Lossless Compression Performance of a Simple Counter-Based Entropy Coder

Armein Z. R. Langi [1,2]

[1]ITB Research Center on Information and Communication Technology
[2]Information Technology RG, School of Electrical Engineering and Informatics
Institut Teknologi Bandung, Jalan Ganeca 10, Bandung, 40116, Indonesia
Email: armein.z.r.langi@stei.itb.ac.id

Abstract. This paper describes the performance of a simple counter based entropy coder, as compared to other entropy coders, especially Huffman coder. Lossless data compression, such as Huffman coder and arithmetic coder, are designed to perform well over a wide range of data entropy. As a result, the coders require significant computational resources that could be the bottleneck of a compression implementation performance. In contrast, counter-based coders are designed to be optimal on a limited entropy range only. This paper shows the encoding and decoding process of counter-based coder can be simple and fast, very suitable for hardware and software implementations. It also reports that the performance of the designed coder is comparable to that of a much more complex Huffman coder.

Keywords: *entropy coder; counter-based coder; lossless compression; Rice coders.*

1 Introduction

Compression schemes reduce the number of bits required to represent signals and data [1]. Image communication and storage are examples of applications that benefit from image compression, because the compression results in (i) faster (real–time) image transmission through bandlimited channels and (ii) lower requirements for storage space. Noncompressed images require many data bits, making it impossible for real–time transmission through bandlimited channels—such as 48 kbit per second (kpbs) and 112 kbps integrated services digital network (ISDN), as well as 9.6 kbps voice–grade telephone or radio channels [1]. For example, transmitting a 256×256 pixels grayscale still–image over the voice–grade line would require at least 54.61 s. Furthermore, one HDTV format needs 60 frames of 1280×720 pixels per second. Using 24 bpp colour pixels, this HDTV format would require an impractical channel capacity of 1,440 Mbps. For example, space explorations by National Aeronautics and Space Administration (NASA) missions generate huge science data, requiring real-time compression [2]. Consequently, international bodies such as Committee for Space Data Systems (CCDS) have defined compression standards for real-time systems [3].

Received December 22nd, 2010, Revised November 17th, 2011, Accepted for publication November 17th, 2011.
Copyright © 2011 Published by LPPM ITB, ISSN: 1978-3086, DOI: 10.5614/itbj.ict.2011.5.3.2

In signal and data compression cases, lossless compression in forms of entropy coders is always needed. As shown in Figure 1, a value quantization block converts any input samples into quantized values according to a prescribed target of compression ratio (CR) [4]. A lossless entropy encoder compresses quantized values to obtain the desired efficient representations (bitstreams). The quantized values can be recovered by inverting encoding process at the decoder. An entropy decoder recovers the values losslessly. A value dequantization block reconstructs the signal samples.

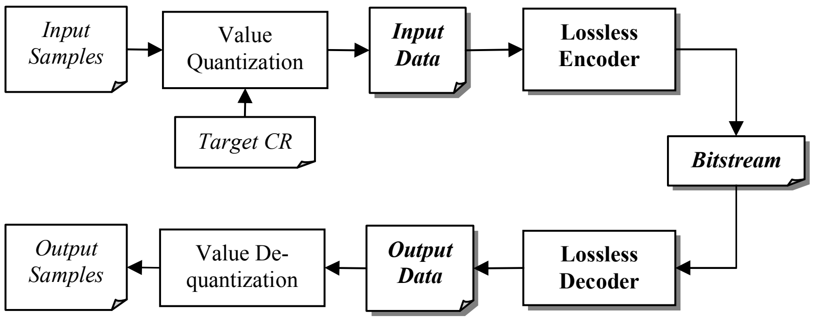

Figure 1 A typical use of lossless coder in an image compression system.

Since there are fast algorithms for value quantization, such as wavelet scalar quantization (WSQ) [5], the entropy coder is usually the time–performance bottleneck. Entropy coders such as zero run length (ZRL) coders and Huffman coders must perform statistical modeling of data distribution and assign shorter length codewords to samples that occur more often. This in effect reduces the average of representation bits, thus achieving on average a compression.

If the codeword table does not have to be transmitted, Huffman code is the shortest length code for statistically independent data. However, the Huffman coder is designed to obtain optimal performance for all range of entropy values [4]. As a result, the encoding cost is high because the coder must create a table of codewords for input data through calculation of data distribution. Such processing can be very time consuming.

This paper then proposes to use counter-based coders [2] because such coders are much simpler. Counter based coder consists of a set of subcoders operating at specific range of entropy values (statistics). Such subcoders can be designed to be optimal only for a narrow range of entropy [2]. Each subcoder is simply a counter, counting number of zeros or ones in a data stream. As a result, the encoder and decoder are very simple and fast.

However we need to study the performance of counter-based coder in practice in the context of transform coding, since the work in [2] was for differential pulse code modulation schemes. Furthermore, Rice coders have always been thought to be a special purpose coder, as compare to Huffman coder. We study and design of such counter coders to be used in transform coder compression scheme. The entropy range of the wavelet coefficients can be used to design a particular Rice coder. We can then have a performance comparable to that of Huffman coder with a fraction of computational costs [6].

2 Entropy Coders

In principle, an entropy coder is a coder that allocates a number of bits to a codeword proportionally to an information value of the codeword [4].

2.1 First Order Entropy

Let us assume that input data consist of a sequence of fixed length codewords coming from a zero-memory information source (i.e., a source that produces statistically independent codewords). The source has an alphabet S having symbols $\{s_1, s_2, \ldots, s_q\}$. For a particular set of input data, the probabilities of symbol occurrences are

$$P(s_1),\ P(s_2),\ \ldots,\ P(s_q) \tag{1}$$

Satisfying

$$\sum_{i=1}^{q} P(s_i) = 1 \tag{2}$$

If a symbol s_i rarely occurs (i.e., $P(s_i)$ is very small), its occurrence has a high *suprising* value. Hence symbol s_i is said to have a high amount of information $I(s_i)$, which is inversely proposional to magnitude of $P(s_i)$. If a symbol s_i occurs, we are said to have received an amount of information $I(s_i)$ from the source, defined as

$$I(s_i) = \log \frac{1}{P(s_i)} \tag{3}$$

Since the probability of receiving an amount of information $I(s_i)$ is $P(s_i)$, the *average* amount of information received for each symbol is called the first-order *entropy* of S, denoted as $H(s)$,

$$H(s) = \sum_{i=1}^{q} P(s_i) I(s_i) = \sum_{i=1}^{q} P(s_i) \log \frac{1}{P(s_i)} \tag{4}$$

In other word, the source S produces an average amount of information H for each occurrence of a symbol. If we use binary logarithm (i.e., $\log_2(1/P(s_i))$) the information amount unit is in bits.

To carry this amount of information, the source needs data resources namely codewords. Using a code W, each symbol s_i needs a distinct codeword w_i. If each codeword w_i needs l_i bits, then average bits required to transfer information from source S using code W is

$$R = \sum_{i=1}^{q} P(s_i) l_i \tag{5}$$

Since we have $I(s_i) \le l_i$, or

$$H(s) \le R \tag{6}$$

This means for a zero-memory source, the entropy is the lower bound of average bits per symbol. In other words, an entropy coder will have an efficient representation of S, i.e, a small R, but will not be smaller than the entropy. The minimum length achievable is the data entropy. Thus the performance of an entropy coder is in how close the resulting R to $H(s)$. It should be noted that it is possible to have $R < H(s)$ if the source happens to have memory. For statistically independent data, the entropy is the simply the first–order entropy [2].

Efficient lossless codes depend on statistical distributions. An entropy coder allocates a number of bits proportional to the amount of information. A rarely occurring symbol will get more bits. Highly frequent symbols get codewords with fewer bits. As a result, we can have an efficient average of bit usages.

2.2 Zero Run-Length Coder

Suppose that we have a source producing many consecutive zero symbols, i.e., $P(s_1)$ is high for s_1 = 0x00. In this case we can use ZRL, which is a very simple code. It simply counts the number of consecutive zeros in an input codeword and produces a zero symbol followed by a number representing the counting result.

For example, consider a source that uses an alphabet consisting of fix 8 bits codewords, {0x00, 0x01, 0x02, …, 0xFF}. If a ZRL encoder encounters a stream of nine input data:

(0x02, 0x01, 0x00, 0x00, 0x00, 0x00, 0x07, 0x00, 0x7F) (7)

it will produce a stream of eight output data

(0x02, 0x01, 0x00, 0x04, 0x07, 0x00, 0x01, 0x7F) (8)

Notice that there are five zeros (0x00) in the input stream, and the first four consecutive zeros are replaced by a zero symbol 0x00 and a counting result 0x04. The last zero is replaced with a zero symbol 0x00 and a counting result 0x01. Given the simple coding rule, a ZRL decoder will be able to recover the original stream from the encoder output stream easily (uniquely decodable) without any data loss.

However notice also that the number of data have been reduced from nine to eight, performing a lossless compression. This compression is effective as long as the input stream contains significant number of consecutive zeros. Otherwise, ZRL will in fact produce an expansion instead of a compression.

2.3 Huffman Coder

The variable-length Huffman coder is a well known and widely used coder because its compression ratio performance is close to optimal (i.e., the actual compression ratio is very close to the input entropy). In an optimal code, occurrences of long codewords are rare hence the number of bits is reduced. A Huffman coder maps the input samples using a table of variable-length codewords. The codewords are designed such that the most frequent data sample has the shortest codeword. Consequently, the resulting bit rate is always within H and $H+1$, regardless of the input entropy H.

A Huffman coder maintains statistical distributions of the source. The coder uses the distribution to estimate the information value of each symbol. It then creates a codeword for each symbol according to its information value. Finally it uses that codeword to represent corresponding symbol from the source.

Thus the process of a Huffman coder is usually computationally costly because it consists of two passes. The first pass reads all data samples to generate statistical distribution, to be used to allocate bits to the codewords. It allocates bits variably for codewords according to a hierarchical tree structure of symbols, in which the hierarchy reflects the importance of the symbols. The first bit is distributed to the symbol with the lowest information values. The remaining bits are distributed by traversing through the tree structure in a way that symbols with more information value receive more bits, while ensuring that the resulting code is always uniquely decodable. The second pass encodes the sample data using the resulting codeword table. This two-pass Huffman code scheme must then include the codeword table in the bitstream to allow decoder to decode data correctly.

A single-pass Huffman code is even more computationally expensive. To avoid having to send the codeword table, both encoder and decoder must maintain an adaptive codeword table. They adapt the table as each incoming data sample changes symbol distribution. As a result, both encoder and decoder must create a new tree structure and the codeword table periodically.

3 Counter-Based Coders

We then propose to use the counter coder as an alternative and more computationally efficient entropy coder. One basic counter coder called Ψ_1 (or PSI–1) works as follows [2]. Given a block of data samples (for *j* = 1, ..., *J*), Ψ_1 replaces every sample in the block with a number of consecutive zero bits '0' followed by a closing one bit '1' (see Table 1). For example, if a sample happens to have a value of 55, the Ψ_1 encodes it using 56 bits, i.e., 55 zeros followed by a one. Hence, the encoding algorithm is simple. The reconstruction is obviously simple too. A Ψ_1 decoder just counts the number of zero bits until a one appears. The counting result is the sample value.

Define now a function $i(j)$ such that a particular data block consists of symbols $\{s_{i(1)}, s_{i(2)}, \ldots, s_{i(J)}\}$. The total number of bits required to encode a data block is (for index *j* and block size *J*)

$$L = J + \sum_{j=1}^{J} x_{i(j)} \tag{9}$$

Notice that this code basically allocates more codeword bits to data samples with higher sample values, i.e., $x_i \geq x_j \leftrightarrow l_i \geq l_j$. This means Ψ_1 code is optimal if the statistical distribution of data samples is monotonically decreasing, i.e., $x_i \geq x_j \leftrightarrow P(s_i) \geq P(s_j)$. The average codeword length of Ψ_1 code is

$$R = \sum_{i=1}^{q} P(s_i) i \tag{10}$$

It has been studied elsewhere [3] that this code is optimal for a (monotonically decreasing distribution) source with a first-order entropy around 1, i.e., $1.5 < H < 2.5$. It should be clear that in a monotonically decreasing distribution, the probability of a sample has a large value is low. As a result, the probability of long codeword length Eq. (11) is low. Thus it is optimal. We say that range is the natural entropy range of Ψ_1. For sources with entropies outside that range, there are several variations of Ψ_1.

Table 1 A codeword table of Ψ_1 Code.

i	**Symbols** s_i	**Samples** x_i	**Sample Data** d_i	**Codewords** w_i	**Length** l_i
1	s_1	0	0000 0000	1	1
2	s_2	1	0000 0001	01	2
3	s_3	2	0000 0010	001	3
4	s_4	3	0000 0011	0001	4
…	…	…	…	…	…
256	s_{256}	255	1111 1111	0000…00001	256

For example, if $H > 2.5$, it is safe to assume that the LSBs of sample data $d_{i(j)}$ are completely random. In this case there is no need to perform any compression on those LSBs. We can then split $d_{i(j)}$ into two portions: *k* LSBs and (8-*k*) MSBs. The MSBs are coded using Ψ_1 code before being sent to bitstream, while the LSBs are sent uncoded. A decoder first recovers the MSBs using Ψ_1 decoder, and then concatenates the results with the uncoded LSBs, resulting in the desired $d_{i(j)}$. It can be shown that this approach has a natural entropy range $1.5 + k < H < 2.5 + k$. This code is called $\Psi_{1,k}$.

For $0.75 < H < 1.5$ range, we can first use the Ψ_1, resulting in a bitstream with many '1' bits. To exploit this, we can cascade it using another simple code. The simple code first complements the results of Ψ_1 (i.e., converting '0' into '1' and vice versa), and group the resulting bit into binary 3-tuples. It finally codes each binary 3-tuples in a way that a tuple with many '0' bits will use short codewords. This code is called Ψ_0.

For sources with entropy $H < 0.75$, we can cascade the coder with another coder, such as ZRL coder described above. A ZRL encoder processes samples, and provides its outputs to a counter coder. In effect the ZRL coder brings the data entropy into counter code natural range.

4 Compression Performance

To observe the performance of this simple counter, we have used a WSQ scheme shown in Figure 1 [5]. Here we used an image sample of Lena.pgm, having a dimension of 512×512 pixels at 8 bit per pixel (bpp) grayscale, hence a size of 2,097,152 bits. The scheme first quantizes the image according to a set of prescribed rates 0.1, 0.2, …,0.9, 1, 2, …,6, and 7 bpp. The resulting data becomes input data to a lossless encoder. In addition to the counter code, we have used a Huffman coder for a comparison. The lossless encoder produces compressed bitstreams. A lossless decoder can then reproduce output data from the bitstreams.

Given a prescribed rate, the value quantization block produces image data and then later the coder produces the bistream. We measured the size of the bitstream. The compression ratio (CR) is the ratio between the original image size (in this case 2,097,152 bits) and the bitstream size. We measured the CR for each prescribed rate for counter coder, and measure similar performance of Huffman coder for comparisons. We expect that the resulting CRs should approximate the prescribed CR.

Table 2 and Figure 2 show the results of the coders for various prescribed rates. Overall the performance of counter coder is comparable to that of Huffman coder. At very low rates Huffman Coder still outperforms counter coder. However in most cases both coders perform better than expected rates.

We can also measure the entropy the image data for each prescribed rate, and compare the CR performances relative over that of the entropy. As shown in Figure 2, for high prescribed CR (thus low entropy), both counter coder and Huffman coder are able to outperform the entropy. For high entropy (low prescribed CR), the counter coder outperforms the Huffman coder. The counter coder performance is comparable to entropy, while Huffman coder underperforms the entropy.

Table 2 A comparison of compression ratio (CR) performance.

Prescribed		CR Performance			Relative Performance Over Entropy	
Rate (bpp)	**CR**	**Entropy**	**Huffman**	**Counter**	**Huffman**	**Counter**
0.1	80.00	44.44	53.33	50.00	120%	113%
0.2	40.00	27.59	32.00	29.63	116%	107%
0.3	26.67	21.05	24.24	22.86	115%	109%
0.4	20.00	16.67	20.00	18.18	120%	109%
0.5	16.00	14.81	17.78	16.33	120%	110%
0.6	13.33	13.33	15.69	14.55	118%	109%
0.7	11.43	11.94	13.79	12.70	116%	106%
0.8	10.00	10.96	12.50	11.59	114%	106%
0.9	8.89	10.13	11.43	10.67	113%	105%
1	8.00	9.30	10.39	9.76	112%	105%
2	4.00	4.97	4.79	4.85	96%	98%
3	2.67	3.16	2.92	3.11	92%	98%
4	2.00	2.30	2.16	2.32	94%	101%
5	1.60	1.79	1.75	1.82	98%	101%
6	1.33	1.47	1.44	1.47	98%	100%
7	1.14	1.24	1.21	1.22	98%	98%

What is the image quality associated with the CR performance level? For each prescribed rate the scheme can produce a bitstream. The decoder scans then use the bitstream to produce a reconstructed image. We then compare the quality of the reconstructed image, as compared to the original image. One way to measure image quality is through a measurement of peak signal to noise ratio (PSNR). This measure compares the two images, and calculates the energy of image differences. PSNR is then the ratio of peak-value energy and that of the image energy. A PSNR level of 30 dB or more is considered good quality.

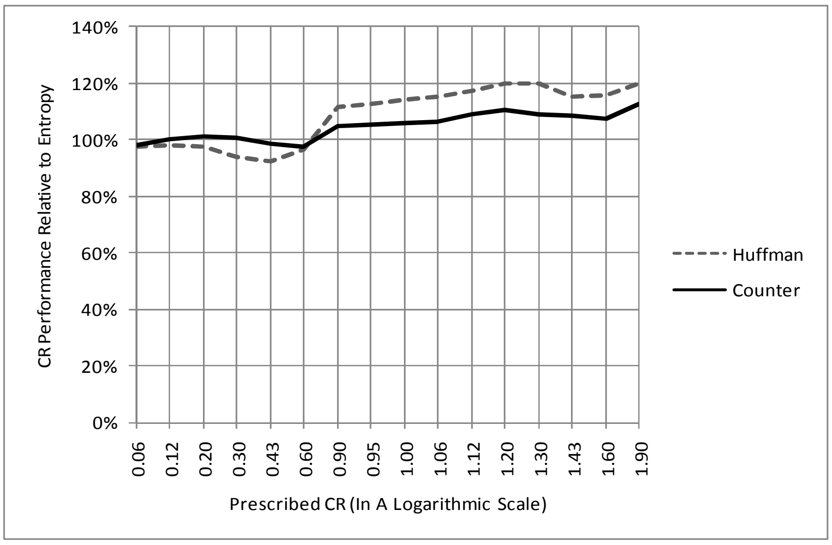

Figure 2 Compression Ratio (CR) performances of counter code as compared to Huffman coder, relative to data entropy.

Table 3 and Figure 3 show compression performances of the coders at various levels of image quality, after being combined with the WSQ scheme shown in Figure 1. The WSQ decoder uses the image data from the bitstream to generate a reconstructed image. We can then compare the reconstructed image with the original one, and the measure its quality in terms of PSNR. Here, counter coder outperforms Huffman coder for high quality compression. Huffman coder outperformed counter coder at lower image quality levels, although both outperform the entropy. Beyond 37 dB PSNR, the CRs of the coders are basically similar.

Table 3 Compression ratio performance over compression quality.

Quality PSNR (dB)	CR Entropy Coder	CR Huffman Coder	CR Counter Coder
28.06	44.4	53.33	50.00
31.56	27.6	32.00	29.63
34.14	16.7	20.00	18.18
36.42	11.0	12.50	11.59
37.24	9.3	10.39	9.76
40.95	5.0	4.79	4.85
49.67	2.3	2.16	2.32
51.14	1.5	1.44	1.47

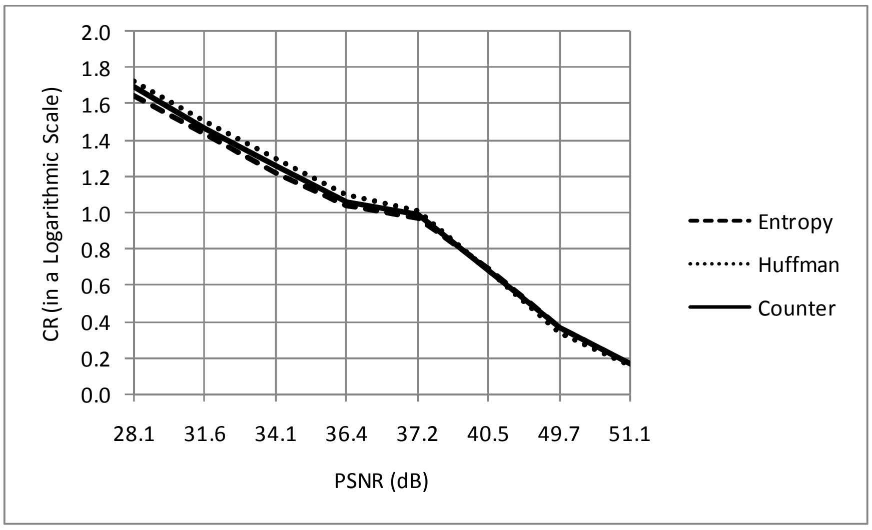

Figure 3 Compression ratio performances in a scalar quantization scheme.

5 Conclusions

Simple counter coders can be used effectively to compress images in a WSQ scheme. The counter coders are optimal for monotonically decreasing distributions. This kind of distribution is natural for indices of the WSQ. In this paper we have shown the counter coder performance, especially CR and PSNR, matches that of Huffman coder. This results in an almost optimal performance comparable to Huffman coding with a fraction of code complexity.

Acknowledgements

This work was supported in part by Riset Unggulan (RU) ITB.

Nomenclature

CR	= Compression ratio
dB	= Decibel
SNR	= Signal to noise ratio
PSNR	= Peak SNR

References

[1] Jayant, N., *Signal Compression: Technology Targets and Research Directions*, IEEE J. Selected Areas Communications, IEEE 0733-8716/92, **10**(5), pp. 796-818, July 1992.

[2] Rice, R.F., *Some Practical Universal Noiseless Coding Coding Techniques, Part III, Module PSI–14,K+*, JPL Publication 91–3, NASA, JPL California Institute of Technology, 124p, November 1991.

[3] CCSDS, *Image Data Compression*, Recommended Standard CCSDS 122.0-B-1, Consultative Committee for Space Data Systems, Nov, 2005. (available at http://public.ccsds.org, accessed 4 March 2011)

[4] Langi, A., *Review of Data Compression Methods and Algorithms*, Technical Report, DSP-RTG–2010–9, Institut Teknologi Bandung, Sep. 2010.

[5] Bradley, J.N. & Brislawn, C.N., *The Wavelet/Scalar Quantization Compression Standard for Digital Fingerprint Images*, Proc. IEEE Int. Symp. Circuits and Systems, London, May 3–June 2, 1994.

[6] Langi, A.Z.R., *An FPGA Implementation of a Simple Lossless Data Compression Coprocessor*, Proc. International Conference on Electrical Engineering and Informatics (ICEEI 2011), Bandung, July 2011.