JET PROPULSION LABORATORY

INTEROFFICE MEMORANDUM
331-94.2-043
June 1, 1994

TO: Distribution

FROM: Aaron Kiely

SUBJECT: Comparison of Huffman and Bit-Wise Arithmetic Coding

1 Introduction

A well-known defect of Huffman coding is that the redundancy (the difference between code rate and source entropy) can be substantial when the source entropy is small. Combining Huffman and zero-runlength coding is a common method of overcoming this problem. Alternatively, a new block-adaptive data compression scheme, bit-wise arithmetic coding, may be used [5]. In this memo we examine the redundancy and overhead associated with block-adaptive Huffman coding, analyze a variation of the zero-runlength coding, and compare the techniques to bit-wise arithmetic coding.

In the sequel, we will be interested in the problem of compressing the output of a uniform quantizer having b bits. At the input of the quantizer is an IID continuous source with probability density function (pdf) $f(x)$. For details on computing the quantizer distortion or entropy, and an analysis of bit-wise arithmetic encoding, see [5].

2 Huffman Coding

2.1 Redundancy

Given the probability distribution on the quantizer output symbols we may use the Huffman algorithm to (non-adaptively) assign a binary codeword to each symbol. The redundancy of Huffman codes has been studied and bounded by many researchers, e.g. [4]. Because each codeword must be at least one bit long, the rate of a Huffman code can never be less than one, so the redundancy is particularly high when the entropy approaches zero. Even when the source entropy exceeds one bit, redundancy may be significant.

Example 1 *Suppose the quantizer output is IID with probability distribution* $\{0, \varepsilon, 1/4 - \varepsilon, 1/2, 1/4 - \varepsilon, \varepsilon, 0, 0\}$ *for small* ε*. The entropy is very nearly* 1.5 *bits, but an ideal Huffman code assigns codewords of length (respectively)* $\{0, 4, 3, 1, 2, 4, 0, 0\}$*, producing an average length of* $1.75 + 3\varepsilon$ *bits, adding approximately* .25 *bits to the rate.*

This redundancy effect also contributes to the rate of block-adaptive Huffman codes.

2.2 Block-Adaptive Huffman Coding

Under block-adaptive Huffman coding, we create a new Huffman code for each block of N quantizer output symbols using the actual symbol frequencies in the block as estimates of the symbol probabilities. It will be convenient to decompose the rate into two components: the *code rate*[1], resulting from codeword bits, and the *overhead rate*, from bits required to identify the code parameters. We examine overhead in Section 2.3. In this section, we show how to estimate the code rate.

Example 2 *Using the distribution of Example 1, in a sequence of N samples, we would expect each of the symbols with probability ε to appear $N\varepsilon$ times. When both symbols appear, the expected code rate is close to 1.75 bits. However, each symbol has a probability of $(1-\varepsilon)^N$ of not appearing in the block. When one doesn't appear, it is not assigned a codeword, so the expected code rate is lower. Consequently, the average code rate is somewhat less than 1.75 bits.*[2]

In principle it is straightforward to compute the code rate:

$$R_{\text{Huff}} = \sum_{x \in \mathcal{A}^N} \Pr[x]\text{Huff}[x]. \tag{1}$$

Here $\mathcal{A}$ is the quantizer output alphabet and $\text{Huff}[x]$ is the rate obtained by block-adaptively Huffman encoding the sequence x. Of course, both $\text{Huff}[x]$ and $\Pr[x]$ depend only on the frequencies of symbols in x for an IID source. Although there is no known closed form expression for $\text{Huff}[x]$, its computation is quite feasible by computer– we simply apply the Huffman algorithm to the symbol frequencies in x. What is much less feasible is the summation over all of $\mathcal{A}^N$.

As a more practical means of estimating the code rate, we can limit the summation to a few typical sequences. We find the "critical" symbol a whose probability of appearing in an N-length block is closest to $1/2$. We construct a sequence x containing one occurrence of a, no occurrences of symbols having probability less than that of a, and all other symbols occurring in x with frequency (nearly) proportional to their probability. We also construct a second typical sequence y that is similar to x except that a does not occur in y.[3] The rate estimate is a weighted sum of $\text{Huff}[x]$ and $\text{Huff}[y]$. This approximation comes surprisingly close to simulations, see Figure 3.

We should note that the code rate may be smaller than the overhead rate. In fact the code rate may be lower than the source entropy. For example, using (1) in the extreme case where $N = 1$, the code rate is zero! Also note that at very high distortion, block-adaptive Huffman coding can achieve rates below one bit (see, e.g., Figure 3) while the non-adaptive Huffman code cannot.

[1] It should be clear from context whether "code rate" refers to a source code (as we intend it here) or channel code (as it is more commonly used).

[2] From this example, we can see that increasing N may actually increase the code rate because it increases the probability that "rare" symbols will appear in a block.

[3] Actually, since the distribution is symmetric, there are two such symbols. For improved accuracy, we construct four different typical sequences.

2.3 Overhead

In block-adaptive Huffman coding we must transmit overhead bits with each block to identify to the decoder the Huffman code used. The problem of minimizing the number of overhead bits is treated in [1, 6]. Here we summarize some of the relevant results.

A Huffman code for 2^b symbols can be identified by specifying the codeword lengths $n_1, n_2, \ldots n_{2^b-1}$. (We can solve the Kraft-McMillan equality $\sum_i 2^{-n_i} = 1$ to find n_{2^b}.) This requires approximately

$$(2^b - 1) \log_2 l(2^b, N)$$

bits of overhead, where $l(i, N)$ is an *a priori* upper bound on codeword length. We may use

$$l(i, N) = \min\{i - 1, L(N)\}$$

where $L(N)$ is the unique integer K satisfying $F_{K+3} > N \geq F_{K+2}$ and F_K is the Kth Fibonacci number. In general, without making use of additional information about the source, it is not possible to use substantially less overhead. We call this overhead technique OH1.

Under OH1, overhead bits are required even for symbols that do not appear in a block. When $f(x)$ is nonincreasing in $|x|$ and the quantizer bin width is large, it is likely that several of the quantizer symbols corresponding to large $|x|$ do not appear in a given block. In such a situation, if we arrange the n_i in order of decreasing magnitude of quantizer reconstruction point, then the list of n_i is likely to begin with several zeros. We can first send b bits identifying z, the length of the run of zeros, and then encode the remaining $2^b - z$ codewords in the manner of OH1. This overhead scheme, which we call OH2, requires

$$b + (2^b - z - 1) \log_2 l(2^b - z, N) \tag{2}$$

bits of overhead. When the entire block consists of N zeros, $z = 2^b - 1$ and we need only b bits to encode the block, resulting in more savings than simply reduced overhead. This has some impact at low rates.

This new scheme costs at most b extra overhead bits compared to OH1, and in practice often saves a significant amount. Using the bound $l(i, N) \leq \frac{\log_2 N}{\log_2((1+\sqrt{5})/2)}$ (from [6]) we find that a sufficient condition for OH2 to give lower overhead than OH1 is that

$$z \geq \frac{b}{\log_2 \left[\frac{\log_2 N}{\log_2((1+\sqrt{5})/2)}\right]} \approx \frac{b}{.526 + \log_2 \log_2 N}.$$

For example, when $N = 256$ and $b = 8$, we get a savings whenever $z \geq 3$.

3 Zero-runlength Encoding

3.1 ZGH encoding

A common means of tackling the redundancy problem at low entropy is to combine Huffman coding with zero-runlength coding. This is more efficient than Huffman coding when zeros

are highly probable. For example, the sequence of quantizer output indices $0, 1, 0, 0, 2, -1$ can be thought of as $0^1, 1, 0^2, 2, 0^0, -1$, i.e., an alternating sequence from the separate alphabets $\{0^i : i \geq 0\}$ and the nonzero integers. We insert the symbol 0^0 between adjacent nonzero symbols to ensure that the sequence is alternating. This allows us to separately encode the two alphabets. First let us consider the non-adaptive case.

Because the source is IID, the runs of zeros are distributed according to

$$\Pr[0^i] = (1 - p_0){p_0}^i$$

where p_j denotes the probability that the quantizer output is index j. We encode runs of zeros using the optimal (Huffman) encoding for this distribution: the Gallager and van Voorhis (GVH) code [3]. The nonzero symbols are separately encoded using the standard block-adaptive Huffman coding procedure. We call this combined technique ZGH (for zero-runlength + GVH + Huffman) coding. There are, of course, many variations of zero-runlength encoding, we choose this one primarily for its simplicity.

To estimate the rate of the ZGH code, consider the encoding of the pair $0^i, S$, where S is some symbol from the nonzero alphabet. The expected number of source symbols represented by this pair is

$$\mathrm{E}[1 + i] \approx 1 + \sum_{i=0}^{\infty} i(1 - p_0)p_0^i = \frac{1}{1 - p_0}.$$

The expected number of encoded bits for the pair is $R_{\mathrm{Huff}}(\mathrm{NZ}) + n_{\mathrm{GVH}}$ where $R_{\mathrm{Huff}}(\mathrm{NZ})$ is the rate obtained from applying a Huffman code to the nonzero symbols, and n_{GVH} is the average length of a GVH codeword. From [3], we find

$$n_{\mathrm{GVH}} = 1 + \lfloor \log_2 l \rfloor + \frac{p_0^k}{1 - p_0^l}$$

where

$$l = \left\lceil \frac{-\log(1 + p_0)}{\log p_0} \right\rceil \tag{3}$$

and $k = 2^{\lfloor \log_2 l \rfloor + 1} - l$. Thus the average rate of the ZGH code is

$$R_{\mathrm{ZGH}} = \frac{R_{\mathrm{Huff}}(\mathrm{NZ}) + n_{\mathrm{GVH}}}{\mathrm{E}[1 + i]} = (1 - p_0)\left[1 + \lfloor \log_2 l \rfloor + \frac{p_0^k}{1 - p_0^l} + R_{\mathrm{Huff}}(\mathrm{NZ})\right].$$

In Figures 1 and 2 we compare the performance of the ZGH and standard Huffman codes for Gaussian and Laplacian sources. As might be expected, the ZGH performs better than Huffman coding alone at low rates but may perform somewhat worse everywhere else.

3.2 Overhead

To use ZGH block-adaptively, the nonzero values in the sequence are transmitted using the block-adaptive Huffman code with OH2 used to transmit the overhead bits. For the runlength portion code, overhead is required only to identify the GVH parameter l to the decoder.

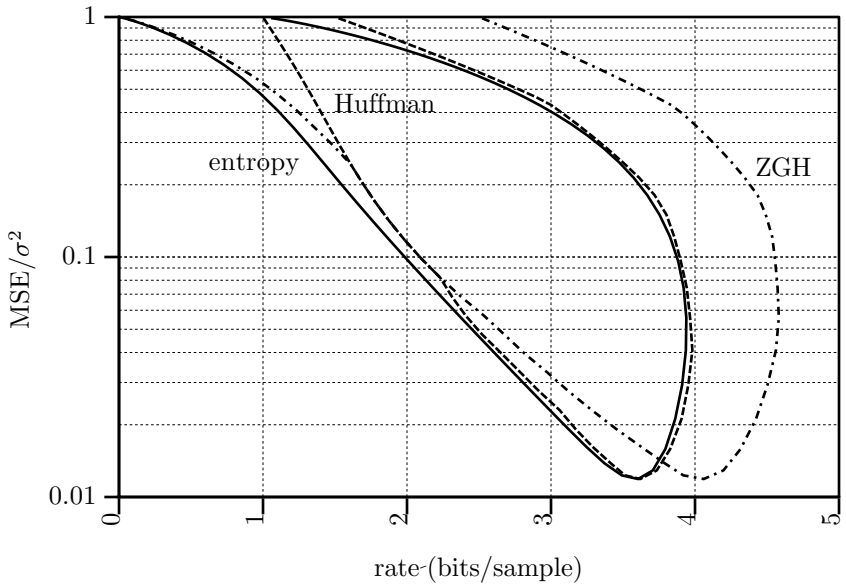

Figure 1: Rate-Distortion, Huffman and ZGH coding for Gaussian source, $b = 4$.

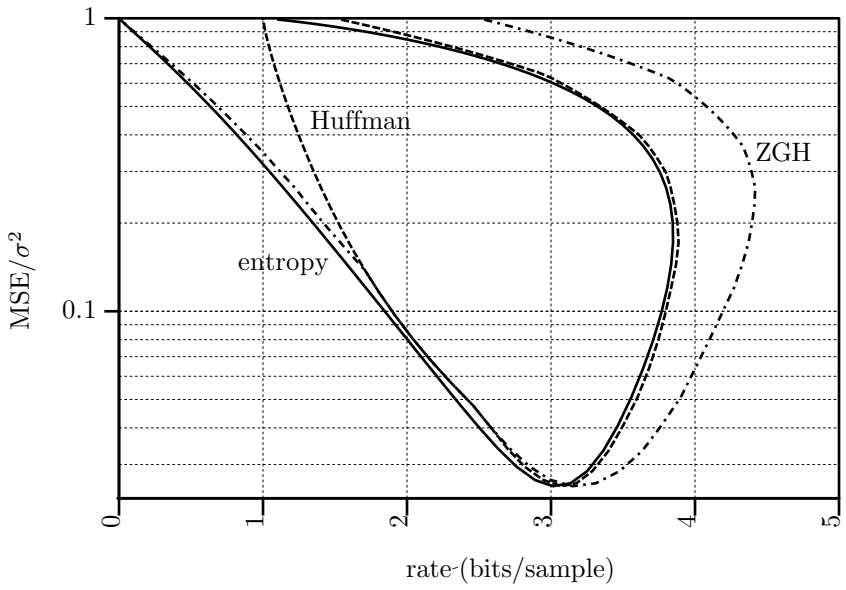

Figure 2: Rate-Distortion, Huffman and ZGH coding for Laplacian source, $b = 4$.

To compute l (see (3)), as an estimate of p_0 we use $\hat{p_0} = I/N$ where I is the number of zeros occurring in the sequence. Since I must be an integer, only certain values of l can be encoded. For $I = N$ we have $l = \infty$, which corresponds to the all zeros sequence, we reserve an extra overhead symbol for it, so that we require only the GVH overhead to transmit the entire sequence. The remaining possible values of l can be counted in the obvious way. See [2] for more details on GVH implementation. For $N = 256$, we require 5 bits of overhead, for $N = 512$, we require 6 bits. Thus we pay a small price in overhead in exchange for lowered rate when sequences contain long runs of zeros.

4 Conclusion

In Figure 3 we plot the estimated and simulated performance of block-adaptive Huffman coding for $N = 256$ and 512 on IID Gaussian and Laplacian sources. The estimated rate uses the technique described in Section 2.2. The simulations use twenty iterations for each point. In each case, OH2 outperforms OH1. In Figure 4 we also plot the performance of bit-wise arithmetic coding for the same sources.

At low rate, long runs of zeros are common, and ZGH efficiently exploits these long runs with low overhead. Consequently ZGH performs best at very low rates. The overhead of ZGH, however, is quite variable and at higher rates the overhead becomes substantial and ZGH does not perform as well as conventional Huffman coding. By contrast, bit-wise arithmetic coding has overhead that is nearly constant over all rates. In a practical adaptive system that uses zero-runlength coding, we would want a method for determining when to switch to conventional Huffman coding. Bit-wise arithmetic coding has the advantage of low redundancy and overhead over all rates, without requiring knowledge of the probability of runs of zeros.

Refinements to source-coding schemes often amount to meta-coding: an additional code layer is added to describe existing lower layers. For example, block-adaptive Huffman coding adds the layer of overhead bits that describe the Huffman code to be used. The refinement of OH2 adds a layer describing the overhead bits. A system that can optionally use zero-runlength encoding would require additional meta-coding to identify whether this option is exercised. In exchange for added complexity, increased meta-coding can offer lower rate, more flexibility, and the ability to trade-off rate between layers.

However, a drawback of extensive meta-coding is that each higher code layer becomes increasingly important: successful decoding of any of the lower code layers is unlikely unless all higher layers are correctly decoded. If the compressed data is to be transmitted over a noisy channel, this suggests that each higher layer should have increased error protection. This in turn would likely require more complex encoding and decoding schemes. Further, added error protection results in a lower channel code rate, which erodes the compression gained by additional meta-coding.

References

[1] Y. S. Abu-Mostafa and R. J. McEliece, "Maximal Codeword Lengths in Huffman Codes," *TDA Progress Reports* 42-110, April-June 1992, pp. 188-193, August 15, 1992.

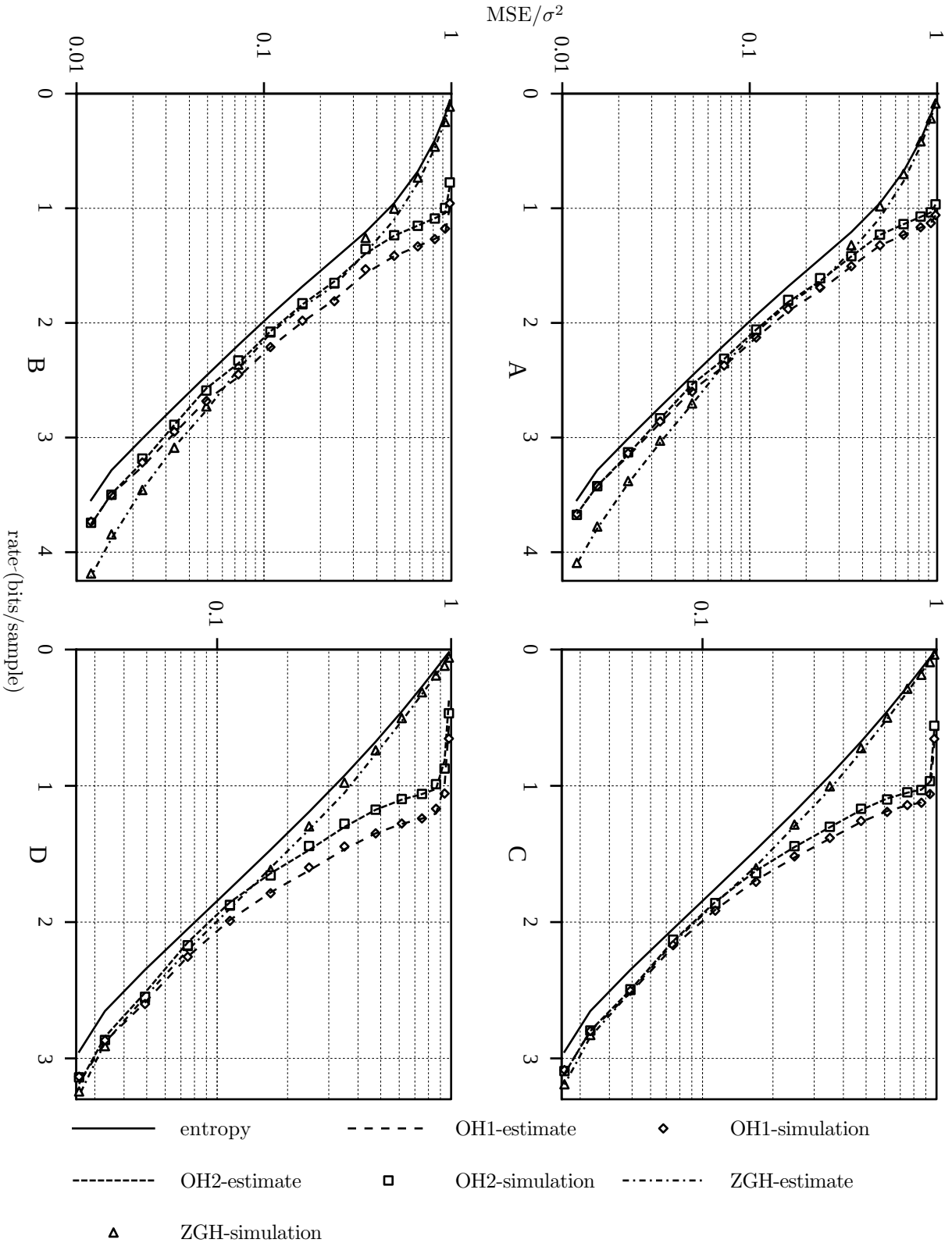

Figure 3: Huffman coding performance for OH1, OH2 and ZGH. a) Gaussian source, $b = 4, N = 512$, b) Gaussian source, $b = 4, N = 256$, c) Laplacian source, $b = 4, N = 512$, d) Laplacian source, $b = 4, N = 256$.

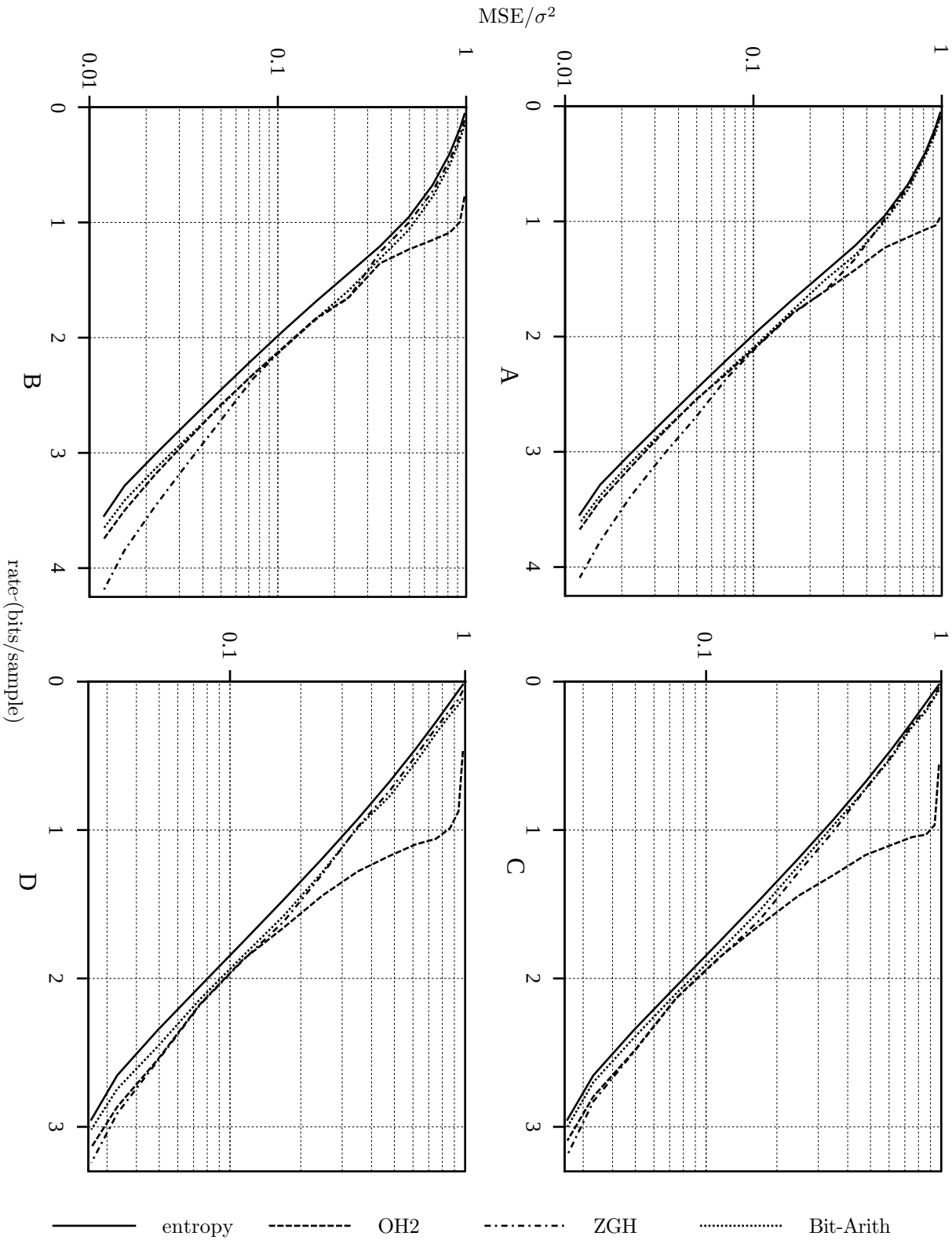

Figure 4: Huffman and bit-wise arithmetic performance, including overhead. a) Gaussian source, $b = 4, N = 512$, b) Gaussian source, $b = 4, N = 256$, c) Laplacian source, $b = 4, N = 512$, d) Laplacian source, $b = 4, N = 256$.

[2] K. Cheung, P. Smyth, and H. Wang, "A High-Speed Distortionless Predictive Image Compression Scheme," *TDA Progress Reports* 42-102, May-June 1990.

[3] R. G. Gallager and D. C. Van Voorhis, "Optimal Source Codes for Geometrically Distributed Integer Alphabets," *IEEE Trans. Inform. Theory* vol. IT-21, pp. 228-230, March 1975.

[4] R. G. Gallager, "Variations on a Theme by Huffman," *IEEE Trans. Inform. Theory* vol. IT-24, no. 6, pp. 668-674, November, 1978.

[5] A. B. Kiely, "Bit-Wise Arithmetic Coding for Data Compression," *TDA Progress Reports* 42-117, January-March 1994, pp. 145-160, May 15, 1994.

[6] R. J. McEliece and T. H. Palmatier, "Estimating the Size of Huffman Code Preambles," *TDA Progress Reports* 42-114, April-June 1993, pp. 90-95, August 15, 1993.

Distribution: Laif Swanson,
Fabrizio Pollara,
Kar-Ming Cheung,
Sam Dolinar,
Laura Ekroot,
Dan Glover,
Max Costa,
Bob McEliece