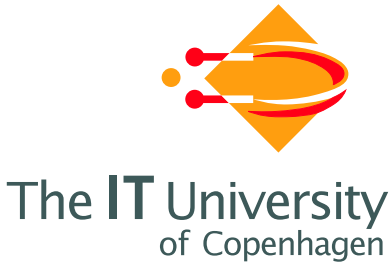

Efficient Entropy Coding for Image Compression

A Technical Report at the ITU

Alexandre Krivoulets

IT University Technical Report Series

TR-2002-TR–2002–13

ISSN 1600–6100 2 2002

ISSN 1600–6100

ISBN 87-7949-018-2

Copies may be obtained by contacting:

IT University of Copenhagen
Glentevej 67
DK-2400 Copenhagen NV
Denmark

Telephone: +45 38 16 88 88
Telefax: +45 38 16 88 99
Web www.it-c.dk

Efficient Entropy Coding Algorithms for Image Compression

Alexandre Krivoulets

Abstract

In this paper we propose entropy coding algorithms for image and video compression systems using binary decomposition. We show that this technique allows for efficient coding algorithms. We propose two decompositions that allow for reduction of the number of coding parameters by a factor of about 10 compared to the JPEG, while the compression performance is not worse. A small number of coding parameters per context (3-4) may be beneficial for multi-context algorithms and for systems with limited memory or hardware resources, e.g., for mobile communication devices.

1 Introduction

Image compression algorithms are fundamentally carried out in two steps: a mapping (or transformation) of the original image into a sequence of descriptors, being another representation of the image, followed by entropy coding of the descriptors. The transformation aims to eliminate redundant information in the image. The kind of transformation depends on the application. The most popular transforms for lossless image compression are prediction techniques and reversible discrete wavelet transform, whereas discrete cosine transform (DCT) and discrete wavelet transform are most used in lossy image and video coding techniques. Transform coefficients and prediction residuals are the examples of descriptors. Efficient entropy coding of the descriptors is essential for any compression system.

It has been observed that a probabilistic model for prediction errors and transform coefficients is well approximated by a continuous off-centered Laplacian distribution [1, 2]:

$$f(x) = \frac{\alpha}{2} \exp^{-\alpha|x-\mu|},$$

where $\alpha > 0$ and $-\infty < \mu < \infty$ are the distribution parameters.

In image compression systems, the transform coefficients are usually quantized (or rounded) using an equidistant quantizer with quantization step $\Delta > 0$ centered at zero (for rounding $\Delta = 1$). After such a quantization, the probability distribution of discrete symbols $i \in Z$ is defined as

$$P(i) = \int_{\frac{\Delta(2i-1)}{2}}^{\frac{\Delta(2i+1)}{2}} f(x)dx$$

and after a little algebra we find

$$P(i) = \begin{cases} 1 - \frac{\sqrt{\theta}}{2}\left(\theta^{\delta} + \theta^{-\delta}\right), & i = 0 \\ \frac{1}{2\sqrt{\theta}}(1-\theta)\theta^{i-\delta}, & i > 0 \\ \frac{1}{2\sqrt{\theta}}(1-\theta)\theta^{|i|+\delta}, & i < 0 \end{cases} \quad (1)$$

where $\theta = \exp^{-\alpha\Delta}$ and $\delta = \mu/\Delta$ are the new distribution parameters. Such a distribution we shall call the off-centered discrete Laplasian distribution (ODLD)[1]. The parameter θ defines the rate of decay of the distribution and δ determines the offset of its center. The introduction of the parameters θ and δ allows us to get rid of the dependence on Δ and α in the probability distribution and thereby the continuous source and quantizer parameters.

Deducing this formula, we assumed, that $-\Delta/2 \le \mu \le \Delta/2$, i.e., $-0.5 \le \delta \le 0.5$. The restriction is justified by the fact, that in practice for transform coefficients the inequality $|\mu| \ll \Delta$ is often true, and for context-based prediction schemes the unit interval containing the center of the distribution can be located by an error feedback loop [3, 4].

A method that approximates optimal coding of sources with the ODLD was first proposed in [5] and is widely used for coding of prediction residuals in image compression algorithms. However, the performance of this technique can be improved by using arithmetic coding. Moreover, the method is not applicable for sources with the entropy less then 1 bit, e.g., for coding of transform coefficients after quantization in loosy image compression techniques. Another ad-hoc method, based on Huffman coding, was developed for the JPEG image compression standard [6] and video compression standards [7] to code AC and DC coefficients of the DCT. Similar technique was used for coding of wavelet transform coefficients in [8]. Despite these methods allow for fast coding and coding, the efficiency also can be improved by about $5 \ldots 15\%$ using arithmetic coding.

However, direct use of m-ary arithmetic coding (m is the size of the source alphabet) may be inefficient for sources with the ODLD. The main problem is the alphabet size. Although formula (1) assumes an infinite alphabet, in practice it is finite, but quite large (e.g., the potential range of DCT coefficients in the JPEG image compression standard is supposed to be in the range $-2047 \ldots 2047$ for images with 8 bits per sample [6]). Using arithmetic coding, one must store and update as many parameters as the alphabet size[2]. Furthermore, such skewed distribution causes waste of the code

[1] Another distribution, called two-sided geometric distribution (TSGD), was proposed in [3] to model the probability distribution of prediction errrors in prediction based image compression algorithms. Although there is no preference among these distributions, in some cases they may give different solutions for optimal coding. We developed the proposed methods based on the ODLD model, even though some results are also applicable for sources with the TSGD.

[2] Of course, for the source (1) one could estimate, store and update only two parameters and calculate the probabilities for the whole alphabet, but this would decrease the coding speed.

space and thereby, reduces the coding efficiency.

An efficient method based on binary tree decomposition of the source alphabet, combined with binary arithmetic coding, was proposed for coding of DC and AC coefficients of the DCT in the JPEG image compression standard [6]. The binary decomposition allows to overcome the aforementioned problems and efficiently code sources with large alphabets and skewed distribution. Using multiplication free arithmetic coding, one may also design fast algorithms.

In this paper, we propose two methods of coding of sources with the ODLD using this technique. The main advantage of the methods is a small number of coding parameters (3-4). This is near to the number of parameters of the distribution (1), and it is less by a factor of about 10 than that of the JPEG (33 parameters), whereas the compression efficiency is not worse. A small number of coding parameters is essential for multi-context algorithms. It also may be beneficial for compression systems with limited memory or hardware resources.

One more benefit is that they are developed for sources with infinite alphabet. This means that, there is no limit imposed on the alphabet size. They are suitable for real sources with any alphabet size without any changes in the algorithm.

This paper is organized as follows. In the next Section we describe the general idea of the binary decomposition technique for source coding. In Section III, we present two decompositions for coding of the ODLD sources and discuss their properties. Finally, in Section IV we present the experimental results and discuss the efficiency and properties of the proposed methods.

2 Source coding using binary tree decomposition

Binary decomposition of source symbols, combined with binary arithmetic coding, is a well known technique for coding of m-ary sources [6, 9, 10]. A general idea of this method is that any proper and complete binary tree with m leaves can be used to represent symbols from an m-ary source $A = \{a_1, a_2, \ldots, a_m\}$ with any probability distribution. A source symbol is represented by the sequence of binary decisions when passing through the tree from the root to the leaf, corresponding to this symbol. The sequence of decisions can be regarded as a sequence of binary symbols generated by a Markov source, modeled by this tree. Each node of the tree corresponds to a state of the source and the tree defines a corresponding directed graph of state transitions. The root node defines the initial state.

A binary tree with m leaves has $K = m - 1$ nodes. To each node η_k, $k = 0, 1, \ldots, K-1$, of the decomposition tree there corresponds a parameter q_k. Each parameter, i.e, probability of a binary symbol (decision) being '0', assumes a value in $[0, 1]$ and specifies a binary probability distribution at the node. These parameters are uniquely defined by the probability distribution of the source symbols. The sequence of binary decisions can be decomposed into K subsequences of statistically independent binary symbols with probability distributions q_k, corresponding to each node. The subsequences can be effectively encoded using some kind of binary coding techniques, e.g., arithmetic coding [11] or Golomb run-length coding [12]. If the statistics of the

source is unknown or changing, one can implement an adaptive coding technique and thereby adjust the coding parameters to the source probability distribution.

Binary arithmetic coding is much simpler for hardware and software implementations than an m-ary arithmetic coding. The model no longer has to produce and maintain cumulative probabilities. A single distribution parameter is sufficient to encode a decision at each node. On the other hand, one has to encode more than one binary event for each input symbol. The average number of binary coding operations per source symbol is defined as

$$\overline{n} = \sum_{a \in A} P(a) n(a),$$

where $n(a)$ is the number of binary decisions to code a. This parameter defines mainly the coding speed. The minimum $\overline{n}$ is achieved when the tree is a Huffman tree for this source. The use of binary decomposition assumes that the decomposition tree is fixed[3] during encoding, and compression is performed by a binary coding technique. Using properties of the distribution (1) we propose two decomposition trees, which allow to reduce the number of coding parameters to 3 or 4.

3 Binary decompositions of alphabet for sources with the ODLD

The first proposed decomposition tree $\mathcal{A}$ is a unary representation of the index in the sequence of symbols of source (1) arranged in non-increasing probability order, that is: $0, +1, -1, +2, -2, \ldots$, if $\delta \geq 0$ or $0, -1, +1, -2, +2, \ldots$, if $\delta \leq 0$. In this tree the path from the root to the leaf i can be defined as

$$\begin{cases} \underbrace{1 \ldots 1}_{2i-1} 0 & \text{if } i > 0, \\ \underbrace{11 \ldots 1}_{2|i|} 0 & \text{if } i \leq 0, \end{cases} \tag{2}$$

if $\delta \geq 0$ or

$$\begin{cases} \underbrace{1 \ldots 1}_{2i} 0 & \text{if } i \geq 0, \\ \underbrace{11 \ldots 1}_{2|i|-1} 0 & \text{if } i < 0, \end{cases} \tag{3}$$

if $\delta \leq 0$. The average number of binary coding operations per source symbol is

$$\bar{n}_{\mathcal{A}} = 1 + \frac{\sqrt{\theta}}{2} \left(\frac{2\theta^{|\delta|} + \theta^{-|\delta|}(1+\theta)}{1-\theta} \right).$$

It can readily be shown, that this decomposition with proper ordering of source symbols (i.e., in non increasing probability order) is a Huffman tree if $\theta < \theta_{max}(\delta)$

[3]Otherwise, we would come to a (dynamic) Huffman coding technique and there would be no need for binary coding (at least for sources with an entropy larger than 1 bit).

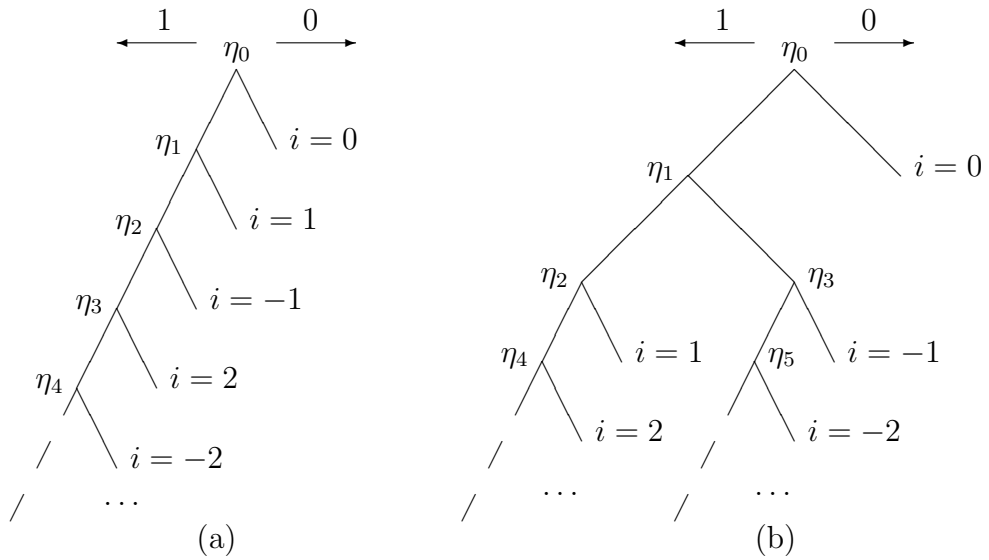

Figure 1: The decomposition trees. (a) The decomposition tree $\mathcal{A}$, (b) The decomposition tree $\mathcal{B}$.

and $\min_{\delta}\{\theta_{max}(\delta)\} = \theta_{max}(\delta = 0) = 1/3$. That is, if $0 < \theta < 1/3$, $-0.5 \leq \delta \leq 0.5$, then no other decomposition can achieve a smaller $\bar{n}$. For $0 < \theta < 1/3$, $\max_{\delta}\{\bar{n}_{\mathcal{A}}\} = \bar{n}_{\mathcal{A}}(\delta = 0) \lesssim 2.44$ and the entropy of the source $H \lesssim 2.36$. Hereafter, a source with the entropy $H \leq 2.36$ bits will be referred to as the *low entropy* source, and a source with $H > 2.36$ bits as the *high entropy* source.

Although the unary tree is optimal for sources (1) with low entropy in terms of the number of coding operations, it can be used for coding of sources with any entropy. Furthermore, it allows for reducing the number of storing parameters to three.

Without loss of generality, assume, that the decomposition tree is defined by (2), regardless whether the symbols are arranged in non-increasing order or not. Let the nodes η_k be indexed in such a way, that $k = 0$ corresponds to the root node, and increment index to each successive node (see Fig. 1*a*).

To code a source symbol i, the encoder codes a sequence of $n = 2i - 1$, if $i > 0$, or $n = 2|i|$, if $i \leq 0$, binary symbols '1' followed by '0'. The binary symbol at the position $j = 1, 2, \ldots, n + 1$ corresponds to a decision at the node η_{j-1} and is coded using probability distribution q_{j-1}. It can readily be verified, that

$$q_k = \begin{cases} 1 - \frac{\sqrt{\theta}}{2}\left(\theta^{\delta} + \theta^{-\delta}\right) & k = 0, \\ \frac{(1-\theta)\theta^{-\delta}}{\theta^{\delta}+\theta^{-\delta}} & k = 2t-1, \\ \frac{(1-\theta)\theta^{\delta}}{\theta^{\delta}+\theta^{1-\delta}} & k = 2t, \end{cases} \tag{2}$$

where $t = \{1, 2, \ldots\}$.

Even though the tree has infinitely many nodes, it is evident, that only three parameters corresponding to the nodes η_0, η_1 and η_2 are necessary to store and update. The last two are used for coding at all odd and even remaining nodes, respectively.

The drawback of this decomposition is that for high entropy sources the tree is not a Huffman tree. A good trade off is a tree, that provides a reasonable number of

binary coding operations per input symbol for a wide range of θ, while having a small number of parameters and a simple data structure.

For high entropy sources, e.g., for coding of the prediction residuals in lossless image compression algorithms, we propose the decomposition tree denoted $\mathcal{B}$. The first decision in this decomposition is whether the symbol i is zero or not (let it be '0' if $i = 0$ and '1' otherwise). If not, then the second decision is whether the symbol is positive or negative (let it be '1' if $i > 0$ and '0' otherwise), and then use unary decomposition of $|i-1|$ (let it be $|i-1|$ '1's followed by '0'). Figure 1*b* shows the tree.

The probability of the decision '0' at the node η_k is defined as

$$q_k = \begin{cases} 1 - \frac{\sqrt{\theta}}{2}\left(\theta^{\delta} + \theta^{-\delta}\right), & k = 0, \\ \frac{\theta^{-\delta}}{\theta^{\delta}+\theta^{-\delta}}, & k = 1, \\ 1 - \theta, & k = 2, 3, \ldots \end{cases}$$

Thus, for this decomposition tree, as well as for the decomposition $\mathcal{A}$, the parameters only at three nodes η_0, η_1 and η_2 are sufficient to store and update. However, we propose to keep the statistics for positive and negative branches separately, i.e., use a separate parameter q_3 for the node η_3. This allows for capturing the statistics if the actual probability distribution of source symbols has different slopes for positive and negative values. So, we propose the total number of parameters for this decomposition to be 4. Decisions at the nodes η_{2t} and η_{2t+1}, $t = 1, 2, \ldots$, are coded using the parameters q_2 and q_3 respectively.

The average number of binary coding operations per input symbol using this decomposition is

$$\bar{n}_{\mathcal{B}} = 1 + \frac{\sqrt{\theta}}{2}\left(\frac{2-\theta}{1-\theta}\right)\left(\theta^{\delta} + \theta^{-\delta}\right).$$

For low entropy sources $\bar{n}_{\mathcal{B}}/\bar{n}_{\mathcal{A}} < 1.34$ and for high entropy sources $0.5 < \bar{n}_{\mathcal{B}}/\bar{n}_{\mathcal{A}} < 1$ (for $H > 7.5$ bit, $0.5 < \bar{n}_{\mathcal{B}}/\bar{n}_{\mathcal{A}} < 0.526$ and if $H \to \infty$, then $\bar{n}_{\mathcal{B}}/\bar{n}_{\mathcal{A}} \to 0.5$, $\forall \delta \in [-0.5, 0.5]$).

4 Experimental results and discussion

In order to evaluate the efficiency of the proposed methods, we implemented them for entropy coding of prediction residuals and DCT coefficients for lossless and lossy compression techniques respectively. The aim was to evaluate the compression and speed efficiency for coding of data with different entropy and for different applications and compare them with the decomposition used in the JPEG standard. Binary coding for both decompositions, as well as for the JPEG decomposition, was implemented using the QM-coder [6]. For testing purposes we used nine 8-bit grayscale images of size 720×576, which are available via the internet [13].

In the first case we implemented a simple prediction technique and coded the prediction residuals. We used the average of the pixel above $X[i-1, j]$ and the pixel to the left, $X[i, j-1]$, as the prediction value for the current pixel $X[i, j]$. This

corresponds to the predictor No.7 of the lossless mode of the JPEG standard. The sequence of prediction residuals was treated as a memoryless source. Table 1 shows the resulting number of bytes of the compressed images. Table 2 gives the number of binary coding operations per pixel. We used this parameter to evaluate the speed performance. The "Diff." column in all tables gives the relative differences of figures between the JPEG decomposition and the proposed ones.

In the second case we coded DCT coefficients, acquired from the JPEG compressed files with the average compression ratio about 7:1[4]. We encoded the coefficients of the same spatial frequency together and used separate statistics for each frequency. We did not use the "end of block" symbol. The reason was to compare the average compression performance for sources with different entropy, which, in essence, were formed by the coefficients with different spatial frequencies. Tables 3 and 4 give the number of bytes of the compressed images and the number of binary coding operations per pixel respectively.

Experimental results show that the compression performance of both proposed decompositions is not worse than that of the JPEG standard, while they have much smaller the number of the parameters. The decompositions $\mathcal{A}$ and $\mathcal{B}$ require to store and update only 3 and 4 parameters, respectively, versus 33 parameters for the JPEG decomposition. (Therefore, branching the tree is also simpler.)

The decomposition $\mathcal{A}$ has an obvious advantage for sources with entropy $H \leq 2.36$ bits, since it has the smallest number of the binary coding operations per input symbol. The decomposition $\mathcal{B}$ has about half the number of coding operations per input symbol than the decomposition $\mathcal{A}$ for high entropy sources. Although, the decomposition $\mathcal{B}$ has about 20% more binary coding operations per symbol than the JPEG decomposition, the real difference of speed performance may be less than this figure. This is because passing the JPEG tree decomposition in the coding process is more costly operation than passing the tree $\mathcal{B}$ (there are fewer nodes to pass). Nonetheless, both decompositions allow for improvement of speed performance in different ways.

A straightforward way is to use one of a number of proposed multiplication free binary arithmetic coders (see, e.g., [14, 15, 16]). Another way is to use run-length Golomb codes. For example, the Rice coding [5] can be viewed as a fast version of the decomposition $\mathcal{A}$, where the decisions are treated as a binary memoryless source and coded using Golomb codes. Although the Rice coding is widely used for coding of high entropy sources, we propose a modification (generalization) of this algorithm for sources with any entropy. In this modification, the encoder adaptively switches the Golomb run-length coding to code the most probable binary symbol of the sequence of decisions, thereby allowing the encoding of sources with low entropy.

The speed improvement of the fast version is for the price of higher redundancy. The relative redundancy of the modified Rice algorithm is upper bounded by 50%, if $H \to 0$, and for $0.05 < H < 1$ it is about 10%-30%. (These figures are valid if the

[4]The reason for such a ratio was to have the ODLD sources in a wide range of distribution parameters (see below in the paragraph). Otherwise, for higher compressed images most of the high frequency coefficients would be zero.

Image	JPEG decomp.	Decomp. $\mathcal{A}$	Diff.	Decomp. $\mathcal{B}$	Diff.
baloon	170043	169425	-0.36%	169574	-0.27%
barb	277707	268953	-3.15%	269895	-2.81%
barb2	276088	268103	-2.89%	269465	-2.40%
board	216994	215733	-0.58%	216432	-0.26%
boats	233511	229546	-1.70%	231318	-0.94%
girl	231568	227700	-1.67%	228466	-1.34%
gold	249334	246321	-1.21%	247885	-0.58%
hotel	260963	255600	-2.01%	255882	-1.94%
zelda	218749	217480	-0.58%	218167	-0.27%
average	237217	233206	-1.69%	234120	-1.30%

Table 1: The number of bytes for lossless image compression.

Image	JPEG decomp.	Decomp. $\mathcal{A}$	Diff.	Decomp. $\mathcal{B}$	Diff
baloon	3.50	4.46	+27.47%	3.66	+4.53%
barb	6.73	17.30	+157.02%	10.27	+52.64%
barb2	6.70	16.63	+148.23%	9.93	+48.22%
board	4.90	8.26	+68.72%	5.69	+16.15%
boats	5.35	9.85	+83.85%	6.50	+21.31%
girl	5.24	8.48	+61.93%	5.81	+10.94%
gold	6.05	11.16	+84.38%	7.19	+18.76%
hotel	6.16	13.48	+118.75%	8.35	+35.52%
zelda	5.00	7.24	+44.60%	5.20	+3.97%
average	5.52	10.76	+95.11%	6.96	+26.10%

Table 2: The average number of binary coding operations per pixel for lossless image compression.

Image	JPEG decomp.	Decomp. $\mathcal{A}$	Diff.	Decomp. $\mathcal{B}$	Diff
baloon	33017	33483	+1.44%	33458	+1.33%
barb	75799	75574	-0.30%	75594	-0.27%
barb2	78377	77808	-0.73%	77898	-0.61%
board	43849	44833	+2.24%	44695	+1.92%
boats	55272	55413	+0.26%	55469	+0.36%
girl	55364	55200	-0.30%	55207	-0.28%
gold	64402	63905	-0.77%	64111	-0.45%
hotel	67595	67545	-0.07%	67579	-0.02%
zelda	42579	42409	-0.40%	42597	+0.04%
average	57361	57352	-0.016%	57400	+0.068%

Table 3: The number of bytes of compressed DCT coefficients.

Image	JPEG decomp.	Decomp. $\mathcal{A}$	Diff.	Decomp. $\mathcal{B}$	Diff
baloon	1.44	2.05	+42.4%	1.66	+15.3%
barb	2.09	3.52	+68.7%	2.57	+23.2%
barb2	2.12	3.41	+60.6%	2.54	+19.3%
board	1.57	2.55	+62.1%	1.95	+23.8%
boats	1.79	2.81	+56.8%	2.14	+19.3%
girl	1.83	2.83	+54.9%	2.16	+18.0%
gold	1.94	2.81	+44.8%	2.21	+13.7%
hotel	1.97	3.48	+76.6%	2.51	+27.5%
zelda	1.61	2.33	+44.6%	1.85	+15.2%
average	1.82	2.87	+57.6%	2.18	+19.7%

Table 4: The average number of binary coding operations per pixel for the DCT coefficients.

source symbols are arranged in non-increasing probability order, or if $\mu \ll \Delta$, i.e., $\delta \approx 0$)

The speed performance of the decomposition $\mathcal{B}$, combined with the QM-coder (like in our experiment), can be improved by using "speed up" mode [6]. Another way is to send the second decision (the sign bit) directly to the output bit stream.This may increase the code length by at most 6.2% (for the worst case when $\delta = \pm 1/2$) for sources with entropy $H \geq 2.36$ bit. If $H < 2.36$ bit, the increase can be more than 6.2% for $\delta \to \pm 1/2$. However, in most applications $\delta \approx 0$ (especially for low entropy sources). That is, in practice, the increase usually is negligibly small.

One more advantage of the proposed methods is that there is no embedded limit on the alphabet size, thereby any value can be encoded and there is no need to adjust the algorithm while designing, for example, a compression system for images with different number of bits per pixel.

5 Conclusion

In this paper we studied a coding of sources with the ODLD using binary decomposition and suggested two methods that allow for efficient coding of sources with such skewed distribution and infinite alphabets. We have shown, that the proposed methods have reduced the number of parameters by a factor of about 10 compared to the JPEG, while the compression efficiency is not worse. This may be essential for multi-contexts algorithms or when designing a compression system with limited memory or hardware resources. Also, we discussed of fast software and hardware implementations.

References

[1] J.O'Neal, "Predictive quantizing differencial pulse code modulation for the transmission of television signals," *Bell Syst. Tech. J.*, vol.45, pp. 689-722, May 1966.

[2] R.C. Reininger, J.D. Gibson "Distributions of the two-dimentional DCT coefficients for images," *IEEE Trans. on Comm.* vol. 31, no.6, pp. 835-839, 1983.

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

[4] M.J. Weinberger, G.Seroussi and G.Sapiro, "The LOCO-I lossless image compression algorithm: Principles and Stamdartization into JPEG-LS," *IEEE Trans. Image Proc.*, vol. 9, no. 8, pp. 1309-1324, Aug., 2000.

[5] R.F. Rice "Some practical universal noisless coding technique," JPL Publication 79-22, Pasadena, California, Mar. 1979.

[6] W.B. Pennebaker, J.L. Mitchell, *JPEG – Still Image Data Compression Standard*. New York: Van Nostrand Reinhold, 1993.

[7] R. Schäfer and T. Sikora, "Digital video coding standards and their role in video communications," *Proceedings IEEE*, vol. 83, no. 6, pp. 907-923, 1995.

[8] A. Said and W.A. Pearlman. "An image multiresolution representation for lossless and lossy compression," *IEEE Trans. Im. Proc.*, vol. 5, no. 9, pp. 1303-1310, 1996.

[9] G.G. Langdon and J.J. Rissanen "A double adaptive file compression algorithm," *IEEE Trans. Comm.*, vol.31, no.11, pp. 1253-1255, 1983.

[10] P.G. Howard, "The design and analysis of efficient lossless data compression systems," Report CS-93-28, Brown Uneversity, Providence, Rhode Island, 1993.

[11] I.H. Witten, R. Neal, and J.G. Cleary, "Arithmetic coding for data compression," *Comm. ACM*, vol.30, no.6, pp. 520-541, 1987.

[12] S.W. Golomb "Run-length encoding," *IEEE Trans. Inform. Theory*, vol.12, pp. 399-401, 1966.

[13] ftp://ftp.csd.uwo.ca/pub/from_wu/images .

[14] G.G. Langdon and J.J. Rissanen "A simple general binary source coding," *IEEE Trans. Inf. Theory*, vol.28, no.5, pp. 800-803, 1982.

[15] W.B. Pennebaker, J.L. Mitchell, G.G. Langdon and R.B.Arps, "An overview of the basic principles of the Q-coder adaptive binary arithmetic coder," *IBM J. Res. Develop.*, vol.32, no.6, pp. 717-726, 1988.

[16] P.G. Howard and J.S. Vitter, "Arithmetic coding for data compression," *Proc. IEEE*, vol.82, no.6, pp. 857-865, Jun. 1994.