Variants of Golomb Coding and the *n*-ary Versions

Na Wang, Sian-Jheng Lin, *Member, IEEE,* Yunghsiang S. Han, *Fellow, IEEE,* and Nenghai Yu

***Abstract*—Golomb coding is a type of entropy encoding scheme for geometric distributions. It consists of two parts, and both parts are coded with variable-length coding, which requires a higher computational effort than fixed-length coding schemes. To solve this issue, the first part of this paper presents a variant of Golomb coding that uses fixed-length coding to code the first part. The simulations show that the proposed coding scheme has a higher throughput than Golomb coding, due to the reduction of arithmetic complexity. In the second part, we discuss the n-ary versions of Golomb coding and the proposed coding scheme.**

***Index Terms*—Entropy encoding, Golomb coding, Geometric distribution.**

I. Introduction

With the advent of the information age, data compression has become an indispensable part of data storage and transmission due to the large amount of data and limited computer storage and network bandwidth. Most compression systems for images, speech and video use adaptive predictors or decorrelation transforms to map blocks of the original data into low-entropy blocks of integers to facilitate entropy coding [1]. Typical entropy encodings used in such systems include arithmetic coding [2], Huffman coding [3] and run-length coding. Among them, run-length coding is a coding scheme that records the number of successive symbols in a sequence. In particular, when its alphabet is independent and identically distributed (i.i.d.) with occurrence probability p, the output stream of run-length coding follows the geometric distribution defined as

$$P(i) = p(1-p)^{i-1}, \tag{1}$$

for $0 < p < 1$ and $i \geq 1$. Furthermore, the geometric distribution also arises in other problems, such as when encoding protocol information in data networks.

A sequence following a geometric distribution can be coded by Golomb coding [4]–[7]. Golomb coding is a variable-to-variable length coding scheme proposed by Golomb in the 1960s, and the code has been studied extensively for image processing [8], [9], data compression [10]–[13] and systems on a chip (SoCs) [14]–[16]. It has been proven optimal for integer sequences following a geometric distribution [17]. In other words, it approximates the coding efficiency of Huffman coding in such sources. However, compared to Huffman coding, the encoding/decoding of Golomb coding does not need to maintain a codebook, and the coding process is performed via integer arithmetic. These advantages make Golomb coding attractive, as integer arithmetic is usually much faster than memory accesses on modern processors.

The Golomb-Rice (GR) coding [18]–[21], introduced by Rice in 1979, is a subset of Golomb coding. Currently, GR coding is used in many audio/image formats, such as Shorten, FLAC, Apple Lossless, MPEG-4 ALS, and FELICS. Golomb coding has an adjustable parameter M that can be any positive integer, and GR code requires M to be a power of two. The requirement eases the implementations of GR coding since multiplication and division by a power of two can be easily implemented by logical operations. Golomb coding requires several integer division operations that can be replaced with bitwise operations in GR coding. Furthermore, Golomb coding consists of two parts: a quotient part and a remainder part. Golomb coding uses variable-length coding to code both parts. In contrast, GR coding uses fixed-length coding to code the remainder part.

The above two reasons show that, compared with GR coding, Golomb coding has inferior throughput when M is not a power of two. To solve this issue, we focus on the coding algorithms for M that are not a power of two. More precisely, this paper presents a class of prefix codes for geometric probability distributions. First, similar to GR coding, the proposed prefix codes use fixed-length coding to code the remainder part. Second, we show that the integer division operations in the encoding process can be replaced with integer multiplication operations.

A code is called n-ary code when each codeword symbol of it is in $\mathbb{Z}_n$. Although many prefix codes are binary $n = 2$, a number of n-ary prefix codes are proposed. For example, the variable-length quantity (VLQ) is a class of universal coding defined in the standard MIDI file format. The VLQ can also be seen as the 256-ary version of Exp-Golomb coding [22], [23], and thus, each input integer is converted into a byte (8 bits) to facilitate processing on modern computer systems. However, to the best of our knowledge, there is no literature exploring arbitrary n-ary Golomb coding. Thus, in the second part of this paper, we discuss the n-ary versions of Golomb coding and the proposed coding. The contributions of this work are summarized as follows.

1) A variant of Golomb coding is proposed. The Golomb coding uses variable-length coding to encode the quotient and remainder parts. In contrast, the proposed coding uses fixed-length coding to encode the remainder part and a variable-length coding to encode another.

Manuscript received December 29, 2019; revised March 5, 2020; accepted August 28, 2020. This work was supported in part by Hundred Talents Program of Chinese Academy of Sciences, Natural Science Foundation of Anhui Province (No. BJ2100330001), National Natural Science Foundation of China (Grant No. 61671007), and Start Fund of Dongguan University of Technology (No. KCYXM2017025). The associate editor coordinating the review of this article and approving it for publication was E. Rosnes. (*Corresponding author: Sian-Jheng Lin.*)

N. Wang, S.-J. Lin and N. Yu are with the school of Information Science and Technology, University of Science and Technology of China, Heifei, 230026, China (email:wn312991@mail.ustc.edu.cn; sjlin@ustc.edu.cn; ynh@ustc.edu.cn.)

Y.S. Han is with the School of Electrical Engineering & Intelligentization, Dongguan University of Technology, Dongguan, China (email:yunghsiangh@gmail.com.)

2) The simulations are presented. The proposed coding scheme involves approximately 20% fewer addition operations, 40% fewer multiplication operations and 20% more bitwise operations during encoding and 40% fewer addition operations, 10% fewer multiplication operations, 50% fewer bitwise operations and 20% more branch operations during decoding than Golomb coding.
3) The n-ary versions of Golomb coding and the proposed coding scheme are presented.

The remainder of the paper is organized as follows. Section II introduces Golomb coding, and Section III introduces the proposed prefix codes. Section IV presents the implementation considerations and the simulation results. Section V discusses the details of n-ary codes. Finally, Section VI concludes this work.

II. Preliminaries

A. Notation

Let $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ denote the ring of integers modulo n. Let $\lfloor x \rfloor$ denote the largest integer that is less than or equal to x. Let $Rem\,(i, j) = i - \left(\left\lfloor \frac{i}{j} \right\rfloor + s\right) j$, where $s = 0$ if $i \geq 0$; otherwise, $s = 1$. Note that $Rem\,(i, j) < 0$ when $i < 0$.[1] Let

$$\mathrm{NC}_j^n\,(i) = \left(i_{j-1}, \ldots, i_1, i_0\right) \tag{2}$$

denote the j-symbol n-ary natural code (NC) of $i = i_0 + i_1 \times n + \cdots + i_{j-1} \times n^{j-1}$, where $i_k \in \mathbb{Z}_n$. Given an n-ary stream S, we construct a FIFO queue accordingly. The operation

$$R \leftarrow \mathrm{Deque}_S^n\,(i)$$

removes i symbols $\{S_i\}_{i=0}^{i-1}$ from the front terminal position in the queue that contains S, and these i symbols form an integer $R = S_0 + S_1 \times n + \cdots + S_{i-1} \times n^{i-1}$. Notably, this paper uses the abbreviations $\mathrm{NC}_j\,(i)$ and $\mathrm{Deque}_S\,(i)$ when $n = 2$.

B. Golomb coding

Golomb coding is used to encode a sequence following a geometric distribution. Precisely, each symbol N of the sequence follows $P\,(N = i) = p\,(1-p)^i$, where $p \in (0, 1)$ and $i = 0, 1, 2, 3, \ldots$. During encoding, the input value N is divided by M to obtain the quotient and the remainder. Then, the quotient is coded by unary coding, and then, the remainder is encoded by truncated binary encoding. The parameter M can be determined by the inequality

$$(1-p)^M + (1-p)^{M+1} \leq 1 < (1-p)^{M-1} + (1-p)^M\,. \tag{3}$$

For a large M, there is very little penalty from selecting [5]

$$M = \left\lfloor \frac{-1}{\log_2\,(1-p)} \right\rfloor. \tag{4}$$

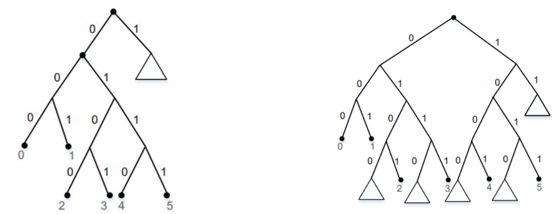
The details of encoding the input N are described below. Let

$$b = \lceil \log_2 M \rceil, \qquad t = 2^b - M. \tag{5}$$

[1]When $i \geq 0$ and $j > 0$, $Rem\,(i, j)$ is the ordinary remainder from integer division; however, when $i < 0$, it is not.

TABLE I
THE GOLOMB CODING AND THE PROPOSED CODING FOR $M = 6$

N	Golomb		Length	Proposed	
	Q	R		R	Q
0	0	00	3	000	
1	0	01	3	001	
2	0	100	4	010	1
3	0	101	4	011	1
4	0	110	4	100	1
5	0	111	4	101	1
6	10	00	4	110	1
7	10	01	4	111	1
8	10	100	5	010	01
9	10	101	5	011	01
10	10	110	5	100	01
11	10	111	5	101	01
12	110	00	5	110	01
13	110	01	5	111	01



(a) (b)

Fig. 1. Golomb coding tree and the proposed coding tree for $M = 6$: (a) the Golomb coding tree and (b) the proposed coding tree

1) N is divided by M to obtain the quotient $q = \left\lfloor \frac{N}{M} \right\rfloor$ and the remainder $r = \mathrm{Rem}\,(N, M)$.
2) Let $\langle QCode \rangle = (\underbrace{1, 1, \cdots, 1}_{q}, 0)$ denote the unary coding of q.
3) Let
$$\langle RCode \rangle = \begin{cases} \mathrm{NC}_{b-1}\,(r) & \text{if } r < t, \\ \mathrm{NC}_b\,(r+t) & \text{otherwise,} \end{cases} \tag{6}$$
denote the truncated binary encoding of r.
4) The codeword is given by $\langle QCode \rangle \langle RCode \rangle$.

When $M = 1$, Golomb coding is equivalent to unary coding. Furthermore, when $M = 2^b$ (i.e., when M is a power of two), it is known that the implementation can be further simplified [24]. First, Step 1) does not require integer division operations, and the results can be obtained via

$$\begin{aligned} q &= N \gg b, \\ r &= N \odot (M-1)\,, \end{aligned} \tag{7}$$

where $\gg$ denotes the bitwise right-shift operation and $\odot$ denotes the bitwise AND operation. Second, Step 3) does not require the conditional statement (if $r < t$), and

$$\langle RCode \rangle = \mathrm{NC}_b\,(r)\,. \tag{8}$$

To illustrate the above algorithm, Table I shows the Golomb coding for $M = 6$ on the left-hand side. Figure 1(a) shows the corresponding Golomb coding tree, and the value of N is labeled at the bottom. In this case, we have $b = \lceil \log_2 6 \rceil = 3$, and $t = 2^b - M = 2$. Thus, Table I includes t codewords of

TABLE II
THE GOLOMB CODING AND THE PROPOSED CODING FOR $M = 4$

N	Golomb Q	Golomb R	Length	Proposed R	Proposed Q
0	0	00	3	00	1
1	0	01	3	01	1
2	0	10	3	10	1
3	0	11	3	11	1
4	10	00	4	00	01
5	10	01	4	01	01
6	10	10	4	10	01
7	10	11	4	11	01
8	110	00	5	00	001
9	110	01	5	01	001
10	110	10	5	10	001
11	110	11	5	11	001

length 3, M codewords of length 4, M codewords of length 5, and so on. Furthermore, Table II shows the Golomb coding at $M = 4$ on the left-hand side. In this case, we have $b = \lceil \log_2 4 \rceil = 2$, and $t = 2^b - M = 0$. Thus, Table II includes M codewords of length 3, M codewords of length 4, and so on.

Next, we discuss the decoding of a code bitstream S, which is a sequence of concatenated codewords. The receiver first determines Q, that is, the number of successive ones before a zero occurs. Then, the receiver finds R, which has b or $b-1$ bits in S. The details are given below.

1) Decode $\langle QCode \rangle$:
   a) $Q \leftarrow 0$.
   b) Read a bit $a \leftarrow \mathrm{Deque}_S(1)$. If $a = 1$, $Q \leftarrow Q + 1$, and repeat this step. The repetition stops when a zero is read.
2) Decode $\langle RCode \rangle$:
   a) $R \leftarrow \mathrm{Deque}_S(b-1)$.
   b) If $R \geq t$, then $a \leftarrow \mathrm{Deque}_S(1)$, and
$$R \leftarrow (R \ll 1) + a - t, \tag{9}$$
   where $\ll$ denotes the bitwise left-shift operation.
3) The value is $N = Q \times M + R$.

Notably, when $M = 2^b$ (i.e., when M is a power of two), Step 2) directly reads b bits without the conditional statement, and the value is denoted by R. In addition, Step 3) calculates the value
$$N = (Q \ll b) + R \tag{10}$$
without integer multiplication operations.

C. The implementation of unsigned integer divisions with constant divisors

Integer division is very time consuming on modern CPUs and should be avoided as much as possible during implementation. It is known that unsigned division by a power of two can be implemented by a logical right-shift operation and a bitwise operation. When the divisor is a constant but not a power of two, [25] presents a method to calculate the quotient and the remainder with a multiplication operation and a shift operation. The basic idea is to multiply by a sort of reciprocal of the divisor d, such as $2^\ell/d$, and the quotient is obtained by right-shifting the value by ℓ bits.

Let us first consider the unsigned division of n by 3 on a 32-bit machine. The calculation steps are as follows:

1) Let $H = \left(2^{33}+1\right)/3$.
2) The quotient and the remainder are given by
$$q = \left\lfloor H \times n / 2^{33} \right\rfloor, \qquad r = n - q \times 3. \tag{11}$$

One can see that when $0 \leq n < 2^{32}$,
$$q = \left\lfloor \frac{2^{33}+1}{3} \frac{n}{2^{33}} \right\rfloor = \left\lfloor \frac{n}{3} + \frac{n}{3 \times 2^{33}} \right\rfloor = \left\lfloor \frac{n}{3} \right\rfloor.$$

Notably, calculating (11) may overflow when using 32-bit arithmetic. To solve this issue, the result $H \times n$ can be stored in a 64-bit integer type. For example, the C implementation of (11) is given by

```
uint32_t q=((uint64_t)H*(uint64_t)n)>>33.
```

The algorithm for unsigned division is described as follows. Given a word size $W \geq 1$ and a divisor d, $1 \leq d < 2^W$, the following provides a way to determine a pair of integers (m,u), $0 \leq m < 2^{W+1}$ and $u \geq W$, such that
$$\left\lfloor \frac{mn}{2^u} \right\rfloor = \left\lfloor \frac{n}{d} \right\rfloor, \tag{12}$$
for $0 \leq n < 2^W$.

1) u is the smallest integer such that
$$2^u > 2^{W-1}\left(d - 1 - Rem\left(2^u - 1, d\right)\right). \tag{13}$$
2) Then,
$$m = \frac{2^u + d - 1 - Rem\left(2^u - 1, d\right)}{d}. \tag{14}$$

III. Proposed code

When M is a power of two, GR codes give implementation modification to improve performance. First, the division and multiplication operations in Golomb coding can be replaced with the bitwise operations given in (7) and (10). Second, $\langle RCode \rangle$ can be encoded with fixed-length coding as given in (8). Golomb coding when $M \neq 2^b$ is then much slower than the coding when $M = 2^b$. In this section, a class of prefix codes for any M is proposed such that $\langle RCode \rangle$ can be encoded with fixed-length coding. First, a class of prefix codes is proposed to encode the remainder part with fixed-length encoding, rather than truncated binary encoding. Second, we show that the codeword lengths of the proposed coding scheme are equal to that of Golomb coding.

A. Code construction

First, the right-hand sides of Tables I and II show two examples of the proposed coding for $M = 6$ and $M = 4$. Figure 1(b) shows the proposed coding tree for $M = 6$. These tables show that the proposed coding scheme has the same codeword lengths as with Golomb coding. Tables I and II show that there are some differences between the two coding schemes. First, when $M = 6$, Golomb coding takes two or three bits to encode the remainder, and the proposed coding scheme always takes three bits to encode it. Second, Golomb coding first encodes the quotient, while the proposed coding scheme first encodes the remainder. Furthermore, one can verify that the codeword

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCOMM.2020.3022396, IEEE Transactions on Communications

Algorithm 1 Proposed encoding algorithm

Require: N and M.
Ensure: A codeword.
Let $b = \lceil \log_2 M \rceil$, $t = 2^b - M$
1: **if** $N < t$ **then**
2: $\langle RCode \rangle \leftarrow NC_b(N)$
3: **return** $\langle RCode \rangle$
4: **else**
5: The quotient $q \leftarrow \left\lfloor \frac{N-t}{M} \right\rfloor$
6: The remainder $r \leftarrow \mathrm{Rem}(N-t, M) + t$
7: $\langle RCode \rangle \leftarrow NC_b(r)$
8: $\langle QCode \rangle \leftarrow (\underbrace{0, 0, \cdots, 0}_{q}, 1)$
9: **return** $\langle RCode \rangle \langle QCode \rangle$
10: **end if**

Algorithm 2 Proposed decoding algorithm

Require: The code stream S, M
Ensure: N
Let $b = \lceil \log_2 M \rceil$, $t = 2^b - M$
1: **if** $S = null$ **then**
2: **return** $null$
3: **else**
4: $R \leftarrow \mathrm{Deque}_S(b)$
5: $Q \leftarrow 0$
6: **if** $R \geq t$ **then**
7: $k \leftarrow \mathrm{Deque}_S(1)$
8: **while** $k = 0$ **do**
9: $Q \leftarrow Q + 1$
10: $k \leftarrow \mathrm{Deque}_S(1)$
11: **end while**
12: **end if**
13: $N = R + Q \times M$
14: **return** N
15: **end if**

length is $\left\lfloor \frac{N-2}{6} \right\rfloor + 4$ in Table I, and the codeword length is $\left\lfloor \frac{N}{4} \right\rfloor + 3$ in Table II.

The details of encoding the input N are described below. Let $b = \lceil \log_2 M \rceil$, and $t = 2^b - M$; then, $N - t$ is divided by M to obtain the quotient $q = \left\lfloor \frac{N-t}{M} \right\rfloor$ if $N \geq t$, and the remainder $r = \mathrm{Rem}(N-t, M) + t$. The quotient q is encoded by unary coding, and the remainder r is encoded by a b-bit binary representation of r. Table I shows some encoding rules of the proposed coding scheme. First, the quotient is encoded by unary coding when $N \geq 2$. Second, the remainder is encoded by a b-bit binary code, and $b = \lceil \log_2 M \rceil = 3$. More precisely, the remainder is a b-bit binary representation of $Rem(N-2, 6) + 2$. Note that $Rem(i, j) < 0$ when $i < 0$. Algorithm 1 describes the proposed encoding procedure. In Algorithm 1, Lines 2–3 handle the case in which $N < t$, and the codeword contains $\langle RCode \rangle$ only. Lines 5–9 handle the case $N \geq t$. In particular, Lines 5–6 calculate the quotient and the remainder of $N - t$. Lines 7–9 generate the codeword.

Additionally, a number of implementation modifications can be applied to improve the throughput of Algorithm 1.

1) In Line 5, division with a constant M can be replaced with multiplication and a right-shift operation (see Section II-C for more details). That is, the instruction $q \leftarrow \left\lfloor \frac{N-t}{M} \right\rfloor$ can be replaced with
$$q \leftarrow m(N-t) \gg u, \tag{15}$$
where m and u are determined by (14) and (13), respectively. For example, when $M = 6$, the instruction $q \leftarrow \left\lfloor \frac{N-t}{6} \right\rfloor$ can be replaced with $q \leftarrow 3(N-t) \gg 4$.
2) In Line 6, the instruction can be replaced with
$$r \leftarrow N - q \times M, \tag{16}$$
considering that multiplication usually takes fewer CPU cycles than a modulus operation.
3) From (16), one can avoid multiplication when $q \in \{0, 1\}$, since
$$r = \begin{cases} N & \text{if } q = 0, \\ N - M & \text{if } q = 1, \\ N - q \times M & \text{otherwise.} \end{cases} \tag{17}$$

Notably, we cannot conclude that the implementation of (17) is always faster than (16), as (17) incurs an additional cost with the if-else statement. However, Lemma 1 shows that, in most cases of the proposed coding scheme, the multiplication operation in (17) is not performed. These cases include q is not calculated, $q = 0$, and $q = 1$.

Lemma 1. *$P(q \in \{null, 0, 1\}) > 1 - 2^{\frac{1-2M}{M+1}} \geq \frac{1}{2}$, where $q = null$ represents the case in which q does not need to be calculated.*

Proof. From (4), we have
$$M + \ell = \frac{-1}{\log_2(1-p)},$$
where $0 \leq \ell < 1$. Note that
$$\begin{aligned} 0 \leq t &= 2^b - M \\ &= 2^{\lceil \log_2 M \rceil} - M \\ &< 2^{1+\log_2 M} - M \\ &= 2M - M \\ &= M. \end{aligned}$$

From the probability mass function of the geometric distribution, we have
$$\begin{aligned} P(q \in \{null\}) &= P(N < t), \text{ and} \\ P(q \in \{0, 1\}) &= P(t \leq N < 2M + t). \end{aligned} \tag{18}$$

Thus,
$$\begin{aligned} P(q \in \{null, 0, 1\}) &= P(1 \leq N < 2M + t) \\ &= \sum_{N=1}^{2M+t-1} p(1-p)^{N-1} \\ &= 1 - (1-p)^{2M+t-1} \\ &= 1 - 2^{\frac{1-2M-t}{M+\ell}} \\ &> 1 - 2^{\frac{1-2M}{M}} \\ &\geq \frac{1}{2}. \end{aligned} \tag{19}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCOMM.2020.3022396, IEEE Transactions on Communications

□

Note that the lemma is true only when the encoder exactly knows the true distribution of the input; M is larger than or equal to one. Moreover, as M increases, the probability $P(q \in \{null, 0, 1\})$ also increases.

It is worth noting that when M is a power of two, the implementation can be further improved. First, as $t = 2^b - M = 0$, Lines 5–6 can be calculated via (7) without any integer division operations. Second, Line 1 does not need the conditional statement, since $t = 0$ and the statement "$N < t$" is always false.

Next, we discuss the decoding of a code bitstream S, which is a sequence of concatenated codewords. We first decode $\langle RCode \rangle$, which has $b = \lceil \log_2 M \rceil$ bits. Then, we decode $\langle QCode \rangle$, that is, the number of successive zeros before a one occurs. Algorithm 2 presents the details. In Algorithm 2, Line 4 reads b bits from the code bitstream S. If $R \geq t$, Lines 7–11 try to decode $\langle QCode \rangle$. Line 13 calculates the decoded value.

Furthermore, when $M = 2^b$ (i.e., when M is a power of two), Line 6 does not need the conditional statement, since $t = 0$ and the statement "$R \geq t$" is always true. Thus, we count the number of zeros preceding the first occurrence of a one, and the number is denoted by Q. In addition, Line 13 can be computed via (10) without any integer multiplication operations.

B. Code length

The following theorems show that the codeword lengths of Golomb coding and the proposed coding scheme are the same.

Theorem 1. *The codeword of N for Golomb coding is* $\left\lfloor \frac{N-(2^b-M)}{M} \right\rfloor + 1 + \lceil \log_2 M \rceil$ *bits.*

Proof. When $N < t = 2^b - M$, $q = 0$, and the length of the codeword is $1 + \lfloor \log_2 M \rfloor$. When $N \geq t$, we consider $N \in \left[2^b + (i-1) \times M, 2^b + i \times M\right)$ for an integer $i \geq 0$. The following discussion divides the interval into two cases.

1) When $N \in \left[2^b + (i-1) \times M, (i+1) \times M\right)$, we have $i = \left\lfloor \frac{N-(2^b-M)}{M} \right\rfloor$. In this case, the quotient's code is $q + 1 = i + 1$ bits, and the remainder's code is b bits. Therefore, the codeword is $(i+1) + b = \left\lfloor \frac{N-(2^b-M)}{M} \right\rfloor + 1 + \lceil \log_2 M \rceil$ bits.
2) When $N \in \left[(i+1) \times M, 2^b + i \times M\right)$, we have $i = \left\lfloor \frac{N-(2^b-M)}{M} \right\rfloor$. In this case, the quotient's code is $q + 1 = i + 2$ bits, and the remainder's code is $b - 1$ bits. Therefore, the codeword is $(i+2) + (b-1) = i + 1 + b = \left\lfloor \frac{N-(2^b-M)}{M} \right\rfloor + 1 + \lceil \log_2 M \rceil$ bits.

□

Theorem 2. *The codeword of N with the proposed coding scheme is* $\left\lfloor \frac{N-(2^b-M)}{M} \right\rfloor + 1 + \lceil \log_2 M \rceil$ *bits.*

Proof. In the proposed coding scheme, the remainder is coded with $b = \lceil \log_2 M \rceil$ bits. As the quotient's codeword is null when $N < 2^b - M$, the following discusses the codeword lengths in two cases.

1) When $0 \leq N < 2^b - M$, the quotient's codeword is null, and the codeword is $\lceil \log_2 M \rceil = 1 + \lfloor \log_2 M \rfloor$ bits, which is equal to that of Golomb coding in the same range.
2) When $N \geq 2^b - M$, we have $q = \left\lfloor \frac{N-(2^b-M)}{M} \right\rfloor$, and the remainder's code is $\lceil \log_2 M \rceil$ bits. Therefore, the codeword is $\left\lfloor \frac{N-(2^b-M)}{M} \right\rfloor + 1 + \lceil \log_2 M \rceil$ bits.

One can see that $\left\lfloor \frac{N-(2^b-M)}{M} \right\rfloor + 1 + \lceil \log_2 M \rceil = 1 + \lfloor \log_2 M \rfloor$ when $0 \leq N < 2^b - M$. Thus, the codeword length is $\left\lfloor \frac{N-(2^b-M)}{M} \right\rfloor + 1 + \lceil \log_2 M \rceil$ for $N \geq 0$. □

IV. Simulation

In this section, we first give some instances to show the number of arithmetic operations used in Golomb coding and the proposed coding scheme. Then, we show the simulations of both coding schemes. Finally, the results are discussed.

A. Arithmetic complexities

Due to the branches used during coding, it is difficult to give the formulas for the exact complexities of Golomb coding and the proposed coding scheme. Instead, we give the average number of arithmetic operations for $M = 2, 3, \ldots, 32$. We implemented the proposed coding scheme and Golomb coding in C and compiled with GCC 7.4.0 with optimization level -O3. These programs are tested on a platform equipped with an Intel(R) Core(TM) i7-7700K CPU @ 3.60 GHz and 8 GB main memory on an Ubuntu 16.04 operating system. The input sequence (1) is generated by gsl_ran_geometric() in the GNU scientific library (GSL) 2.4, where

$$p = 1 - 2^{\frac{-1}{M+0.5}} \tag{20}$$

is deduced from (4).

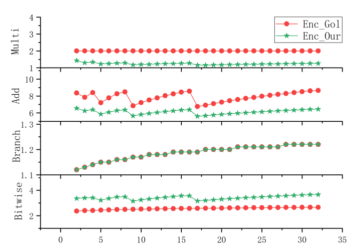


Fig. 2. The average number of arithmetic operations in Golomb coding and the proposed coding scheme during encoding for $M = 2, 3, \ldots, 32$. For simplicity, we abbreviate Addition and Multiplication as Add and Multi, respectively

We first encode a sequence of 2^{11} integers following distribution (20); then, we count the number of operations required in coding this sequence in running time. For each operation $\Gamma \in$ {Addition, Multiplication, Branch, Bitwise}, the average number of operations required by an integer is defined as

$$\Gamma/\text{integer} = \frac{\text{The total times of } \Gamma \text{ executed}}{\text{The number of integers in the sequence}}.$$

Table III lists the average number of operations used in each symbol for $M = 12, 16$. Notably, the term Addition counts the

TABLE III
NUMBER OF OPERATIONS IN GOLOMB CODING AND THE PROPOSED CODING SCHEME

Operations/integer (Enc./Dec.)	$M = 16$				$M = 12$	
	Gol_A	Our_A	Gol_B	Our_B	Gol_A	Our_A
Addition	8.57/11.77	6.38/6.77	6.57/6.77	5.57/6.77	7.78/10.59	6.05/6.12
Multiplication	2/1	1.28/1	0/0	0/0	2/1	1.23/0.85
Branch	1.19/1.19	1.19/1.19	0.19/0.19	0.19/0.19	1.18/0.80	1.18/1.03
Bitwise	2.57/7.19	3.57/3.19	4.57/5.19	4.57/4.19	2.54/6.05	3.39/3.03

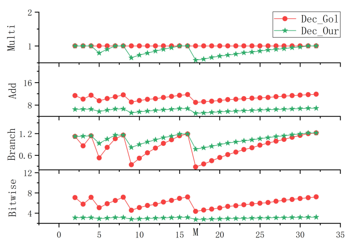


Fig. 3. The average number of arithmetic operations in Golomb coding and the proposed coding scheme during decoding for $M = 2, 3, \ldots, 32$

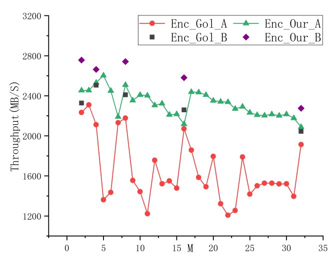


Fig. 4. Encoding performance of Golomb coding and the proposed coding scheme for $M = 2, 3, \ldots, 32$

number of + and − operations, the term Multiplication counts the number of × operations, and the term Bitwise counts the number of logical operations $\ll$, $\gg$, & and ~ used in the algorithms. It can be seen that fewer arithmetic operations are used in the proposed coding scheme than in Golomb coding. Additionally, Figures 2–3 show the average number of arithmetic operations required by an integer in both coding schemes during encoding and decoding, respectively. The simulation shows that the proposed coding scheme involves approximately 20% fewer addition operations, 40% fewer multiplication operations and 20% more bitwise operations during encoding and 40% fewer addition operations, 10% fewer multiplication operations, 50% fewer bitwise operations and 20% more branch operations during decoding than Golomb coding.

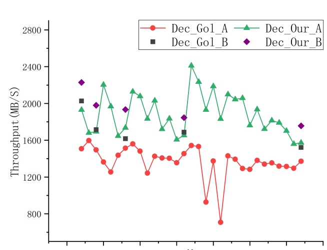


Fig. 5. Decoding performance of Golomb coding and the proposed coding scheme for $M = 2, 3, \ldots, 32$

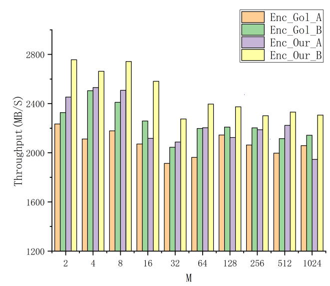


Fig. 6. Encoding performance of Golomb coding and the proposed coding scheme when M is a power of two

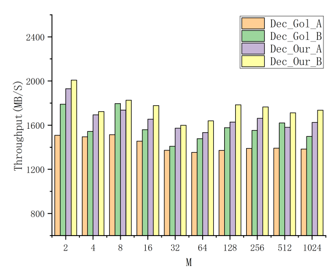


Fig. 7. Decoding performance of Golomb coding and the proposed coding scheme when M is a power of two

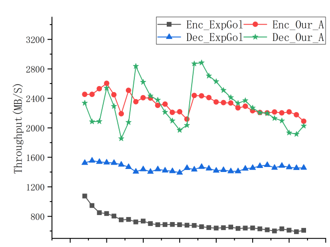


Fig. 8. Performance of Exp-Golomb coding and the proposed coding scheme for $M = 2, 3, \ldots, 32$

B. Throughput

This subsection shows the simulations of Golomb coding and the proposed coding scheme. We implemented both coding schemes in C, and each code has two implementations, A and B, where implementation A is for any values of M, and implementation B is only for values of M that are a power of two. In particular, implementation B adopts the improvement given in (7) and (10). The input sequence follows the geometric distribution given in (20). [2] Figure 4 and Figure 5 show the throughput of two implementations A and B during encoding and decoding, respectively. The throughput is defined as

$$\text{Throughput} = \frac{\text{Size of input data (MB)}}{\text{Time consumed (Second)}},$$

where MB stands for a megabyte $\left(8 \times 2^{20} \text{ bits}\right)$.

The simulation shows that the proposed coding has up to a 30% (30%) better throughput than Golomb coding during encoding (decoding). Additionally, Figure 6 and Figure 7 show the throughput during encoding and decoding, respectively, for $M = 2, 4, \ldots, 1024$. As implementation B uses the optimization tricks given in (7) and (10), its throughput is better than that of implementation A. In addition, Figure 8 shows the throughputs of Exp-Golomb coding and the proposed coding scheme during encoding and decoding for $M = 2, 3, \ldots, 32$. It can be seen that the proposed coding scheme has up to a 70% (35%) better throughput than Exp-Golomb coding during encoding (decoding). Notably, the CPU throughput measurements may vary across machines since they depend on many CPU behaviors, such as CPU cache or register transfers.

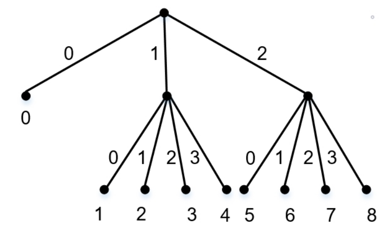
C. Discussion

Due to the fixed-length coding and the optimization mechanism given in (15)–(17), the proposed coding scheme has higher throughput than Golomb coding in Figure 4 and Figure 5. As shown in Figure 4 and Figure 5, when M is not a power of two, the throughput of the proposed coding scheme is much better than that of Golomb coding during both encoding and decoding, which is consistent with the results listed in Table III.

Next, we discuss the peaks occurring in Figure 4 and Figure 5 when M is a power of two. For Golomb coding in Figure 4, we have $t = 2^b - M = 0$ for an M that is a power of two; thus, the second branch in (6) is always executed. Although in Figure 2, Golomb coding performs more arithmetic operations when M is a power of two than when M is not a power of two. Due to the branch prediction technique used in CPU design that attempts to guess the outcome of a conditional operation and prepare for the most likely result, the local maximum appears in "Enc_Gol_A". In Figure 5, (9) is performed when M a power of two, and this operation gives the local minima in "Dec_Gol_A". In the proposed coding, Lines 6–12 in Algorithm 2 are performed when M is a power of two, and this operation gives the local minima in "Dec_Our_A".

Finally, we discuss the performances in Figure 6 and Figure 7. In our C implementation, "Enc_Our_B" has one fewer subtraction operation than "Enc_Gol_B", and "Dec_Our_B" has one fewer bitwise_not operation than "Dec_Gol_B". In addition, although q and r in "Enc_Our_A" require more operations than "Enc_Gol_B", the performances of both codes are close due to the help of the optimization mechanisms presented in Section III-A.

[2]The source code is available at https://github.com/wn312991/VariantsofGolombCoding.git.



(a)

r	Code(Value)
0	0 (0)
1	10 (4)
2	11 (5)
3	12 (6)
4	13 (7)
5	20 (8)
6	21 (9)
7	22 (10)
8	23 (11)

(b)

Fig. 9. Truncated 4-ary encoding when $M = 9$, where r is the integer to encode: (a) the coding tree, and (b) the codebook

V. n-ARY CODING

The n-ary code is a class of codes where each codeword symbol is in $\mathbb{Z}_n$. This section discusses the n-ary version of Golomb coding and the proposed coding scheme. Both n-ary versions use a parameter M to divide the input value into two parts, termed the quotient Q and the remainder R. Then, both parts are encoded into n-ary codes. We first introduce truncated n-ary encoding. Then, n-ary Golomb coding is proposed, and the coding efficiency is analyzed. Finally, the n-ary version of the proposed coding scheme is provided, and the compression performance is discussed.

A. Truncated n-ary encoding

The conventional Golomb coding scheme encodes the remainder part with truncated binary encoding. Therefore, this subsection presents the truncated n-ary encoding scheme that will be used in n-ary Golomb coding.

The coding tree of truncated n-ary encoding possesses the following properties. First, the root has $n-1$ children (the reason is explained in the later subsection), and other internal nodes have n children. Second, the tree is completely filled on every level except for the last level, and all the nodes in the last level are as far to the right as possible. With the above definitions, the generated codewords have $b-1$ or b symbols, where b is the height of the tree. The following shows that the number of leaves is a multiple of $n-1$. That is,

$$M = k \times (n-1), \tag{21}$$

for $k \in \mathbb{N}$. First, if the tree does not have any interval nodes, the tree has $n-1$ leaves by the definition. Second, if a leaf is replaced by an internal node, the tree will increase by $n-1$ leaves. This completes the proof. Figure 9(a) shows the 4-ary coding tree with $M = 9$ leaves, and the value of the remainder r is labeled at the bottom. Figure 9(b) lists the codewords $a_0 a_1$ for $r = 0, 1, \ldots, 8$, and the integers in parentheses denote the corresponding decimal values $4a_0 + a_1$.

Theorem 3. *The coding tree of truncated n-ary encoding with $M = k \times (n-1)$ leaves possesses the following properties.*

1) *The height of the tree is $b = \lceil \log_n k \rceil + 1$.*
2) *The tree has $t = n^{b-1} - k$ leaves in the $(b-1)$-th layer.*

Proof. A coding tree with height b has at most $(n-1) \times n^{b-1}$ leaves. Then, we obtain the inequality

$$\begin{aligned} &(n-1) \times n^{b-2} < M \leq (n-1) \times n^{b-1} \\ \Rightarrow &1 + \log_n k \leq b < 2 + \log_n k \\ \Rightarrow &b = \lceil \log_n k \rceil + 1. \end{aligned} \tag{22}$$

Next, we consider t. Note that the $(b-1)$-th layer has $(n-1) \times n^{b-2} - t$ internal nodes. Then, we have

$$\begin{aligned} &n \times \left((n-1) \times n^{b-2} - t\right) + t = M \\ \Rightarrow &t = n^{b-1} - k. \end{aligned} \tag{23}$$

□

As shown in Figure 9(b), the codeword table can be divided into the upper and the lower parts. The upper part covers the t codewords of size $b-1$, and the lower part covers $M - t$ codewords of size b. When the input $N < t$, the codeword is the n-ary representation of N and is encoded with $(b-1)$ symbols. Otherwise, the codeword is the n-ary representation of $N + t \times (n-1)$ and is encoded with b symbols. It is worth noting that when $t = 0$, the codeword is always represented by b symbols. Given $N \in \mathbb{Z}_M$, the encoding can be written as a function

$$\mathcal{T}_{n,k}(N) = \begin{cases} \mathrm{NC}_{b-1}^{n}(N) & \text{if } N < t, \\ \mathrm{NC}_{b}^{n}(N + t \times (n-1)) & \text{otherwise.} \end{cases} \tag{24}$$

In decoding, we first read $(b-1)$ symbols, which form a number R. If $R < t$, this step corresponds to the upper part of the codebook, and the decoded N is R. Otherwise, we should read one more symbol a from the code bitstream. Then, N is the value R subtracted by $t \times (n-1)$. The decoding function

$$\mathcal{T}_{n,k}^{-1} : \mathbb{Z}_n^b \rightarrow \mathbb{Z}_M \tag{25}$$

is defined as follows.

1) $R \leftarrow \mathrm{Deque}_S^n(b-1)$.
2) If $R \geq t$, $a \leftarrow \mathrm{Deque}_S^n(1)$, and
$$R \leftarrow R \times n + a - t \times (n-1).$$
3) Return R.

B. n-ary Golomb coding

The n-ary Golomb coding scheme requires a parameter M, that is, the number of leaves of the coding tree defined in (21). The coefficients b and t are defined in Theorem 3. Given the input value N, the encoding steps are as follows.

1) N is divided by M to obtain the quotient $q = \lfloor N/M \rfloor$ and the remainder $r = Rem(N, M)$.
2) Let $\langle QCode \rangle = \underbrace{(n-1, n-1, \cdots, n-1)}_{q}$, where $n - 1 \in \mathbb{Z}_n$.
3) Let $\langle RCode \rangle = \mathcal{T}_{n,k}(r)$ be defined in (24).
4) The codeword is given by $\langle QCode \rangle \langle RCode \rangle$.

TABLE IV
THE GOLOMB CODING SCHEME AND THE PROPOSED CODING SCHEME FOR $(n, M) = (4, 6)$

N	Golomb Q	Golomb R	Length	Proposed R	Proposed Q
0		0	1	0	
1		1	1	1	
2		20	2	2	1
3		21	2	3	1
4		22	2	2	2
5		23	2	3	2
6	3	0	2	2	3
7	3	1	2	3	3
8	3	20	3	2	01
9	3	21	3	3	01
10	3	22	3	2	02
11	3	23	3	3	02
12	33	0	3	2	03
13	33	1	3	3	03

The n-ary Golomb coding scheme consists of two parts: $\langle QCode \rangle$ and $\langle RCode \rangle$, where $\langle QCode \rangle = (n-1, \ldots, n-1)$ consists of a series of values $n-1 \in \mathbb{Z}_n$. To distinguish the two parts during decoding, the first symbol of $\langle RCode \rangle$ should not be $n-1$. Thus, we require that the root of the coding tree of truncated n-ary encoding scheme has only $n-1$ children.

To illustrate the above description, Table IV shows the n-ary Golomb coding scheme at $n = 4, M = 6$ on the left-hand side. In this case, we have $b = \lceil \log_4 2 \rceil + 1 = 2$ and $t = 4^{2-1} - 2 = 2$. Thus, Table IV shows t codewords of length 1, followed by M codewords of length 2, M codewords of length 3, and so on. It is worth noting that we can obtain the conventional Golomb coding scheme by letting $n = 2$.

During decoding, we first decode Q, which is the number of successive $(n-1)$s in the code. Then, we decode R, which has b or $b-1$ symbols in the code bitstream. The details are given below.

1) Decode $\langle QCode \rangle$:
   a) $Q \leftarrow 0$.
   b) Read symbol $a \leftarrow \mathrm{Deque}_S^n(1)$. If $a = n-1$, $Q \leftarrow Q + 1$, and repeat this step. The repetition stops until the symbol read is not $n-1$.
2) $\langle RCode \rangle = \mathcal{T}_{n,k}^{-1}(S)$ defined in (25).
3) The value is $N = Q \times M + R$.

C. Analysis

First, we analyze the best value of M for a geometric distribution. From (21), M is determined by k; thus, we first discuss the codeword length of Golomb coding and then discuss the value of k in the following theorems.

Theorem 4. *The codeword length for encoding N with the n-ary Golomb coding scheme is*

$$Length = \begin{cases} \left\lfloor \frac{N}{M} \right\rfloor + b & \text{if } t = 0, \\ \left\lfloor \frac{N}{M} \right\rfloor + \left\lfloor \frac{\mathrm{Rem}(N,M)}{n^{b-1}-k} \right\rfloor + b - 1 & \text{otherwise.} \end{cases} \tag{26}$$

Proof. In n-ary Golomb coding, the length of $\langle QCode \rangle = \underbrace{(n-1, n-1, \cdots, n-1)}_{q}$ is $q = \left\lfloor \frac{N}{M} \right\rfloor$. From Section V-A,

$\langle RCode\rangle$ always has b symbols when $t = 0$; $\langle RCode\rangle$ has $b-1$ symbols if $\mathrm{Rem}(N, M) < t$; otherwise, it has b symbols. The codeword length of $\langle RCode\rangle$ can be written as $\left\lfloor \frac{\mathrm{Rem}(N,M)}{t} \right\rfloor + b - 1$. In summary, the codeword length of n-ary Golomb coding is given by (26). □

One can verify that the result of Theorem 4 for $n = 2$ is the same as the results of Theorems 1 and 2.

Theorem 5. *The average codeword length of n-ary Golomb coding for a geometric distribution is given in* (27).

Proof. From Theorem 4, the average codeword length is discussed in the following.

1) When $t = 0$, we have

$$\begin{aligned}
&L_1(n,p,k)\\
&=\sum_{j=0}^{\infty}\left(\left\lfloor \frac{j}{M}\right\rfloor + b\right) p(1-p)^j\\
&=\sum_{j=0}^{\infty} bp(1-p)^j + \sum_{j=0}^{\infty}\left\lfloor \frac{j}{M}\right\rfloor p(1-p)^j\\
&=b + \sum_{j=0}^{\infty}\left\lfloor \frac{j}{M}\right\rfloor p(1-p)^j\\
&=b + \sum_{h=0}^{\infty}\sum_{j=hM}^{(h+1)M-1} hp(1-p)^j\\
&=b + \left(1-(1-p)^M\right)\sum_{h=0}^{\infty} h(1-p)^{hM}\\
&=b + \frac{(1-p)^M}{1-(1-p)^M}\\
&=\lceil \log_n k\rceil + 1 + \frac{(1-p)^{k(n-1)}}{1-(1-p)^{k(n-1)}}.
\end{aligned}$$

2) When $t > 0$, we have

$$\begin{aligned}
&L_1(n,p,k)\\
&=\sum_{j=0}^{\infty}\left(\left\lfloor \frac{j}{M}\right\rfloor + \left\lfloor \frac{\mathrm{Rem}(j,M)}{n^{b-1}-k}\right\rfloor + b - 1\right) p(1-p)^j\\
&=\frac{(1-p)^M}{1-(1-p)^M} + b - 1 + \sum_{j=0}^{\infty}\left\lfloor \frac{\mathrm{Rem}(j,M)}{n^{b-1}-k}\right\rfloor p(1-p)^j.
\end{aligned} \tag{28}$$

Let

$$T = n^{b-1} - k, \qquad G = \left\lfloor \frac{M-1}{T}\right\rfloor. \tag{29}$$

The term in (28) is derived by

$$\begin{aligned}
&\sum_{j=0}^{\infty}\left\lfloor \frac{\mathrm{Rem}(j,M)}{T}\right\rfloor p(1-p)^j\\
&=\sum_{L=0}^{M-1}\sum_{Z=0}^{\infty}\left\lfloor \frac{\mathrm{Rem}(L+ZM,M)}{T}\right\rfloor p(1-p)^{L+ZM}\\
&=p\sum_{L=T}^{M-1}\sum_{Z=0}^{\infty}\left\lfloor \frac{L}{T}\right\rfloor (1-p)^{L+ZM}\\
&=p\sum_{L=T}^{M-1}\left\lfloor \frac{L}{T}\right\rfloor \frac{(1-p)^L}{1-(1-p)^M}\\
&=\frac{p}{1-(1-p)^M}\left(\sum_{x=1}^{G-1} x \sum_{L=Tx}^{T(x+1)-1}(1-p)^L\right.\\
&\quad\left. +G\sum_{L=GT}^{M-1}(1-p)^L\right)\\
&=\frac{1}{1-(1-p)^M}\left(\sum_{x=1}^{G-1} x\left((1-p)^{Tx}-(1-p)^{T(x+1)}\right)\right.\\
&\quad\left. +G(1-p)^{TG} - G(1-p)^M\right)\\
&=\frac{1}{1-(1-p)^M}\left(\sum_{x=1}^{G-1}(1-p)^{Tx} - (G-1)(1-p)^{TG}\right.\\
&\quad\left. +G(1-p)^{TG} - G(1-p)^M\right)\\
&=\frac{1}{1-(1-p)^M}\left(\sum_{x=1}^{G}(1-p)^{Tx} - G(1-p)^M\right)\\
&=\frac{(1-p)^T\left(1-(1-p)^{TG}\right)}{\left(1-(1-p)^M\right)\left(1-(1-p)^T\right)} - \frac{G(1-p)^M}{1-(1-p)^M}.
\end{aligned}$$

Then, we have

$$\begin{aligned}
&L_1(n,p,k)\\
&=\frac{(1-p)^M}{1-(1-p)^M} + b - 1 + \sum_{j=0}^{\infty}\left\lfloor \frac{\mathrm{Rem}(j,M)}{n^{b-1}-k}\right\rfloor p(1-p)^j\\
&=\frac{(1-p)^T\left(1-(1-p)^{TG}\right)}{\left(1-(1-p)^M\right)\left(1-(1-p)^T\right)} + b - 1\\
&\quad+\frac{(1-G)(1-p)^M}{1-(1-p)^M}\\
&=\frac{(1-p)^{n^{\lceil \log_n k\rceil}-k}\left(1-(1-p)^{\left(n^{\lceil \log_n k\rceil}-k\right)\left\lfloor \frac{k(n-1)-1}{n^{\lceil \log_n k\rceil}-k}\right\rfloor}\right)}{\left(1-(1-p)^{k(n-1)}\right)\left(1-(1-p)^{n^{\lceil \log_n k\rceil}-k}\right)}\\
&\quad+\lceil \log_n k\rceil + \frac{\left(1-\left\lfloor \frac{k(n-1)-1}{n^{\lceil \log_n k\rceil}-k}\right\rfloor\right)(1-p)^{k(n-1)}}{1-(1-p)^{k(n-1)}}.
\end{aligned}$$

□

TABLE V
THE OPTIMAL VALUE OF k FOR A CERTAIN (n,p) IN (27)

$n\backslash p$	2^{-1}	4^{-1}	8^{-1}	16^{-1}	32^{-1}	64^{-1}
2	1	2	4	8	21	42
3	1	3	3	9	13	35
4	1	1	4	6	16	25
5	1	1	5	7	8	25
6	1	1	2	6	10	36
7	1	1	2	7	10	12
8	1	1	2	8	10	14
9	1	1	2	9	9	16
10	1	1	2	2	10	16

$$L_1(n,p,k)=\begin{cases}\lceil \log_n k\rceil+1+\frac{(1-p)^{k(n-1)}}{1-(1-p)^{k(n-1)}} & \text{if } t=0,\\ \frac{(1-p)^{n^{\lceil \log_n k\rceil}-k}\left(1-(1-p)^{\left(n^{\lceil \log_n k\rceil}-k\right)\left\lfloor \frac{k(n-1)-1}{n^{\lceil \log_n k\rceil}-k}\right\rfloor}\right)}{\left(1-(1-p)^{k(n-1)}\right)\left(1-(1-p)^{n^{\lceil \log_n k\rceil}-k}\right)}+\lceil \log_n k\rceil \\ +\frac{\left(1-\left\lfloor \frac{k(n-1)-1}{n^{\lceil \log_n k\rceil}-k}\right\rfloor\right)(1-p)^{k(n-1)}}{1-(1-p)^{k(n-1)}} & \text{otherwise.}\end{cases} \tag{27}$$

Algorithm 3 Encoding algorithm for the n-ary proposed coding scheme

Require: N, n, k.
Ensure: A codeword.

Let $b = \lceil \log_n k \rceil + 1$, $t = n^{b-1} - k$
if $N < t$ **then**
  $\langle RCode \rangle \leftarrow \mathrm{NC}_{b-1}^{n}(N)$
  return $\langle RCode \rangle$
else
  The quotient $c \leftarrow \left\lfloor \frac{N-t}{M} \right\rfloor$
  The remainder $r \leftarrow \mathrm{Rem}(N-t, k) + t$
  Let $\langle QCode \rangle = (\underbrace{0, 0, \cdots, 0}_{c}, \beta)$, where $c = \left\lfloor \frac{N-t}{M} \right\rfloor$, $\beta = \left\lfloor \frac{N-cM-t}{k} \right\rfloor + 1 \in \mathbb{Z}_n$
  $\langle RCode \rangle \leftarrow \mathrm{NC}_{b-1}^{n}(r)$
  return $\langle RCode \rangle \langle QCode \rangle$
end if

Algorithm 4 Decoding algorithm for the n-ary proposed coding scheme

Require: The code bitstream S
Ensure: N

Let $b = \lceil \log_n k \rceil + 1$, $t = n^{b-1} - k$
if $S = null$ **then**
  return $null$
else
  $R \leftarrow \mathrm{Deque}_S^n(b-1)$
  $c \leftarrow 0$
  if $R \geq t$ **then**
    $a \leftarrow \mathrm{Deque}_S^n(1)$
    while $a = 0$ **do**
      $c \leftarrow c + 1$
      $a \leftarrow \mathrm{Deque}_S^n(1)$
    end while
    $s \leftarrow \mathrm{Deque}_S^n(1)$
  end if
  $N = R + c \times M + k \times (s-1)$
  return N
end if

Given n and p, the objective is to determine the value of k to minimize $L_1(n,p,k)$ in (27). However, it is difficult to give the close form via (27). Instead, Table V gives the optimal value of k for certain n and p values via the mathematical software.

D. n-ary proposed coding

Subsection V-B shows that both parts in n-ary Golomb coding are coded with variable-length coding. In this subsection, we present a class of prefix codes whose remainder part is coded by fixed-length coding. Upon introducing the encoding algorithm, Table IV gives an example of the n-ary proposed coding scheme for $(n, M) = (4, 6)$ on the right-hand side of the table. In this case, the n-ary Golomb coding scheme uses one or two symbols to encode the remainder, while the n-ary proposed coding scheme always uses one symbol to encode it.

Let

$$b = \lceil \log_n k \rceil + 1, \qquad t = n^{b-1} - k. \tag{30}$$

The following introduces the quotient part and the remainder part.

1) Construction of $\langle QCode \rangle$: Let

$$\langle QCode \rangle = \begin{cases} () & \text{if } 0 \leq N < t, \\ (\underbrace{0, 0, \cdots, 0}_{c}, \beta) & \text{otherwise,} \end{cases} \tag{31}$$

TABLE VI
THE COMPRESSION RATIO OF n-ARY CODING FOR $M = 9765$

n	Ratio
2	0.476531
4	0.492469
8	0.529687
16	0.570922
32	0.609082
64	0.658359

where () is the null string and

$$c = \left\lfloor \frac{N-t}{M} \right\rfloor, \qquad \beta = \left\lfloor \frac{N-cM-t}{k} \right\rfloor + 1. \tag{32}$$

2) Construction of $\langle RCode \rangle$: First, we discuss the design philosophy of the proposed $\langle RCode \rangle$.

1) Case $t = 0$: In n-ary Golomb coding, from Section V-A, when $t = 0$, the truncated n-ary encoding tree is a complete tree of height b. In addition, the codeword length of $\langle QCode \rangle$ is zero when $0 \leq N < M$. Thus, the shortest codeword in n-ary Golomb coding has b symbols. For the proposed coding scheme, $\langle QCode \rangle$ has at least one symbol when $c = 0$. Thus, $\langle RCode \rangle$ should have $b - 1$ symbols. Therefore, we should use $b - 1$ symbols in $\mathbb{Z}_n$ to express $\langle RCode \rangle$.

TABLE VII
THE THROUGHPUT OF n-ARY GOLOMB CODING AND n-ARY PROPOSED CODING SCHEME FOR $n = 2, 4, 8$, $M = 21$

n	Enc_n_Gol(MB/S)	Enc_n_Our(MB/S)	Dec_n_Gol(MB/S)	Dec_n_Our(MB/S)
2	937.56	1099.34	1698.25	2337.62
4	930.13	1000.58	1325.99	1803.64
8	1001.73	1103.93	1003.27	1218.23

TABLE VIII
THE THROUGHPUT OF n-ARY GOLOMB CODING AND n-ARY EXP-GOLOMB FOR $n = 2, 256$

n	Enc_n_Exp-Gol(MB/S)	Enc_n_Gol(MB/S)	Dec_n_Exp-Gol(MB/S)	Dec_n_Gol(MB/S)
2	644.40	937.56	1455.21	1698.25
256	1071.46	1163.94	1285.26	1440.71

2) Case $t > 0$: In the proposed coding scheme, $\langle QCode \rangle = ()$ is the null string when $0 \le N < t$. To ensure that the codeword length of the n-ary proposed coding scheme is equal to that of the n-ary Golomb coding scheme, the length of $\langle RCode \rangle$ in the proposed coding scheme is the same as the shortest codeword of the n-ary Golomb coding scheme. The shortest n-ary Golomb coding scheme has $b-1$ symbols when $\langle QCode \rangle = ()$ is the null string, which means that $\langle RCode \rangle$ in the n-ary proposed coding scheme has $b-1$ symbols. As the $\langle RCode \rangle$ in the proposed coding scheme is fixed-length coding, the length of $\langle RCode \rangle$ in the n-ary proposed coding scheme should always have $b-1$ symbols.

According to the above discussion, $\langle RCode \rangle$ is encoded by $\mathrm{NC}^n_{b-1}(r)$. Algorithm 3 describes the n-ary proposed encoding algorithm. In Algorithm 3, Lines 1–3 handle the case in which $N < t$, and the codeword only contains $\langle RCode \rangle$ to ensure the codeword length is the same as that of the n-ary Golomb coding scheme. Lines 5–9 handle the case in which $N \ge t$. In particular, Lines 5–6 calculate the quotient and the remainder. Lines 7–9 generate the codeword. During decoding, we first decode $\langle RCode \rangle$, whose codeword always has $b-1$ symbols; then, we decode $\langle QCode \rangle$. Algorithm 4 presents the details of the decoding method. In Algorithm 4, Line 4 reads $b-1$ symbols from the code bitstream S. If $R \ge t$, Lines 7–13 try to decode $\langle QCode \rangle$. Line 14 calculates the decoded value.

Theorem 6. *For an input N, the codeword length of the n-ary proposed coding scheme is*

$$Length = \begin{cases} b-1 & \text{if } 0 \le N < t, \\ b + \left\lfloor \frac{N-t}{M} \right\rfloor & \text{if } N \ge t \ge 0, \end{cases} \tag{33}$$

which is the same as that of the n-ary Golomb coding scheme.

Proof. When $t > 0$, from Section V-D2, when $0 \le N < t$, $\langle QCode \rangle = ()$, $\langle RCode \rangle$ has $b-1$ symbols, so the n-ary proposed coding scheme has $b-1$ symbols; otherwise, it has $b - 1 + \left\lfloor \frac{N-t}{M} \right\rfloor + 1 = b + \left\lfloor \frac{N-t}{M} \right\rfloor$ symbols. In summary, we have

$$Length = \begin{cases} b-1 & \text{if } 0 \le N < t, \\ b + \left\lfloor \frac{N-t}{M} \right\rfloor & \text{if } N \ge t > 0. \end{cases} \tag{34}$$

When $t = 0$, from Section V-D2, the codeword lengths of $\langle RCode \rangle$ and $\langle QCode \rangle$ are $b-1$ and $\left\lfloor \frac{N}{M} \right\rfloor + 1$, respectively. Thus, the codeword length is

$$b - 1 + \left\lfloor \frac{N}{M} \right\rfloor + 1 = \left\lfloor \frac{N}{M} \right\rfloor + b. \tag{35}$$

Combining (34) and (35), we obtain (33).

It is easy to see that when $t = 0$, the codeword length of the n-ary proposed coding scheme is the same as that of the n-ary Golomb coding scheme (26). Next, we show that for integers N and $t > 0$, the codeword length of the n-ary proposed coding scheme (34) is also the same as that of the n-ary Golomb coding scheme (26).

1) When $0 \le N < t$, the n-ary Golomb coding scheme given in (26) requires $\lfloor N/M \rfloor + \left\lfloor \frac{\mathrm{Rem}(N,M)}{t} \right\rfloor + b - 1 = 0 + 0 + b - 1 = b - 1$ symbols, which is equal to that of the n-ary proposed coding scheme.
2) When $iM + t \le N < (i+1)M$, we have $i = \left\lfloor \frac{N-t}{M} \right\rfloor$. The codeword length of the n-ary Golomb coding scheme given in (26) requires $\lfloor N/M \rfloor + \left\lfloor \frac{\mathrm{Rem}(N,M)}{t} \right\rfloor + b - 1 = i + 1 + b - 1 = b + i$ symbols, which is equal to (34).
3) When $(i+1)M \le N < (i+1)M + t$, we have $i = \left\lfloor \frac{N-t}{M} \right\rfloor$. The codeword length of the n-ary Golomb coding scheme given in (26) requires $\lfloor N/M \rfloor + \left\lfloor \frac{\mathrm{Rem}(N,M)}{t} \right\rfloor + b - 1 = i + 1 + 0 + b - 1 = b + i$ symbols, which is also equal to (34).

□

E. Simulation and discussion

In this subsection, we first give the simulation results to show the compression ratio and the throughput for n-ary coding schemes. Then, the benefits of n-ary coding schemes are discussed.

1) Simulation results: In the first simulation, we show the compression ratio of n-ary codings for $n \in \{n_i\}_{i=1}^{z}$, $z \in \mathbb{N}$. First, the input sequence, which consists of 8000 32-bit integers, is generated by the GNU scientific library with (20), which is determined by M. From (21), we have

$$M = i \times \mathrm{lcm}\left(n_1 - 1, n_2 - 1, \cdots n_z - 1\right), \qquad i \in \mathbb{N}.$$

Figure VI presents the simulation results for $i = 1$, $M = 9765$ and $n = 2, 4 \ldots, 64$. The compression ratio is defined as the number of bits of the compressed file divided by the number of bits of the original file. As one can see, the larger n, the poorer compression ratio.

Next, we show the throughput of n-ary codings. To facilitate the implementations on conventional computers, this simulation considers the n-ary coding schemes for $n = 2^l$ (i.e., for an n that is a power of two). Table VII tabulates the throughput of n-ary Golomb coding and the n-ary proposed coding scheme for $n = 2, 4, 8$ and $M = 21$. Table VIII shows the throughput of n-ary Golomb coding and n-ary Exp-Golomb for $n = 2, 256$. It can be seen that the n-ary Golomb coding scheme ($n = 2, 256$) has a higher throughput than n-ary Exp-Golomb coding during encoding and decoding.

2) Benefits of n-ary coding schemes: As shown in the above simulation results, the n-ary coding schemes have poorer compression ratios and lower throughput on conventional computers. However, the n-ary coding schemes are still important in other aspects. First, as stated in Section I, Exp-Golomb coding has a non-binary version. However, to our knowledge, Golomb coding lacks a non-binary version, and the major work of this paper is to give the code construction. Second, in backward-adaptive coding, the input symbol is encoded by a coding system chosen from among multiple encoding schemes [26]–[31]. To align the alphabet of output symbols, we usually require that all the coding schemes use the same alphabet. Thus, when backward-adaptive coding uses an n-ary coding scheme and Golomb coding, we usually require that Golomb coding should also be n-ary. Third, the proposed n-ary coding schemes can be used in nonbinary computer systems, such as ternary computers. A potential future application of ternary computers is the circuit-based commercial quantum computer developed by IBM in 2019. It uses a quantum ternary state rather than the typical qubit.

VI. Conclusion

In this paper, we propose an entropy coding scheme for geometric distributions. The length of the codeword is equal to that of Golomb coding. However, the remainder of the proposed coding scheme uses fixed-length coding, which can significantly reduce the arithmetic complexity. The simulation shows that the proposed coding involves approximately 20% fewer addition operations, 40% fewer multiplication operations and 20% more bitwise operations during encoding and 40% fewer addition operations, 10% fewer multiplication operations, 50% fewer bitwise operations and 20% more branch operations during decoding than Golomb coding. In addition, the n-ary versions for both coding schemes are proposed and analyzed.

Acknowledgment

The authors would like to thank Dr. Eirik Rosnes and the anonymous reviewers for their helpful comments.

References

[1] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. 23, no. 3, pp. 337–343, May 1977.

[2] P. G. Howard and J. S. Vitter, "Arithmetic coding for data compression," *Proc. IEEE*, vol. 82, no. 6, pp. 857–865, June 1994.

[3] R. Hashemian, "Memory efficient and high-speed search Huffman coding," *IEEE Trans. Commun*, vol. 43, no. 10, pp. 2576–2581, Oct. 1995.

[4] H. Hashempour and F. Lombardi, "Application of arithmetic coding to compression of VLSI test data," *IEEE Trans. Comput*, vol. 54, no. 9, pp. 1166–1177, Sep. 2005.

[5] S. Golomb, "Run-length encodings (corresp.)," *IEEE Trans. Inf. Theory*, vol. 12, no. 3, pp. 399–401, July 1966.

[6] G. Gong, T. Helleseth, and P. V. Kumar, "Solomon W. Golomb—mathematician, engineer, and pioneer," *IEEE Trans. Inf. Theory*, vol. 64, no. 4, pp. 2844–2857, April 2018.

[7] Robinson, "Optimum Golomb rulers," *IEEE Trans. Comput*, vol. C-28, no. 12, pp. 943–944, Dec. 1979.

[8] J. Ding, H. Chen, and W. Wei, "Adaptive Golomb code for joint geometrically distributed data and its application in image coding," *IEEE Trans. Circuits Syst. Video Technol*, vol. 23, no. 4, pp. 661–670, April 2013.

[9] Z. Jiang, W. D. Pan, and H. Shen, "Universal Golomb–Rice coding parameter estimation using deep belief networks for hyperspectral image compression," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens*, vol. 11, no. 10, pp. 3830–3840, Oct. 2018.

[10] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS," *IEEE Trans. Image Process*, vol. 9, no. 8, pp. 1309–1324, Aug. 2000.

[11] S. Yang and P. Qiu, "Efficient integer coding for arbitrary probability distributions," *IEEE Trans. Inf. Theory*, vol. 52, no. 8, pp. 3764–3772, Aug. 2006.

[12] J. E. Tate, "Preprocessing and Golomb–Rice encoding for lossless compression of phasor angle data," *IEEE Trans. Smart Grid*, vol. 7, no. 2, pp. 718–729, March 2016.

[13] N. Merhav, G. Seroussi, and M. J. Weinberger, "Optimal prefix codes for sources with two-sided geometric distributions," *IEEE Trans. Inf. Theory*, vol. 46, no. 1, pp. 121–135, Jan. 2000.

[14] A. Chandra and K. Chakrabarty, "Test data compression and decompression based on internal scan chains and Golomb coding," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst*, vol. 21, no. 6, pp. 715–722, June 2002.

[15] ——, "Low-power scan testing and test data compression for system-on-a-chip," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst*, vol. 21, no. 5, pp. 597–604, May 2002.

[16] ——, "System-on-a-chip test-data compression and decompression architectures based on Golomb codes," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst*, vol. 20, no. 3, pp. 355–368, March 2001.

[17] R. Gallager and D. van Voorhis, "Optimal source codes for geometrically distributed integer alphabets (corresp.)," *IEEE Trans. Inf. Theory*, vol. 21, no. 2, pp. 228–230, March 1975.

[18] D. Brunello, G. Calvagno, G. A. Mian, and R. Rinaldo, "Lossless compression of video using temporal information," *IEEE Trans. Image Process*, vol. 12, no. 2, pp. 132–139, Feb. 2003.

[19] R. Sugiura, Y. Kamamoto, N. Harada, and T. Moriya, "Optimal Golomb-Rice code extension for lossless coding of low-entropy exponentially distributed sources," *IEEE Trans. Inf. Theory*, vol. 64, no. 4, pp. 3153–3161, April 2018.

[20] H. Kim, J. Lee, H. Kim, S. Kang, and W. C. Park, "A lossless color image compression architecture using a parallel Golomb-Rice hardware CODEC," *IEEE Trans. Circuits Syst. Video Technol*, vol. 21, no. 11, pp. 1581–1587, Nov. 2011.

[21] W. D. Leon-Salas, S. Balkir, K. Sayood, and M. W. Hoffman, "An analog-to-digital converter with Golomb-Rice output codes," *IEEE Trans. Circuits Syst., II, Exp. Briefs*, vol. 53, no. 4, pp. 278–282, April 2006.

[22] T.-C. Chen, Y.-W. Huang, C.-Y. Tsai, B.-Y. Hsieh, and L.-G. Chen, "Architecture design of context-based adaptive variable-length coding for h.264/avc," *IEEE Trans. Circuits Syst., II, Exp. Briefs*, vol. 53, no. 9, pp. 832–836, Sep. 2006.

[23] M. F. Brejza, T. Wang, W. Zhang, D. Al-Khalili, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Exponential Golomb and Rice error correction codes for generalized near-capacity joint source and channel coding," *IEEE Access*, vol. 4, pp. 7154–7175, Jun. 2016.

[24] J. E. Tate, "Preprocessing and Golomb-Rice encoding for lossless compression of phasor angle data," *IEEE Trans. Smart Grid*, vol. 7, no. 2, pp. 718–729, March 2016.

[25] H. Warren, *"Integer Division By Constants" in Hacker's delight*. NJ, USA: Addison-Wesley, 2012, ch. 10, pp. 205–232.

[26] H. S. Malvar, "Fast adaptive encoder for bi-level images," in *Proceedings DCC 2001. Data Compression Conference*, March 2001, pp. 253–262.

[27] ——, "Lossless adaptive encoding of finite alphabet data," *U.S. Patent 6 477 280*, Nov. 2002.

[28] ——, "Lossless adaptive encoding and decoding of integer data," *U.S. Patent 7 015 837*, Mar. 2006.

[29] ——, "Adaptive encoding and decoding of bi-level images," *U.S. Patent 6 990 242*, Jan. 2006.

[30] ——, "Lossless and near-lossless audio compression using integer-reversible modulated lapped transforms," in *Proceedings DCC 2007. Data Compression Conference*, March 2007, pp. 323–332.

[31] ——, "Lossless adaptive Golomb/Rice encoding and decoding of integer data using backward-adaptive rules," *U.S. Patent 7 580 585*, Aug. 2009.

Na Wang received the B.Eng. degree in electronic information engineering from the Tianjin University of Technology (TJUT), Tianjin, China, in 2015. She is currently pursuing the Ph.D. degree with the University of Science and Technology of China (USTC). Her research focuses on entropy-coding algorithms and data compression.

Sian-Jheng Lin received the B.Sc., M.Sc., and Ph.D. degrees in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 2004, 2006, and 2010, respectively. From 2010 to 2014, he was a postdoc with the Research Center for Information Technology Innovation, Academia Sinica. From 2014 to 2016, he was a postdoc with the Electrial Engineering Department at King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia. He was a part-time lecturer at Yuanpei University from 2007 to 2008, and at Hsuan Chuang University From 2008 to 2010. He is currently a project researcher with the School of Information Science and Technology at University of Science and Technology of China (USTC), Hefei, China. In recent years, his research focuses on the algorithms for MDS codes and its applications to storage systems.

Yunghsiang S. Han received B.Sc. and M.Sc. degrees in electrical engineering from the National Tsing Hua University, Hsinchu, Taiwan, in 1984 and 1986, respectively, and a Ph.D. degree from the School of Computer and Information Science, Syracuse University, Syracuse, NY, in 1993. He was from 1986 to 1988 a lecturer at Ming-Hsin Engineering College, Hsinchu, Taiwan. He was a teaching assistant from 1989 to 1992, and a research associate in the School of Computer and Information Science, Syracuse University from 1992 to 1993. He was, from 1993 to 1997, an Associate Professor in the Department of Electronic Engineering at Hua Fan College of Humanities and Technology, Taipei Hsien, Taiwan. He was with the Department of Computer Science and Information Engineering at National Chi Nan University, Nantou, Taiwan from 1997 to 2004. He was promoted to Professor in 1998. He was a visiting scholar in the Department of Electrical Engineering at University of Hawaii at Manoa, HI from June to October 2001, the SUPRIA visiting research scholar in the Department of Electrical Engineering and Computer Science and CASE center at Syracuse University, NY from September 2002 to January 2004 and July 2012 to June 2013, and the visiting scholar in the Department of Electrical and Computer Engineering at University of Texas at Austin, TX from August 2008 to June 2009. He was with the Graduate Institute of Communication Engineering at National Taipei University, Taipei, Taiwan from August 2004 to July 2010. From August 2010 to January 2017, he was with the Department of Electrical Engineering at National Taiwan University of Science and Technology as Chair Professor. Now he is with School of Electrical Engineering & Intelligentization at Dongguan University of Technology, China. He is also a Chair Professor at National Taipei University from February 2015. His research interests are in error-control coding, wireless networks, and security. Dr. Han was a winner of the 1994 Syracuse University Doctoral Prize and a Fellow of IEEE. One of his papers won the prestigious 2013 ACM CCS Test-of-Time Award in cybersecurity.

Nenghai Yu received his B.S. degree in 1987 from Nanjing University of Posts and Telecommunications, M.E. degree in 1992 from Tsinghua University and Ph.D. degree in 2004 from University of Science and Technology of China, where he is currently a professor. His research interests include multimedia security, multimedia information retrieval, video processing, information hiding and security, privacy and reliability in cloud computing.