Adaptive Run-Length / Golomb-Rice Encoding of Quantized Generalized Gaussian Sources with Unknown Statistics

Henrique S. Malvar
Microsoft Research
One Microsoft Way
Redmond, WA 98052, USA

Abstract

We present a simple and efficient entropy coder that combines run-length and Golomb-Rice encoders. The encoder automatically switches between the two modes according to simple rules that adjust the encoding parameters based on the previous output codeword, and the decoder tracks such changes. This adaptive Run-Length/Golomb-Rice (RLGR) coder has a fast learning rate, making it suitable for many practical applications, which usually involve encoding small source blocks. We study the encoding of generalized Gaussian (GG) sources after quantization with uniform scalar quantizers with deadzone, which are good source models in multimedia data compression, for example. We show that, for a wide range of source parameters, the RLGR encoder has a performance close to that of the optimal Golomb-Rice and Exp-Golomb coders designed with knowledge of the source statistics, and in some cases the RLGR coder improves coding efficiency by 20% or more.

1. Introduction

Multimedia data compression has been a topic of increased interest. Most compression systems for images, video, audio and speech use adaptive predictors and/or decorrelating transforms, coupled with quantizers, to map blocks of the original data into low-entropy blocks of integers that are suitable for entropy coding [1]. Typical entropy coders used in such systems include Run-Length, Huffman, arithmetic, and variations of Golomb-Rice coders [2], [3].

One of the main problems in designing efficient entropy coders for such applications is that typically there are large variations in the statistics of blocks of integers to be encoded. Studies have shown that in most cases the data prior to quantization have probability distributions that can be significantly more concentrated near zero than a Gaussian distribution. For image and audio compression, the distributions are well modeled by generalized Gaussian distributions with parameter varying from $\nu = 0.5$ (more peaked at the origin) to $\nu = 1$ (Laplacian) [4], [5], although some subsets of the sources symbols (e.g. DC coefficients) may be better modeled by $\nu = 2$ (Gaussian).

Golomb-Rice (GR) coders have been used widely in modern compression systems [2], [6]. The motivation is that GR coders are optimal or nearly optimal for integer sources with two-sided geometric (TSG) distributions [7], which approximate quite closely the

distributions of uniformly quantized Laplacian sources. In other words, GR codes approximate optimal Huffman codes for such sources, with minimal loss in efficiency (typically less than 5 %). The main advantages of GR codes are that the encoding tables can be easily changed by changing a single integer parameter k, and that output codewords can be easily computed for the corresponding input symbols, so that no explicit tables are actually needed. That makes GR coders quite attractive for modern processors, for which computations are much faster than memory accesses.

For integer sources coming from scalar quantization of multimedia data with relatively large step sizes, the distribution of the integer data to be encoded can be significantly more peaked around zero than a Laplacian, especially if the quantizer has a deadzone [8]. That motivates the search for an entropy encoder that can handle such distributions.

In this paper we consider the encoding of a broader class of sources than the TSG, namely those generated by uniform deadzone quantization of continuous sources with generalized Gaussian (GG) distribution. We propose a Run-Length/Golomb-Rice (RLGR) coder that performs quite well for a broad range of GG distributions and quantization parameters. These parameters do not need to be known or estimated, because the RLGR coder uses a backward adaptation strategy that automatically adjusts its own parameters to nearly optimal values for any source in the class. In Section 2 we discuss the GG distribution and deadzone quantizer models, and in Section 3 we describe the RLGR coder. In Section 4 we discuss the performance of the RLGR coder by comparing it to the GR and Exp-Golomb (EG) encoders. We conclude by noting the that RLGR encoder's efficiency is close to that of GR and EG coders, with fast adaptation to source statistics and better performance for more peaked distributions.

2. Quantized generalized Gaussian sources

The unit-variance generalized Gaussian (GG) distribution is defined by the following probability density function (PDF):

$$f(x) = \frac{\nu\eta(\nu)}{2\Gamma(1/\nu)} \exp\left[-\left(\eta(\nu)|x|\right)^{\nu} \right] \tag{1}$$

where ν is a shape parameter, $\Gamma(\cdot)$ is the Gamma function, and

$$\eta(\nu) \triangleq \sqrt{\Gamma(3/\nu)/\Gamma(1/\nu)} \tag{2}$$

For $\nu = 2$ the GG PDF is a Gaussian, and for $\nu = 1$ it a Laplacian. For $\nu < 1$ the distribution gets more peaked at $x = 0$ and has longer tails.

As discussed in the Introduction, the GG distribution can be quite useful in modeling transform coefficients in multimedia compression. In lossy compression, entropy is reduced by quantizing those coefficients, usually with a uniform scalar quantizer, as shown in Fig. 1. The quantizer maps input values within each bin to integers …, –2, –1, 0, 1, 2, … Note that the center bin can be made wider (the deadzone) by setting $\alpha > 1$. The reason for that deadzone is that it approximates optimal entropy-constrained quantizers for GG sources [8]. In practice, usually the central quantization bin varies in width from δ to 2δ, that is $0 \leq \alpha \leq 1$.

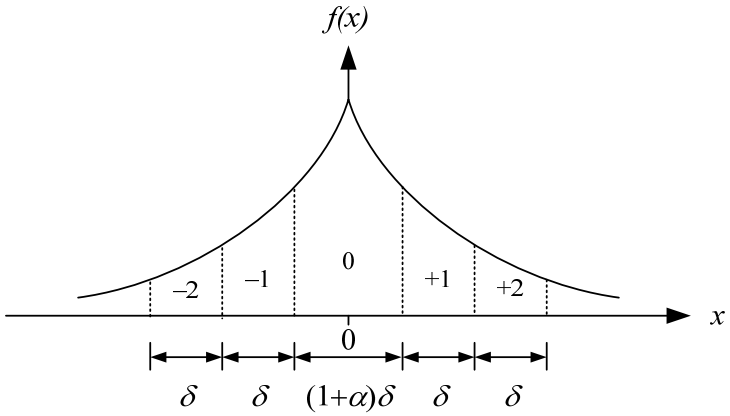

Figure 1. The generalized Gaussian PDF and the uniform quantization bins with step size δ. Parameter α controls the width of the central bin, generating a "deadzone" (no deadzone if $\alpha = 0$).

Let us call r the output of the quantizer, which is the integer variable to be encoded. From Fig. 1 we see that the probability distribution of r can be computed by

$$P(r=k)=\begin{cases}\displaystyle\int_{-(1+\alpha)\delta/2}^{(1+\alpha)\delta/2} f(x)\,dx, & k=0\\ \displaystyle\int_{(2|k|-1+\alpha)\delta/2}^{(2|k|+1+\alpha)\delta/2} f(x)\,dx, & k\neq 0\end{cases} \tag{3}$$

Given the probabilities above, we can compute the entropy of the integer source and the average code length of any prefix encoder by

$$H=\sum_r -P(r)\log_2\left(P(r)\right),\quad L_{av}=\sum_r P(r)\,L(r) \tag{4}$$

where $L(r)$ is the length of the codeword associated with input symbol r. We then define the code efficiency as the ratio H/L_{av} (e.g. if $H/L_{av}=1/2$ the efficiency is 50%).

GR coders are designed for encoding nonnegative integers, since they were originally designed to encode run lengths from binary memoryless sources [9]; the GR coder with parameter k is defined by the encoding rule in Table 1. We need to map the source of integers r generated by the quantizer to an equivalent source generating nonnegative integers u. That can be done by the folding and interleaving mapping [7]

$$u=M(r)=\begin{cases}2r, & r\geq 0\\ 2|r|-1, & r<0\end{cases} \tag{5}$$

Input value	Output codeword
u	$G(u,k) = \underbrace{111\cdots110}_{\substack{\text{prefix,}\\ p \text{ bits}}} \underbrace{b_{k-1}b_{k-2}\cdots b_0}_{\substack{\text{suffix } z,\\ k \text{ bits}}}$

Table 1. Golomb-Rice (GR) encoder with parameter k. The prefix length is given by $p = \lfloor u/2^k \rfloor$, and the suffix value z is the remainder of $u/2^k$.

That mapping converts a Laplacian input r to an output u whose probability distribution is nearly exponential, for which the Golomb-Rice coder is optimal or nearly optimal [7], depending on the quantization step size δ. As discussed in [7], the efficiency of the code can be increased by constructing the truly optimal code, which for some ranges of PDF parameters replaces the GR coder by one of three additional Huffman coders, whose code trees are variations of that of the GR coder. Note that from Eqns. (3)–(5) it is easy to compute the efficiency of the GR coder for any set of parameters $\{\nu, \delta, \alpha\}$.

Although the GR coder is nearly optimal for GG sources with $\nu = 1$ (Laplacian), they are not robust to mismatches between the assumed PDF of u and the actual one. In other words, efficiency decays rapidly if a suboptimal k is used, or if $\nu << 1$. To alleviate these problems, modifications in the codeword generation rule have been proposed [10]–[12]; these correspond to changes in the intrinsic Huffman tree. A particularly simple and useful rule is that of the Exp-Golomb (EG) coders [10]. Whereas in the GR coder the number of codewords of any length is constant, for the EG coder the number of codewords of a particular length grows exponentially. We discuss the performances of the GR and EG coders for the quantized GG source in Section 4.

3. The RLGR coder

Our main goal with the RLGR coder is to define an adaptive encoding rule in which the GR parameter k is adjusted based on the actual data to be encoded. Motivated by our previous work in [6], we use a backward adaptation rule, that is, a rule that depends only on output codewords. Other adaptive GR coders use a forward adaptation rule [2], in which source statistics over the previous N symbols are computed, and then the corresponding optimal k is used to encode the next block of N symbols. The main disadvantages of forward adaptation are a delay in adaptation (a block is encoded using k computed from a previous block, but that delay can be reduced via recursive estimation), and the fact that computation of the optimal k is only valid if the shape of the actual PDF of the data matches the model. As we show in Section 4, the backward adaptation rule performs well in terms of setting good (but not necessarily optimal) values for k for a broad range of values for the parameters $\{\nu, \delta, \alpha\}$. That is particularly important for changes in the deadzone parameter α, because as we increase α we increase the probability of zero, altering significantly the shape of the PDF.

Additionally, the RLGR coder addresses the cases of large quantization steps or small values of the GG parameter ν by adding a run-length mode that encodes more efficiently

$k = 0$ "no run" mode	input symbol = u	code = $GR(u, k_R)$
$k > 0$ run mode	string of m symbols $u = 0$, with $m = 2^k$	code = 0
	string of m symbols $u = 0$ (with $m < 2^k$) followed by symbol $u \neq 0$	code = 1 + bin(m, k) + $GR(u - 1, k_R)$

Table 2. Run-Length/Golomb-Rice (RLGR) encoder with parameters k and k_R. bin(m, k) means the binary representation of m using k bits.

strings of zeros. As we can see in Table 2, the encoding rule of the proposed RLGR coder is quite simple. There are now two parameters, k and k_R. When $k = 0$, the RLGR coder is identical to a GR coder with parameter k_R. When $k > 0$, the RLGR coder operates in run mode, where input strings of the form 000...u are mapped to an output codeword that equals either a single bit 0 for a complete run of 2^k zero symbols, or a longer string that specifies a partial run length and the nonzero value that follows. The run mode improves performance for PDFs that are more peaked around $x = 0$ than a Laplacian, such as those corresponding to a large quantization steps size (e.g. $\delta \approx 1$), a significant quantization deadzone ($\alpha > 0$), or a low GG shape parameter ($\nu < 1$).

3.1. Adaptation

To encode a quantized GG source with the RLGR encoder, we need to find the appropriate values of the RLGR parameters k and k_R. It would be extremely difficult to find analytic formulas for computing these parameters for a given set of source parameters $\{\nu, \delta, \alpha\}$. Plus, in practice, estimating these parameters for a given data block may also be difficult. Thus we use the backward adaptive rule described below. It is inspired on the backward adaptation ideas in [6]: basically, if long output codewords are generated, the RLGR parameters are adjusted upwards (which tends to reduce the codeword length for large input symbol values), and vice versa.

After encoding an input symbol or an input string using a code $GR(u, k_R)$, as indicated in Table 2, we adapt the GR parameter k_R using the rules in Table 3, where $k_{RP} = L\, k_R$, with L equal to a power of 2 (e.g. $L = 4$), so that $k_R = k_{RP} >> \log_2(L)$, and $p = \left\lfloor u / 2^{k_R} \right\rfloor = u >> k_R$ (where >> denotes a right-shift operation). Note that this provides for "fractional adaptation" of k and k_R in a way that is exactly reproducible at the decoder, similar to [6]. To adjust the RLGR main parameter k, we use the rule in Table 4, where $k_P = L\, k$, again with L equal to a power of 2, so that $k = k_P >> \log_2(L)$. The parameters $\{U_0, D_0, U_1, D_1\}$ control the speed of adaptation; in practice a good set of values is $\{U_0, D_0, U_1, D_1\} = \{3, 1, 2, 1\}$, with $L = 4$.

$p = 0$	decrease k_R by setting $k_{RP} \leftarrow k_{RP} - 2$
$p = 1$	no change in k_R
$p > 1$	increase k_R by setting $k_{RP} \leftarrow k_{RP} + p + 1$

Table 3. Adaptation rule for the RLGR parameter k_R, using fractional adaptation. It is applied immediately after the encoder outputs a codeword $GR(u, k_R)$, with $p = \left\lfloor u / 2^{k_R} \right\rfloor$.

$k = 0$	$u = 0 : k_P \leftarrow k_P + U_0$	
	$u > 0 : k_P \leftarrow k_P - D_0$	
$k > 0$	complete run	$k_P \leftarrow k_P + U_1$
	partial run	$k_P \leftarrow k_P - D_1$

Table 4. Adaptation rule for the RLGR main parameter k, using fractional adaptation. It is applied immediately after the encoder outputs a full codeword; the new value of k is $\lfloor k_P / L \rfloor$.

4. Performance

In Figs. 2–4 we compare the efficiency (entropy / average code length) of our new RLGR coder to those of GR and EG coders, for a wide range of quantization step sizes (from "high fidelity" $\delta = 0.01$ to "low fidelity" $\delta = 1$) and four values for the GG parameter ν. Each figure corresponds to a different value for the deadzone parameter, from $\alpha = 0$ (no deadzone) to $\alpha = 1$, which corresponds to the largest deadzone typically used in practice (the center quantization bin has twice the width δ of the other quantization steps).

We can make several interesting observations from the efficiency plots in Figs. 2–4. We note that GR coders perform almost as well for Gaussian sources ($\nu = 2$) as they do for Laplacian sources ($\nu = 1$). In both cases, though, the curves for each k are relatively narrow, highlighting that efficiencies above 90% can be obtained only if the optimal value for the GR parameter k is chosen.

The EG coders don't perform as well as the GR coders if the PDF is not heavily peaked, e.g. in the Laplacian and Gaussian cases. Fore heavily peaked sources ($\nu < 0.5$), though, the EG coder efficiency surpasses that of the GR coder. For example, in Fig. 2 for $\nu = 0.25$ and $\delta \leq 0.4$, the EG coder can maintain efficiencies above 90%, whereas the efficiency of the GR code can drop below 80%. That's expected, since the EG code lengths don't grow as fast as those of the GR coder, for large values of the input symbol. The loss in efficiency in the EG codes for Laplacian sources ($\nu = 1$) is compensated by reduced sensitivity to picking the right value for the parameter k. For example, in Fig. 2, if we consider the range of step sizes for which the optimal GR code has parameter $k = 4$,

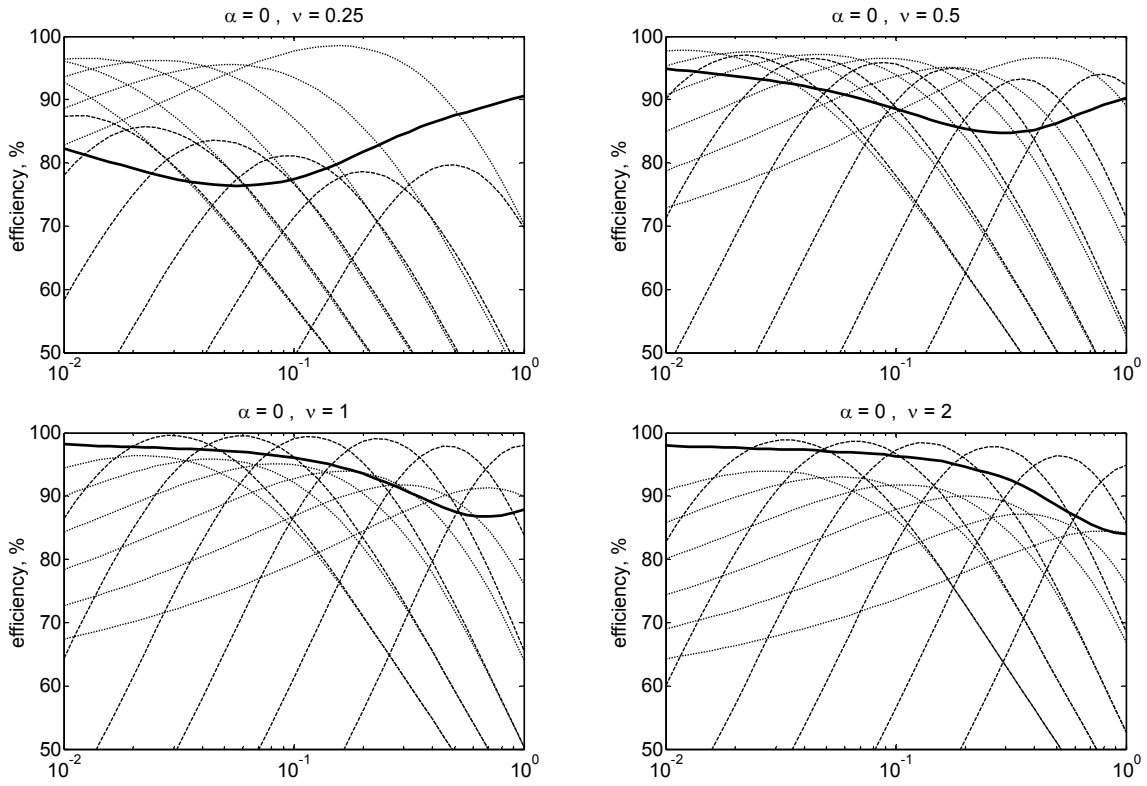

Figure 2. Code efficiency as a function of the quantization step size δ varying from 0.01 to 1.0, for a generalized Gaussian source with parameter $\nu = 0.25$, 0.5, 1 (Laplacian), and 2 (Gaussian), with a quantization deadzone parameter $\alpha = 0$. Solid line: RLGR coder; dashed lines: GR coder with parameter varying from $k = 0$ (rightmost curve) to $k = 5$; dotted lines: EG coder with parameter varying from $k = 0$ (rightmost curve) to $k = 5$.

if we use a GR coder with $k = 4$ we get efficiencies in excess of 97%, but if we use $k = 1$ the efficiency drops below 50%. For the EG code, if we also consider the range of step sizes for which the optimal EG code has parameter $k = 4$, if we use an EG coder with $k = 4$ the efficiency is high, about 95%, but if we use $k = 1$ the efficiency drops to about 80%, a better scenario than with the GR coder.

The efficiency of the RLGR coder is comparable to that of the GR coder, but not quite as high. Even in cases where the RLGR coder operates exclusively in "no run" mode (e.g. for very small quantization step sizes), its efficiency is always strictly below that of the optimal GR coder. That's because the adaptation rule in Table 3 forces the GR parameter to oscillate, typically between the optimal value and that value +/– 1, reducing efficiency. However, the penalty is usually small. For example, in Fig. 2 for the Laplacian case ($\nu = 1$) and $\delta \leq 0.2$, the RLGR coder maintains an efficiency of 95% or better. As expected, the main advantage of the RLGR coder is when the probability of the symbol zero increases significantly, e.g. in Fig. 4 for $\alpha = 1$, $\nu = 0.5$ and $\delta = 1$; that's because in those cases the RLGR coder operates mostly in run mode.

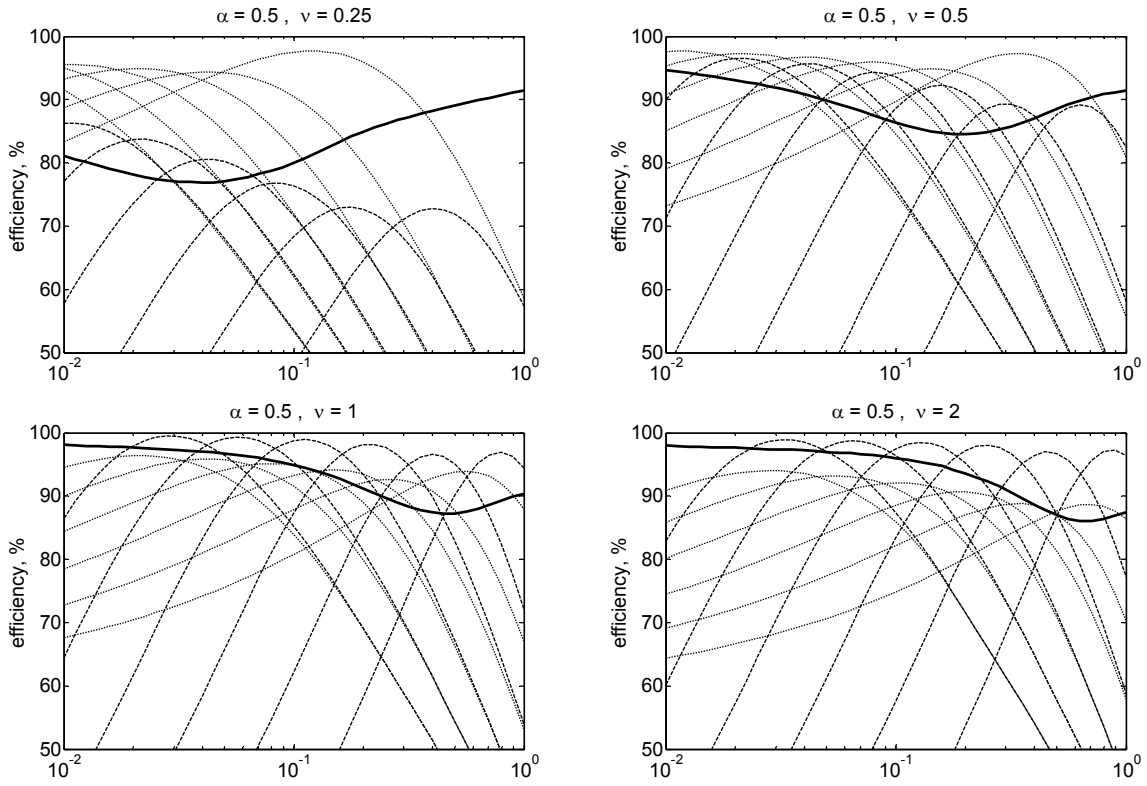

Figure 3. Code efficiency as a function of the quantization step size δ varying from 0.01 to 1.0, for a generalized Gaussian source with parameter ν = 0.25, 0.5, 1 (Laplacian), and 2 (Gaussian), with a quantization deadzone parameter α = 0.5. Solid line: RLGR coder; dashed lines: GR coder with parameter varying from k = 0 (rightmost curve) to k = 5; dotted lines: EG coder with parameter varying from k = 0 (rightmost curve) to k = 5.

We see that the RLGR coder performs reasonably well for all source and quantization parameter values, except for very low values of the GG parameter ν (e.g. ν = 0.25). As we discussed before, for multimedia applications the data to be encoded usually has $\nu \geq$ 0.5, for which the loss in efficiency of the RLGR coder in comparison to the GR coder is typically less then 5%, and in only a few regions it gets to 10% or so. On the other hand, for high quantization steps ($\delta \approx 1$) and significant deadzones ($\alpha > 0.5$), the RLGR coder can provide higher efficiency than the GR coder, by as much as 20%, especially for $\nu \approx 0.5$.

The RLGR coder adapts to new data quickly, as shown in Fig. 5, where we encode a block of 400 symbols. The first 200 symbols are from a quantized GG with ν = 2, α = 0.5, and δ = 0.1, and the next 200 corresponds to ν = 0.5, α = 0.5, and δ = 0.5. We see from Fig. 5 that it takes only about 30 symbols or so for the adaptation rule to move the parameters to their appropriate values for the new set of source parameters.

Finally, we recall that our adaptation rules work well, but they are not necessarily optimal; it may be possible to derive better rules that will reduce the efficiency gaps.

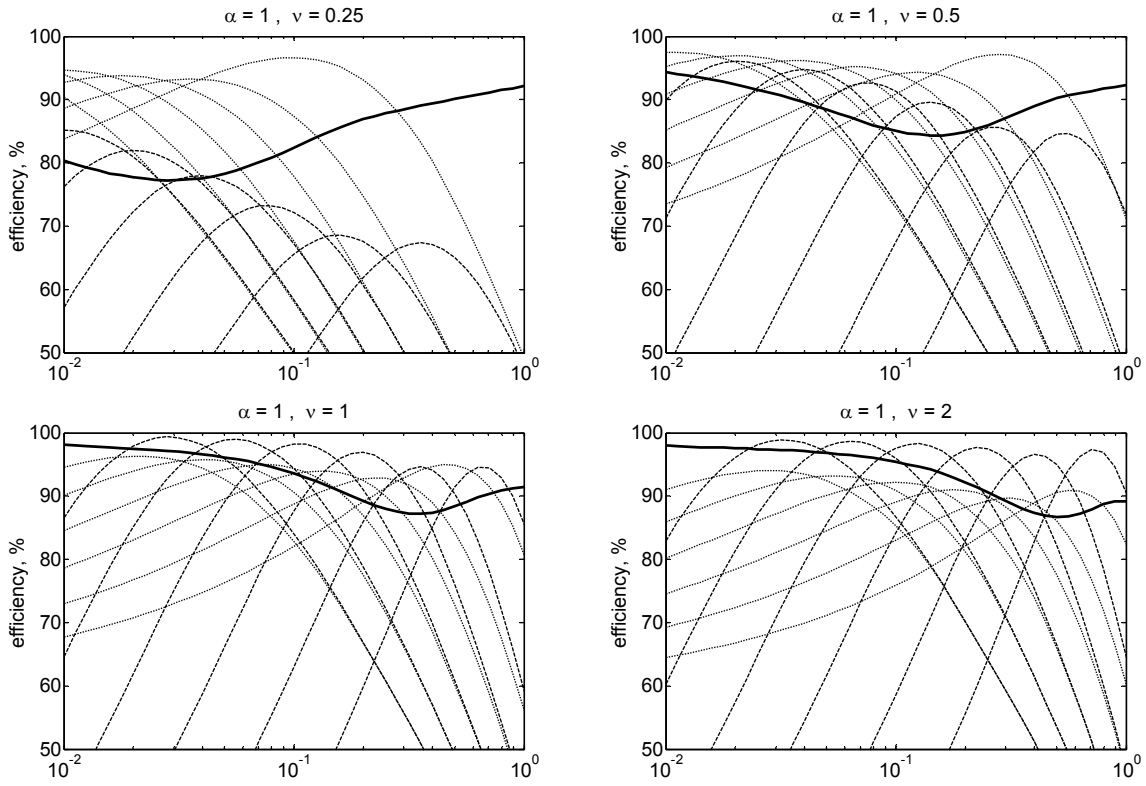

Figure 4. Code efficiency as a function of the quantization step size δ varying from 0.01 to 1.0, for a generalized Gaussian source with parameter ν = 0.25, 0.5, 1 (Laplacian), and 2 (Gaussian), with a quantization deadzone parameter α = 1. Solid line: RLGR coder; dashed lines: GR coder with parameter varying from k = 0 (rightmost curve) to k = 5; dotted lines: EG coder with parameter varying from k = 0 (rightmost curve) to k = 5.

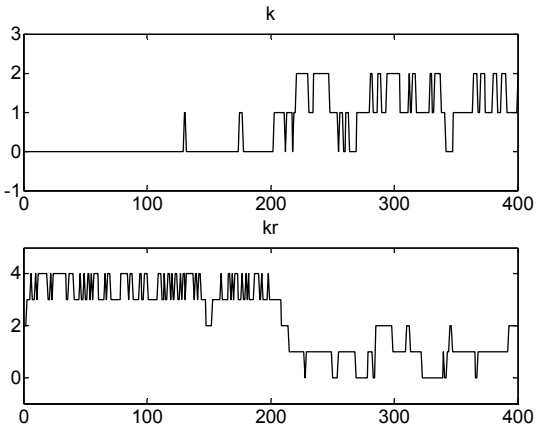

Figure 5. Fast learning rate for the RLGR coder. The source emits 400 symbols, and source and quantization parameters change at symbol 200. The RLGR parameters k and k_R quickly adapt to the new source statistics.

5. Conclusion

We have presented an entropy coder that combines run-length and Golomb-Rice encoding and uses simple backward-adaptive rules. The RLGR coder can be efficiently implemented in modern processors, and has coding efficiencies that are close (within 5% to 10%) to that of optimal Golomb-Rice coders, for a wide range of input sources that can be modeled as scalar quantized symbols from generalized Gaussian sources. The run mode of the RLGR coder increases its efficiency for sources with PDFs highly peaked at the origin, such as in low bit-rate coding or other scenarios that are appropriately modeled by generalized Gaussians with low shape parameter. The main advantage of the RLGR coder is that no estimation of source PDF parameters is needed, thanks to the backward adaptation rule.

References

[1] K. Sayood, *Introduction to Data Compression.* Morgan Kaufmann, 1996.

[2] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS," *IEEE Trans. Image Processing*, vol. 9, pp. 1309–1324, Aug. 2000.

[3] I. E. G. Richardson, *H.264 and MPEG-4 Video Compression, Video Coding for Next-Generation Multimedia.* Wiley, 2003.

[4] R. L. Joshi and T. R. Fischer, "Comparison of generalized Gaussian and Laplacian modeling in DCT image coding," *IEEE Signal Processing Letters,* vol. 2, pp. 81–82, May 1995.

[5] R. Yu, X. Lin, S. Rahardja, and C. C. Ko, "A statistics study of the MDCT coefficient distribution for audio," *Proc. IEEE Int. Conf. on Multimedia and Expo,* Taipei, Taiwan, vol. 2, pp. 1483–1486, June 2004.

[6] H. S. Malvar, "Fast progressive wavelet coding," *Proc. Data Compression Conf.*, Snowbird, UT, pp. 336–343, Mar. 1999.

[7] N. Merhav, G. Seroussi, and M. J. Weinberger, "Optimal prefix codes for sources with two-sided geometric distributions," *IEEE Trans. on Information Theory,* vol. 46, pp. 121–135, Jan. 2000.

[8] G. J. Sullivan, "On embedded scalar quantization," *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing,* Montreal, Canada, vol. 4, pp. 605–608, May 2004.

[9] R. G. Gallager and D. C. van Voorhis, "Optimal source codes for geometrically distributed integer alphabets," *IEEE Trans. Information Theory*, vol. IT-21, pp. 228–230, Mar. 1975.

[10] J. Teuhola, "A compression method for clustered bit-vectors," *Information Processing Letters*, vol. 7, pp. 308–311, Oct. 1978.

[11] S. Xue and B. Oelmann "Hybrid Golomb codes for a group of quantised GG sources," *IEE Proc. – Vision, Image and Signal Processing,* vol. 150, pp. 256–260, Aug. 2003.

[12] J. Wen and J. D. Villasenor, "Structured prefix codes for quantized low-shape-parameter generalized Gaussian sources," *IEEE Trans. on Information Theory,* vol. 45, pp. 1307–1314, May 1999.