

# Entropy Coding Using Equiprobable Partitioning

Yuxing Han, Jiangtao Wen and John D. Villasenor

**Abstract**—We present a simple source coding algorithm for independent and identically distributed (i.i.d.) sources that gives coding efficiency performance close to that of arithmetic coding, but with much lower computational complexity and much higher robustness to mismatches between the assumed and actual symbol probabilities. The method is based on the principle that the probability of occurrence of a symbol sequence is determined by the total number of occurrences of each member of the symbol alphabet, but not by the order of occurrence. Thus, the coding of a string of symbols can be accomplished in three steps. First, the sequence length  $M$  is encoded using an exp-Golomb code. Second, the symbol occurrence frequencies are coded using exp-Golomb codes. Third, a set of fixed length codes are used to select among the equiprobable candidate sequences. In contrast with arithmetic coding, which involves significant computation during the process of encoding and decoding, in the method described here the actual encoding and decoding are extremely simple. Furthermore, the proposed algorithm is robust to mismatches between the assumed and actual symbol probabilities.

**Index Terms**—Huffman codes, arithmetic codes, entropy codes

## I. INTRODUCTION

Entropy coding is used over an extremely wide range of applications. The most commonly used entropy coding methods are various forms of Huffman [1] and arithmetic [2] codes. Huffman codes are known to be the optimal prefix codes for coding symbols from a static i.i.d source one at a time. Furthermore, for an information source in which the symbol probabilities are all inverse powers of two, Huffman codes achieve the entropy (for binary Huffman codes; analogous efficiencies also exist for the non-binary case), so no more efficient code is possible. However, as is well known, for the more common case where symbol probabilities are not inverse powers of two, the restriction to integer-length codewords means that Huffman codes fail to achieve the entropy. Joint Huffman coding of groups of multiple symbols as opposed to individual symbols allows entropy to be approached but at a table size cost that grows exponentially with the size of the group. Arithmetic coding achieves better coding efficiency than Huffman coding by providing a single codeword to represent an entire sequence without exponential complexity growth, thereby enabling a length  $M$  sequence  $x^M$  to be represented using at most  $l(x^M) = \lceil -\log_2 p(x^M) \rceil + 1$  bits, where  $p(x^M)$  is the probability of the realization [3]. While various complexity-reduced, integer-only versions of arithmetic coding have been

developed to address this issue (see, for example, [4]), the process of arithmetic coding remains inherently complex due to the need to repeatedly calculate the size of the partitioned intervals and represent their upper and lower bounds using finite precision. In [4], the authors compared some known, highly optimized arithmetic coding algorithms, and showed that the encoding/decoding speeds of these algorithms is significantly (in some cases by several orders of magnitude) slower than that of Huffman encoding. Thus, the efficiency advantage of arithmetic coding over Huffman coding comes at a significant complexity cost. By contrast, the entropy coding approach here was motivated by the desire to create an entropy-approaching algorithm in which the complexity is off-loaded to off-line computation, thereby greatly simplifying the encoding and decoding process.

The basic idea underlying the present paper is that the overall probability of a sequence is dependent only on the number of occurrences of the symbols in the sequence, and not on the order in which those symbols occur in a given realization. Consider an i.i.d. source with an alphabet of size  $K$ , containing symbols  $s_i$ ,  $1 \leq i \leq K$ , each with corresponding probability  $p_i$ . The number of occurrences of symbol  $i$  in a given sequence is denoted by  $N(s_i)$  and the total length of the sequence is denoted by  $M$ . If, for example,  $K = 2$ , each of the  $\frac{M!}{N(s_1)!(M-N(s_1))!}$  sequences of length  $M$  containing  $N(s_1)$  occurrences of  $s_1$  and  $M - N(s_1)$  occurrences of  $s_2$  is equally likely. Once one knows  $M$  and  $N(s_1)$ , the identification of the specific sequence among the equiprobable candidates can be simply accomplished using a fixed length code with at most one bit of inefficiency. The coding task therefore involves efficiently conveying  $M$  and  $N(s_i)$ ,  $1 \leq i \leq K$ , and then sending a series of fixed length codes that select among the possible sequences, given  $M$  and  $N(s_i)$ .

Entropy coding schemes based on the idea equiprobable partitioning and fixed-length indexing have been proposed several decades ago, using the description “enumerative source coding” [5] and “variable-to-fixed-length coding” [6]. However, unlike arithmetic and Huffman codes, these have not seen wide adoption in practical systems. For example, in [7], the author pointed out two main drawbacks of the then-existing enumerative coding proposals: First, when the input block size is large, the computational and storage complexities associated with the known enumerative coding approaches became prohibitive (for example, the storage requirements increased as a cubic function of input length). This eliminated the biggest potential advantage of enumerative coding over arithmetic encoding. Second, when the block size is small, the overhead incurred in sending the side

Yuxing Han and John D. Villasenor are with the University of California, Los Angeles, CA 90095-1594 (email: ericahan@ee.ucla.edu, villa@ee.ucla.edu)

Jiangtao Wen is with Morphbius Technology Incorporated, La Jolla, CA 92037 (email: jtwen@morphbius.com)

information using the earlier schemes, especially when coded using fixed length codes, became significant, thereby making the overall coding efficiency lower than other source coding techniques. In this paper, we introduce a method that resolves both of these disadvantages, leading to a coding scheme that is both efficient and attractive from a computational and storage standpoint.

## II. ENCODING STEPS

The encoding process consists of three basic steps. In step 1, the sequence length  $M$  is encoded. This can be accomplished using any predetermined prefix or fixed-length code. In the work presented here  $M$  is coded using an exp-Golomb code. Exp-Golomb codes were first described by Teuhola [8], are parameterized by  $k$ , and have  $2^k$  codewords of shortest length,  $2^{k+1}$  codewords of next shortest length, etc. Exp-Golomb codes support description in terms of a unary|binary concatenation. The unary term is  $j + 1$  bits in length, and identifies the codeword belonging to the  $j$ th set of codewords, where the first set of codewords consists of all equal-length shortest codewords, etc. The binary term contains  $k + j$  bits and selects among the  $2^{k+j}$  codewords in the  $j$ th set. Thus, for an exp-Golomb code with parameter  $k$ , the codeword for representing a sequence of length  $M$  will have length  $k+1+2 \lfloor \log_2 \lfloor (M-1)/2^k \rfloor + 1 \rfloor$  bits. Computing an exp-Golomb code representing  $M$  can be accomplished with approximately  $j = \lfloor \log_2 \lfloor (M-1)/2^k \rfloor + 1 \rfloor$  comparisons. Thus, the cost both in bits and computation to encode  $M$  increases logarithmically with  $M$ . A more detailed description of exp-Golomb codes including an illustration of code trees is found in [9].

Next, in step 2, the number of symbol occurrences  $N(s_i)$  are encoded for  $1 \leq i \leq K - 1$  (Note that  $N(s_K)$  does not need to be explicitly sent, as its value is implied given  $M$  and knowledge of the other  $N(s_i)$  values). This is most efficiently accomplished by coding differentially with respect to the expected number of occurrences given  $M$  and the symbol probabilities. Again, an exp-Golomb code can be used, with an appropriate mapping to enable expressing negative, zero, and positive integers. Since zero is the most likely differential code value, it is given the shortest exp-Golomb codeword. 1 and  $-1$  are given the two next shortest codewords, and so on. This requires  $k + 1 + 2 \lfloor \log_2 \lfloor (N(s_i) - Mp_i)/2^k \rfloor + 1 \rfloor$  bits for each of the  $K - 1$  values  $N(s_i)$ . The computational complexity for generating each of these codewords is almost identical to the complexity noted above for step 1. There is no requirement that the same exp-Golomb parameter  $k$  be used for coding each  $N(s_i)$ .

In the third and final step, the locations of the  $N(s_i)$  occurrences of each symbol  $s_i$  within the sequence are encoded. This process is best illustrated by example. Consider a source alphabet of size  $K = 3$  with symbols A, B, and C which produces the following sequence of length  $M = 5$ : BACAA. We assume that the information that the sequence has length five and contains three A's, one B, and one C has been encoded as described in steps 1 and 2 above.

We first consider only the locations of the A's, and temporarily represent the sequence to be coded as XAXAA, where X indicates any other symbol than A. There are  $\binom{5}{3} = 10$  equally probable ways of distributing three A's in a sequence of length five. Thus, the specific locations of the A's can be represented using a fixed-length code of length  $\lceil \log_2(10) \rceil = 4$  bits with at most one bit of inefficiency. The actual bit sequence to represent this is determined through a lexicographic mapping in which the ten possibilities are ordered from AAAXX to XXAAA. The appropriate position in the list can be computed recursively using the relation  $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$ . In  $\binom{4}{2} = 6$  of the 10 possible sequences, A is not the first symbol, and in the other  $\binom{4}{3} = 4$  sequences, A is the first symbol. Since XAXAA does not start with A, it will at a position  $\geq 6$  in the lexicographic ordering (where the first element of the list has position zero), and a pointer into the list can be temporarily set to the value 6. Of the four possible sequences of length 4 containing three A's,  $\binom{3}{2} = 3$  contain A as the first element (AAAX, AAXA, AXAA) and  $\binom{3}{3} = 1$  (XAAA) does not. In this case the final four symbols in the five-symbol sequence are AXAA, so this is element number two in the triple (AAAX, AAXA, AXAA) (where again, counting begins at zero). Thus, the pointer position for XAXAA is  $6+2=8$ , which is represented in binary form by 1000.

The next step is to encode the position of the single B. There are  $\binom{2}{1} = 2$  possibilities BAXAA and XABAA, so a single bit is used (binary 0, since in this case it is BAXAA that occurs). Once the positions of the A's and B's are known, all remaining symbols must be C. So, the codeword for identifying BACAA among all length five sequences containing three A's, one B, and one C is 1000 0.

Alternatively, the encoding in step 3 can also be accomplished by using subtraction using the corresponding complement symbols. Again with reference to the same example, in the first part of step 3, instead of encoding XAXAA, consider its complement AXAXX. There are  $\binom{5}{3} = 10$  equally probable ways of distributing three X's in a sequence of length five, ordered lexicographically from the zeroth possibility (AAXXX) to the 9th possibility (XXXAA). Applying the same approach as before, AXAXX can be encoded as 1. Thus, the codeword for encoding the locations of A should be  $9-1=8$ , the same result as was achieved via direct encoding. While either method (directly encoding the sequence, or encoding via its complement) produces the same codeword, different computational complexities are involved. In the direct example above, two additions ( $0+6$  and  $6+2$ ) were used; in the complement method a single subtraction ( $9-1$ ) was needed. Further details of this tradeoff will be discussed in sec. IV.

## III. ENCODED SEQUENCE LENGTH AND EFFICIENCY

The three steps in the algorithm are the encoding of 1) the sequence length  $M$ ,  $M \geq 1$  2) the numbers of symbol occurrences  $N(s_i) - Mp_i$  and 3) the symbol locations. As noted earlier, encoding the sequence length  $M$  in step 1 requires  $k + 1 + 2 \lfloor \log_2 \lfloor (M-1)/2^k \rfloor + 1 \rfloor$  bits. Step 2

utilizes at most  $K - 1$  codewords with respective lengths given by  $k + 1 + 2 \lceil \log_2 \lfloor (N(s_i) - Mp_i)/2^k \rfloor + 1 \rceil$ ,  $1 \leq i \leq K - 1$ .  $N(s_i) \geq 0$ ,

$$1 \leq i \leq \min \left\{ K - 1, \arg \min_j \left( \sum_{i=1}^j N(s_i) = M \right) \right\}$$

Step 3, coding of the symbol locations, involves sending a series of fixed length codewords, each of which identifies the locations of all occurrences of symbol  $s_i$  within the sequence. The first such codeword requires  $\lceil \log_2 \binom{M}{N(s_1)} \rceil$  bits, the second codeword requires  $\lceil \log_2 \binom{M-N(s_1)}{N(s_2)} \rceil$  bits, and in general the  $i$ th codeword requires  $\lceil \log_2 \binom{M-N(s_1)-\dots-N(s_{i-1})}{N(s_i)} \rceil$  bits. A maximum of  $K - 1$  such fixed length codewords are needed; the number will be lower if there are symbols for which  $N(s_i) = 0$ .

The number of bits for encoding  $M$  in step 1 using exp-Golomb codes grows logarithmically with  $M$ . In the limit of large sequence length  $M$ , the cost in bits per symbol of sending  $M$  approaches zero. Similarly, the cost in bits per symbol of sending the symbol location information also grows logarithmically with  $M$ . By contrast, the cost to send the symbol location information grows approximately linearly with  $M$ , in proportion to the entropy (the growth is initially sublinear due to subtracted terms proportional to  $\log M$ . However, the effect of these  $\log M$  terms becomes negligible for large  $M$ ). To illustrate this, consider the fixed length codeword of length  $\lceil \log_2 \binom{M}{N(s_1)} \rceil$  used to convey the location of the  $N(s_1)$  instances of the symbol  $s_1$ . In the limit of large  $M$ ,  $N(s_1) \approx p_1 M$  and as a result,

$$\binom{M}{N(s_1)} \approx \frac{M!}{[p_1 M]! [M - p_1 M]!} \quad (1)$$

Using Stirling's approximation,  $n! \approx n^n e^{-n} \sqrt{2\pi n}$ , gives

$$\log_2 \binom{M}{N(s_1)} \approx MH(p_1) - \frac{1}{2} \log_2 M - C \quad (2)$$

where  $C = \frac{1}{2} \log_2(2\pi) + \frac{1}{2} \log_2(p_1) + \frac{1}{2} \log_2(1 - p_1)$  is a constant and  $H(p_1) = -p_1 \log_2 p_1 - (1 - p_1) \log_2(1 - p_1)$  is the entropy of a binary source in which one of the symbols has probability  $p_1$ . The sublinear character of Eq. (2) is a consequence of the reduction in entropy associated with knowledge that there are  $N(s_1)$  occurrences of symbol  $s_1$ . More generally, the cost for sending the symbol location information for symbol  $s_i$  is

$$\begin{aligned} & \log_2 \left( \binom{M - \sum_{j=1}^{i-1} N(s_j)}{N(s_i)} \right) \\ & \approx \left( M - \sum_{j=1}^{i-1} N(s_j) \right) H \left( \frac{p_i}{1 - \sum_{j=1}^{i-1} p_j} \right) \\ & - \frac{1}{2} \log_2 \left( M - \sum_{j=1}^{i-1} N(s_j) \right) - C_i \end{aligned} \quad (3)$$

where

$$\begin{aligned} C_i &= \frac{1}{2} \log_2(2\pi) \\ & + \frac{1}{2} \log_2 \left( \frac{p_i}{1 - \sum_{j=1}^{i-1} p_j} \right) + \frac{1}{2} \log_2 \left( 1 - \frac{p_i}{1 - \sum_{j=1}^{i-1} p_j} \right) \end{aligned} \quad (4)$$

The relation between entropy and codeword length is

$$\begin{aligned} & H(X^M) \\ &= H(M) + H(N(s_1), \dots, N(s_M) | M) \\ &+ H(X^M | N(s_1), \dots, N(s_M)) \\ &\leq H(M) + H(N(s_1) | M) + \dots + H(N(s_M) | M) \\ &+ \log_2 \left[ \binom{M}{N(s_1)} \dots \binom{M - \sum_{j=1}^{K-2} N(s_j)}{N(s_{K-1})} \right] \quad (5) \\ &= H(M) + H(N(s_1) | M) + \dots + H(N(s_M) | M) \\ &+ \left[ \sum_{i=1}^{K-1} \log_2 \left( \binom{M - \sum_{j=1}^{i-1} N(s_j)}{N(s_i)} \right) \right] \\ &\leq l(X^M) \end{aligned}$$

where  $K$ ,  $k_1$  and  $k_2$  are respectively the size of alphabet, the parameter of the exp-Golomb code used in step 1, and the parameter of the exp-Golomb code used in step 2, and  $H(X^M)$  is the entropy of a sequence with length  $M$ . For an i.i.d source,  $H(X^M) = M * H(X)$  where  $H(X) = -\sum_{i=1}^K p_i \log p_i$  is the entropy for the given source distribution. Also,

$$\begin{aligned} & l(X^M) \\ &\leq (k_1 + 1 + 2 \lceil \log_2 \lfloor (M - 1)/2^{k_1} \rfloor + 1 \rceil) + \\ &\left( \sum_{i=1}^{K-1} (k_2 + 1 + 2 \lceil \log_2 \lfloor (N(s_i) - Mp_i)/2^{k_2} \rfloor + 1 \rceil) \right) + \\ &\left( \sum_{i=1}^{K-1} \left\lceil \log_2 \left( \binom{M - \sum_{j=1}^{i-1} N(s_j)}{N(s_i)} \right) \right\rceil \right) \\ &< (k_1 + 1 + 2 \lceil \log_2 \lfloor \frac{(M-1)}{2^{k_1}} \rfloor + 1 \rceil) \\ &+ (K - 1) (k_2 + 1 + 2 \lceil \log_2 \lfloor \frac{M}{2^{k_2}} \rfloor + 1 \rceil) \\ &+ \left\lceil \log_2 \left( \binom{M}{N(s_1)} \dots \binom{M - \sum_{j=1}^{K-2} N(s_j)}{N(s_{K-1})} \right) \right\rceil \\ &+ K - 1 \\ &< (k_1 + 1 + 2 \lceil \log_2 \lfloor \frac{(M-1)}{2^{k_1}} \rfloor + 1 \rceil) \\ &+ (K - 1) (k_2 + 1 + 2 \lceil \log_2 \lfloor \frac{M}{2^{k_2}} \rfloor + 1 \rceil) \\ &+ H(X^M | N(s_1) \dots N(s_K)) + K \\ &< (k_1 + 1 + 2 \lceil \log_2 \lfloor \frac{(M-1)}{2^{k_1}} \rfloor + 1 \rceil) \\ &+ (K - 1) (k_2 + 1 + 2 \lceil \log_2 \lfloor \frac{M}{2^{k_2}} \rfloor + 1 \rceil) + H(X^M) + K \end{aligned} \quad (6)$$

Thus, the cost of sending the sequence is at most the true entropy plus at most  $K$  logarithm terms. As  $M$  becomes large,  $H(X^M)$  becomes the dominant contribution to sequence length.

The only inherent inefficiencies in the overall coding approach lie in the exp-Golomb representations in steps 1 and 2 and in the ceiling operations in the fixed length codeword lengths  $\lceil \log_2 \binom{M - N(s_1) - \dots - N(s_{i-1})}{N(s_i)} \rceil$  in step 3. The exp-Golomb inefficiencies vanish with large  $M$ , though the inefficiency due to the ceiling operation creates a persistent deviation from entropy that is bounded at a total of  $K - 1$  bits and on average is about  $K/2$  bits.

It is instructive to compare this approach with Huffman coding. The expected length of the Huffman code for an  $M$ -symbol input sequence is  $M\bar{l}$ , where  $\bar{l}$  is the average number of bits per symbol used by the Huffman code.  $\bar{l}$  is lower bounded by the entropy  $H$  of the source (with the bound achieved only when all symbol probabilities are inverse powers of two), and therefore, while the total code length

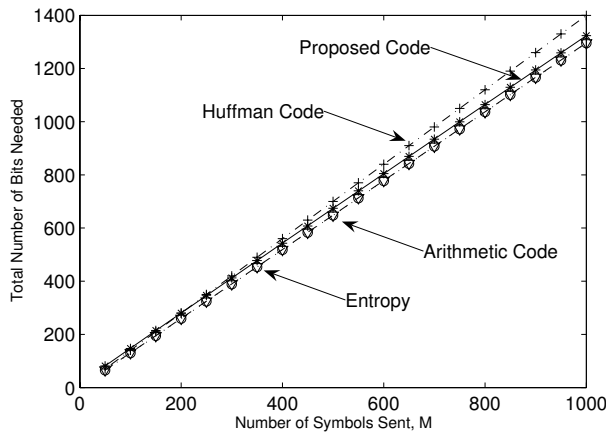


Fig. 1. Total number of bits needed to code a source with alphabet size  $K = 3$  and symbol probabilities (0.1 0.3 0.6). The sequence length is  $M$  symbols. Coding parameters for the exp-Golomb code in steps 1 and 2 are  $k_1 = 4$  and  $k_2 = 0$  respectively. For this figure as well as the other figures and tables, the length for the arithmetic code is calculated using  $l(x^M) = \lceil -\log_2 p(x^M) \rceil + 1$ . [3]

$M\bar{l}$  is a linear function of  $M$ , its slope will generally be greater than  $MH$ , resulting in a coding inefficiency relative to entropy that remains fixed in percentage terms in the limit of large  $M$ .

By contrast, the proposed code uses the information regarding  $M$  and  $N(s_i)$  to partition the set of all possible sequences of length  $M$  into groups of equiprobable sequences sharing a specific set of size  $K - 1$  of values  $N(s_i)$ . Since sequences within these groups are equiprobable, use of a fixed length code to identify specific symbol locations is optimal (other than the cost of the ceiling operation as discussed above), and the overall code length closely tracks the entropy.

Fig. 1 shows a comparison between coding performances for the proposed code, traditional Huffman coding, and arithmetic coding as a function of sequence length  $M$ . The size of the source alphabet is  $K = 3$  with symbol probabilities (0.1 0.3 0.6). In these results, the parameter  $k$  for the exp-Golomb encoding of  $M$  is four while the parameters for exp-Golomb encoding of  $N(s_1)$  and  $N(s_2)$  are zero. Since the symbol input probabilities are not inverse powers of two, the Huffman code is suboptimal and thus gives a higher slope than the curve corresponding to entropy. The curve for the proposed coding method is approximately parallel to the entropy, with an offset due to the costs of overhead information regarding sequence length  $M$  and number of occurrences of symbols  $s_i$ ,  $N(s_i)$ , and to the ceiling operation in the fixed length codeword lengths  $\lceil \log_2 \left( \frac{M - N(s_1) - \dots - N(s_{i-1})}{N(s_i)} \right) \rceil$ . The arithmetic code is more complex than the code proposed here, but approaches entropy more closely. The code lengths shown in this figure and in the other figures/tables of this section are averages based on trials using 5000 realizations.

Fig. 2 shows, for the same source used in Fig. 1, the contributions to the overall bit rate of the three steps of 1)

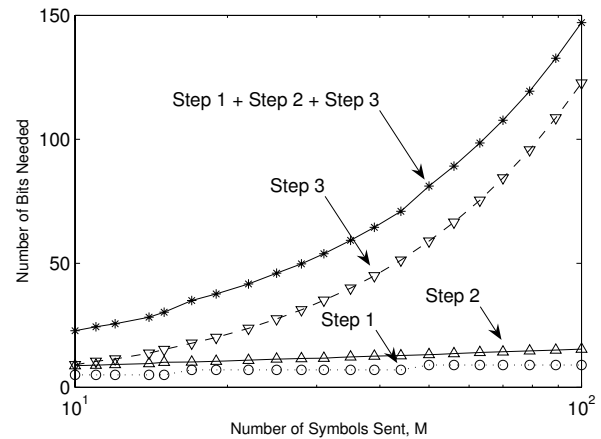


Fig. 2. Number of bits needed in each of the three coding steps to code a source with alphabet size  $K = 3$  and symbol probabilities (0.1 0.3 0.6). Coding parameters for the exp-Golomb code in steps 1 and 2 are  $k_1 = 4$  and  $k_2 = 0$  respectively.

encoding of  $M$  (using an exp-Golomb code with parameter  $k = 4$ ), 2) encoding of  $N_1$  and  $N_2$  (using an exp-Golomb code with parameter  $k = 0$ ), and 3) fixed length encoding of the symbol locations. The horizontal axis of Fig. 2 is sequence length  $M$ , plotted logarithmically. The figure shows that as expected for exp-Golomb code, the number of bits needed in steps 1 and 2 grows approximately according to the logarithm of  $M$  (and thus the corresponding curves appear approximately linear on this plot). The number of bits needed to code symbol locations (the 3rd step) grows approximately exponentially on the semi-log scale of Fig. 2, or approximately linearly in an absolute sense. As  $M$  increases, this becomes the dominant contributor to the overall code length.

Table I examines the coding performance of the proposed algorithm for short sequence lengths and provides a comparison with Huffman coding and arithmetic coding. Again, the same source as in Fig. 1 is used. As expected, for small  $M$ , the overhead incurred by entropy coding  $M$  and  $N_1$  and  $N_2$  renders the proposed algorithm less efficient than Huffman coding and arithmetic coding with perfect knowledge of the source distribution. However, as  $M$  grows, the efficiency of the proposed algorithm improves, and it eventually becomes more efficient than Huffman encoding and closely tracks that of the arithmetic encoder.

Notably, the fixed length codes used to convey the symbol positions are completely independent of the symbol probabilities  $p_i$ . While the symbol probabilities are used in sending  $N(s_i)$ , this cost becomes vanishingly small (in percentage terms) with large  $M$ . Thus, in contrast with Huffman and nonadaptive arithmetic codes in which mismatches between the assumed and actual probability distributions can lead to very significant coding inefficiencies, the present approach is inherently robust to any i.i.d. symbol probability distribution. (Adaptive arithmetic codes will of course be robust to mismatch, but they still entail significantly more computational complexity than the proposed approach.) The mismatch performance is illustrated in Fig. 3, which shows

TABLE I

NUMBER OF BITS NEEDED FOR CODING THE  $K = 3, p_i=(0.1 \ 0.3 \ 0.6)$  SOURCE FOR SEQUENCE LENGTH  $M$ .

$M$	Entropy	Arith. Code	Huff. Code	Prop. Code
10	13	14	14	23
50	65	66	70	81
90	117	117	126	134
130	168	169	182	189
170	220	221	238	241
210	272	273	294	293
250	324	325	350	347

TABLE II

PERFORMANCE UNDER PROBABILITY MISMATCH: NUMBER OF BITS NEEDED FOR CODING THE  $K = 3, p_i=(0.1 \ 0.3 \ 0.6)$  SOURCE WHEN THE ENCODER ASSUMES THE SOURCE DISTRIBUTION IS (0.6 0.3 0.1).

$M$	Entropy	Arith. Code	Huff. Code	Prop. Code
10	13	27	19	31
50	65	131	95	95
90	117	234	171	150
130	168	337	247	207
170	220	441	323	260
210	272	544	399	312
250	324	648	475	367

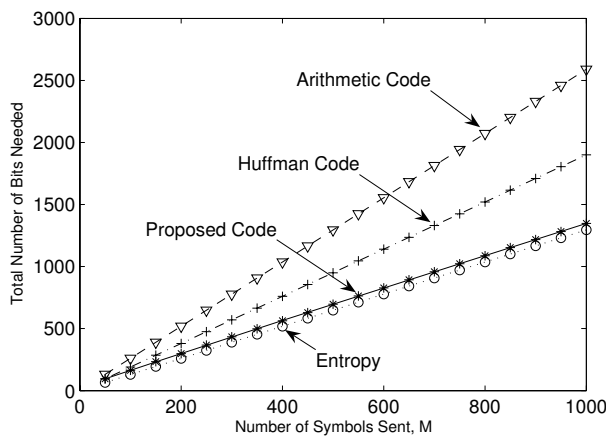


Fig. 3. Performance under probability mismatch: Number of bits needed for coding the  $K = 3, p_i=(0.1 \ 0.3 \ 0.6)$  source when the encoder assumes the source distribution is (0.6 0.3 0.1).

the coding performance on the same  $K = 3, p_i=(0.1 \ 0.3 \ 0.6)$  source as used in Fig. 1, but in which the symbol probabilities assumed by the encoders are (0.6 0.3 0.1). Both the Huffman and arithmetic codes have significantly higher slopes than the entropy curve. By contrast, the proposed code again tracks the slope of the entropy curve, with the only penalty due to mismatch occurring in a slightly less efficient representation of  $N(s_i)$ .

For example, for the source studied in Fig. 3, the overhead of coding  $M, N_1$  and  $N_2$  increases logarithmically, reaching an average about 15 bits for sending  $M$  and about 25 bits for sending  $N_1$  and  $N_2$  for an input length of about  $M = 1000$  symbols when the encoder knows the correct symbol probabilities, as contrasted with about 15 bits for sending  $M$  and about 50 bits for sending  $N_1$  and  $N_2$  when the incorrect (0.6 0.3 0.1) probabilities are assumed. The probability mismatch case is further examined in Table II for short sequence lengths  $M$ . In this case, even for relatively small  $M$  the proposed code outperforms both the Huffman and arithmetic code.

#### IV. STORAGE REQUIREMENT AND COMPLEXITY

##### A. Storage requirement

The computational cost of generating exp-Golomb codes for steps 1 and 2 is very small as noted earlier. Generating the fixed length codes in step 3 requires access to  $\binom{m}{n}$ ,  $1 \leq m \leq M, 0 \leq n \leq m$ . The values  $\binom{m}{n}$ , or equivalently, the entries in Pascal's triangle, can either be pre-computed and stored in a table, or generated as needed on the fly. In either case, the recursion  $\binom{m}{n} = \binom{m-1}{n-1} + \binom{m-1}{n}$  can be used. Thus, the cost for computing the  $M$ th row in Pascal's triangle given the  $(M-1)$ th row will be at most  $(M-1)$  additions. A Pascal's triangle with  $M$  rows contains approximately  $M^2/2$  entries, though symmetries reduce the storage requirement by a factor of two. The entries can be efficiently stored using an exp-Golomb code, which is well suited to representing arbitrarily large integers using prefix codes. Storing the Pascal's triangle entries up to the  $M$ th row using an exp-Golomb code with  $k = 0$  will require

$$\sum_{m=1}^M \sum_{n=0}^{\lceil \frac{m}{2} \rceil} \left( 1 + 2 \left\lceil \log_2 \left[ \left[ \binom{m}{n} - 1 \right] + 1 \right] \right\rceil \right)$$

bits.

However, in the typical case, only a subset of all possible  $m, n$  combinations will be needed, corresponding to jointly typical values of  $m$  and  $n$  as  $M$  becomes large. Fig. 4 shows the percentage of the Pascal's triangle table entries which are used in coding symbols from a binary source in which the probability of one of the symbols varies from 0.1 to 0.9. As can be seen from the figure, as  $M$  becomes large, the percentages tend to converge to between about 30% to 65% of the entire Pascal's triangle. Thus, it is the most efficient for both the encoder and decoder to store these 30% to 65% entries as opposed to the entire triangle. If a highly atypical sequence occurs and requires an entry not previously stored, one can always the recursion relation noted earlier calculate the corresponding entry. Of course, in the case where a probability mismatch is suspected, from the standpoint of table storage it is the best to assume the worst case,  $p_i = 0.5$ . Fig. 5 illustrates the storage requirement for these entries when an exp-Golomb code with  $k = 0$  is used, again for the case of a binary source.  $p = 0.5$  leads to the largest

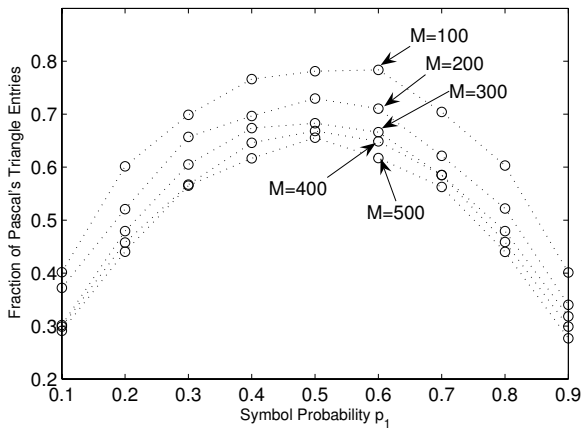


Fig. 4. Total fraction of Pascal's triangle entries used in coding 5000 realizations of a binary source with symbol probabilities  $p_1$  and  $1 - p_1$ . Results for sequence lengths  $M = 100$  to 500 are shown.

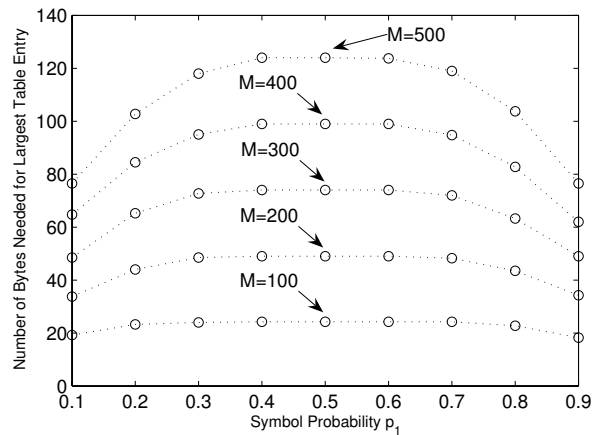


Fig. 6. Number of bytes needed to store the largest Pascal's triangle entry used in coding 5000 realizations of a binary source with symbol probabilities  $p_1$  and  $1 - p_1$ . Results for sequence lengths  $M = 100$  to 500 are shown. Storage is accomplished using an exp-Golomb code with parameter  $k = 0$ .

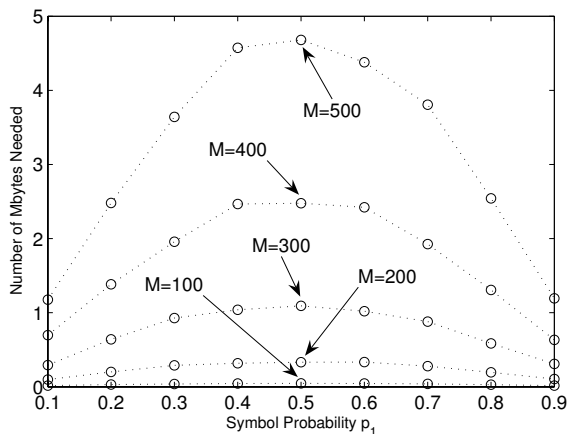


Fig. 5. Total number of Mbytes needed to store Pascal's triangle entries used in coding 5000 realizations of a binary source with symbol probabilities  $p_1$  and  $1 - p_1$ . Results for sequence lengths  $M = 100$  to 500 are shown. Each entry is stored using an exp-Golomb code with parameter  $k = 0$ .

storage requirements because the entries in the middle of Pascal's triangle have the largest values and require the most precision. Fig. 6 illustrates the number of bytes needed to store the largest of these entries.

Storage requirements can be reduced by sending the symbol location information in step 3 according to ascending probability order. For example, in a source with alphabet size  $K = 5$  with symbols A, B, C, D, E and respective probabilities (0.6, 0.1, 0.1, 0.1, 0.1), it is better to send the information about the locations of symbol A last. As shown in Fig. 7, this leads to a storage requirement that is about four times lower relative to sending the locations of symbol A first.

**B. Complexity**

Once the Pascal's triangle information is available, the cost for generating the fixed length code for conveying the  $N(s_i)$  places where symbol  $s_i$  occurs will depend on whether direct

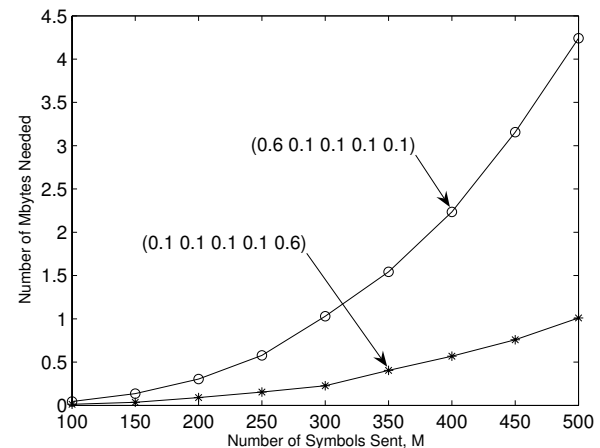


Fig. 7. Relationship of table size to the order in which symbol location information is sent. Results are averages based on coding 1000 realizations of a source with alphabet size  $K = 5$  with probabilities (0.6, 0.1, 0.1, 0.1, 0.1). The vertical axis gives the table size in Mbytes, and the horizontal axis shows the sequence length  $M$ . The upper curve corresponds to sending the symbol locations for the most probable symbol first, while the lower curve corresponds to sending the symbol locations for the most probable symbol last.

or complement method is used. If the direct method is used, the cost of conveying the locations of symbol  $s_i$  will be at most  $M - N(s_1) - \dots - N(s_i)$  additions with the appropriate Pascal's triangle entries. If the complement method is used, the cost will be at most  $N(s_i)$  additions. Thus, to minimize the number of additions one should choose between the direct and the complement method by identifying  $\min(N(s_i), M - N(s_1) - \dots - N(s_i))$ . Obviously, the total number of additions needed for encoding will be less than  $M$ . No comparisons are needed during the encoding.

In contrast with encoding of the fixed length codes in step 3, which requires additions but no comparisons, in decoding both additions and comparisons are needed. If the direct method is used, decoding the locations of  $s_i$  will

require at most  $M - N(s_1) - \dots - N(s_i)$  additions and  $M - N(s_1) - \dots - N(s_{i-1})$  comparisons. If the complement method is used, the cost will be at most  $N(s_i)$  additions and  $M - N(s_1) - \dots - N(s_{i-1})$  comparisons; as with encoding, the less computationally expensive method should be chosen.

#### V. APPLICATION TO GENERALIZED GAUSSIAN SOURCES

The highly peaked wide-tailed pdf's that are encountered in many applications are often modeled using the family of generalized Gaussian (GG) pdf's. The GG is a family of pdf's parameterized by a shape parameter  $\nu$ . When  $\nu$  is 1 the GG is Laplacian and  $\nu = 2$  it is Gaussian; as  $\nu \rightarrow \infty$  the distribution approaches uniformity. GG distributions have been well studied. The two-sided GG pdf can be expressed as

$$f(x) = C_1 \exp(-C_2 |x|^\nu)$$

where  $C_1$  and  $C_2$  are given by

$$\begin{aligned} C_1 &= \nu \eta(\sigma, \nu) / 2\Gamma(1/\nu) \\ C_2 &= [\eta(\sigma, \nu)]^\nu \end{aligned}$$

and

$$\eta(\sigma, \nu) = \frac{1}{\sigma} \left[ \frac{\Gamma(3/\nu)}{\Gamma(1/\nu)} \right]^{1/2}$$

The variable  $\nu$  is the shape parameter and the standard deviation of the source is given by  $\sigma$ . In associating a discrete source with a GG, we use a uniform scalar quantizer of step size  $\delta$  and a deadzone at the origin of width  $(1 + \alpha)\delta$ ,  $\alpha \geq 0$ . A quantized GG can be used to model positive, nonnegative, and two-sided discrete sources. As discussed in [9] and elsewhere, Golomb-Rice and exp-Golomb codes are known to perform well in coding quantized GG sources.

We consider the performance of the proposed code on an i.i.d. positive integer source produced by quantizing  $M = 1000$  values drawn from a GG with shape parameter  $\nu = 0.7$  and a deadzone size of  $\alpha = 0.5$ . Fig. 8 shows the coding performance of the proposed code as well as the performance of an exp-Golomb code with parameter  $k = 0, 1, 2$  for step sizes (expressed as a function of standard deviation  $\sigma$ ) ranging from  $10^{-2} \leq \delta/\sigma \leq 10^0$ . The encoder in the proposed code assumes  $\delta/\sigma = 0.3$ , so its efficiency is highest at that value. For the entire range of quantization step sizes, the proposed code outperforms the exp-Golomb code with parameters  $k = 0$  and  $k = 1$ . The exp-Golomb code with  $k = 2$  slightly outperforms the proposed code for some  $\delta$  values, but the difference is very small (1-2 percentage points). In all cases, the proposed code is more robust to variations in the quantization step size, and leads to efficiencies that stay well above 85%. By contrast, the efficiencies of the  $k = 2$  exp-Golomb codes dips below 50%, while the efficiencies of the  $k = 1$  and  $k = 0$  codes go below 60% and 80% respectively.

When the shape parameter of GG source becomes small the efficiency of the proposed code is lower. Fig. 9 shows the coding efficiency when the shape parameter is  $\nu = 0.3$  and a deadzone size of  $\alpha = 0.5$ . Both the proposed code (using parameters  $k_1 = 10$ ,  $k_2 = 0$ ) and exp-Golomb codes

with parameters  $k=0, 1$ , and 2 are shown. For this shape parameter the proposed code is superior to the exp-Golomb code for larger step sizes but inferior at smaller step sizes. This is due to the long tail in the source distribution generated when small step sizes are used. Each very large input symbol has a small probability of occurring in a sequence of length  $M = 1000$ , leading to many cases in which  $N(s_i)$  will be zero. Coding each of these  $N(s_i) = 0$  occurrences costs at least two bits, leading to significant efficiency loss when a large fraction of the possible symbols do not occur in a typical realization.

A low complexity and effective solution to coding integer distributions with long-tail characteristics is to find a cut-off region for the quantized GG inputs, similar to the partitioning and differentiated treatment of inputs inside and outside the typical set in Shannon's proof of his source coding theorem [3]. Thus, the set of possible positive integers can be divided into a high probability class and a low probability class, using a pre-defined cut-off probability known both to the encoder and decoder. In applying the proposed code for sequences of inputs containing both high probability and low probability integers, high probability integers can be directly coded as described earlier, while low probability integers in the input sequence are coded with the aid of an ESCAPE symbol. The ESCAPE symbol is designated as the integer  $N$  satisfying  $N = \arg \min (p_n < T)$ , where  $T$  is the cut-off probability threshold. After the total numbers of occurrences and the exact locations of the high probability inputs and the ESCAPE symbol of value  $N$  are coded, the encoder appends the concatenations of the exp-Golomb codewords for  $x - N$  for each low probability input  $x$  in the input sequence. Fig. 10 shows the performance of the proposed code with cut-off regions as compared with exp-Golomb codes. As can be seen, the performance of the proposed code improves significantly while remaining robust to input variations. In theory it is possible to also modify exp-Golomb encoding to utilize a cutoff and ESCAPE code, though in practice this would actually lower the exp-Golomb coding performance.

In contrast with Huffman and arithmetic codes, in the proposed code the encoder must be in possession of the entire sequence prior to beginning transmission. However, given the speed of present day processors and data transmission rates and the simplicity of the algorithm, in most entropy coding applications (including real-time video and audio compression algorithms) this requirement would not have any significant impact on the overall performance of the system.

#### VI. CONCLUSIONS

We have presented an entropy coding scheme for an i.i.d. source based on partitioning all possible sequences of length  $M$  into sets of equiprobable sequences that share the same number of occurrences of each of the symbols but differ in the specific symbol locations. The sequence is encoded by first sending the sequence length  $M$  using an exp-Golomb (or other suitable) code, then sending the number of occurrences

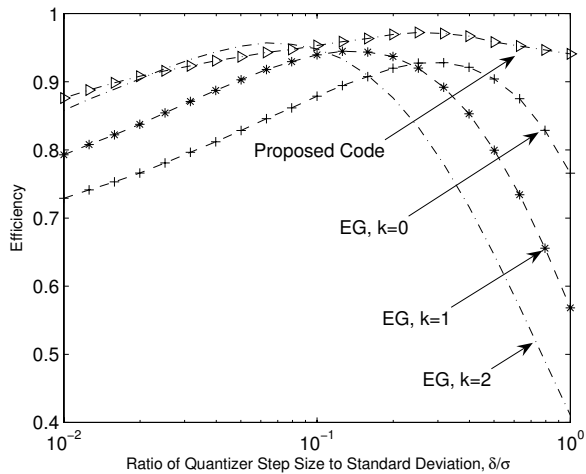


Fig. 8. Coding efficiency of the proposed code (using parameters  $k_1 = 10$ ,  $k_2 = 0$ ) and of exp-Golomb (EG) codes with parameters  $k = 0, 1, 2$  for coding a quantized generalized Gaussian (GG) source with length  $M = 1000$ . The curves shown are averages based on coding 1000 realizations. Source symbols are generated by quantizing a GG source with shape parameter  $\nu = 0.7$  and quantizer deadzone size  $\alpha = 0.5$ . The horizontal axis shows quantizer step sizes  $\delta$  ranging from  $\delta/\sigma = 10^{-2}$  to  $10^0$ .  $\sigma$  is the standard deviation of the source prior to quantization. The proposed code quantizer assumes  $\delta/\sigma = 0.3$ .

$N(s_i)$  of each of the  $K$  symbols in the alphabet, and finally sending a set of fixed length codes to identify the symbol locations. The algorithm provides performance similar to that of arithmetic coding with significantly lower computational complexity. In addition, it is inherently robust to mismatches between the assumed and actual symbol probabilities. Experimental results confirm that the code achieves high efficiency when applied to coding of quantized generalized Gaussian sources.

#### REFERENCES

- [1] R. Gallager, "Variations on a theme by Huffman," *IEEE Transactions on Information Theory*, vol. 24, no. 6, pp. 668–674, Nov. 1978.
- [2] A. Moffat, R. M. Neal, and I. H. Witten, "Arithmetic coding revisited," *ACM Trans. Inf. Syst.*, vol. 16, no. 3, pp. 256–294, July 1998.
- [3] T. Cover and J. Thomas, *Elements of Information Theory*. New York, NY, USA: Wiley, 1991.
- [4] A. Said, "Comparative analysis of arithmetic coding computational complexity," *Hewlett-Packard Laboratories Report*, pp. HPL-2004-75, 2004.
- [5] T. Cover, "Enumerative source encoding," *IEEE Transactions on Information Theory*, vol. 19, no. 1, pp. 73–77, Jan. 1973.
- [6] J. P. M. Schalkwijk, "An algorithm for source coding," *IEEE Transactions on Information Theory*, vol. 18, no. 3, pp. 395–399, May 1972.
- [7] L. Oktem, *Hierarchical Enumerative Coding and Its applications in Image Compression*. Tampere, Finland: Tampere University of Technology Ph.D. Dissertation, 1999.
- [8] J. Teuhola, "A compression method for clustered bit-vectors," *Inform. Processing Lett.*, vol. 7, pp. 308–311, Oct. 1978.
- [9] J. Wen and J. Villasenor, "Structured prefix codes for quantized low-shape-parameters generalized Gaussian sources," *IEEE Transactions on Information Theory*, vol. 45, no. 4, pp. 1307–1314, May 1999.

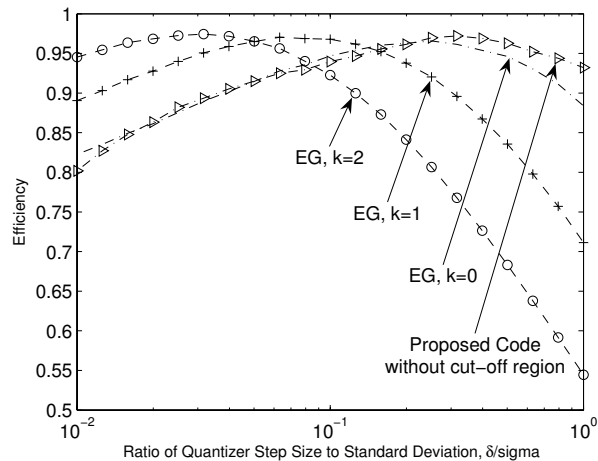


Fig. 9. Coding efficiency of the proposed code (using parameters  $k_1 = 10$ ,  $k_2 = 0$ ) and of exp-Golomb (EG) codes with parameters  $k = 0, 1, 2$  for coding a quantized generalized Gaussian (GG) source with length  $M = 1000$ . The curves shown are averages based on coding 1000 realizations. Source symbols are generated by quantizing a GG source with shape parameter  $\nu = 0.3$  and quantizer deadzone size  $\alpha = 0.5$ . The horizontal axis shows quantizer step sizes  $\delta$  ranging from  $\delta/\sigma = 10^{-2}$  to  $10^0$ .  $\sigma$  is the standard deviation of the source prior to quantization. The proposed code quantizer assumes  $\delta/\sigma = 0.3$ . No cut-off is used.

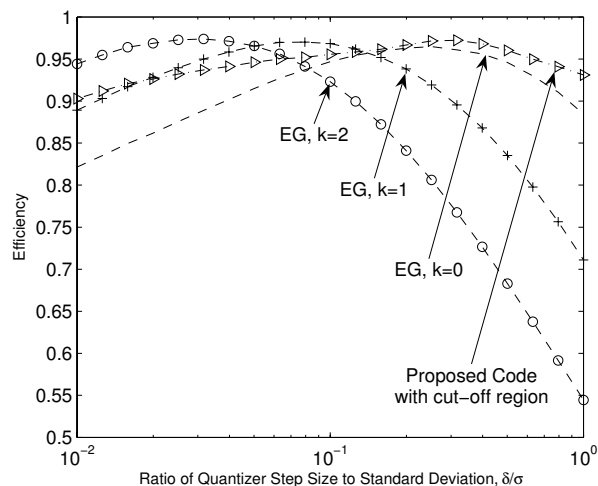


Fig. 10. Same as Fig. 9, but with a cut-off region at  $N = \arg \min (p_n < 10^{-4})$  for  $M = 1000$ . Note that in contrast with Fig. 9, the performance of the proposed code remains good for small step sizes.