

A Novel Bit-Wise Adaptable Entropy Coding Technique

Aaron Kiely and Matthew Klimesh

Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive, Mail Stop 238-420, Pasadena, CA 91109

e-mail: {aaron, klimesh}@shannon.jpl.nasa.gov

Abstract — We present a novel entropy coding technique which is adaptable in that each bit to be encoded may have an associated probability estimate which depends on previously encoded bits. The technique may have advantages over arithmetic coding. The technique can achieve arbitrarily small redundancy and admits a simple and fast decoder.

I. INTRODUCTION

We examine the problem of compressing a sequence of bits b_1, b_2, \dots from a random source, where each bit has an associated probability estimate $p_i = \text{Prob}[b_i = 0]$ which may depend on source bit values prior to index i . This dependence encompasses adaptive probability estimation as well as correlations or memory in the source. Efficient compression requires a bit-wise adaptable coder. Arithmetic coding is the traditional approach in this situation; here we describe an alternative.

We present some highlights of our new compression technique, which we call recursive interleaved entropy coding. This technique generalizes methods described in [1, 2]. For more details on our technique, including practical encoding and decoding algorithms, see [3].

II. AN ENTROPY CODING TECHNIQUE

Without loss of generality we assume that $p_i \geq 1/2$ for each index i since the encoder and decoder can invert b_i when this is not the case. We partition the region $[1/2, 1]$ into several narrow intervals, each of which has a bin that stores a list of bits. We place source bit b_i into the bin with the interval that contains p_i . Since each interval spans a small range, we think of each bin as representing some nominal probability value.

Bits in the leftmost bin, which contains probability $1/2$, are transmitted without further processing. For every other bin we specify an exhaustive prefix-free set of codewords. When bits in a bin form one of these codewords, we delete these bits and place new bits in other bins. (The rules for formation of codewords are not straightforward; see [3].) The mapping from codewords to output bits can be described with a binary tree. Each codeword is assigned to a terminal node, non-terminal nodes are labeled with a destination bin, and the branch labels (each a zero or one) indicate the output bits.

For example, Figure 1 shows a possible tree for a bin with nominal probability 0.9 and codeword set $\{00, 01, 1\}$. If the codeword 00 is formed in the bin (this occurs with probability approximately 0.81), we place a zero in the bin containing 0.81. If instead the codeword is 1, we place a one in the bin containing 0.81, then we place a zero in the bin containing 0.53

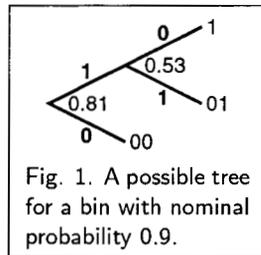


Fig. 1. A possible tree for a bin with nominal probability 0.9.

because, given that the codeword is not 00, the probability that the codeword is 1 is approximately 0.53.

In practice, bins are identified by indices rather than nominal probability values, starting with index 1 for the leftmost bin. Our goal is to have bits flow into this bin, where they are transmitted. To accomplish this, we impose the constraint on our trees that each output bit from the tree for bin j must be placed in a bin with index strictly less than j . Although this mapping can be made without regard for nominal probability values, one expects better compression if (the mapping causes) the bits in a bin all have nearly the same probability-of-zero. Thus, we would like the tree for a bin with nominal probability p to produce output bits that all have probability-of-zero in the range $[1/2, p)$. Such a tree is said to be *useful at p* .

Theorem 1 For any given probability value $p \in (1/2, 1)$, there exists a tree that is useful at p .

This can be proved by constructing an infinite family of trees such that for any $p \in (1/2, 1)$ there is a tree from the family that is useful at p . We say that such a family of useful trees is *complete*.

Theorem 2 Given a complete family of useful trees, for any $\epsilon > 0$ and $\delta > 0$ there exists a coder using only trees from this family and a constant c for which the following holds: For any n and any sequence of bits b_1, \dots, b_n whose associated probability-of-zero estimates p_1, \dots, p_n are all in the range $[\delta, 1 - \delta]$, the sequence will be compressed to at most

$$c + (1 + \epsilon) \sum_{i=1}^n \begin{cases} -\log_2 p_i, & \text{if } b_i = 0 \\ -\log_2(1 - p_i), & \text{if } b_i = 1 \end{cases} \text{ bits.}$$

III. PERFORMANCE

In a test designed to measure the speed of decoding when isolated from source modeling, a 10-bin coder provided about 35% faster decoding than a binary arithmetic coder from [4]. Encoding is presently about three times slower than decoding. In this test, redundancy was 0.0032 bits/source bit for our coder, and 0.0019 bits/source bit for the arithmetic coder. Details can be found in [3].

REFERENCES

- [1] F. Ono, S. Kino, M. Yoshida, and T. Kimura, "Bi-Level Image Coding with MELCODE — Comparison of Block Type Code and Arithmetic Type Code," *Proc. IEEE Global Telecommunications Conference (GLOBECOM '89)*, pp. 0255–0260, Nov. 1989.
- [2] P. G. Howard, "Interleaving Entropy Codes," *Proc. Compression and Complexity of Sequences 1997*, pp. 45–55, 1998.
- [3] A. Kiely and M. Klimesh, "An Adaptable Binary Entropy Coder," *Proc. 2001 IEEE Data Compression Conference, Snowbird, Utah, March 27–29, 2001*.
- [4] A. Moffat, R. M. Neal, and I. H. Witten, "Arithmetic Coding Revisited," *ACM Transactions on Information Systems*, vol. 16, no. 3, pp. 256–294, July, 1998.

The work described was funded by the TMOD Technology Program and performed at the Jet Propulsion Laboratory, California Institute of Technology under contract with the National Aeronautics and Space Administration.