IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 18 July 2023, accepted 29 July 2023, date of publication 11 August 2023, date of current version 21 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3304329

RESEARCH ARTICLE

On the Non-Approximate Successive Cancellation Decoding of Binary Polar Codes With Medium Kernels

ZHILIANG HUANG, (Member, IEEE), ZONGSHENG JIANG, SHUIHONG ZHOU, AND XIAOYAN ZHANG

School of Mathematics, Physics and Information Engineering, Zhejiang Normal University, Jinhua 321004, China

Corresponding author: Zhiliang Huang (zlhuang@zjnu.cn)

This work was supported in part by the National Nature Science Foundation of China under Grant 61401399; in part by the Research Fund of the National Mobile Communications Research Laboratory, Southeast University, under Grant 2016D05; and in part by the Zhejiang Provincial Nature Science Foundation of China under Grant LY18F010017 and Grant LY17F010003.

ABSTRACT Polar codes constructed by large kernels can attain better finite length performance than those originating from Arıkan's 2×2 kernel. However, the successive cancellation (SC) decoding for these polar codes is impractical even for relatively small kernel size of m because complexity of the *kernel computation* grows exponentially with m. This research shows when $m > 2$, there exists a large amount of *like terms* in the kernel computation which yields a ground for facilitating the decoding. By transferring the kernel computation from the probability domain to the likelihood ratio domain (l-domain), the so-called l-formula method provides an efficient way to combine the like terms in the kernel computation for kernels up to size 11. However, the l-formula method becomes intractable for kernel size beyond 11. To further reduce the computational complexity, this paper proposes a W-formula method which transforms the kernel computation into the probability pair domain (W-domain). Advanced from the l-domain, the numerator and denominator of the likelihood ratio are considered separately, which eases the restrictions of combining like terms. The W-formula method can combine much more like terms resulting in a significant reduction on the number of sub-formulas for medium kernels ($m \leq 16$). Furthermore, in the W-domain, sub-formulas become regular and there exist many common sub-formulas whose computations can be shared. Being able to handle kernels of size up to 16, we show that the W-formula based SC decoding achieves a significant complexity reduction over the existing non-approximate SC decoding (the l-formula based SC decoding).

INDEX TERMS Large kernel, medium kernel, polar codes, SC decoding, W-formula.

I. INTRODUCTION

Polar codes were invented by Arıkan [1] as the first family of capacity-achieving codes with explicit construction method and low encoding and decoding complexity over binary input discrete memoryless channels (B-DMCs). The original polar codes are constructed based on a 2×2 kernel matrix $G_2 = \left(\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right)$ and its nth Kronecker power $G_2^{\otimes n}$ corresponds to a linear code with length $N = 2^n$. The kernel matrix can be replaced by larger matrices G_m where the kernel size $m > 2$. It has been know that polar codes constructed by large kernels may obtain better finite length performance [2].

Given a kernel matrix, its finite-length analysis can be assessed in terms of the *error exponent* [2] or the *scaling exponent* [3], [4]. It was shown that the block error probability of the G_2 based polar codes under the successive cancellation (SC) decoding is $O(2^{-2^{n(0.5-\epsilon)}})$ for any $\epsilon > 0$, where 0.5 is called the *error exponent* of the kernel. The error exponent can exceed 0.5 for large kernel matrix [2]. The scaling exponent μ is a constant that depends only on the kernel G_m and the channel W for which $N = O((I(W) - R)^{-\mu})$, where $I(W)$ is the symmetric capacity of W and R is the

The associate editor coordinating the review of this manuscript and approving it for publication was Zihuai Lin.

code rate [3], [4]. For the binary erasure channel, μ equals to 3.627 for G_2. In [3], Fazeli and Vardy constructed an 8×8 kernel with $\mu = 3.577$ and a 16×16 kernel with $\mu = 3.356$.

Many work have been dedicated to designing large kernels with larger error exponents [2], [5] [6] or smaller scaling exponents [3], [4]. However, complexity of a straightforward SC decoder for a G_m based polar code behaves as $O(2^m N \log N)$ [2]. The straightforward SC decoder becomes infeasible even with a moderate value of m.

There exist several work that address the complexity challenge of the straightforward SC decoder for polar codes with large kernels. These work can be divided into three categories. The first category is the approximate algorithm. An approximate decoding algorithm called the *sequential decoding* was introduced by Miloslavkaya and Trifonov [7] for decoding polar codes with an arbitrary binary kernel. The basic idea of *sequential decoding* is to design a metric that can be used to compare paths with different lengths. Because the designed metric is an approximation, the *sequential decoding* is an approximate algorithm. The *window decoding* [8] is another efficient decoding technique for polar codes with large kernels. Trifonov [8] introduced the window decoding for polar codes with non-binary Reed-Solomon (RS) kernels. Trofimiuk and Trifonov [9], [10] extended the window decoding for polar codes with some $2^t \times 2^t$ binary kernels by exploiting the relationship between the considered kernels and $G_2^{\otimes t}$. Abbasi and Viterbo [11] further improved the work of [9] and significantly reduced the computational cost of the window decoding. One drawback of the window decoding is that the block length of codes is limited to 2^t where t is a positive integer. Trifonov [12] proposed a universal method for kernel processing based on the recursive trellis representation of the codes. The window decoding and the recursive trellis based decoding algorithms use the *max* operator to approximate the *summation* operator in the computation of kernel processing. Therefore, they are approximate algorithms. Gupta et al. [13] proposed a *polar list decoding* for large polarization kernels and this method can be used to process kernels of size up to 64. Again, the polar list decoding is an approximate decoding algorithm.

The second category is the non-approximate algorithm for some special kernels. Buzaglo et al. [14], [15] proposed a permuted SC decoder to efficiently decode a special type of kernels called the permuted kernels. Two 16×16 permuted kernels that have better scaling exponents than Arikan's kernel are presented in [15]. It was shown in [15] that polar codes with the permuted kernels obtained slight improvement in error performance compared with polar codes with Arikan's kernel.

The third category is the non-approximate algorithm for *general* kernels (no limit to the kernels). Transforming the computation from probability domain into the likelihood domain (l-domain) is a common practice to reduce computational cost in decoding polar codes. Some efforts have been made to reduce the computational cost of the kernel computation by transforming the computation into the l-domain [16], [17] [18]. We refer the kernel computation in the l-domain as l-formulas. The papers [16], [17], and [18] are dedicated to obtaining l-formulas for large kernels. Bonik et al. [16] and Wang et al. [17] presented the l-formulas for the G_3 and G_4 kernels, respectively. Huang et al. [18] later proposed a so-called l-formula method, which can obtain l-formulas for any binary linear kernels. However, the kernel computation computed by l-formulas remains complex when the kernel size exceeds 11 [18]. It should be noted that the kernel computation computed by l-formulas has no approximation.

In this study, we follow the way of the third category. We focus on the complexity reduction of the non-approximate SC decoding for polar codes with general medium kernel, i. e., $2 < m \leq 16$. The factor of 2^m in the complexity of the straightforward SC decoding is due to a simple summation over all possibilities for the *kernel computation*. This research shows that a large number of *like terms* exist in the kernel computation for large kernels. Basically, the l-formula method provides an effective way to combine these like terms resulting in significant reduction of computational cost for small kernels ($m \leq 11$). However, as the kernel size increases, the relationship between terms becomes complex and makes combining like terms less efficient in the l-domain. To further combine like terms, we proposes a W-formula method, which transforms the kernel computation into the probability pair domain (W-domain). The W-formula method can combine many more like terms over the l-formula method. Like the operators in l-domain, we design similar operators which are used to combine like terms in the W-domain. Compared to the l-formula method, we consider the numerator and denominator of the ratio separately in the W-formula method, which eases the restrictions of combining like terms. In the W-domain, many more like terms can be combined than that in the l-domain leading to a significant reduction of the computational cost. Another important advantage of the W-formula method is that the formula in the W-domain becomes *regular* resulting in many common computations. In general, a formula consists of many sub-formulas and the number of sub-formulas increases rapidly with increasing the kernel size. Like the l-formula method, the kernel computation in the W-domain is called the W-formulas and a W-formula consists of many sub-formulas. The difference is that many common sub-formulas (their computations can be shared) appear in some W-formulas for large kernels. This feature which does not exist in the l-domain results in a multiple times reduction of the computation. With the above two advantages, our results show that the W-formula based SC decoding achieves significant reduction of the computation cost by comparing with the l-formula based SC decoding for optimal medium kernels in the literature.

The structure of this paper is organized as follows. In Section II, we introduce polar codes with large kernels, notations of W-formula, and definitions of combination channel and its algebra. In Section III, we introduce advantages and challenges of the l-formula method. In Section IV, we describe in details how to obtain W-formulas for an arbitrary binary kernel matrix. In Section V, we present how to use W-formulas to do the kernel computation. Section VI provides examples of W-formulas for some optimal kernels. The advantages of the W-formula method are presented in Section VII. In Section VIII, we present computational cost analysis of the W-formula based SC decoder. We conclude the paper in Section IX.

II. PRELIMINARIES

A. POLAR CODES WITH LARGE KERNELS

Let $W : \{0,1\} \to \mathcal{Y}$ denote a generic B-DMC with input alphabet $\{0,1\}$, output alphabet $\mathcal{Y}$, and transition probabilities $W(y|x), x \in \{0,1\}, y \in \mathcal{Y}$. We use a_1^N to denote a row vector $(a_1, \cdots, a_N)$. For a general kernel matrix G_m,[1] $W_m : \{0,1\}^m \to \mathcal{Y}^m$ is defined by

$$W_m(y_1^m|u_1^m) \triangleq \prod_{i=1}^{m} W(y_i|(u_1^m G_m)_i), \tag{1}$$

where $(u_1^m G_m)_i$ is the i-th element of $u_1^m G_m$.

The bit channels $W_m^{(i)} : \{0,1\} \to \mathcal{Y}^m \times \{0,1\}^{i-1}$, where $1 \le i \le m$, are defined by

$$W_m^{(i)}(y_1^m, u_1^{i-1}|u_i) \triangleq \frac{1}{2^{m-1}} \sum_{u_{i+1}^m \in \{0,1\}^{m-i}} W_m(y_1^m|u_1^m). \tag{2}$$

For simplicity, we omit the alphabet $\{0,1\}^{m-i}$ of u_{i+1}^m in the rest of the paper.

For the SC decoding, the basic recursive formulas are [1]

$$W_m^{(i)}(y_1^m, u_1^{i-1}|u_i) = \frac{1}{2^{m-1}} \sum_{u_{i+1}^m} W(y_1|(u_1^m G_m)_1) \cdots W(y_m|(u_1^m G_m)_m). \tag{3}$$

Let $v_k = (u_1^m G_m)_k$, equation (3) can be rewritten as

$$W_m^{(i)}(y_1^m, u_1^{i-1}|u_i) = \frac{1}{2^{m-1}} \sum_{u_{i+1}^m} W(y_1|v_1) \cdots W(y_m|v_m). \tag{4}$$

For simplicity, we use $W_m^{(i)}(u_i)$ to denote $W_m^{(i)}(y_1^m, 0_1^{i-1}|u_i)$.

In (4), $W(y_k|v_k)$ is referred as a *channel expression* and $\sum_{u_{i+1}^m} W(y_1|v_1) \cdots W(y_m|v_m)$ is referred as an *expression*. We use *dim* to denote the dimension of $\{0,1\}^{m-i}$ (alphabet of u_{i+1}^m); that is $dim = m - i$ for u_{i+1}^m. And we denote the *dim* of u_{i+1}^m as the *dim* of an expression $\sum_{u_{i+1}^m} W(y_1|v_1) \cdots W(y_m|v_m)$. We sometimes denote an expression as a letter E and dim_E denote the *dim* of E. Note that we ignore the constant $\frac{1}{2^{m-1}}$ in the computation of $W_m^{(i)}(y_1^m, u_1^{i-1}|u_i)$ by W-formulas.

[1] All kernels used in this study are linear kernels given in [6].

The bit channel likelihood ratio is

$$l_m^{(i)}(y_1^m, u_1^{i-1}) = \frac{W_m^{(i)}(y_1^m, u_1^{i-1}|u_i = 0)}{W_m^{(i)}(y_1^m, u_1^{i-1}|u_i = 1)}, \tag{5}$$

where $i = 1, \ldots, m$. Again, for simplicity, we use $l_m^{(i)}$ to denote $l_m^{(i)}(y_1^m, 0_1^{i-1})$.

Note that v_k in (4) is a linear combination of some terms in u_1^m, i.e., $v_1 = u_5 + u_6 + u_7$. Let $\mathcal{S}_i$ and $\mathcal{S}_j$ denote the set of variables contained in v_i and v_j, respectively. The operators $\sqcap$, $\sqsubset$, $=$, and $\lessdot$ are used to indicate the relationship between v_i and v_j. Denote $v_i \sqcap v_j = \emptyset$ when $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$, $v_i \sqsubset v_j$ when $\mathcal{S}_i \subset \mathcal{S}_j$, $v_i = v_j$ when $\mathcal{S}_i = \mathcal{S}_j$, and $u_i \lessdot v_k$ when $u_i \in \mathcal{S}_k$. We also call v_k contains u_i for the last case. Denote $==$ as the *equal to* operator. Then, the value of $v_i == v_j$ is *true* if $v_i = v_j$ and *false* otherwise. Let $v_2 = u_7 + u_8$. Then $v_1 + v_2 = u_5 + u_6 + u_8$. Let $\bar{v}_k = v_k + 1$. Denote $1_{v_k=0}$ as the indicator function of equation $v_k = 0$; thus $1_{v_k=0}$ equals 1 if $v_k = 0$ and 0 otherwise.

B. NOTATIONS OF W-FORMULA

Let $B_{y_i} \triangleq (B(y_i|0), B(y_i|1)) = (W(y_i|0), W(y_i|1))$. Because there is a one-to-one correspondence between y_i and i, we also use B_i to denote B_{y_i}. Define

$$B_i^{-1} = (B(y_i|1), B(y_i|0))$$

$$L(B_i) = B(y_i|0),$$

$$S(B_i) = B(y_i|0) + B(y_i|1),$$

$$B_i \diamond B_j = (S(B_i \cdot B_j), S(B_i^{-1} \cdot B_j)),$$

$$B_i \cdot B_j = (B(y_i|0)B(y_j|0), B(y_i|1)B(y_j|1)).$$

The priority for operators -1, $\cdot$ and $\diamond$ is $\cdot < \diamond < -1$, i.e., $B_i \cdot B_j^{-1} \diamond B_k = B_i \cdot ((B_j^{-1}) \diamond B_k)$.

C. COMBINATION CHANNEL AND ITS ALGEBRA

Two channel expressions in (4) can be combined as a *combination channel* if they satisfy certain conditions. Basically, we have two types of combination channel: the *zero-variable-type* and the *one-variable-type*. For two given channel expressions $W(y_1|v_1)$ and $W(y_2|v_2)$, if $v_1 == v_2$, they can be combined as a combination channel of the *zero-variable-type*. Let

$$W((y_1^2)_z|v_1) \triangleq W(y_1|v_1)W(y_2|v_1), \tag{6}$$

where z denotes the *zero-variable-type*. Then, $B_{(y_1^2)_z} \triangleq (W((y_1^2)_z|0), W((y_1^2)_z|1)) = B_1 \cdot B_2$.

For two channel expressions $W(y_1|v_1)$ and $W(y_2|v_2)$ in (4), if $v_2 \sqsubset v_1$ and $v_2 \sqcap v_i = \emptyset, i = 3, \ldots, m$, then they can be combined as a combination channel of the *one-variable-type*. For example, let

$$W((y_1^2)_o|s_1) \triangleq \sum_{v_2} W(y_1|v_1)W(y_2|v_2), \tag{7}$$

where o denotes the *one-variable-type* and $s_1 = v_1 - v_2$. The equation (7) can be rewritten as

$$W((y_1^2)_o|s_1) \triangleq \sum_{v_2} W(y_1|v_2 + s_1)W(y_2|v_2).$$

Then, $B_{(y_1^2)_o} \triangleq (W((y_1^2)_o|0), W((y_1^2)_o|1)) = B_1 \diamond B_2$.

The combination channel can be combined again. For example, let

$$W(((y_1^2)_o, y_3)_z|s_1) \triangleq W((y_1^2)_o|s_1)W(y_3|s_1), \tag{8}$$

we will have $B_{((y_1^2)_o,y_3)_z} = B_{(y_1^2)_o} \cdot B_3 = B_1 \diamond B_2 \cdot B_3$.

III. THE *l*-FORMULA AND ITS CHALLENGES

Our basic purpose is to reduce the computational cost of (3). The complexity of straightforward calculation of (3) is $O(2^m)$. In [18], an *l*-formula method is proposed to reduce the computation cost of (3). The *l*-formula behaves well for kernels with size up to 11. However, complexity of the *l*-formula method increases fast with kernel size m and it becomes unacceptable when $m > 11$.

A. THE *l*-FORMULA

In the *l*-formula method, we focus on the computation of $l_m^{(i)}$ in (5). It has been shown [18] that $l_m^{(i)}$ can be computed by a formula of $l_1, \cdots, l_m$, which are connected by $\diamond$ and $\times$ for an arbitrary kernel G_m and these formulas are called the *l*-formulas. The $\diamond$ operator is the key factor that the *l*-formula method can reduce the computational cost of (5). Basically, the $\diamond$ operator is able to reduce the dimension of computation. We use an example to illustrate the $\diamond$ operator. Consider

$$W(y_1^2|u_1) = \sum_{u_2} W(y_1|u_1 + u_2)W(y_2|u_2), \tag{9}$$

we have

$$l_{y_1^2} \triangleq \frac{W(y_1^2|0)}{W(y_1^2|1)} = l_1 \diamond l_2. \tag{10}$$

It can be seen that the summation of (9) can be simplified into one $\diamond$ in (10).

For optimal binary kernels provided in [6], all the first bit channels' transition probabilities (at the kernel level) have the following form

$$W_m^{(1)}(y_1^m|u_1) = \sum_{u_2^m} W(y_1|u_1 + \cdots + u_m)W(y_2|u_2)\cdots W(y_m|u_m), \tag{11}$$

where $m = 2, \cdots, 16$. By using the *l*-formula generating procedures [18], we have

$$l_{y_1^m} \triangleq \frac{W_m^{(1)}(y_1^m|0)}{W_m^{(1)}(y_1^m|1)} = l_1 \diamond l_2 \diamond \cdots \diamond l_m. \tag{12}$$

The complexity of computation (11) is $O(2^m)$, while the *l*-formula of (12) reduces it to $O(m)$. Certainly, the above example is a *perfect* case for the *l*-formula method. In general cases, the *l*-formula method has its challenges.

It should be noted that the definition of $\diamond$ in the *W*-formula method reserves the advantage of $\diamond$ in the *l*-formula method. Similar to (12), we have

$$B_{y_1^m} \triangleq (W_m^{(1)}(y_1^m|0), W_m^{(1)}(y_1^m|1)) = B_1 \diamond B_2 \diamond \cdots \diamond B_m. \tag{13}$$

B. CHALLENGES OF THE *l*-FORMULA

(1) Too many sub-formulas: An *l*-formula consists of many sub-formulas. The computational cost of computing an *l*-formula is determined by the number of sub-formulas that it contains. In the *l*-formula generating algorithm, the number of sub-formulas of certain *l*-formula increases as we increase the kernel size m. The total number of sub-formulas makes the computation infeasible for optimal kernel G_m when $m > 11$ [18]. Avoiding the rapid growth of the number of sub-formulas of *l*-formulas motivates the proposed *W*-formula method.

(2) No common part in l-formula: Unfortunately, sub-formulas in the *l*-formula are not in common and their computation cannot be shared.

IV. *W*-FORMULAS

In this section, we propose a procedure to generate *W*-formulas for an arbitrary kernel G_m. First, definitions of the *W*-formula and sub-formula are provided. Second, the outline of the *W*-formula Generating Algorithm is described as a recursive algorithm and some functions are denoted. Third, details and proofs of the functions are presented. Fourth, the *W*-formula generating algorithm for any kernel size is proposed by using these functions. Finally, an example is provided to illustrate the *W*-formula Generating Algorithm.

A. DEFINITION OF THE *W*-FORMULA

Definition 1 (W-formula): The *W*-formula for a $W_m^{(i)}(y_1^m, u_1^{i-1}|u_i)$ is *defined* as a formula of $B_1, \cdots, B_m$ ($B_1, \cdots, B_m$ may appear more than once) connected by operators -1, L, S, $\cdot$, and $\diamond$, which can be used to compute the value of $W_m^{(i)}(y_1^m, u_1^{i-1}|u_i)$.

Definition 2 (sub-formula): A sub-formula is defined as a formula of $B_1, \cdots, B_m$ ($B_1, \cdots, B_m$ appear at most once) connected by operators -1, $\cdot$, and $\diamond$.

Note that the result of computing a *W*-formula is a value and the result of computing a sub-formula is a pair (two values like B_i).

For G_2, a straightforward calculation based on (3) results in

$$\begin{aligned} W_2^{(1)}(y_1^2|0) &= S(B_1 \cdot B_2), \\ W_2^{(1)}(y_1^2|1) &= S(B_1^{-1} \cdot B_2), \\ W_2^{(2)}(y_1^2, u_1|0) &= L(B_1^{1-2u_1} \cdot B_2), \\ W_2^{(2)}(y_1^2, u_1|1) &= L(B_1^{-(1-2u_1)} \cdot B_2^{-1}), \end{aligned}$$

which are called the *W*-formulas of G_2.

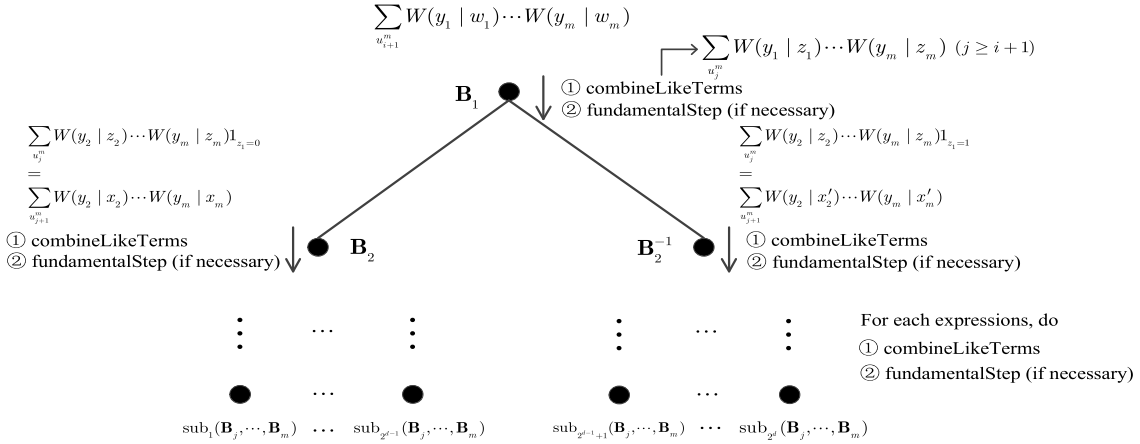


FIGURE 1. The recursive *W*-formula generating procedure.

B. OUTLINE OF THE W-FORMULA GENERATING ALGORITHM

The W-formula Generating algorithm can be described as a recursive algorithm (see Figure 1). The Generating procedure starts from an *expression* $\sum_{u_{i+1}^m} W(y_1|w_1)\cdots W(y_m|w_m)$ which is the result of implementing the **Early processing** (introduce later). The algorithm first implements the function **combineLikeTerms** on the expression, which combines like terms as much as possible. Suppose that the resulting expression is $\sum_{u_j^m} W(y_1|z_1)\cdots W(y_m|z_m)$ $(j \geq i+1)$ after implementing the function **combineLikeTerms**. If $j == m$ $(dim = 1)$, then $\sum_{u_j^m} W(y_1|z_1)\cdots W(y_m|z_m)$ can be expressed as a sub-formula and the algorithm is finished; otherwise, the algorithm implements the function **fundamentalStep** on the resulting expression and it is broken into two short expressions. The two short expressions have the similar form as the original expression. Therefore, we can do the same procedure on them (First, **combineLikeTerms**. Second, **fundamentalStep** if necessary). By implementing the **fundamentalStep** function for all 2^i expressions in one level and generating 2^{i+1} expressions to the next level is called one iteration. Because the *dim* of expressions for the next level decrease at least 1 ($m-i$ becomes $m-j$, $j \geq i+1$), the algorithm will end at most $m-i-1$ iterations (After $m-i-1$ iterations, the *dim* of expressions become 1 and they can be expressed as a sub-formula). In Figure 1, $\text{sub}_i(B_j,\cdots,B_m)$ denotes a sub-formula, which contains $B_j,\cdots,B_m$ exactly once (different in order). The number d is the depth of the tree. The value of d is equivalent to the number of iterations ($d \leq m-i-1$).

Before going to the recursive algorithm described in Figure 1, we can do the **Early processing** on (3) to simplify subsequent processing (see Figure 2). The **Early processing** contains two functions **hideKnownValues** and **standardExpressionTransform**. The function **hideKnownValues** can ignore the known values u_1^{i-1} at first and just consider them in the final step. The function **standardExpressionTransform** brings some benefits for the function **combineLikeTerms**.

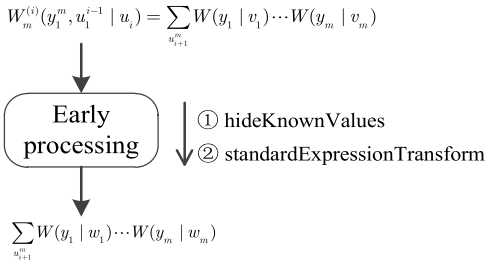


FIGURE 2. Early processing.

C. DETAILS OF FUNCTIONS

The function **combineLikeTerms** involves two functions **zeroVariableCombine** and **oneVariableCombine**. Next, we will introduce **hideKnownValues**, **standardExpressionTransform**, **zeroVariableCombine**, **oneVariableCombine**, **combineLikeTerms**, and **fundamentalStep** in detail.

1) HIDE KNOWN VALUES

Consider the ith bit channel of a kernel G_m. Let G_A and G_B be the submatrices of G_m consisting of the first $i-1$ rows and the last $m-i+1$ rows, respectively. We have $W(y_j|(u_1^m G_m)_j) = W(y_j|a_j + (u_i^m G_B)_j)$, where $a_j = (u_1^{i-1} G_A)_j, j = 1,\cdots,m$.

Consider the following equation

$$W_m^{(i)}(y_1^m, 0_1^{i-1}|u_i) = \sum_{u_{i+1}^m} W(y_1|(u_i^m G_B)_1)\cdots W(y_m|(u_i^m G_B)_m). \quad (14)$$

Because $a_j s, j = 1,\ldots,m$ are known values, we can obtain the W-formula of (14) instead of (3) at first. Then, we substitute B_j by $(B_j)^{1-2a_j}$ for each j in the obtained W-formula and the resulting W-formula is the W-formula of (3). This fact is given in the following proposition.

Proposition 1 (Hide Known Values): Assume a W-formula of $W_m^{(i)}(y_1^m, 0_1^{i-1}|u_i)$ is $f_i(B_1,\cdots,B_m)$, which is obtained

Algorithm 1 hideKnownValues(E)

Input: $E = \sum_{u_{i+1}^m} W(y_1|(u_1^m G_m)_1)\cdots W(y_m|(u_1^m G_m)_m)$

Output: $\acute{E} = \sum_{u_{i+1}^m} W(y_1|((0_1^i, u_{i+1}^m)G_m)_1)\cdots W(y_m|((0_1^i, u_{i+1}^m)G_m)_m)$

1: The implementation process is straightforward and omitted.

by implementing Algorithm 7 (introduce later) on (14). Then, the W-formula of $W_m^{(i)}(y_1^m, u_1^{i-1}|u_i)$ is $f_i(B_1^{(1-2a_1)}, \cdots, B_m^{(1-2a_m)})$.

Proof: For each $j \in \{1, \cdots, m\}$, we have

$$(W(y_j|a_j + (u^m_{i,u_i=0}G_B)_j), W(y_j|a_j + (u^m_{i,u_i=1}G_B)_j)) = (W(y_j|(u^m_{i,u_i=0}G_B)_j), W(y_j|(u^m_{i,u_i=1}G_B)_j))^{1-2a_j}.$$

Therefore, each B_j in the W-formula of $W_m^{(i)}(y_1^m, 0_1^{i-1}|u_i) = f_i(B_1, \cdots, B_m)$ becomes $B_j^{1-2a_j}$ in the W-formula of $W_m^{(i)}(y_1^m, u_1^{i-1}|u_i) = f_i(B_1^{(1-2a_1)}, \cdots, B_m^{(1-2a_m)})$. ∎

The pseudo code of function **hideKnownValues** is described in Algorithm 1. Because the implementation process of **hideKnownValues** is straightforward, it is omitted.

2) STANDARD EXPRESSION TRANSFORM

Definition 3 (Standard expression): A standard expression has the following form

$$\sum_{u_{i+1}^m} W(y_1|v_1)\cdots W(y_i|v_i)W(y_{i+1}|u_{i+1})\cdots W(y_m|u_m); \quad (15)$$

that is $v_j = u_j$ when $j = i+1, \cdots, m$.

Lemma 1 (Standard expression transform [18]): Given an expression $W_m^{(i)}(y_1^m, 0_1^{i-1}|u_i)$ defined by a lower triangular matrix G_m, it can be transformed into a standard expression.

Proof: This lemma is included and proved in [18] (Section III, Lemma 1). ∎

Let $A = \{i+1, \ldots, m\}$. Let G_{AA} denote the submatrix of G_m consisting of the array of elements $((G_m)_{ij})$ with $i \in A$ and $j \in A$. We can do linear row transforms on last $m-i$ rows of G_m and make G_{AA} an identity matrix. We use function **rowTransform** to denote this procedure; that is $\acute{G}_m$ = **rowTransform**(G_m) ($\acute{G}_{AA}$ is identity matrix). In [18], we show that the expression $\sum_{u_{i+1}^m} W(y_1|((0_1^i, u_{i+1}^m)\acute{G}_m)_1)\cdots W(y_m|((0_1^i, u_{i+1}^m)\acute{G}_m)_m)$ is equivalent to the original expression and it is an standard expression because of $(0_1^i, u_{i+1}^m)\acute{G}_m)_j = u_j$ when $j = i+1, \ldots, m$. The pseudo code of function **standardExpressionTransform** is described in Algorithm 2.

3) ZERO VARIABLE COMBINATION

Proposition 2 (Zero-variable combination): Given an expression E as follows

$$E = \sum_{u_{i+1}^m} W(y_1|v_1)W(y_2|v_1)W(y_3|v_3)\cdots W(y_m|v_m), \quad (16)$$

we have

$$E = \sum_{u_{i+1}^m} W((y_1^2)_z|v_1)W(y_3|v_3)\cdots W(y_m|v_m),$$

where $W((y_1^2)_z|v_1) \triangleq W(y_1|v_1)W(y_2|v_1)$. Furthermore, we have

$$B_{(y_1^2)_z} \triangleq (W((y_1^2)_z|v_1 = 0), W((y_1^2)_z|v_1 = 1)) = B_1 \cdot B_2.$$

The proof is immediate and omitted.

The Proposition 2 describes one type of the **zeroVariableCombine** function. It has another three forms as follows.

$$W((\bar{y}_1, y_2)_z|v_1) \triangleq W(y_1|\bar{v}_1)W(y_2|v_1), \quad (17)$$

$$W((y_1, \bar{y}_2)_z|v_1) \triangleq W(y_1|v_1)W(y_2|\bar{v}_1), \quad (18)$$

$$W((\bar{y}_1, \bar{y}_2)_z|v_1) \triangleq W(y_1|\bar{v}_1)W(y_2|\bar{v}_1). \quad (19)$$

Correspondingly, we have $B_{(\bar{y}_1,y_2)_z} = B_1^{-1} \cdot B_2$, $B_{(y_1,\bar{y}_2)_z} = B_1 \cdot B_2^{-1}$, $B_{(\bar{y}_1,\bar{y}_2)_z} = B_1^{-1} \cdot B_2^{-1}$.

In (17), $W((\bar{y}_1, y_2)_z|v_1)$ is referred as a *combination* channel expression and $\bar{y}_1$ means that the corresponding channel expression in the right side of (17) is $W(y_1|\bar{v}_1)$. The two channel expressions in the right side of (16), (17), (18), or (19) are denoted as a channel expression pair of the *zero-variable-type*. The **zeroVariableCombine** function is to combine all possible channel expression pairs of the zero-variable-type for a given expression.

In Algorithm 3, we introduce a boolean array **arrayVisited**. The size of **arrayVisited** is m. The value **arrayVisited**[i] denotes that the channel expression $W(y_i|v_i)$ is visited or not. The initial values of **arrayVisited** are all **false**. In the line 6 of Algorithm 3, we set **arrayVisited**[j] to **true** because the channel expression $W(y_j|v_j)$ is combined into $W(y_i, y_j|v_i)$. Then, the channel expression $W(y_j|v_j)$ will never be visited again in the algorithm. The details of the **zeroVariableCombine** function are presented in Algorithm 3. Note that the input and output in the algorithm are just examples.

4) ONE VARIABLE COMBINATION

Proposition 3 (One-variable combination): Given an expression E as follows:

$$E = \sum_{u_{i+1}^m} W(y_1|v_1 + v_2)W(y_2|v_2)W(y_3|v_3)\cdots W(y_m|v_m)$$

and assuming $v_2 = u_{i+1}$, $u_{i+1} \preccurlyeq v_1 + v_2$ and $v_2 \sqcap v_k = \emptyset$, for $k = 3, \cdots, m$, we have

$$E = \sum_{u_{i+2}^m} W((y_1^2)_o|v_1)W(y_3|v_3)\cdots W(y_m|v_m)$$

Algorithm 2 standardExpressionTransform(E)

Input: $E = \sum_{u_{i+1}^m} W(y_1|((0_1^i, u_{i+1}^m)G_m)_1)\cdots W(y_m|((0_1^i, u_{i+1}^m)G_m)_m)$

Output: $\acute{E} = \sum_{u_{i+1}^m} W(y_1|w_1)\cdots W(y_i|w_i)W(y_{i+1}|u_{i+1})\cdots W(y_m|u_m)$, where $w_j = (0_1^i, u_{i+1}^m)\acute{G}_m)_j, j = 1, 2, \ldots, m$

1: $\acute{G}_m =$ **rowTransform**(G_m); ▷ Submatrix $\acute{G}_{AA}$ of $\acute{G}_m$ is an identity matrix

2: **return** $\acute{E} = \sum_{u_{i+1}^m} W(y_1|((0_1^i, u_{i+1}^m)\acute{G}_m)_1)\cdots W(y_m|((0_1^i, u_{i+1}^m)\acute{G}_m)_m)$

Algorithm 3 zeroVariableCombine(E, *zeroFlag*, **arrayVisited**)

Input: $E = \sum_{u_{i+1}^m} W(y_1|v_1)W(y_2|v_2)\cdots W(y_m|v_m)$, *zeroFlag*, **arrayVisited**

Output: $E = \sum_{u_{i+1}^m} W((y_1^2)_z|v_1)W(y_3|v_3)\cdots W(y_m|v_m)$ ▷ suppose $v_1 == v_2$ is true

1: **for** $i = 1, 2, \ldots, m$ **do**
2: **if arrayVisited**$[i] ==$ **false then**
3: **for** $j = i+1, 2, \ldots, m$ **do**
4: **if arrayVisited**$[j] ==$ **false and** $v_i == v_j$ **then**
5: $W(y_i|v_i) \leftarrow W((y_i, y_j)_z|v_i)$; ▷ $W((y_i, y_j)_z|v_i) \triangleq W(y_i|v_i)W(y_j|v_i)$
6: **arrayVisited**$[j] =$ **true**
7: *zeroFlag* $=$ **true**
8: **end if**
9: **end for**
10: **end if**
11: **end for**
12: **return** $\sum_{u_{i+1}^m} W((y_1^2)_z|v_1)W(y_3|v_3)\cdots W(y_m|v_m)$

where $W((y_1^2)_o|v_1) \triangleq \sum_{v_2} W(y_1|v_1+v_2)W(y_2|v_2)$. Furthermore, we have $B_{(y_1^2)_o} = B_1 \diamond B_2$.

Proof:

$$E = \sum_{u_{i+1}^m} W(y_1|v_1+v_2)W(y_2|v_2)W(y_3|v_3)\cdots W(y_m|v_m) \tag{20}$$

$$\overset{(a)}{=} \sum_{u_{i+2}^m}\left(\sum_{v_2} W(y_1|v_1+v_2)W(y_2|v_2)\right)\cdots W(y_m|v_m) \tag{21}$$

$$\overset{(b)}{=} \sum_{u_{i+2}^m} W((y_1^2)_o|v_1)W(y_3|v_3)\cdots W(y_m|v_m) \tag{22}$$

where (a) follows from the fact that only v_2 and v_1+v_2 contain u_{i+1}. Then, we can define $W((y_1^2)_o|v_1) \triangleq \sum_{v_2} W(y_1|v_1+v_2)W(y_2|v_2)$ and (b) follows. ■

The Proposition 3 describes one type of the **oneVariableCombine** function. It has another three forms as following.

$$W((\bar{y}_1, y_2)_o|v_1) \triangleq \sum_{v_2} W(y_1|\overline{v_1+v_2})W(y_2|v_2), \tag{23}$$

$$W((y_1, \bar{y}_2)_o|v_1) \triangleq \sum_{v_2} W(y_1|v_1+v_2)W(y_2|\bar{v}_2), \tag{24}$$

$$W((\bar{y}_1, \bar{y}_2)_o|v_1) \triangleq \sum_{v_2} W(y_1|\overline{v_1+v_2})W(y_2|\bar{v}_2). \tag{25}$$

Correspondingly, we have $B_{(\bar{y}_1,y_2)_o} = B_1^{-1} \diamond B_2$, $B_{(y_1,\bar{y}_2)_o} = B_1 \diamond B_2^{-1}$, $B_{(\bar{y}_1,\bar{y}_2)_o} = B_1^{-1} \diamond B_2^{-1}$.

The two channel expressions in the right side of (20), (23), (24), and (25) are denoted as a channel expression pair of the *one-variable-type*. The details of the **oneVariableCombine** function are presented in Algorithm 4. We first find a v_k that contains a single variable (lines 3 to 8). Suppose the found single variable is v_k which contains a single variable u_j. Then, we construct a set $\mathcal{S}$ contained the indices of $v_l, l \neq k, l \in \{1, \ldots, m\}$ which contain u_j (lines 10 to 15). According to Proposition 3, if $\mathcal{S}$ contains only one element, the channel expressions $W(y_k|v_k)$ and $W(y_p|v_p)$ can be combined (suppose the element in $\mathcal{S}$ is p) (lines 16 to 22).

5) COMBINE LIKE TERMS

Generally speaking, the function **combineLikeTerms** combines like terms for an expression as much as possible. There are mainly two types of like terms: the zero-variable-type and the one-variable-type corresponding to the functions **zeroVariableCombine** and **oneVariableCombine**. Given an expression E, the *dim* of E will not reduce after implementing the **zeroVariableCombine** function on E. But the *dim* of E may reduce 1 after implementing the **oneVariableCombine** function on E. That's why we name the two functions as **zeroVariableCombine** and **oneVariableCombine**.

The pseudo code of function **combineLikeTerms** is described in Algorithm 5. Note that we just combine one possible channel expression pair of the one-variable-type in the **oneVariableCombine**. Because after combining a channel expression pair of the one-variable-type, it may appear some channel expression pairs of the zero-variable-type and we should combine them at first. In Algorithm 5, the *while* loop is broken only if both *zeroFlag* and *oneFlag* are false. It means the function **combineLikeTerms** is ended only if there are no like terms of two types to be combined in the expression E.

If $dim_E == 1$, E can be combined to only one channel expression and to be a sub-formula. For

Algorithm 4 oneVariableCombine(E, *oneFlag*, **arrayVisited**)

Input: $E = \sum_{u_{i+1}^m} W(y_1|v_1)W(y_2|v_2)\cdots W(y_m|v_m)$

Output: $E = \sum_{u_{i+1}^m} W(y_1|v_1)W(y_2|v_2)\cdots W(y_m|v_m)$ or $\sum_{u_{i+1}^{j-1},u_{j+1}^m} W(y_1|v_1)\cdots W(y_{l-1}|v_{l-1})W((y_l,y_k)_o|v_l - v_k)W(y_{l+1}|v_{l+1})\cdots W(y_m|v_m)$ ▷ suppose $v_k = u_j$, $u_j \lessdot v_l$ and $v_k \sqcap v_{\acute{k}} = \emptyset$, for $\acute{k} = 1,\ldots,l-1,l+1,\ldots,m$

for $j = i+1, i+2, \ldots, m$ **do**
  $uFlag =$ **flase**;
  for $k = i+1, i+2, \ldots, m$ **do** ▷ After for loop, $v_k = u_j$
    if **arrayVisited**$[k] ==$ **false and** $v_k == u_j$ **then**
      $uFlag =$ **true**;
      break;
    end if
  end for
  if $uFlag ==$ **true then**
    $\mathcal{S} \leftarrow \varnothing$;
    for $l = i+1, i+2, \ldots, m$ **do**
      if $l! = k$ and $u_j \lessdot v_l$ **then**
        $\mathcal{S} \leftarrow \mathcal{S} \bigcup l$;
      end if
    end for
    if $|\mathcal{S}| == 1$ **then**
      $W((y_p,y_k)_o|v_p - v_k) \triangleq \sum_{u_j} W(y_p|v_p)W(y_k|v_k)$; ▷ Suppose the only element in $\mathcal{S}$ is v_p
      $W(y_p|v_p) \leftarrow W((y_p,y_k)_o|v_p - v_k)$;
      arrayVisited$[k] =$ **true**;
      oneFlag $=$ **true**;
      return $\sum_{u_{i+1}^{j-1},u_{j+1}^m} W(y_1|v_1)\cdots W(y_{p-1}|v_{p-1})W((y_p,y_k)_o|v_p - v_k)W(y_{p+1}|v_{p+1})\cdots W(y_m|v_m)$;
    end if
  end if
end for
return $\sum_{u_{i+1}^m} W(y_1|v_1)W(y_2|v_2)\cdots W(y_m|v_m)$; ▷ Do nothing

Algorithm 5 combineLikeTerms(E, **arrayVisited**)

Input: $E = \sum_{u_i^m} W(y_k|v_1)\cdots W(y_m|v_m)$, **arrayVisited**

Output: $E = \sum_{u_j^m} W(y_k|w_1)\cdots W(y_m|w_m)$ ▷ Updated by combining all like terms

while true do
  zeroFlag $=$ **false**, *oneFlag* $=$ **false**;
  $E \leftarrow$ **zeroVariableCombine**(E, *zeroFlag*, **arrayVisited**);
  $E \leftarrow$ **oneVariableCombine**(E, *oneFlag*, **arrayVisited**);
  if *zeroFlag* $=$ **false and** *oneFlag* $==$ **false then**
    return E;
  end if
end while

```
struct Node {
    String localFormula;
    struct Node *lChild;
    struct Node *rChild;
}
```

FIGURE 3. Node structure.

example, let $E = \sum_{u_i} W(y_{i_1}|u_i)W(y_{i_2}|\overline{u}_i)W(y_{i_3}|u_i)$. Define $\sum_{u_i} W(y_{i_1},\overline{y}_{i_2},y_{i_3}|u_i) = W(y_{i_1},\overline{y}_{i_2}|u_i)W(y_{i_3}|u_i)$ and $W(y_{i_1},\overline{y}_{i_2}|u_i) = W(y_{i_1}|u_i)W(y_{i_2}|\overline{u}_i)$. Then, $E = \sum_{u_i} W(y_{i_1},\overline{y}_{i_2},y_{i_3}|u_i)$. So, we have $E = S(B_{y_{i_1}}\cdot\overline{B}_{y_{i_2}}\cdot B_{y_{i_3}})$. In the tree structure of a W-formula, we will drop S from expression of E. Therefore, $B_{y_{i_1}}\cdot\overline{B}_{y_{i_2}}\cdot B_{y_{i_3}}$ is the sub-formula for this example. Therefore, the result of the **combineLikeTerms** function may be an expression or a sub-formula.

6) FUNDAMENTAL STEP

Suppose a $v_k = u_{j_1} + u_{j_2} + \cdots$ and $j_1 < j_2 < \cdots$. We refer the u_{j_1} as the first term of v_k.

Lemma 2 (Fundamental step for the W-formula): Given an ith bit channel expression E as in (26), shown at the bottom of the next page, and assume the first term in v_k is u_j, we have $x_l = v_l + v_k$ if $u_j \lessdot v_l$; otherwise $x_l = v_l$, for $l = k+1, \cdots, m$ in the left part of (27). In the right part of as in (27), shown at the bottom of the next page, if $u_j \lessdot v_l$, $x'_l = \overline{v_l + v_k}$; otherwise $x'_l = v_l$.

Proof: Let $M \triangleq W(y_{k+1}|v_{k+1})\cdots W(y_m|v_m)$ and define

$$W(y_{k+1}^m|0) \triangleq W(y_{k+1}^m|v_k = 0) \triangleq \sum_{u_{i+1}^m} M\cdot 1_{v_k=0},$$

$$W(y_{k+1}^m|1) \triangleq W(y_{k+1}^m|v_k = 1) \triangleq \sum_{u_{i+1}^m} M\cdot 1_{v_k=1}.$$

Algorithm 6 fundamentalStep(E, **arrayVisited**)

Input: $E = \sum_{u_{i+1}^m} W(y_k|v_k)\cdots W(y_i|v_i)\cdots W(y_m|v_m)$

Output: B_k, $\sum_{u_{i+1}^{j-1},u_{j+1}^m} W(y_{k+1}|x_{k+1})\cdots W(y_m|x_m)$, $\sum_{u_{i+1}^{j-1},u_{j+1}^m} W(y_{k+1}|x'_{k+1})\cdots W(y_m|x'_m)$

1: u_j = **firstTerm**(v_k); ▷ $i+1 \le j \le m$
2: **for** $l = k+1, k+2, \ldots, m$ **do**
3: **if** $u_j \lessdot v_l$ **then**
4: $x_l \leftarrow \underline{v_l + v_k}$;
5: $x'_l \leftarrow v_l + v_k$;
6: **else**
7: $x_l \leftarrow v_l$;
8: $x'_l \leftarrow v_l$;
9: **end if**
10: **end for**
11: **arrayVisited**[k] = **true**;
12: **return** B_k, $\sum_{u_{i+1}^{j-1},u_{j+1}^m} W(y_{k+1}|x_{k+1})\cdots W(y_m|x_m)$, $\sum_{u_{i+1}^{j-1},u_{j+1}^m} W(y_{k+1}|x'_{k+1})\cdots W(y_m|x'_m)$;

Then

$$E = \sum_{u_{i+1}^m}(W(y_k|v_k)M1_{v_k=0} + W(y_m|v_k)M1_{v_k=1}) \tag{28}$$

$$= W(y_k|0)W(y_{k+1}^m|0) + W(y_k|1)W(y_{k+1}^m|1) \tag{29}$$

$$= S(B_k \cdot (W(y_{k+1}^m|0), W(y_{k+1}^m|1))) \tag{30}$$

$$= S(B_k \cdot (\sum_{u_{i+1}^m} M \cdot 1_{v_k=0}, \sum_{u_{i+1}^m} M \cdot 1_{v_k=1})) \tag{31}$$

$$\overset{(a)}{=} S(B_k \cdot (\sum_{u_{i+1}^{j-1},u_{j+1}^m} W(y_{k+1}|x_{k+1})\cdots W(y_m|x_m) \tag{32}$$

$$, \sum_{u_{i+1}^{j-1},u_{j+1}^m} W(y_{k+1}|x'_{k+1})\cdots W(y_m|x'_m)), \tag{33}$$

where (a) follows from that: By substituting u_j by $u_j + v_k$ for v_l in each channel expression of M in (31) if $u_j \lessdot v_l$, the expression $\sum_{u_{i+1}^{j-1},u_{j+1}^m} W(y_{k+1}|x_{k+1})\cdots W(y_m|x_m)$ in (32) is obtained. Suppose $u_j \lessdot v_l$ and let $\grave{v}_l = u_j + v_l$. Then $v_l = u_j + \grave{v}_l$. Because of $1_{v_k=0}$, we just need to consider $v_k = 0$ in $M \cdot 1_{v_k=0}$. Therefore, we have $u_j = u_j + v_k$ ($v_k = 0$). Thus, we have $x_l \leftarrow u_j + \grave{v}_l = u_j + v_k + \grave{v}_l = v_l + v_k$. The expression $\sum_{u_{i+1}^{j-1},u_{j+1}^m} W(y_{k+1}|x'_{k+1})\cdots W(y_m|x'_m)$ in (33) can be obtained by the similar derivation. ■

Obtaining the first term of v_k is referred as the function u_j = *firstTerm*(v_k). The Algorithm 6 describes details of implementing the **fundamentalStep** function for an given expression. In Algorithm 6, the channel expression starts at k. This means the values of **arrayVisited**[0],. . .,**arrayVisited**[$k-1$] are supposed to true. The Lemma 2 shows the correctness of the **fundamentalStep** function.

The B_k in (27) is referred as a *local formula*. It should be noted that the channel expression $W(y_k|v_k)$ in (26) can be a combination channel expression.

D. HIGH-LEVEL FUNCTION

We now describe the high-level function of our algorithm, given in Algorithm 7. Recall the outline of the W-formula generating algorithm, each expression in the same level will be broken into two short expressions and the expressions in the last level correspond to sub-formulas. Therefore, a full binary tree is suitable for describing the W-formula generating algorithm. In Algorithm 7, the function **createBinaryTree**($m-i$) is used to create a full binary tree with level $m-i$ (level starts at 0, ends at $m-i-1$). For a node, we denote its level as l and its index in the current level as j. We have three undefined data structures in Algorithm 7. The first one is **arrayPointer**[l][j], which denotes the pointer to point to the node with index (l, j), the second one is $E_{(l,j)}$, which denotes the expression for the node with index (l, j), and the last one is $B_{(l,j)}$, which denotes the local formula for the node with index (l, j).

The level of the tree may be less than $m - i$ since the **oneVariableCombine** function may reduce the *dim* of an expression without breaking it into two short expressions.

$$E = \sum_{u_{i+1}^m} W(y_k|v_k)W(y_{k+1}|v_{k+1})\cdots W(y_m|v_m) \tag{26}$$

$$= S(B_k \cdot (\underbrace{\sum_{u_{i+1}^{j-1},u_{j+1}^m} W(y_{k+1}|x_{k+1})\cdots W(y_m|x_m)}_{\text{left part}}, \underbrace{\sum_{u_{i+1}^{j-1},u_{j+1}^m} W(y_{k+1}|x'_{k+1})\cdots W(y_m|x'_m)}_{\text{right part}})) \tag{27}$$

Algorithm 7 W-Formula Generating Algorithm, Main Loop

Input: A kernel matrix G_m, index i and $B_1, \cdots, B_m$
Output: **arrayPointer**[0], [0] ▷ a pointer of the root node
▷ Starts from $E_{(0,0)} \leftarrow (W_m^{(i)}(y_1^m, u_1^{i-1}|u_i) = \sum_{u_{i+1}^m} W(y_1|v_1)\cdots W(y_m|v_m))$

1: **arrayPointer**[0], [0] $\leftarrow$ **createBinaryTree**($m-i$);
2: $E_{(0,0)} \leftarrow$ **hidingKnownValues**($E_{(0,0)}$);
3: $E_{(0,0)} \leftarrow$ **standardExpressionTransform**($E_{(0,0)}$);
4: $dim \leftarrow dim_{E_{(0,0)}}$, $l \leftarrow 0$, **arrayVisited**[$0, \cdots, m-1$] $\leftarrow$ [**false**, $\cdots$, **false**];
5: **while** $dim > 1$ **do**
6: **for** $j = 0, 1, \ldots, 2^l - 1$ **do**
7: **arrayTempVisited** $\leftarrow$ **arrayVisited**;
8: $E_{(l,j)} \leftarrow$ **combineLikeTerms**($E_{(l,j)}$, **arrayTempVisited**);
9: **if** $dim_{E_{(l,j)}} == 1$ **then**
10: (∗**arrayPointer**[l][j]).*localFormula* $\leftarrow B_{(l,j)}$; ▷ Drop S from $E_{(l,j)}$
11: **end if**
12: **if** $dim_{E_{(l,j)}} > 1$ **then**
13: $(B_{(l,j)}, E_{(l+1,2j)}, E_{(l+1,2j+1)}) \leftarrow$ **fundamentalStep**($E_{(l,j)}$, **arrayTempVisited**);
14: (∗**arrayPointer**[l][j]).*localFormula* $\leftarrow B_{(l,j)}$;
15: **end if**
16: **end for**
17: **arrayVisited** $\leftarrow$ **arrayTempVisited**;
18: $l \leftarrow l + 1$;
19: $dim \leftarrow dim_{E_{(l,0)}}$;
20: **end while**
21: **return arrayPointer**[0], [0];

But, for simplicity, we still create a full binary tree with level $m-i$ and its actual level is clear in Algorithm 7.

In the line 10 of Algorithm 7, $B_{(l,j)}$ denotes a sub-formula because of $dim_{E_{(l,j)}} = 1$. In the line 13, $B_{(l,j)}$, $E_{(l+1,2j)}$, and $E_{(l+1,2j+1)}$ correspond to the outputs of Algorithm 6 as B_k, $\sum_{u_{i+1}^{j-1}, u_{j+1}^m} W(y_{k+1}|x_{k+1})\cdots W(y_m|x_m)$, and $\sum_{u_{i+1}^{j-1}, u_{j+1}^m} W(y_{k+1}|x'_{k+1})\cdots W(y_m|x'_m)$, respectively.

Each node in the tree has a data field *localFormula*, which is used to save the local formula (non-leaf node) or the sub-formula (leaf node) generated in the algorithm. The data structure of the node is given in Figure 3. The pointers *lChild* and *rChild* point to the node's left and right child nodes, respectively. For simplicity, we omit the assignment of pointers *lChild* and *rChild* for each node and just suppose that we have an pointer **arrayPointer**[l][j] to point to each node (l, j) in the tree. After the main loop finished, the *localFormula* of each node in the tree is assigned and the tree corresponds to the W-formula for a given bit channel $W_m^{(i)}(y_1^m, u_1^{i-1}|u_i)$.

E. EXAMPLE

In Figure 4, we use $W_7^{(4)}(y_1^7, u_1^3|u_4 = 0)$ to illustrate the W-formula generating algorithm. The kernel matrix is

$$G_7 = \begin{pmatrix} 1&0&0&0&0&0&0\\ 1&1&0&0&0&0&0\\ 1&0&1&0&0&0&0\\ 1&0&0&1&0&0&0\\ 1&1&1&0&1&0&0\\ 1&1&0&1&0&1&0\\ 1&0&1&1&0&0&1 \end{pmatrix}.$$

Based on the definition (3) and the hideKnownValues function, we obtain

$$E_{(0,0)} = \sum_{u_5^7} W(y_1|u_5+u_6+u_7)W(y_2|u_5+u_6)W(y_3|u_5 + u_7)W(y_4|u_6+u_7)W(y_5|u_5)W(y_6|u_6)W(y_7|u_7).$$

The W-formula in Figure 4 can be viewed as the following formula.

$$W_7^{(4)}(y_1^7|0) = S(B_1 \cdot (S(B_2 \cdot B_7 \cdot (B_4 \cdot B_5) \diamond (B_3 \cdot B_6)), S(B_2^{-1} \cdot B_7 \cdot (B_4 \cdot B_5^{-1}) \diamond (B_3^{-1} \cdot B_6)))).$$

Note that all S are dropped in the tree structure of the W-formula. Based on the hideKnownValues function, the W-formula of $W_7^{(4)}(y_1^7, u_1^3|0)$ is

$$W_7^{(4)}(y_1^7, u_1^3|0) = S(B_1^{(1-2(u_1+u_2+u_3))} \cdot (S(B_2^{(1-2u_2)} \cdot B_7 \cdot (B_4 \cdot B_5) \diamond (B_3^{(1-2u_3)} \cdot B_6)), S(B_2^{(2u_2-1)} \cdot B_7 \cdot (B_4 \cdot B_5^{-1}) \diamond (B_3^{(2u_3-1)} \cdot B_6)))).$$

V. COMPUTATION ON W-FORMULAS

In the previous section, we present the method to generate the W-formula for a given expression in the general case. Actually, there exist some special cases for the W-formula. In this section, we first present all the special cases for the W-formula. Then, we provide the procedures to compute the value of an expression based on its W-formula.

A. GENERAL CASE

The computation of the W-formula in general is described in Figure 5. Consider a non-leaf node v in the tree structure of an W-formula, its left child node and right child node are denoted as v_l and v_r, respectively. Suppose the computation result of local formula on v is B_v, the value which is passed by v_l is s_l, and the value which is passed by v_r is s_r. Then the value computed on v is $S(B_v \cdot (s_l, s_r))$. Now we consider a leaf node v. Then the value computed on v is $S(B_v)$. Recall the equation (27), these computation rules are easy to obtain.

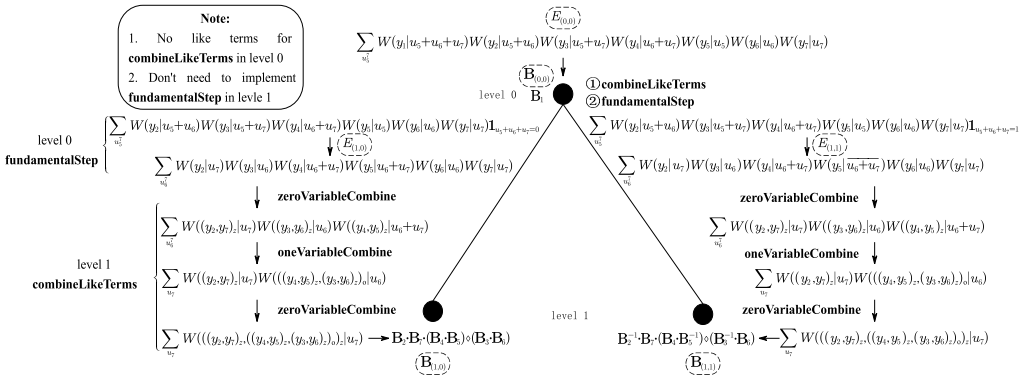


FIGURE 4. An example for generating W-formula.

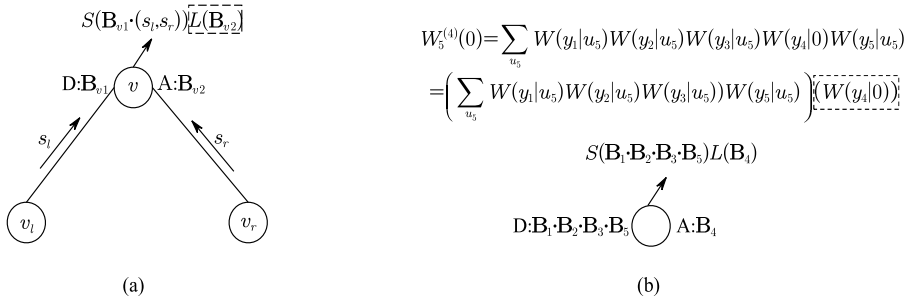


FIGURE 5. Explanation of letter D. (a) Computation on the non-leaf node. (b) Computation on the leaf node.

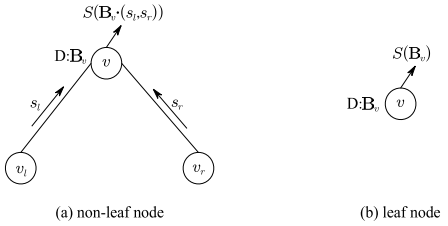


FIGURE 6. Explanation of letter A. (a) General case; (b) Example for $W_5^{(4)}(y_1^5, 0_1^4|0)$.

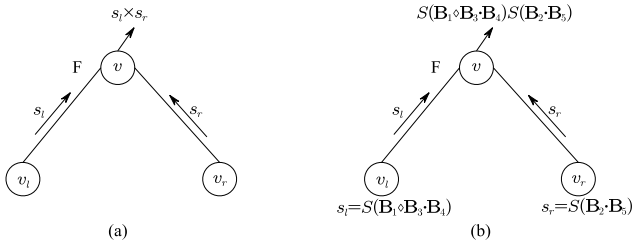


FIGURE 7. Explanation of letter F. (a) General case; (b) Example for $W_5^{(2)}(y_1^5, 0|0)$.

B. SPECIAL CASES

In Figure 5, we use a letter D to denote the local formula on the node v. In the following, we will introduce two letters A and F for a node v.

1) LETTER A

For some expressions, they may contain constant channel expressions such as $W(y_i|0)$ or $W(y_i|1)$. These constant channel expressions correspond to the letter A. For a node v, suppose that the computation result of its D part is a pair B_{v1}, the computation result of its A part is a pair B_{v2}, the value which is passed by its left child node is s_l, and the value which is passed by its right child node is s_r. Then the value computed at node v is $S(B_{v1} \cdot (s_l, s_r))L(B_{v2})$, which is passed to its parent node or the final result. The process on node v with letter D and A is described in Fig. 6(a). Two points about letter A should be noted: 1) the letter A only appears in the root node; 2) The letter A is not necessary.

An example is given in Fig. 6(b) to explain the process on a node v with letter D and A. The computation result of D part is $B_{v1} = B_1 \cdot B_2 \cdot B_3 \cdot B_5$ and the computation result of A part is $B_{v2} = B_4$. Therefore, the value computed at node v is $S(B_1 \cdot B_2 \cdot B_3 \cdot B_5)L(B_4)$.

2) LETTER F

For some expressions, they can be broken into two independent expressions. In this case, we introduce a letter F. A node v is denoted by a letter F and its two children nodes v_l and v_r correspond to the two independent expressions. For a node v,

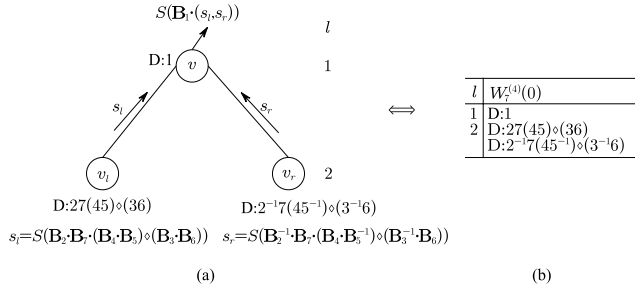


FIGURE 8. (a) W-formula for $W_7^{(4)}(0)$; (b) Simple form.

suppose the value which is passed by its left child node is s_l, and the value which is passed by its right child node is s_r. Then the value computed at node v is $s_l \times s_r$. The process on node v with letter F in general is described in Fig. 7(a) and the example for $W_5^{(2)}(y_1^5, 0|0)$ is given in Fig. 7(b).

VI. W-FORMULAS FOR SOME OPTIMAL KERNELS

In this section, we first introduce a *simple form* for the W-formulas in the tree structure. Second, we present the W-formulas in the simple form for several optimal kernels given in [6]. These W-formulas will reveal some useful properties of the computation reduction for the W-formula based SC decoding.

A. THE SIMPLE FORM

To express W-formulas more efficient (save space), we introduce a *simple* form for the W-formulas. The *simple* form describes the W-formula in a table. In the simple form, we use a number i ($1 \le i \le m$) to denote B_i and ij to denote $B_i \cdot B_j$. I.e., $27(45) \diamond (36)$ implies $B_2 \cdot B_7(B_4 \cdot B_5) \diamond (B_3 \cdot B_6)$. The D, A, and F parts for a node form a row in the table. Then these rows for nodes in the same level are arranged in column. For example, the W-formula of $W_7^{(4)}(0)$ and its simple form is described in Figure 8 (a) and (b), respectively.

B. EXAMPLES OF W-FORMULAS

We present W-formulas of optimal kernels G_7 [6] and the last level of $W_{10}^{(5)}(0)$ in the *simple* form in Examples VI-B and VI-B. In these tables, i is the ith bit channel and l is the level of the tree.

Example 1 (W-formulas for G_7):

i	l	$W_7^{(i)}(0)$	$W_7^{(i)}(1)$
1	0	D:1 ⋄ 2 ⋄ 3 ⋄ 4 ⋄ 5 ⋄ 67	1
2	0	F	
	1	D:1 ⋄ 3 ⋄ 47	1
		D:2 ⋄ 56	2
3	0	D:1 ⋄ 452 ⋄ 63 ⋄ 7	13
4	0	D:1	1
	1	D:$27(45) \diamond (36)$	4
		D:$2^{-1}7(45^{-1}) \diamond (3^{-1}6)$	4
5	0	D:$(14) \diamond (26)$37A:5	1235
6	0	D:1347A:256	1246
7	0	A:1234567	1347

Example 2 (The last level of $W_{10}^{(5)}(0)$):

l	$W_{10}^{(5)}(0)$
3	D:$(46) \diamond (78)59$
	D:$(46^{-1}) \diamond (7^{-1}8)59$
	D:$(4^{-1}6^{-1}) \diamond (78)59$
	D:$(4^{-1}6) \diamond (7^{-1}8)59$
	D:$(4^{-1}6) \diamond (7^{-1}8)59$
	D:$(4^{-1}6^{-1}) \diamond (78)59$
	D:$(46^{-1}) \diamond (7^{-1}8)59$
	D:$(46) \diamond (78)59$

VII. ADVANTAGES OF W-FORMULAS

Compared with the l-formulas, two factors make a significant reduction of computation for the W-formulas. One is the reduction on the number of sub-formulas and the other is the repetition of sub-formulas.

A. REDUCTION ON THE NUMBER OF SUB-FORMULAS

The number of sub-formulas of a W-formula (l-formula) is decided by the times of combining like terms in the W-formula (l-formula) generating algorithm. The more like terms are combined, the fewer sub-formulas are generated in the W-formula (l-formula). We will show that much more like terms can be combined in the W-formula generating algorithm than the l-formula generating algorithm.

First, let's see the definition of the sign between two expressions. We define the sign of $W(y_i|v_j)$ as $+$ and the sign of $W(y_i|\overline{v}_j)$ as $-$. The combination of channel expressions occurs between two channel expressions. We define *the sign of two channel expressions* as the multiplication of sign of each channel expression. The sign multiplication rules of two channel expressions are $+\times+=+, +\times-=-, -\times+=-$, and $-\times-=+$.

Now, we introduce a concept *channel consistency* for two channel expressions. In the l-formula generating algorithm, we consider the ratio of two expressions. Suppose the two expressions are E_0 and E_1. The *channel consistency* means that the two channel expressions in E_0 have the same sign with two corresponding channel expressions (at the same positions) in E_1.

In the l-formula generating algorithm, only if the two channel expressions satisfy the *channel consistency*, they can be combined. But, we don't need this restriction (*channel consistency*) in the W-formula generating algorithm because the expressions E_0 and E_1 are processed separately.

Let $L_l(m)$ and $L_W(m)$ denote the number of sub-formulas of $l_m^{(5)}$ and $W_m^{(5)}(0)$, respectively. Table 1 shows the number of sub-formulas comparison of $l_m^{(5)}$ and $W_m^{(5)}(0)$, $m = 9, \cdots, 16$. It can be seen that the W-formula method obtains a significant reduction on the number of sub-formulas over the l-formula method.

B. REPETITION OF SUB-FORMULAS

As we can see from Table 1, the W-formula achieves a significant complexity advantage over the l-formula. However,

TABLE 1. The number of sub-formulas of *l*-formula and *W*-formula for some optimal kernels: $L_l(m)$ vs. $L_W(m)$.

m	*error exponent*	$L_l(m)$	$L_W(m)$
9	0.4616	12	4
10	0.4691	27	8
11	0.4774	90	16
12	0.4921	261	16
13	0.4883	793	64
14	0.4941	2328	64
15	0.5006	6894	256
16	0.5182	21890	512

TABLE 2. The comparison of the number of sub-formulas for $W_m^{(5)}(0)$ between considering and not considering the repetition of sub-formulas for some optimal kernels: $L_W(m)$ vs. $L_{W_{\text{rep}}}(m)$.

m	*error exponent*	$L_W(m)$	$L_{W_{\text{rep}}}(m)$
9	0.4616	4	4
10	0.4691	8	4
11	0.4774	16	16
12	0.4921	16	16
13	0.4883	64	16
14	0.4941	64	32
15	0.5006	256	32
16	0.5182	512	32

it can be seen that the number of sub-formulas of $W_m^{(5)}(0)$ still grows fast with respect to kernel size m when $m \geq 12$. We will show that there are many repeated sub-formulas in the W-formula and their computation can be shared.

In the W-formula generating algorithm, an expression is broken into two short expressions by implementing **fundamentalStep** function of the W-formula. Unlike the **fundamentalStep** function of l-formula, we can always choose the same channel expressions in the **fundamentalStep** function of the W-formula for the two short expressions. This feature makes that each sub-formula contains the same number and order of B_is and the differences among them are the sign. We define the sign of B_i as $+$ and the sign of B_i^{-1} as $-$. The sign of a B_i in a sub-formula is decided by the sign change process of the ith channel expression $W(y_i|v_i)$ (from level i to $i+1$, the sign of $W(y_i|v_i)$ may or may not change) in each level in the W-formula generating procedure. Because the sign of a B_i has only two states, two sub-formulas can have the same sign for each B_i and they are the repetition of sub-formulas.

For example, considering the W-formula $W_{10}^{(5)}(0)$ in Example VI-B, it can be seen that the first and eighth, the second and seventh, the third and sixth, and the fourth and fifth sub-formulas are the same. The same sub-formulas just need to be computed once. Therefore, the number of sub-formulas of $W_{10}^{(5)}(0)$ which needs to be computed reduces from 8 to 4.

Let $L_W(m)$ and $L_{W_{\text{rep}}}(m)$ denote the number of sub-formulas of $W_m^{(5)}(0)$, which need to be computed, by not considering and considering the repetition of sub-formulas, respectively. Table 2 shows the comparison of $L_W(m)$ and $L_{W_{\text{rep}}}(m)$, for $m = 9, 10, \cdots, 16$. It can be seen that the number of sub-formulas of W-formulas which need to be computed are significantly reduced by considering repetition of sub-formulas for medium kernels.

TABLE 3. The average number of sub-formulas for optimal kernels up to size 16: $C_l(m)$ vs. $C_W(m)$.

m	*error exponent*	$\frac{2^{m+1}-2}{m}$	$C_l(m)$	$C_W(m)$
2	0.5000	3.0	1.0	2.0
3	0.4206	4.7	1.0	2.0
4	0.5000	7.5	1.0	2.0
5	0.4306	12.4	1.0	2.0
6	0.4513	21.0	1.2	2.0
7	0.4579	36.2	2.0	2.3
8	0.5000	63.7	3.6	2.8
9	0.4616	113.5	3.8	3.1
10	0.4691	204.6	6.0	3.4
11	0.4774	372.1	17.4	6.5
12	0.4921	682.5	48.5	8.0
13	0.4883	1260.1	94.9	11.1
14	0.4941	2340.4	278.5	17.1
15	0.5006	4368.9	792.5	21.9
16	0.5182	8191.8	2486.4	24.6

TABLE 4. The average number of operations for kernel computation: (a) the straightforward computation; (b) the *l*-formula method; (c) the *W*-formula method.

m	*error exponent*	(a)	(b)	(c)
2	0.5000	3.0	2.0	4.0
3	0.4206	9.3	3.7	8.7
4	0.5000	22.5	2.0	4.0
5	0.4306	49.6	7.8	19.2
6	0.4513	105.0	11.3	25.3
7	0.4579	217.7	21.9	33.7
8	0.5000	446.3	42.8	44.8
9	0.4616	908.4	52.1	56.7
10	0.4691	1841.4	83.8	67.4
11	0.4774	3721.8	246.0	118.5
12	0.4921	7505.5	673.8	171.0
13	0.4883	15121.8	1271.5	240.2
14	0.4941	30425.6	3790.6	372.0
15	0.5006	61165.1	10736.6	481.3
16	0.5182	122878.1	34145.0	630.1

C. COMPARISON OF THE NUMBER OF SUB-FORMULAS

Given a kernel G_m, let w_i denote the number of sub-formulas in the W-formulas of $W_N^{(i)}(0)$, $i = 1, \ldots, m$. Let $C_W(m)$ denote the average number of sub-formulas for a kernel G_m such as $C_W(m) = 2^{\frac{\sum_i w_i}{m}}$, where the factor 2 comes from the W-formulas of $W_N^{(i)}(1)$, $i = 1, \ldots, m$. Similarly, let

TABLE 5. The average number of operations of the kernel computation for some 16 × 16 kernels.

kernels	*error exponent*	*scaling exponent*	l-formula	W-formula
$\mathbf{G}_{16}$	0.51828	3.362	34145.0	630.1
$\mathbf{G}_{16}$_Trofimiuk	0.51828	3.346	39098.7	608.1
$\mathbf{G}_{16}$_Fazeli	0.51828	3.356	37408.3	664.6

$C_l(m)$ denote the average number of sub-formulas in the l-formulas of $\frac{W_N^{(i)}(0)}{W_N^{(i)}(1)}$, $i = 1, \ldots, m$. Table 3 presents the average number of sub-formulas comparisons for optimal kernels with size $m = 2, 3, \cdots, 16$. In Table 3, a column of $(2^{m+1} - 2)/m$ is added. This column is viewed as the average number of sub-formulas for the kernel computation of the straightforward SC decoding. For a sub-formula of W-formula, it contains $B_1, \ldots, B_m$ at most once. Consider the equation (3), for each realization of u_{i+1}^m, a term of (3) (right hand) contains $w(y_1|0)$ or $w(y_1|1)$, ..., $w(y_m|0)$ or $w(y_m|1)$ once. Therefore, a term of (3) is viewed as sub-formula like the W-formula. And the average number of terms in the $W_N^{(i)}(0)$ and $W_N^{(i)}(1)$, $i = 1, \ldots, m$ of (3) is $(2^{m+1} - 2)/m$. Table 3 shows that the W-formula method achieves a significantly improvement on the average number of sub-formulas for medium kernels.

VIII. COMPUTATION COST ANALYSIS OF W-FORMULAS

The complexity of W-formula or l-formula based SC decoding is proportional to the average number of sub-formulas. However, the number of operations involved in the sub-formula of W-formula and l-formula are different. In this section, we compare the W-formulas based SC decoding with the l-formulas based SC decoding with respect to the exact number of operations.

A. LOG DOMAIN

In practice, we will transfer the domain to logarithmic domain in the SC decoding to avoid the problem of processing very small numbers. Our simulations are done in the logarithmic domain. However, logarithm and exponential operations will be involved in the logarithmic domain, which makes complexity comparison more difficult. Therefore, the complexity comparison will be done in the non-logarithmic domain.

It should be noticed that the comparison results in the non-logarithmic domain can be directly generalized to the logarithmic domain. We count the number of multiplications in the non-logarithmic domain and the number of additions in the logarithmic domain. Almost all of the additions in the logarithmic domain comes from the multiplications in the non-logarithmic domain. Therefore, the comparison results in the non-logarithmic domain are proportional to the results in the logarithmic domain.

B. COMPARISON OF THE NUMBER OF OPERATIONS

Because the complexity of SC decoding is proportional to the computation of the kernel computations, we just consider the computation cost of the kernel computation. The number of multiplications in (3) dominates the computation cost of the kernel computation for the straightforward SC decoding. For the W-formula, a $\cdot$ operation contains two multiplications and one addition. A $\diamond$ operation contains four multiplications and two additions. For the l-formula, a $\cdot$ operation contains one multiplication and a $\diamond$ operation contains one multiplication, one division, and two additions. Because the number of multiplications is more than additions and the execution speed of multiplication is much lower than addition, we neglect additions in our analysis. Furthermore, we assume one division is equivalent to two multiplications to simplify the comparison.

In the W-formula method, we have to compute $W_N^{(i)}(1)$ as well as $W_N^{(i)}(0)$. Obviously, the number of operations needed for computing $W_N^{(i)}(1)$ is the same as $W_N^{(i)}(0)$. Although a lot of computations of computing $W_N^{(i)}(0)$ and $W_N^{(i)}(1)$ based on W-formulas can be shared (see examples in Section VI), we just count the number of operations of computing $W_N^{(i)}(0)$ and double it in Table 4.

The exact average number of operations (multiplications) of the kernel computation by straightforward computation, l-formula and W-formula are summarized in Table 4. It can be seen that the W-formula based SC decoding achieves a significant reduction of the number of operations compared with the straightforward SC decoding and the l-formula based SC decoding, for $m \geq 10$.

The exact average number of operations of the kernel computation for three 16×16 kernels are provided in Table 5. The G_{16} kernel is the linear kernel provided in [6]. The G_{16}_Fazeli and G_{16}_Trofimiuk kernels which are generated based on heuristic algorithm presented in [3] are provided in [3] and [9], respectively. It can be seen that the W-formula based SC decoding behaves almost the same in terms of the number of operations for three kernels. Again, the W-formula based SC decoding achieves a significant reduction of computations compared with the l-formula based SC decoding for three kernels.

The error performance of polar codes with medium kernels under non-approximate SC decoding can refer to our another paper [19].

IX. CONCLUSION

1) Discussion: The basic task of the W-formula method is to reduce the computational cost of (3). Actually, the computation of (3) is an instance of the classic marginalize product-of-functions (MPF) problem. The *factor graph* and the *sum-product* algorithm (message passing rules) are the

main method to solve the MPF problem [20]. As far as we know, there is no effective method for the exact computation of the MPF problem when the corresponding factor graph has cycles. Therefore, the W-formula method actually provides a way to effective compute the exact marginal of the MPF problem (its factor graph has cycles) at medium scale (the number of variables is less than 16).

2) Conclusion: Given a kernel G_m ($m \times m$ matrix), the computational cost of the *kernel calculation* of SC decoding is $O(2^m)$ in general. In order to reduce the computational cost of the kernel computation, this paper has proposed a W-formula method to obtain the simplified formulas (W-formulas) of the kernel computation. Our results show that the W-formulas yields a significant reduction of the computational cost of the kernel calculation for medium kernels ($m \leq 16$) by comparing with the existing non-approximate methods.

REFERENCES

[1] E. Arıkan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051–3073, Jul. 2009.

[2] S. B. Korada, E. Sasoglu, and R. Urbanke, "Polar codes: Characterization of exponent, bounds, and constructions," *IEEE Trans. Inf. Theory*, vol. 56, no. 12, pp. 6253–6264, Dec. 2010.

[3] A. Fazeli and A. Vardy, "On the scaling exponent of binary polarization kernels," in *Proc. 52nd Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Monticello, IL, USA, Sep. 2014, pp. 797–804.

[4] S. Hamed Hassani, K. Alishahi, and R. L. Urbanke, "Finite-length scaling for polar codes," *IEEE Trans. Inf. Theory*, vol. 60, no. 10, pp. 5875–5898, Oct. 2014.

[5] N. Presman, O. Shapira, S. Litsyn, T. Etzion, and A. Vardy, "Binary polarization kernels from code decompositions," *IEEE Trans. Inf. Theory*, vol. 61, no. 5, pp. 2227–2239, May 2015.

[6] H.-P. Lin, S. Lin, and K. A. S. Abdel-Ghaffar, "Linear and nonlinear binary kernels of polar codes of small dimensions with maximum exponents," *IEEE Trans. Inf. Theory*, vol. 61, no. 10, pp. 5253–5270, Oct. 2015.

[7] V. Miloslavskaya and P. Trifonov, "Sequential decoding of polar codes with arbitrary binary kernel," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Hobart, TAS, Australia, Nov. 2014, pp. 376–380.

[8] P. Trifonov, "Binary successive cancellation decoding of polar codes with Reed–Solomon kernel," in *Proc. IEEE Int. Symp. Inf. Theory*, Honolulu, HI, USA, Jun. 2014, pp. 1–5.

[9] G. Trofimiuk and P. Trifonov, "Efficient decoding of polar codes with some 16×16 kernels," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Guangzhou, China, Nov. 2018, pp. 1–5.

[10] G. Trofimiuk and P. Trifonov, "Window processing of binary polarization kernels," *IEEE Trans. Commun.*, vol. 69, no. 7, pp. 4294–4305, Jul. 2021.

[11] F. Abbasi and E. Viterbo, "Large kernel polar codes with efficient window decoding," *IEEE Trans. Veh. Technol.*, vol. 69, no. 11, pp. 14031–14036, Nov. 2020.

[12] P. Trifonov, "Recursive trellis processing of large polarization kernels," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Melbourne, VIC, Australia, Jul. 2021, pp. 2090–2095.

[13] B. Gupta, H. Yao, A. Fazeli, and A. Vardy, "Polar list decoding for large polarization kernels," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Madrid, Spain, Dec. 2021, pp. 1–6.

[14] S. Buzaglo, A. Fazeli, P. H. Siegel, V. Taranalli, and A. Vardy, "Permuted successive cancellation decoding for polar codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, San Francisco, CA, USA, Jun. 2017, pp. 2618–2622.

[15] S. Buzaglo, A. Fazeli, P. H. Siegel, V. Taranalli, and A. Vardy, "On efficient decoding of polar codes with large kernels," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW)*, Aachen, Germany, Mar. 2017, pp. 1–6.

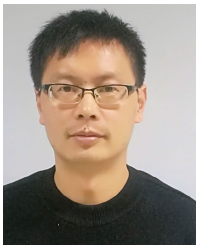
[16] G. Bonik, S. Goreinov, and N. Zamarashkin, "Construction and analysis of polar and concatenated polar codes: Practical approach," Jul. 2012, *arXiv:1207.4343*.

[17] X. Wang, Z. Zhang, and L. Zhang, "On the SC decoder for any polar code of length $N=l^n$," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, İstanbul, Turkey, Apr. 2014, pp. 485–489.

[18] Z. Huang, S. Zhang, F. Zhang, C. Duan, F. Zhong, and M. Chen, "Simplified successive cancellation decoding of polar codes with medium-dimensional binary kernels," *IEEE Access*, vol. 1, pp. 5253–5270, 2018.

[19] S. Zhang, Z. Huang, G. Chen, and M. Chen, "A fast two-phase Monte Carlo method for constructing polar codes with arbitrary binary kernel," *IEEE Access*, vol. 7, pp. 131609–131615, 2019.

[20] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.



ZHILIANG HUANG (Member, IEEE) received the B.S. degree from the School of Materials Science and Engineering, Wuhan Institute of Technology, in 2004, the M.S. degree from the College of Physical and Electric Information Engineering, Zhejiang Normal University, in 2009, and the Ph.D. degree from the School of Information Science and Engineering, Southeast University, in 2013. In 2015, he was a Visiting Researcher with Bilkent University, Ankara, Turkey. He is currently an Associate Professor with the School of Mathematics, Physics and Information Engineering, Zhejiang Normal University. His current research interests include modern coding theory and signal processing for digital communications.

ZONGSHENG JIANG received the B.S. degree from the School of Electronics and Information Engineering, Taizhou University, in 2019. He is currently pursuing the degree with the College of Mathematics and Computer Science, Zhejiang Normal University. His current research interest includes 5G polarization code improvement in modern coding theory.

SHUIHONG ZHOU received the B.S. degree from the School of Electronics and Information Engineering, Shenyang Ligong University, in 2006, and the M.S. degree from the School of Mathematics, Physics and Information Engineering, Zhejiang Normal University, in 2009. She is currently a Laboratory Master with the College of Mathematics and Computer Science, Zhejiang Normal University. Her current research interests include modern coding theory and wireless communications.

XIAOYAN ZHANG received the B.S. degree from the School of Electronics and Information Engineering, South-Central Minzu University, in 2002, the M.S. degree from the School of Communication Engineering, Hangzhou Dianzi University, in 2008, and the Ph.D. degree from the School of Electrical Engineering and Computer Science, Ningbo University, in 2019. In 2016, she was a Visiting Researcher with The Chinese University of Hong Kong. She is currently a Lecturer with the School of Mathematics, Physics and Information Engineering, Zhejiang Normal University. Her current research interests include modern coding theory and wireless communications.