similar to that of Lemma 2. \square

Suppose that a subsequence $m[k+l-1, k]$ of the binary representation of m is a sequence of zeros, i.e., $m[k+l-1, k] = (0 \dots 0)_2$. If the binary addition $m[k-1, 0] + b^*[k-1, 0]$ generates a carry 0, then $b^*[k+l-1, k]$ must be in the form of $(0 \dots 0)_2$. On the other hand, if the binary addition $m[k-1, 0] + b^*[k-1, 0]$ generates a carry 1, then $b^*[k+l-1, k]$ must be in one of the forms of $(00 \dots 000)_2$, $(00 \dots 001)_2$, $(00 \dots 011)_2$, \dots , $(01 \dots 111)_2$, $(11 \dots 111)_2$. Mathematically, this is stated by the following two lemmas. The proofs of these are similar to that of Lemma 2.

²We can choose a d large enough so that it guarantees that $m_d = 0$. Further, if $m_d = 0$, there exists an optimal solution of a^* and b^* such that their binary representations have $d+1$ bits.

Lemma 4

If for some $1 \leq l \leq d+1$ and $0 \leq k \leq d+1-l$, such that $m[k+l-1, k] = (0 \dots 0)_2$, and

$$m[k-1, 0] + b^*[k-1, 0] = a^*[k-1, 0],$$

then $b^*[k+l-1, k] = (0 \dots 0)_2$. \square

Lemma 5

If for some $1 \leq l \leq d+1$ and $0 \leq k \leq d+1-l$, such that $m[k+l-1, k] = (0 \dots 0)_2$ and

$$m[k-1, 0] + b^*[k-1, 0] = 2^k + a^*[k-1, 0],$$

then there exists a $0 \leq r \leq l$, such that $b^*[k+l-1, k+r] = (0 \dots 0)_2$ and $b^*[k+r-1, k] = (1 \dots 1)_2$. \square

If the conditions of Lemma 5 are satisfied and if the subsequence $b^*[k+l-1, k]$ of the optimal solution b^* is of the form $(0 \dots 01 \dots 1)_2$, then $b' = 2^{k+l} b^*[d, k+l] + b^*[k-1, 0]$ and $a' = m + b'$ also form an optimal solution. Mathematically, this is stated by the following lemma.

Lemma 6

For any $2 \leq l \leq d+1$ and $0 \leq k \leq d+1-l$, such that

$$m[k+l-1, k] = (0 \dots 0)_2 \quad (7)$$

and

$$m[k-1, 0] + b^*[k-1, 0] = 2^k + a^*[k-1, 0], \quad (8)$$

if the optimal solution of b^* satisfies that

$$b^*[k+l-1, k+r] = (0 \dots 0)_2, b^*[k+r-1, k] = (1 \dots 1)_2, \quad (9)$$

for some $1 \leq r \leq l-1$, then $b' = 2^{k+l} b^*[d, k+l] + b^*[k-1, 0]$ and $a' = m + b'$ also form an optimal solution. \square

PROOF (SKETCH). By Equation (7), (8), and (9), we have

$$a^* = 2^{k+l} a^*[d, k+l] + 2^{k+r} + a^*[k-1, 0]. \quad (10)$$

Now consider $b' = 2^{k+l} b^*[d, k+l] + b^*[k-1, 0]$. Then,

$$a' = 2^{k+l} a^*[d, k+l] + 2^k + a^*[k-1, 0]. \quad (11)$$

By Equation (9) and the fact that $r \geq 1$, we have

$$B(b') = \sum_{i=0}^{k-1} b_i^* + \sum_{i=k+l}^d b_i^* < B(b^*) = \sum_{i=0}^{k-1} b_i^* + r + \sum_{i=k+l}^d b_i^* \leq t.$$

Further, by Equation (10), and (11), we have

$$B(a') = \sum_{i=0}^{k-1} a_i^* + 1 + \sum_{i=k+l}^d a_i^* = B(a^*)$$

Since a^* and b^* form an optimal solution, a' and b' also form an optimal solution. \square

Note that the optimal solution b' in Lemma 6 satisfies that $b'[k+l-1, k] = (0 \dots 0)_2$. Based on Lemma 5 and 6, when a subsequence $m[k+l-1, k]$ of the binary representation of m is a sequence of zeros and the binary addition $m[k-1, 0] + b^*[k-1, 0]$ generates a carry 1, we can restrict $b^*[k+l-1, k]$ to be either in the form of $(0 \dots 0)_2$ or in the form of $(1 \dots 1)_2$. Together with Lemma 2, 3, and 4, we can summarize the choices of $b^*[k+l-1, k]$ for different combinations of $m[k+l-1, k] \in \{(1 \dots 1)_2, (0 \dots 0)_2\}$ and the carry of the binary sum $m[k-1, 0] + b^*[k-1, 0]$ in Table 2.

Table 2: The choices of $b^*[k+l-1, k]$ according to different combinations of $m[k+l-1, k] \in \{(1 \dots 1)_2, (0 \dots 0)_2\}$ and the carry of the binary sum $m[k-1, 0] + b^*[k-1, 0]$.

		$m[k+l-1, k]$	
		$(1 \dots 11)_2$	$(0 \dots 00)_2$
$m[k-1, 0] + b^*[k-1, 0]$ has carry	0	$(0 \dots 00)_2$ or $(0 \dots 01)_2$	$(0 \dots 00)_2$
	1	$(0 \dots 00)_2$	$(0 \dots 00)_2$ or $(1 \dots 11)_2$

3.3 A Dynamic-Programming-Based Solution to the Optimal Subtraction Problem

In this section, we show a solution to the optimal subtraction problem based on dynamic programming. The idea is that we can treat the binary representation of m as a concatenation of alternating subsequences of zeros and ones and construct an optimal solution based on the optimal choices shown in Table 2.

First, we define a more general minimization problem.

Definition 5

Given integers m , i , l and ξ , such that $m = (m_d \dots m_0)_2$ with $m_d = 0$, $0 \leq i \leq d$ and $\xi \in \{0, 1\}$, we define the problem $Q(i, l, \xi)$ to be the following minimization problem:

Minimize cost $B(a)$, subject to the constraints that $a, b \geq 0$ are integers whose binary representation have $d+1-i$ bits, $a = m[d, i] + b + \xi$, and $B(b) \leq l$.

Let the optimal solution of the problem $Q(i, l, \xi)$ be $a = a^*(i, l, \xi)$ and $b = b^*(i, l, \xi)$. Let $A(i, l, \xi)$ be the minimum cost. \square

Note that the binary representations of $a^*(i, l, \xi)$ and $b^*(i, l, \xi)$ have $d+1-i$ bits. By a similar notation as that in Definition 2, for two integers $0 \leq k \leq j \leq d-i$, we write $a^*(i, l, \xi)[j, k]$ to denote a subsequence from bit j to bit k of the binary sequence of $a^*(i, l, \xi)$. Since $a^*(i, l, \xi) = b^*(i, l, \xi) + m[d, i] + \xi$, we only need to find the optimal solution of $b^*(i, l, \xi)$.

Notice that when $l < 0$, there are no feasible solutions to the problem $Q(i, l, \xi)$. Thus, when $l < 0$, we define $A(i, l, \xi) = \infty$, $a^*(i, l, \xi) = b^*(i, l, \xi) = \phi$. In what follows, we consider the problem $Q(i, l, \xi)$ with $l \geq 0$.

By Definition 5, the optimal subtraction problem is the problem $Q(0, t, 0)$. The following four theorems show that the problem $Q(i, l, \xi)$ exhibits *optimal substructure* [9]: an optimal solution to the problem contains within it optimal solutions to subproblems.

Theorem 3

Suppose that for some $0 \leq i < j \leq d$, $m[j-1, i] = (0 \dots 0)_2$. Given $l \geq 0$, and $\xi = 0$, then $A(i, l, \xi) = A(j, l, 0)$ and the optimal solution $b^*(i, l, \xi) = [b^*(j, l, 0) : (0 \dots 0)_2]$, where the constant suffix sequences are of length $j-i$. \square

PROOF (SKETCH). Since $m[j-1, i] = (0 \dots 0)_2$ and the carry-in ξ to the addition $m[j-1, i] + b^*(i, l, \xi)[j-i-1, 0]$ is 0, this case corresponds to the conditions of Lemma 4. Therefore,

$$b^*(i, l, \xi)[j-i-1, 0] = (0 \dots 0)_2 \quad (12)$$

and hence,

$$B(b^*(i, l, \xi)) = B(b^*(i, l, \xi)[d-i, j-i]). \quad (13)$$

Since $a^*(i, l, \xi) = b^*(i, l, \xi) + m[d, i] + \xi$, then by Equation (12), we have

$$a^*(i, l, \xi)[j-i-1, 0] = (0 \dots 0)_2 \quad (14)$$

and

$$a^*(i, l, \xi)[d-i, j-i] = m[d, j] + b^*(i, l, \xi)[d-i, j-i]. \quad (15)$$

Therefore, due to Equation (14), we have

$$B(a^*(i, l, \xi)) = B(a^*(i, l, \xi)[d-i, j-i]). \quad (16)$$

Let $u = a^*(i, l, \xi)[d-i, j-i]$ and $v = b^*(i, l, \xi)[d-i, j-i]$. Then, based on Equation (15), we have $u = m[d, j] + v$ and based on Equation (13), we have $B(v) \leq l$. Therefore, u and v form a feasible solution to the problem $Q(j, l, 0)$. Further, based on Equation (16), we have $B(u) = B(a^*(i, l, \xi))$.

We could prove by contraposition that u and v form an optimal solution to the problem $Q(j, l, 0)$. Therefore, $A(j, l, 0) = B(u) = B(a^*(i, l, \xi)) = A(i, l, \xi)$ and

$$v = b^*(i, l, \xi)[d-i, j-i] = b^*(j, l, 0).$$

Combining the above equation with Equation (12), we prove the theorem. \square

Theorem 4

Suppose that for some $0 \leq i < j \leq d$, $m[j-1, i] = (0 \dots 0)_2$. Given, $l \geq 0$, and $\xi = 1$, then

$$A(i, l, \xi) = \min\{A(j, l, 0) + 1, A(j, l-j+i, 1)\}.$$

If $A(i, l, \xi) = A(j, l, 0) + 1$, then the optimal solution $b^*(i, l, \xi) = [b^*(j, l, 0) : (0 \dots 00)_2]$, where the constant suffix sequences are of length $j-i$.

If, on the other hand, $A(i, l, \xi) = A(j, l-j+i, 1)$, then the optimal solution $b^*(i, l, \xi) = [b^*(j, l-j+i, 1) : (1 \dots 11)_2]$, where the constant suffix sequences are of length $j-i$. \square

PROOF. The proof is similar to that of Theorem 3. The only difference is that, since this case corresponds to the conditions of Lemma 5, $b^*(i, l, \xi)[j-i-1, 0]$ can be either $(0 \dots 0)_2$ or $(1 \dots 1)_2$. We need to compare the minimum costs for the two cases of $b^*(i, l, \xi)[j-i-1, 0]$ and choose the smaller one. \square

Similarly, we can prove the following two theorems.

Theorem 5

Suppose that for some $0 \leq i < j \leq d$, $m[j-1, i] = (1 \dots 1)_2$. Given $l \geq 0$, and $\xi = 0$, then

$$A(i, l, \xi) = \min\{A(j, l, 0) + j-i, A(j, l-1, 1)\}.$$

If $A(i, l, \xi) = A(j, l, 0) + j-i$, then the optimal solution $b^*(i, l, \xi) = [b^*(j, l, 0) : (0 \dots 00)_2]$, where the constant suffix sequences are of length $j-i$.

If, on the other hand, $A(i, l, \xi) = A(j, l-1, 1)$, then the optimal solution $b^*(i, l, \xi) = [b^*(j, l-1, 1) : (0 \dots 01)_2]$, where the constant suffix sequences are of length $j-i$. \square

Theorem 6

Suppose that for some $0 \leq i < j \leq d$, $m[j-1, i] = (1 \dots 1)_2$. Given $l \geq 0$, and $\xi = 1$, then $A(i, l, \xi) = A(j, l, 1)$ and the optimal solution $b^*(i, l, \xi) = [b^*(j, l, 1) : (0 \dots 0)_2]$, where the constant suffix sequences are of length $j-i$. \square

We treat m as a concatenation of $p \geq 1$ alternating subsequences of zeros and ones prefixing with a subsequence of zeros (since $m_d = 0$), that is, there exist $p+1$ integers $0 = i_0 < i_1 < \dots < i_p = d+1$, such that

$$m[i_{p-k}-1, i_{p-k-1}] = \begin{cases} (0 \dots 0)_2, & \text{if } 0 \leq k \leq p-1 \text{ is even} \\ (1 \dots 1)_2, & \text{if } 0 \leq k \leq p-1 \text{ is odd} \end{cases} \quad (17)$$

If $p = 1$, then m is just a sequence of zeros and thus, $m = 0$. The solution to the optimal subtraction problem is $a^* = b^* = 0$.

Now we suppose that $p \geq 2$. We could assume that the last subsequence of m , $m[i_1-1, i_0]$, is a sequence of ones. Otherwise, the last subsequence $m[i_1-1, i_0]$ contains all zeros. Since the input carry is 0, this case corresponds to the situation of Theorem 3 with $i = 0$ and $j = i_1$. Therefore, to solve the problem $Q(0, t, 0)$, it is equivalent to solve the problem $Q(i_1, t, 0)$, which is the optimal subtraction problem on $m' = m[d, i_1]$, whose last subsequence is a sequence of ones.

Since the first subsequence is a sequence of zeros and the last subsequence is a sequence of ones, the total number of alternating subsequences p is even. Let $p = 2q$, where $q \geq 1$ is an integer. Then, for any $0 \leq r \leq q - 1$, $m[i_{2r+1} - 1, i_{2r}]$ is a sequence of ones and $m[i_{2r+2} - 1, i_{2r+1}]$ is a sequence of zeros. We have the following two theorems which give recursive relations on the optimal solutions of the problem $Q(i_{2r}, l, 0)$ and $Q(i_{2r+1}, l, 1)$, respectively.

Theorem 7

For any $0 \leq r \leq q - 2$ and $l \geq 0$, the optimal solution of the problem $Q(i_{2r}, l, 0)$ relates to the optimal solution of the problem $Q(i_{2r+1}, l - 1, 1)$ and the optimal solution of the problem $Q(i_{2r+2}, l, 0)$ in the following way:

$$A(i_{2r}, l, 0) = \min\{A(i_{2r+1}, l - 1, 1), A(i_{2r+2}, l, 0) + i_{2r+1} - i_{2r}\}.$$

If $A(i_{2r}, l, 0) = A(i_{2r+1}, l - 1, 1)$, then $b^*(i_{2r}, l, 0) = [b^*(i_{2r+1}, l - 1, 1) : (0 \dots 01)_2]$, where the constant suffix sequences are of length $i_{2r+1} - i_{2r}$.

If, on the other hand, $A(i_{2r}, l, 0) = A(i_{2r+2}, l, 0) + i_{2r+1} - i_{2r}$, then $b^*(i_{2r}, l, 0) = [b^*(i_{2r+2}, l, 0) : (0 \dots 00)_2]$, where the constant suffix sequence is of length $i_{2r+2} - i_{2r}$. \square

PROOF. Since $m[i_{2r+1} - 1, i_{2r}] = (1 \dots 1)_2$, by Theorem 5,

$$A(i_{2r}, l, 0) = \min\{A(i_{2r+1}, l - 1, 1), A(i_{2r+1}, l, 0) + i_{2r+1} - i_{2r}\}. \quad (18)$$

If $A(i_{2r}, l, 0) = A(i_{2r+1}, l - 1, 1)$, then

$$b^*(i_{2r}, l, 0) = [b^*(i_{2r+1}, l - 1, 1) : (0 \dots 01)_2],$$

where the constant suffix sequences are of length $i_{2r+1} - i_{2r}$.

If, on the other hand, $A(i_{2r}, l, 0) = A(i_{2r+2}, l, 0) + i_{2r+1} - i_{2r}$, then

$$b^*(i_{2r}, l, 0) = [b^*(i_{2r+2}, l, 0) : (0 \dots 00)_2], \quad (19)$$

where the constant suffix sequences are of length $i_{2r+2} - i_{2r}$.

Since $m[i_{2r+2} - 1, i_{2r+1}] = (0 \dots 0)_2$, by Theorem 3,

$$A(i_{2r+1}, l, 0) = A(i_{2r+2}, l, 0), \quad (20)$$

and

$$b^*(i_{2r+1}, l, 0) = [b^*(i_{2r+2}, l, 0) : (0 \dots 00)_2], \quad (21)$$

where the constant suffix sequences are of length $i_{2r+2} - i_{2r+1}$.

Therefore, by Equation (18) and (20), we have

$$A(i_{2r}, l, 0) = \min\{A(i_{2r+1}, l - 1, 1), A(i_{2r+2}, l, 0) + i_{2r+1} - i_{2r}\}.$$

If $A(i_{2r}, l, 0) = A(i_{2r+2}, l, 0) + i_{2r+1} - i_{2r}$, then based on Equation (19) and (21), we have

$$b^*(i_{2r}, l, 0) = [b^*(i_{2r+2}, l, 0) : (0 \dots 00)_2],$$

where the constant suffix sequence is of length $i_{2r+2} - i_{2r}$. \square

Theorem 8

For any $0 \leq r \leq q - 2$ and $l \geq 0$, the optimal solution of the problem $Q(i_{2r+1}, l, 1)$ relates to the optimal solution of the problem $Q(i_{2r+2}, l, 0)$ and the optimal solution of the problem $Q(i_{2r+3}, l - i_{2r+2} + i_{2r+1}, 1)$ in the following way:

$$A(i_{2r+1}, l, 1) = \min\{A(i_{2r+2}, l, 0) + 1, A(i_{2r+3}, l - i_{2r+2} + i_{2r+1}, 1)\}.$$

If $A(i_{2r+1}, l, 1) = A(i_{2r+2}, l, 0) + 1$, then

$$b^*(i_{2r+1}, l, 1) = [b^*(i_{2r+2}, l, 0) : (0 \dots 00)_2],$$

where the constant suffix sequences are of length $i_{2r+2} - i_{2r+1}$.

If, on the other hand,

$$A(i_{2r+1}, l, 1) = A(i_{2r+3}, l - i_{2r+2} + i_{2r+1}, 1),$$

then

$$b^*(i_{2r+1}, l, 1) = [b^*(i_{2r+3}, l - i_{2r+2} + i_{2r+1}, 1) : (0 \dots 01 \dots 1)_2],$$

where the constant suffix sequence has $i_{2r+3} - i_{2r+2}$ zeros and $i_{2r+2} - i_{2r+1}$ ones. \square

PROOF. The proof is similar to that of Theorem 7. \square

The problem $Q(i_{p-1}, l, 1)$ and $Q(i_{p-2}, l, 0)$ are the base cases of the recursion. Their solutions are straightforward and shown by the following lemma.

Lemma 7

For any $l \geq 0$, $A(i_{p-1}, l, 1) = 1$ and $b^*(i_{p-1}, l, 1) = 0$. For any $l \geq 1$, $A(i_{p-2}, l, 0) = 1$ and $b^*(i_{p-2}, l, 0) = 1$. For $l = 0$, $A(i_{p-2}, l, 0) = i_{p-1} - i_{p-2}$ and $b^*(i_{p-2}, l, 0) = 0$. \square

Let

$$\tau = 1 + \sum_{r=1}^{q-1} (i_{2r} - i_{2r-1}). \quad (22)$$

For the optimal subtraction problem $Q(0, t, 0)$ on $m > 0$, when $t \geq \tau$, it is not hard to show that the minimum cost is $A(0, t, 0) = 1$, as stated by the following theorem.

Theorem 9

Given $m > 0$, if $t \geq \tau$, then $A(0, t, 0) = 1$, and $a^* = 2^{i_{p-1}}$ and $b^* = 2^{i_{p-1}} - m$. \square

Based on Theorem 9, we only need to use the recursive relations shown in Theorem 7 and 8 to solve the optimal subtraction problem when $t < \tau$. The procedure to solve the optimal subtraction problem is shown in Algorithm 1.

Algorithm 1 OSP(m, t): the procedure to solve the optimal subtraction problem with $m \geq 0$ and $t \geq 0$.

```

1: {Given two integers  $m \geq 0$  and  $t \geq 0$ , return the minimum cost  $C$  and the optimal solution  $a^*$  and  $b^*$ .}
2: if  $m = 0$  then  $C \leftarrow 0$ ;  $a^* \leftarrow 0$ ;  $b^* \leftarrow 0$ ; return ( $C, a^*, b^*$ );
3: ( $m, ntz$ )  $\leftarrow$  RmvTailZero( $m$ ); {Remove trailing zeros and  $ntz$  stores the number of trailing zeros.}
4: ( $i_0, i_1, \dots, i_p$ )  $\leftarrow$  SubSeq( $m$ ); {Get the position indices of the alternating subsequences.}
5:  $q \leftarrow p/2$ ;  $\tau \leftarrow 1 + \sum_{r=1}^{q-1} (i_{2r} - i_{2r-1})$ ;
6: if  $t \geq \tau$  then
7:    $C \leftarrow 1$ ;  $a^* \leftarrow 2^{i_{p-1}}$ ;  $b^* \leftarrow a^* - m$ ;
8:    $a^* \leftarrow [a^* : \text{zero}(ntz)]$ ;  $b^* \leftarrow [b^* : \text{zero}(ntz)]$ ; {Append trailing zeros.}
9:   return ( $C, a^*, b^*$ );
10: for  $l \leftarrow 0$  to  $t$  do  $T(p - 1, l) \leftarrow 1$ ;
11:  $T(p - 2, 0) \leftarrow i_{p-1} - i_{p-2}$ ;
12: for  $l \leftarrow 1$  to  $t$  do  $T(p - 2, l) \leftarrow 1$ ;
13: for  $r \leftarrow q - 2$  to  $0$  do
14:   for  $l \leftarrow 0$  to  $t$  do
15:      $T(2r + 1, l) \leftarrow \min\{T(2r + 2, l) + 1, T(2r + 3, l - i_{2r+2} + i_{2r+1})\}$ ;
16:   for  $l \leftarrow 0$  to  $t$  do
17:      $T(2r, l) \leftarrow \min\{T(2r + 1, l - 1), T(2r + 2, l) + i_{2r+1} - i_{2r}\}$ ;
18:  $C \leftarrow T(0, t)$ ;
19:  $b^* \leftarrow \text{OptSln}(T, t, i_0, \dots, i_p)$ ; {Construct the optimal solution  $b^*$ .}
20:  $a^* \leftarrow m + b^*$ ;  $a^* \leftarrow [a^* : \text{zero}(ntz)]$ ;  $b^* \leftarrow [b^* : \text{zero}(ntz)]$ ;
21: return ( $C, a^*, b^*$ );

```

The procedure to solve the optimal subtraction problem begins by handling the trivial case $m = 0$ (Line 2). Then, it removes the trailing zeros of m by calling the function RmvTailZero(m), which returns the new m and the number of trailing zeros ntz (Line 3). In Line 4, it calls the function SubSeq(m) to get the position indices of the alternating subsequences of zeros and ones of m , according to Equation (17). In Lines 6–9, it deals with the case that $t \geq \tau$, in which the solution is given by Theorem 9. In Line 8, the function zero(n) generates a sequence of n zeros. The optimal solution for

the original m is formed by appending ntz zeros to the optimal solution for the new m .

If $t < \tau$, then a bottom-up dynamic-programming-based approach is used to solve the problem based on the recursive relations shown in Theorem 7 and 8. The procedure uses an auxiliary table T of p rows and $t + 1$ columns to store the optimal costs for sub-problems $Q(i_k, l, \xi)$. For any $0 \leq k \leq p - 1$ and $0 \leq l \leq t$, table entry $T(k, l)$ is defined as

$$T(k, l) = \begin{cases} A(i_k, l, 0), & \text{if } k \text{ is even} \\ A(i_k, l, 1), & \text{if } k \text{ is odd} \end{cases}$$

The procedure begins from the base cases in Lines 10–12. By Lemma 7, for any $0 \leq l \leq t$, $T(p - 1, l) = A(i_{p-1}, l, 1) = 1$, and

$$T(p - 2, l) = A(i_{p-2}, l, 0) = \begin{cases} i_{p-1} - i_{p-2}, & \text{if } l = 0 \\ 1, & \text{if } 1 \leq l \leq t \end{cases}$$

In Lines 13–17, the procedure computes $T(k, l)$ in decreasing order of k from $p - 3$ to 0. By Theorem 7 and 8, we have that when $0 \leq r \leq q - 2$,

$$\begin{aligned} T(2r + 1, l) &= A(i_{2r+1}, l, 1) \\ &= \min\{A(i_{2r+2}, l, 0) + 1, A(i_{2r+3}, l - i_{2r+2} + i_{2r+1}, 1)\} \\ &= \min\{T(2r + 2, l) + 1, T(2r + 3, l - i_{2r+2} + i_{2r+1})\}. \end{aligned} \quad (23)$$

and

$$\begin{aligned} T(2r, l) &= A(i_{2r}, l, 0) \\ &= \min\{A(i_{2r+1}, l - 1, 1), A(i_{2r+2}, l, 0) + i_{2r+1} - i_{2r}\} \\ &= \min\{T(2r + 1, l - 1), T(2r + 2, l) + i_{2r+1} - i_{2r}\}. \end{aligned} \quad (24)$$

After getting the table T , the optimal cost is $C = T(0, t)$ and the optimal solution b^* is constructed based on T by the procedure $\text{OptSln}(T, t, i_0, \dots, i_p)$, which is shown in Algorithm 2. The optimal solution a^* is $a^* = m + b^*$. The final optimal solution is formed by appending ntz zeros to a^* and b^* (Line 20).

Algorithm 2 $\text{OptSln}(T, t, i_0, \dots, i_p)$: the procedure to construct the optimal solution b^* from the optimal cost table T .

```

1: {Given the optimal cost table  $T$ , integer  $t \geq 0$ , and the position indices
of subsequences  $i_0, \dots, i_p$ , return the optimal solution  $b^*$ .}
2:  $k \leftarrow 0$ ;  $l \leftarrow t$ ;  $b^* \leftarrow \phi$ ; {Let the initial  $b^*$  be an empty sequence.}
3: while  $k < p - 2$  do
4:   if  $k$  is even then
5:     if  $T(k, l) = T(k + 1, l - 1)$  then
6:        $b^* \leftarrow [\text{zero}(i_{k+1} - i_k - 1) : 1 : b^*]$ ;  $l \leftarrow l - 1$ ;  $k \leftarrow k + 1$ ;
7:     else  $\{T(k, l) = T(k + 2, l) + i_{k+1} - i_k\}$ 
8:        $b^* \leftarrow [\text{zero}(i_{k+2} - i_k) : b^*]$ ;  $k \leftarrow k + 2$ ;
9:     else  $\{k \text{ is odd}\}$ 
10:    if  $T(k, l) = T(k + 1, l) + 1$  then
11:       $b^* \leftarrow [\text{zero}(i_{k+1} - i_k) : b^*]$ ;  $k \leftarrow k + 1$ ;
12:    else  $\{T(k, l) = T(k + 2, l - i_{k+1} + i_k)\}$ 
13:       $b^* \leftarrow [\text{zero}(i_{k+2} - i_{k+1}) : \text{one}(i_{k+1} - i_k) : b^*]$ ;
14:       $l \leftarrow l - i_{k+1} + i_k$ ;  $k \leftarrow k + 2$ ;
15:    if  $k = p - 2$  and  $l \geq 1$  then  $b^* \leftarrow [1 : b^*]$ ;
16:    else  $\{k = p - 1, \text{ or } k = p - 2 \text{ and } l = 0\}$   $b^* \leftarrow [0 : b^*]$ ;
17:    return  $b^*$ ;

```

Algorithm 2 constructs the optimal solution b^* based on the optimal cost table T . In the procedure, the function $\text{zero}(n)$ generates a sequence of n zeros and the function $\text{one}(n)$ generates a sequence of n ones. b^* is constructed from its end to its beginning, based on the optimal substructure shown in Theorem 7 and 8 (Lines 3–14) and the base case solution shown in Lemma 7 (Lines 15–16).

Time complexity: When $t \geq \tau$, the time complexity of Algorithm 1 is $O(d)$, where d is the number of bits in the binary representation of m minus 1. When $t < \tau$, the part of Algorithm 1 that constructs table T has time complexity as $O(p \cdot t)$. Since $p \leq d + 1$ and $t < \tau < i_{p-2} - i_1 + 1 < d + 1$, therefore, the time complexity of constructing table T is $O(d^2)$. The procedure to construct optimal solution, Algorithm 2, has time complexity as $O(d)$. Thus, the

total time complexity of Algorithm 1 is $O(d^2)$ when $t < \tau$. Thus, in either the case $t \geq \tau$ or the case $t < \tau$, the time complexity of Algorithm 1 is $O(d^2)$.

Example 4

Solve the optimal subtraction problem for $t = 5$ and

$$m = (011001001111011100)_2.$$

Solution: We first remove the trailing zeros of m and get the new $m = (0110010011110111)_2$. By examining m , we get $p = 8$, $q = 4$, and the position indices of the alternating subsequences of m are $i_0 = 0, i_1 = 3, i_2 = 4, i_3 = 8, i_4 = 10, i_5 = 11, i_6 = 13, i_7 = 15$, and $i_8 = 16$. By Equation (22), we get $\tau = 6 > t$. Therefore, we need to construct the table T to get the optimal cost. The table T constructed is shown in Table 3. Note that in the table, the row indices range from 7 to 0 and the column indices range from 0 to 5.

Table 3: Table T constructed by Algorithm 1 to get the optimal cost of the optimal subtraction problem for $m = (0110010011110111)_2$ and $t = 5$.

k	l					
	0	1	2	3	4	5
7	1*	1	1	1	1	1
6	2	1	1	1	1	1
5	3	2	1*	1	1	1
4	3	2	2	1*	1	1
3	4	3	3	2*	1	1
2	7	4	3	3	2	1
1	8	4	3	3	2*	1
0	10	7	4	3	3	2*

From Table 3, we get the optimal cost for $t = 5$ is 2. Based on the optimal cost, we can construct the optimal solution b^* by calling Algorithm 2. The entries in the table T with a superscript * are those entries that lead to the optimal cost for $t = 5$. Finally, we get the optimal solution for b^* is $b^* = (0001110000001001)_2$. After appending trailing zeros back, we eventually have

$$\begin{aligned} b^* &= (000111000000100100)_2, \\ a^* &= (100000010000000000)_2. \quad \square \end{aligned}$$

3.4 Achieving the Lower Bound on the Minimum Number of Cubes

We could get the lower bound on the minimum number of cubes η to cover m minterms by solving the optimal subtraction problem on m with $t = \tau - 1$. As shown in Algorithm 1, by solving that optimal subtraction problem, we can get $T(0, 0), T(0, 1), \dots, T(0, \tau - 1)$. Further, by Theorem 9, we know that for $k \geq \tau$, $T(0, k) = 1$. Therefore, for all $k \geq 0$, we know $T(0, k)$, the optimal cost of the optimal subtraction problem on m with $t = k$. Then, by Corollary 1, for any number i such that $T(0, 2^{i-1} - 1) > 2^{i-1}$, the minimum number of cubes η is greater than i . Therefore, a lower bound η_l on η is the smallest number η_l such that $T(0, 2^{\eta_l} - 1) \leq 2^{\eta_l}$, which can be obtained by examining the array $T(0, k)$.

Example 5

For the m of Example 4, we show how to get the lower bound on the minimum number of cubes η to cover m minterms. By Example 4, we have $\tau = 6$ and the last row of Table 3 gives $T(0, 0), \dots, T(0, \tau - 1)$. By examining the last row, we can see that when $\eta_l = 2$, $T(0, 2^{\eta_l} - 1) = T(0, 1) = 7 > 2^{\eta_l} = 2$. However, when $\eta_l = 3$, $T(0, 2^{\eta_l} - 1) = T(0, 3) = 3 < 2^{\eta_l} = 4$. Therefore, the smallest number η_l such that $T(0, 2^{\eta_l} - 1) \leq 2^{\eta_l}$ is $\eta_l = 3$. Thus a lower bound on η is 3. \square

4. EXPERIMENTAL RESULT

We tested our proposed algorithm on 53 two-level logic benchmarks that accompany the two-level logic minimizer Espresso [10]. For each benchmark, we counted the number of minterms covered by the set of cubes in that circuit³ and take that number as the input m to our program. Due to the page limit, we only list the results for 15 of those benchmarks in Table 4. The second column of the table shows how many cubes each circuit has after it was minimized by Espresso [11]. The third column shows the number of inputs of each circuit and the fourth column shows the number of minterms covered by each circuit, in hexadecimal form. The fifth column lists the lower bound on the minimum number of cubes to cover the corresponding number of minterms. The sixth column lists the upper bound according to Theorem 1. For all benchmarks, the runtime of our algorithm was negligible.

From Table 4, we can see that all upper bounds are less than the numbers of cubes of the corresponding circuits after two level minimization, which is not surprising. The benchmark circuits can be viewed as a random assignments of minterms. Although the proposed upper-bound solution is not sophisticated, it still has an advantage over these solutions, based on random assignments of minterms.

Surprisingly, for some of the benchmarks, e.g., `newapla1`, `newcond`, `bca`, the upper bound is equal to the lower bound; this indicates that the solution corresponding to the upper bounds is actually optimal. For some of the other benchmarks, e.g., `dc2`, `exp`, `in1`, the gap between the upper and lower bound is small; this indicates that the solution corresponding to the upper bound is sub-optimal, but quite good.

In Table 5, we list the numbers of benchmarks of the total 53 benchmarks that we tested with gaps between the upper and lower bounds of 0, 1, 2, 3, and more than 3. For about 10% of the benchmarks, the lower bound equals the upper bound. For nearly 20%, the gap is one. Thus, we conclude that our method solves the problem of minimizing the number of cubes to cover a given number of minterms for a very respectable percentage of test cases.

Table 4: Lower bound and upper bound on the minimum number of cubes to cover a specific number of minterms derived from benchmark circuits.

circuit	#cubes	#inputs	#minterms (hex)	lower bound	upper bound
<code>newapla1</code>	6	12	109	3	3
<code>dc2</code>	10	8	C3	3	4
<code>exp</code>	14	8	59	3	4
<code>newtpla</code>	14	15	DF2	3	8
<code>shift</code>	21	19	7FF01	2	12
<code>b3</code>	22	32	CDD60000	4	10
<code>newcond</code>	28	11	288	3	3
<code>b4</code>	29	33	1EFEA8C00	4	16
<code>in2</code>	30	19	66950	4	8
<code>in5</code>	37	24	8C2900	3	6
<code>in1</code>	55	16	6900	3	4
<code>ex7</code>	58	16	DBEA	3	11
<code>bca</code>	72	26	94000	3	3
<code>x1dn</code>	80	27	49E0D80	3	10
<code>ts10</code>	128	22	7FFF8	2	16

Table 5: Numbers and percentages of benchmarks with gaps between upper bounds and lower bounds equal to 0, 1, 2, 3 and larger than 3.

bound gap	0	1	2	3	> 3
#circuits	6	9	5	9	24
percentage (%)	11	17	9	17	45

³We ignore the output part of the cubes; we assume that the number of outputs is one.

5. CONCLUSION AND FUTURE WORK

This work is part of our broader effort to develop a methodology to synthesize digital circuits that implement probabilistic computation. The approach is motivated by concerns over variability and defects. When cast in terms of probabilities, digital computation becomes more robust. The paradigm provides significant tolerance to soft errors (i.e., bit flips). This tolerance scales gracefully to high error rates.

In this paper, we considered the problem of synthesizing two-level combinational logic to generate arbitrary probability values from *unbiased* input probability value of 0.5. With n inputs, optimizing the logic to generate a probability value $\frac{m}{2^n}$ entails minimizing the number of cubes to cover exactly m minterms. In this problem, the only requirement is that the union of a set of cubes covers m minterms in the Boolean space; which m minterms are covered does not matter. We provided both an upper bound and a strong lower bound on the minimum number of cubes to cover a given number of minterms. The lower bound is obtained by solving an instance of what we call the optimal subtraction problem on m . We proposed a dynamic-programming-based solution.

A future direction of research is to develop more sophisticated upper bounds. This, we estimate, would further reduce the gap between the lower bound and the upper bound for many test cases. For test cases where the upper bound is still significantly larger than the lower bound, we intend to develop a search-based algorithm. This algorithm would begin at the lower bound and search upwards for a solution.

6. REFERENCES

- [1] 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.
- [2] L. Chakrapani, P. Korkmaz, B. Akgul, and K. Palem, "Probabilistic system-on-a-chip architecture," *ACM Transactions on Design Automation of Electronic Systems*, vol. 12, no. 3, pp. 1–28, 2007.
- [3] W. Qian, M. D. Riedel, K. Barzagan, and D. Lilja, "The synthesis of combinational logic to generate probabilities," in *International Conference on Computer-Aided Design*, 2009, pp. 367–374.
- [4] R. K. Brayton, C. McMullen, G. D. Hachtel, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.
- [5] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: A multiple-level logic optimization system," *IEEE Transactions on Computer-Aided Design*, vol. 6, no. 6, pp. 1062–1081, 1987.
- [6] G. De Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Optimal state assignment for finite state machines," *IEEE Transactions on Computer-Aided Design*, vol. 4, no. 3, pp. 269–285, 1985.
- [7] T. Villa and A. Sangiovanni-Vincentelli, "NOVA: State assignment of finite state machines for optimal two-level logic implementation," *IEEE Transactions on Computer-Aided Design*, vol. 9, no. 9, pp. 905–924, 1990.
- [8] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, "Multilevel logic synthesis," *Proceedings of the IEEE*, vol. 78, no. 2, pp. 264–300, 1990.
- [9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Dynamic programming," in *Introduction to Algorithms, Second Edition*. MIT Press, 2001, ch. 15, pp. 323–369.
- [10] "espresso-book-examples." [Online]. Available: <http://embedded.eecs.berkeley.edu/pubs/downloads/espresso/index.htm>
- [11] R. L. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," *IEEE Transactions on Computer-Aided Design*, vol. 6, no. 5, pp. 727–750, 1987.