Simple Fast and Adaptive Lossless Image Compression Algorithm

Roman Starosolski*

December 20, 2006

This is a preprint of an article published in
Software—Practice and Experience, 2007, 37(1):65-91, DOI: 10.1002/spe.746

Abstract

In this paper we present a new lossless image compression algorithm. To achieve the high compression speed we use a linear prediction, modified Golomb–Rice code family, and a very fast prediction error modeling method. We compare the algorithm experimentally with others for medical and natural continuous tone grayscale images of depths of up to 16 bits. Its results are especially good for big images, for natural images of high bit depths, and for noisy images. The average compression speed on Intel Xeon 3.06 GHz CPU is 47 MB/s. For big images the speed is over 60 MB/s, i.e., the algorithm needs less than 50 CPU cycles per byte of image.

KEY WORDS: lossless image compression; predictive coding; adaptive modeling; medical imaging; Golomb–Rice codes

1 Introduction

Lossless image compression algorithms are generally used for images that are documents and when lossy compression is not applicable. Lossless algorithms are especially important for systems transmitting and archiving medical data, because lossy compression of medical images used for diagnostic purposes is, in many countries, forbidden by law. Furthermore, we have to use lossless image compression when we are unsure whether discarding information contained in the image is applicable or not. The latter case happens frequently while transmitting images by the system not being aware of the images' use, e.g., while transmitting them directly from the acquisition device or transmitting over the network images to be processed further. The use of image compression algorithms could improve the transmission throughput provided that the compression algorithm complexities are low enough for a specific system. Some systems such as medical CT scanner

*Silesian University of Technology, Institute of Computer Science, Akademicka 16, 44–100 Gliwice, Poland (e-mail: Roman.Starosolski@polsl.pl)

systems require rapid access to large sets of images or to volumetric data that are further processed, analyzed, or just displayed. In such a system, the images or volume slices are stored in the memory since mass storage turns out to be too slow—here, the fast lossless image compression algorithm could virtually increase the memory capacity allowing processing of larger sets of data.

An image may be defined as a rectangular array of pixels. The pixel of a grayscale image is a nonnegative integer interpreted as the intensity (brightness, luminosity) of the image. When image pixel intensities are in the range $[0, 2^N - 1]$, then we say that the image is of N bit depth, or that it is an N-bit image. Typical grayscale images are of bit depths from 8 to 16 bits.

Grayscale image compression algorithms are used as a basis for color image compression algorithms and for algorithms compressing other than images 2-dimensional data characterized by a specific smoothness. These algorithms are also used for volumetric 3-dimensional data. Sometimes such data, as a set of 2-dimensional images, is compressed using regular image compression algorithms. Other possibilities include preprocessing volumetric data before compressing it as a set of 2-dimensional images or using algorithms designed exclusively for volumetric data—the latter are usually derived from regular image compression algorithms.

We could use a universal algorithm to compress images, i.e., we could simply encode a sequence of image pixels extracted from an image in the raster scan order. For a universal algorithm such a sequence is hard to compress. Universal algorithms are usually designed for alphabet sizes not exceeding 2^8 and do not exploit directly the following image data features: images are 2-dimensional data, intensities of neighboring pixels are highly correlated, and the images contain noise added to the image during the acquisition process—the latter feature makes dictionary compression algorithms perform worse than statistical ones for image data [1]. Modern grayscale image compression algorithms employ techniques used in universal statistical compression algorithms. However, prior to statistical modeling and entropy coding the image data is transformed to make it easier to compress.

Many image compression algorithms, including CALIC [2, 3], JPEG-LS [4], and SZIP [5], are predictive, as is the algorithm introduced in this paper. In a predictive algorithm, we use the predictor function to guess the pixel intensities and then we calculate the prediction errors, i.e., differences between actual and predicted pixel intensities. Next, we encode the sequence of prediction errors, which is called the residuum. To calculate the predictor for a specific pixel we usually use intensities of small number of already processed pixels neighboring it. Even using extremely simple predictors, such as one that predicts that pixel intensity is identical to the one in its left-hand side, results in a much better compression ratio, than without the prediction. For typical grayscale images, the pixel intensity distribution is close to uniform. Prediction error distribution is close to Laplacian, i.e., symmetrically exponential [6, 7, 8]. Therefore entropy of prediction errors is significantly smaller than entropy of pixel intensities, making prediction errors easier to compress.

The probability distribution of symbols to be encoded is estimated by the data model. There are two-pass compression algorithms that read data to be compressed twice. During the first pass the data is analyzed and the data model is built. During the second pass the data is encoded using information stored in the model. In a two-pass algorithm we have to include along with the encoded data the data model itself, or information allowing reconstructing the model by the decompression algorithm. In the adaptive modeling

we do not transmit the model; instead it is built on-line. Using a model built for all the already processed symbols we encode specific symbol immediately after reading it. After encoding the symbol we update the data model. If the model estimates conditional probabilities, i.e., if the specific symbol's context is considered in determining symbol's probability then it is the context model, otherwise the model is memoryless. As opposed to universal algorithms that as contexts use symbols directly preceding the current one, contexts in some of the image compression algorithms are formed by the pixel's 2-dimensional neighborhood. In context determination we use pixel intensities, prediction errors of neighboring pixels, or some other function of pixel intensities. High number of intensity levels, especially if the context is formed of several pixels, could result in a vast number of possible contexts—too high a number considering the model memory complexity and the cost of adapting the model to actual image characteristics for all contexts or of transmitting the model to the decompression algorithm. Therefore, in image compression algorithms we group contexts in collective context buckets and estimate probability distribution jointly for all the contexts contained in the bucket. The context buckets were firstly used in the Sunset algorithm that evolved into Lossless JPEG, the former JPEG committee standard for lossless image compression [9].

After the probability distribution for the symbol's context is determined by a data model, the symbol is encoded using the entropy coder. In order to encode the symbol s optimally, we should use $-\log_2(\mathrm{prob}(s))$ bits, where $\mathrm{prob}(s)$ is the probability assigned to s by the data model [10]. Employing an arithmetic entropy coder we may get arbitrarily close to the above optimum, but practical implementations of arithmetic coding are relatively slow and not as perfect as theoretically possible [11]. For entropy coding we also use prefix codes, such as Huffman codes [12], which are much faster in practice. In this case we encode symbols with binary codewords of integer lengths. The use of prefix codes may lead to coding results noticeably worse than the above optimum, when probability assigned by the data model to the actual symbol is high. In image compression, as in universal compression algorithms, we use both methods of entropy coding, however knowing the probability distribution of symbols allows some improvements. Relatively fast Huffman coding may be replaced by a faster entropy coder using a parametric family of prefix codes, i.e., Golomb or Golomb–Rice family [13, 14].

The algorithms used for comparisons in this paper employ two more methods to improve compression ratio for images. Some images contain highly compressible smooth (or 'flat' [7]) regions. It appears that modeling algorithms and entropy coders, tuned for typical image characteristics, do not obtain best results when applied to such regions. Furthermore, if we encode pixels from such a region using prefix codes, then the resulting code length cannot be less than 1 bit per pixel, even if the probability estimated for a symbol is close to 1. For the above reasons some compression algorithms detect smooth regions and encode them in a special way. For example, in the JPEG-LS algorithm, instead of encoding each pixel separately, we encode, with a single codeword, the number of consecutive pixels of equal intensity. In the CALIC algorithm, we encode in a special way sequences of pixels that are of at most two intensity levels—a method aimed not only at smooth regions, but for bilevel images encoded as grayscale also.

The other method, probably firstly introduced in the CALIC algorithm, actually employs modeling to improve prediction. This method is called the bias cancellation. The prediction error distribution for the whole image usually is close to Laplacian with 0 mean. The mean of the distribution for a specific context, however, may locally vary with location within the image. To make the distribution centered at 0 we estimate a

local mean of the distribution and subtract it from the prediction error. The contexts, or context buckets, used for modeling the distribution mean may differ from contexts used for modeling distribution of prediction errors after the bias cancellation.

The performance of the predictive algorithm depends on the predictor function used. The predictors in CALIC and JPEG-LS are nonlinear and can be considered as switching, based on local image gradients, among a few simple linear predictors. More sophisticated schemes are used in some recent algorithms to further improve the compression ratios. For example, in the APC algorithm the predictor is a linear combination of a set of simple predictors, where the combination coefficients are adaptively calculated based on the least mean square prediction error [15]. Another interesting approach is used in the EDP algorithm, where the predictor is a linear combination of neighboring pixels and the pixel coefficients are determined adaptively based on the least-square optimization [16]. To reduce the time complexity of the EDP algorithm the optimization is performed only for pixels around the edges. Compared to a CALIC algorithm which, because of its compression speed, is by some authors considered to be of research use rather than of practical use, the two latter algorithms obtain speeds significantly lower.

Another method of making the image data easier compressible, different from the prediction, is to use 2-dimensional image transforms, such as DCT or wavelet transform. In transform algorithms, instead of pixel intensities, we encode a matrix of transform coefficients. The transform is applied to the whole image, or to an image split into fragments. We use transforms for both lossless and lossy compression. Transform algorithms are more popular in lossy compression, since for a lossy algorithm we do not need the inverse transform to be capable of losslessly reconstructing the original image from transform coefficients encoded with finite precision. The new JPEG committee standard of lossy and lossless image compression, JPEG2000, is a transform algorithm employing a wavelet transform [17, 18]. Apart from lossy and lossless compressing and decompressing of whole images, transform algorithms deliver many interesting features (progressive transmission, region of interest coding, etc.), however, in respect to the lossless compression speed and ratio, better results are obtained by predictive algorithms.

In this paper, we introduce a simple, fast, and adaptive lossless grayscale image compression algorithm. The algorithm, designed primarily to achieve the high compression speed, is based on the linear prediction, modified Golomb–Rice code family and a very fast prediction error modeling method. The operation of updating the data model, which is based on the data model known from the FELICS algorithm [19], although fast as compared to many other modeling methods would be the most complex element of the algorithm. Therefore we apply the reduced model update frequency method that increases the overall compression speed by a couple of hundred percent at the cost of worsening the compression ratio by a fraction of a percent. The algorithm is capable of compressing images of high bit depths, actual implementation is for images of bit depths up to 16 bits per pixel. The algorithm originates from an algorithm designed for 8-bit images only [20]. We analyze the algorithm and compare it with other algorithms for many classes of images. In the experiments, we use natural continuous tone grayscale images of various depths (up to 16 bits), various sizes (up to about 4 millions of pixels) and various classes of medical images (modalities: CR, CT, MR, and US). Nowadays, consumer acquisition devices, such as cameras or scanners, produce images of ever growing sizes and high nominal depths, often attaining 16 bits. The quality of the acquisition process seems to fall behind the growth of acquisition resolution and bit depth—typical high bit depth images are noisy. Natural images used in research were acquired using a high

Table 1: Predictors used in the research.

$\mathsf{Pred0}(X) = 0$	$\mathsf{Pred3}(X) = C$	$\mathsf{Pred6}(X) = B + (A - C)/2$
$\mathsf{Pred1}(X) = A$	$\mathsf{Pred4}(X) = A + B - C$	$\mathsf{Pred7}(X) = (A + B)/2$
$\mathsf{Pred2}(X) = B$	$\mathsf{Pred5}(X) = A + (B - C)/2$	$\mathsf{Pred8}(X) = (3A + 3B - 2C)/4$

quality film scanner. To analyze the algorithm performance on noisy data special images with added noise were prepared. We also generated non-typical easily compressible and incompressible pseudo-images to estimate the best-case and the worst-case performance of compression algorithms.

2 METHOD DESCRIPTION

2.1 Overview

Our algorithm is predictive and adaptive; it compresses continuous tone grayscale images. The image is processed in a raster-scan order. Firstly, we perform prediction using a predictor selected from a fixed set of 9 simple linear predictors. Prediction errors are reordered to obtain probability distribution expected by the data model and the entropy coder, and then output as a sequence of residuum symbols. For encoding residuum symbols we use a family of prefix codes based on the Golomb–Rice family. For fast and adaptive modeling we use a simple context data model based on a model of the FELICS algorithm [19] and the method of reduced model update frequency [20]. The algorithm was designed to be simple and fast. We do not employ methods such as detection of smooth regions or bias cancellation. Decompression is a simple reversal of the compression process. With respect to both time and memory complexity the algorithm is symmetric.

The algorithm described herein originates from an algorithm designed for images of 8-bit depth, which obtained high compression speed but could not be just simply extended to higher bit depths [20]. The most significant differences between these algorithms are reported in Section 2.6.

2.2 Prediction

To predict the intensity of a specific pixel X, we employ fast linear predictors that use up to 3 neighboring pixels: the left-hand neighbor (A), the upper neighbor (B), and the upper-left neighbor (C). We use 8 predictors of the Lossless JPEG algorithm (Table 1, Pred0–Pred7) [9], and one a bit more complex predictor Pred8, that actually returns an average of Pred4 and Pred7. Predictors are calculated using integer arithmetic. We select a single predictor for the whole image, however for pixels of the first row and the first column some predictors cannot be calculated—in this case we use simpler predictors (e.g., Pred2 for the first column).

If there is a subtraction operation in a calculation of the predictor, then its value may be out of the nominal range of pixel intensities $[0, 2^N - 1]$, where N denotes image bit depth. In such a case, we take the closest value from the above range. We compress the residuum symbol that is a difference between the actual (X) and the predicted ($\mathsf{Pred}(X)$)

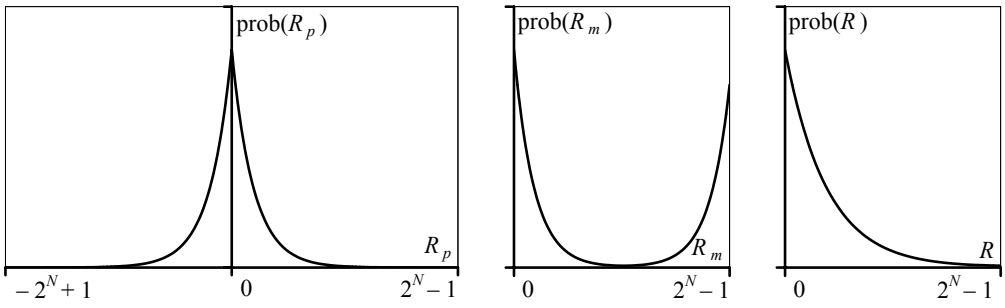

Figure 1: Probability distribution of prediction errors: a) before modulo reduction, b) after modulo reduction, and c) after reordering.

pixel intensity, i.e., $R_p = X - \mathsf{Pred}(X)$. Since both X and $\mathsf{Pred}(X)$ are in the range $[0, 2^N - 1]$, R_p is in the range $[-2^N + 1, 2^N - 1]$. To encode such a symbol directly, using the natural binary code, we would need $N + 1$ bits, i.e., we would expand the N-bit image data before the actual compression. Fortunately, we may use N-bit symbols to encode prediction errors since for a specific pixel we have only 2^N values of R_p (range $[-\mathsf{Pred}(X), 2^N - 1 - \mathsf{Pred}(X)]$). The $\mathsf{Pred}(X)$ may be calculated by both the compression and the decompression algorithm prior to processing the pixel X. Instead of the above-mentioned formula we use $R_m = (X - \mathsf{Pred}(X)) \mathrm{mod} 2^N$. For decompression we use $X = (R_m + \mathsf{Pred}(X)) \mathrm{mod} 2^N$.

The code family we use to encode the residuum requires residual values to be ordered in a descending order of probability. For typical images, before a modulo reduction, the distribution is close to symmetrically exponential (Laplacian), however after that reduction it no longer descends (Fig. 1). We reorder residual values to get the probability distribution close to exponential by simply picking symbols: first, last, second, last but one and so on:

$$R = \begin{cases} 2R_m & \text{for } R_m < 2^{N-1} \\ 2(2^N - R_m) - 1 & \text{for } R_m \geq 2^{N-1} \end{cases}$$

2.3 The code family

The code family used is based on the Golomb–Rice (GR) family, i.e., on the infinite family of prefix codes, that is a subset of a family described by Golomb [13] (Golomb family), rediscovered independently by Rice [14]. GR codes are optimal for encoding symbols from an infinite alphabet of exponential symbol probability distribution. Each code in the GR family is characterized by a nonnegative integer rank k. In order to encode the nonnegative integer i using the GR code of rank k we firstly encode the codeword prefix: $\lfloor i/2^k \rfloor$ using a unary code, then the suffix: $i \mathrm{mod} 2^k$ using a fixed length k-bit natural binary code.

Prediction errors are symbols from a finite alphabet. The probability distribution of these symbols is only close to the exponential. To encode the prediction errors we use a limited codeword length variant of the GR family [21]. For encoding residuum symbols of image of N-bit depth, that is for alphabet size 2^N, we use family of N codes. We limit the codeword length to $l_{\max} > N$. For each code rank $0 \leq k < N$ we define the threshold

Table 2: The code family for integers in range $[0, 15]$, codeword length limited to 8 bits.

Integer	Code			
	$k = 0$	$k = 1$	$k = 2$	$k = 3$
0	0•	0•0	0•00	0•000
1	10•	0•1	0•01	0•001
2	110•	10•0	0•10	0•010
3	<u>1110•</u>	10•1	0•11	0•011
4	1111•0000	110•0	10•00	0•100
5	1111•0001	110•1	10•01	0•101
6	1111•0010	1110•0	10•10	0•110
7	1111•0011	<u>1110•1</u>	10•11	<u>0•111</u>
8	1111•0100	1111•000	110•00	1•000
9	1111•0101	1111•001	110•01	1•001
10	1111•0110	1111•010	110•10	1•010
11	1111•0111	1111•011	<u>110•11</u>	1•011
12	1111•1000	1111•100	111•00	1•100
13	1111•1001	1111•101	111•01	1•101
14	1111•1010	1111•110	111•10	1•110
15	1111•1011	1111•111	111•11	1•111

$\pi_k = \min((l_{\max} - N)2^k, 2^N - 2^k)$. We encode the nonnegative integer $0 \leq i < 2^N$, in the following way: if $i < \pi_k$ then we use the GR code of rank k, in the opposite case we output a fixed prefix: $\pi_k/2^k$ ones, and then suffix: $i - \pi_k$ encoded using fixed length $\lceil \log_2(2^N - \pi_k) \rceil$-bit natural binary code.

Sample codewords are presented in Table 2. The separator is inserted between prefix and suffix of the codeword for legibility only. Some codewords are underlined. For a specific code the underlined codeword and codewords above the underlined one are identical to their equivalents in the GR code.

Use of the code family significantly simplifies the compression algorithm. To encode a certain residuum symbol we just select a code rank based on the information stored in the data model and simply output a codeword assigned to this symbol by the code of the selected rank.

Limiting the codeword length is a method used in several other algorithms, including JPEG-LS. It is introduced to reduce data expansion in case of selecting in the data model a code of improper (too small) rank to encode symbol of high value—coding images we deal with alphabet sizes up to 2^{16} and code of rank $k = 0$ assigns $(i + 1)$-bit codeword to symbol i. Apart from limiting the codeword length, the advantage of the presented family over the original GR codes is that it contains, as the code of rank $k = N - 1$, the N-bit natural binary code. Using natural binary code we may avoid the data expansion even when coding the incompressible data.

2.4 The data model

The modified data model known from the FELICS algorithm invented by Howard and Vitter [19] is used. For prediction errors of pixels in the first column of an image a prediction error of the above pixel is used as a context, for prediction errors of the remaining pixels the preceding residuum symbol, i.e., a prediction error of pixel's left-hand neighbor, is used as a context.

The method of selecting code rank in the data model of the FELICS algorithm is fast and simple. For each context we maintain an array of N counters, one counter for each code rank k. Each counter stores the code length we would have if we used code of rank k to encode all symbols encountered in the context so far. To encode a specific symbol in a specific context we simply use the rank that in that context would give the shortest code so far. After coding the symbol we update the counters in the current context. For each code rank we increase its counter by the length of the codeword assigned to the encoded symbol by code of this rank. Periodically, when the smallest counter in a specific context reaches a certain threshold all the counters in this context are halved, causing the model to assign more importance to the more recently processed data.

Although one symbol only is used to determine the context we use collective context buckets. In the FELICS algorithm data model, for each context, at least one symbol has to be encoded before we are able to estimate the code rank based on actual image data. The first symbol in a given context, or a few first symbols, may be encoded using an improper code rank. Since we deal with alphabet sizes up to 2^{16}, the number of pixels encoded in a non-optimal way may worsen the overall compression ratio. Furthermore, due to an exponential prediction error probability distribution, some contexts may appear in the whole image a couple of times only. For the above reasons we group contexts of higher values in collective context buckets. In our case we maintain a single array of counters for all the contexts contained in the bucket. The number of contexts contained in the bucket grows exponentially in respect to the bucket number, starting with the bucket containing the single context. This way we reduce the FELICS model memory complexity of $O(2^{N+1})$ to $O(N^2)$.

If there are some codes equally good for encoding a specific symbol according to the criterion of the FELICS data model, then the code of the smallest rank is selected, which may cause an improper selection of small code ranks and lead to data expansion. To reduce the effects of the improper rank selection at the beginning of the coding, Howard and Vitter suggest assigning a small initial penalty to the counters for small ranks. We used a simple method that works not only at the beginning of the coding, but also when the image data characteristics change during the compression. We avoid the risk of data expansion by selecting, from among all the equally good codes, the one of the highest rank [22].

2.5 The reduced model update frequency method

The motivation for introducing the reduced model update frequency method is the observation of typical image characteristics that change gradually for almost all the image area or even are invariable. In order to adapt to gradual changes, we may sample the image, i.e., update the data model, less frequently than each time the pixel gets coded. Instead we update the model after the coding of selected pixels only. We could simply pick every i-th symbol to update the model, but such a constant period could interfere with the image structure. Therefore each time, after updating the model with some sym-

```
delay := 0
while not EOF
   read symbol
   compress symbol
   if delay = 0 then
      update model
      delay := random(range)
   else
      delay := delay-1
   endif
endwhile
```

Figure 2: The reduced model update frequency method.

bol, we select randomly a number of symbols to skip before next update of the model (Fig. 2). The number of symbols to skip is selected regardless of the actual value of the symbol used to update the model as well as of it's context (the `delay` variable in the Fig. 2 is a global one). In order to permit the decoder to select the same number we use a pseudo-random number generator. For just avoiding the interference with an image structure, even the simplest pseudo-random number generator, should suffice. We use the fixed pseudo-random number generator seed—this way we avoid storing the seed along with the compressed image and make the compression process deterministic.

By selecting the range of the pseudo-random numbers we may change the model update frequency, i.e., the probability of updating the model after coding a symbol. This way we control the speed of adapting the model and the speed of the compression process. At the beginning of compression, the data model should adapt to the image data characteristics as quickly as possible. We start compression using all symbols in data modeling and then we gradually decrease the model update frequency, until it reaches some minimal value.

The method is expected to vastly improve the compression speed without significantly worsening the compression ratio. In case of the algorithm, from which the described algorithm originates, the method allowed to increase the compression speed by about 250% at the cost of worsening the ratio by about 1% [20]. Similar, to a certain extent, approach was used in the EDP algorithm, where the predictor coefficients are determined in an adaptive way by means of relatively complex LS optimization. As reported in [16], performing the above optimization only for pixels around the edges allows a decrease of the time complexity by an order of magnitude at the cost of a negligible worsening of the compression ratio. Note that, as opposed to the reduced update frequency method, in EDP location the pixels for which the time consuming operation is performed (or skipped) depends on the image contents.

2.6 Differences from the predecessor algorithm

The most significant differences between the described algorithm and the one from which it originates [20] are the code family and the data model. Actually, the described algorithm is simpler compared to its predecessor.

The code family used in the previous algorithm was based on the Golomb codes. It was a limited codeword length family, it contained the natural binary code, ordering of codes in the family was altered compared to original Golomb family. Generating codewords from that family was not as simple as in case of the family presented in Section 2.3, however, it was not a problem for the algorithm of 8-bit image compression. As opposed to 16-bit depth, for 8-bit depth whole family may be precomputed and stored in the array of a reasonable size.

The data model of the predecessor algorithm was also more sophisticated. To aid fast adaptation to characteristics of the image data at the beginning of the compression, the model used variable number of collective context buckets. The compression started with a single bucket (containing all the contexts), that was subsequently divided into smaller ones. By using this method the compression ratio for the smallest images was improved by over 1%. For the described algorithm, the use of variable number of buckets resulted in worsening the average compression ratio (by about 0.2%); only in case of some small images the ratios were negligibly improved (by less than 0.1%). As opposed to its predecessor, in the data model of the described algorithm we select, from among all the equally good codes, the one of the greatest rank. The above feature, along with using different code family, seems to be simpler and more efficient, than using the variable number of buckets. Furthermore, giving the variable number of buckets up we reduce the modeling memory and time complexity.

2.7 Complexity analysis

Time complexity

The fast adaptive compression algorithms are of the linear time complexity, in our case:

$$T(n) = n(c_p + c_c + c_m) = n(c_p + c_c + pc_u) = O(n),$$

where n is the number of pixels in the image, c_p denotes prediction complexity (per pixel), c_c—coding, c_m—modeling, c_u—single model update, and p—the update frequency. Prediction and coding are implemented as short sequences of simple integer operations. The model update is more complex since to update the model we have to compute lengths of N codewords, where N is the image bit depth. Updating the model less frequently we accelerate the slowest part of the compression process.

Memory complexity

The data model requires $O(N^2)$ bytes, where N is the image bit depth, for storing N counters for each of cN buckets, $c \approx 1$. To perform prediction we need $O(w)$ bytes, where w is the image width, since for some of the predictor functions we need the pixel's upper-left neighbor, i.e., memory for storing at least $w + 1$ pixels.

Actual implementation is aimed at maximizing the compression speed, rather than at reducing the memory complexity. Depending on image bit depth and endianness of the CPU it requires from about $7w + 2000$ to about $12w + 4000$ bytes.

3 EXPERIMENTAL RESULTS

3.1 Algorithm implementation

The algorithm was implemented in ANSI C language; the implementation may be downloaded from `http://sun.iinf.polsl.gliwice.pl/~rstaros/sfalic/`. Algorithm processes the image row by row. After the row has been inputted, the prediction for the whole row is performed and a resulting row of residuum symbols is stored in a memory buffer. The row of residuum symbols is compressed to another memory buffer and then output. After updating the model, the number of symbols to be skipped before the next update is selected by picking a pseudo-random number and reducing it modulo 2^m, where m is a nonnegative integer. Therefore following frequencies $p = 2/(2^m + 1)$ may be used: 100% (the full update frequency), 66.6%, 40%, 22.2%, 11.8%, 6.06%, 3.08%, 1.55%, 0.778%, 0.390%, 0.195%, ... We start coding the image with the full update frequency. Then, each time d pixels are coded, we decrease the frequency, until we reach the target update frequency. The number of buckets in the model, as a function of image bit depth, may be also selected. We tested a few following model structures (below are numbers of contexts in the consecutive buckets):

a) 1, 1, 1, 2, 2, 4, 4, 8, 8, ... ;

b) 1, 2, 4, 8, 16, ... ;

c) 1, 4, 16, 64, ...

In an actual implementation, special variants of some functions were prepared for images of depths up to 8 bits. Optimizations are possible when the alphabet size is small. For example, reordering of prediction errors or finding a bucket for specific context may be done using single table-lookup to increase the compression speed; buffers for image rows may be allocated for 8-bit pixels instead of 16-bit ones to reduce implementation's memory requirements.

3.2 Procedure

An HP Proliant ML350G3 computer equipped with two Intel Xeon 3.06 GHz (512 KB cache memory) processors and Windows 2003 operating system was used to measure the performance of algorithm implementations. Single-threaded application executables of the described algorithm, and other algorithms used for comparisons, were compiled using Intel C++ Compiler, version 8.1. To minimize effects of the system load and the input-output subsystem performance on obtained results the executable was run several times. The time of the first run was ignored and the collective time of other runs (executed for at least one second, and at least 5 times) was measured and then averaged. The time measured was a sum of time spent by the processor in an application code and in kernel functions called by the application, as reported by the operating system after application execution. Since we measure the execution time externally we actually include the time of initializing the program by the operating system into our calculations; this time may be significant for the smallest images.

The compression speed is reported in megabytes per second [MB/s], where 1 MB = 2^{20} bytes. Since we used PGM P5 image representation, the pixel size is 2 bytes for image depth over 8 bits, 1 byte in the opposite case. The compression ratio is in bits per pixel

[bpp]: $8e/n$, where e is the size in bytes of the compressed image including the header, n—number of pixels in the image.

3.3 Test image set

A new diverse set of medical and natural continuous tone grayscale test images was prepared to evaluate the performance of lossless image compression algorithms. The main reason of preparing the new set was that, to our best knowledge, there was no publicly available set of test images containing big, high quality images, which were originally acquired with an actual 16-bit depth. The set contains natural continuous tone grayscale images of various bit depths (up to 16 bits), various sizes (up to about 4 millions of pixels) and medical images of various modalities (CR, CT, MR, and US). In the set, image groups were defined, to permit performance analysis based on average results for the whole group, rather than on results for single images.

The biggest group, *normal*, is for evaluating algorithms' performance in a typical case. A collection of smaller groups permits us to analyze or compare results with respect to images' bit depths, sizes, or medical image modality. The set contains also non-typical images, which do not belong to the *normal* group. To analyze the algorithms' performance on noisy data special images with added noise were prepared. To estimate the best-case and the worst-case performance of algorithms, easily compressible and incompressible pseudo-images were also generated. Below, we describe the image groups; details of individual images are reported in [23]. The set contains about one hundred images. It is not as large as, for example, the set used by Clunie in an extensive study on lossless compression of medical images [24] which contained over 3600 images but, on the other hand, moderate size of the set allowed making it publicly available—it may be downloaded from `http://sun.iinf.polsl.gliwice.pl/~rstaros/mednat/`.

Group of natural continuous tone images, i.e., group of images acquired from scenes available for the human eye (photographic images), was constructed as follows. Four images were acquired from a 36mm high quality diapositive film (Fuji Provia/Velvia) using Minolta Dimage 5400 scanner (Fig. 3). In order to minimize the noise, the acquisition was first done at the device's maximum depth of 16 bits, optical resolution 5400dpi and using multiple sampling of each pixel (16 times or 4 times in case of branches image). One image (flower) was softened by setting the scanner focus too close. Then the images' resolution was reduced 3 times. These images formed a group of 16-bit big images, and then were subject to further resolution reduction (3 and 9 times) and to bit depth reduction (to 12 and to 8 bits). The set contains following groups of natural images:

- *natural*—36 natural images of various sizes and bit depths,
- *big*—12 natural images of various bit depths and size approximately 4000000 pixels,
- *medium*—12 natural images of various bit depths and size approx. 440000 pixels,
- *small*—12 natural images of various bit depths and size approx. 49000 pixels,
- *16bpp*—12 natural images of various sizes and 16-bit depth,
- *12bpp*—12 natural images of various sizes and 12-bit depth,
- *8bpp*—12 natural images of various sizes and 8-bit depth.

Figure 3: Sample *natural* images.

Groups of medical images were composed of CR, CT, MR, and US images of various anatomical regions, acquired from devices of several vendors. Most of the medical images are from collections of medical images available on the Internet; the origin of individual images is reported in [23]. In case of medical CR, CT, and MR images we report the nominal bit depth. The actual number of intensity levels may be smaller than implied by the bit depth, by an order of magnitude or even more. The set contains the following groups of medical images:

- *medical*—48 medical CR, CT, MR, and US images,
- *cr*—12 medical CR images, nominal depth: 10 to 16 bits, average size approximately 3500000 pixels,
- *ct*—12 medical CT images, nominal depth: 12 to 16 bits, average size approximately 260000 pixels,
- *mr*—12 medical MR images, nominal depth of 16 bits, average size approximately 200000 pixels,
- *us*—12 medical US images, 8-bit depth, average size approximately 300000 pixels.

To evaluate algorithms' performance in a typical case, the *normal* group was defined. The *normal* group contains all 84 *natural* and *medical* images. The average results of compressing images from the *normal* group are used as a measure of algorithms' performance for continuous tone grayscale images. Unless indicated otherwise, we report the average results for this group.

Other groups contained in the set, non-typical images:

- *noise*—9 images with added noise, created using branches image of various bit depths (8, 12, and 16 bits) and medium size (approximately 440000 pixels). Noise was added using: $v_1 = v_0(1 - a) + ra$, where v_0 denotes original pixel intensity,

v_1—intensity after adding noise, r—random value of uniform distribution (range $[0, 2^N - 1]$, where N is image bit depth) and a is the amount of noise. We prepared images using $a = 0.1, 0.2, 0.5$,

- *empty*—3 pseudo-images, intensity of all pixels equals 0, nominal depth of 8, 12, and 16 bits, size approximately 440000 pixels,
- *random*—3 pseudo-images, random intensities of pixels (uniform distribution), bit depth of 8, 12, and 16 bits, size approximately 440000 pixels.

The set described above contains no images traditionally used for comparisons of image compression algorithms. To verify observations made using the above set for traditional 8-bit test images and to make comparisons to results reported in other studies possible, additional experiments were performed using the popular Waterloo BragZone GreySet2 set of test images (downloaded from: `http://links.uwaterloo.ca/BragZone/GreySet2/`).

3.4 Parameter selection for the algorithm

The parameter selection was based on the average compression speed and the average compression ratio for images from the *normal* group. The threshold, that triggers dividing all the counters in a certain context when the smallest counter reaches it, was selected for each update frequency and for each number of buckets individually as the average best value for predictors 1 to 8. This way we do not favor any specific update frequency, predictor, or model structure. Knowing these parameters, however, we could simply use a fixed threshold for all the update frequencies. Using for all the update frequencies the threshold selected for the update frequency and the number of buckets of the default parameter set listed below, would simplify the algorithm and change the compression ratio, for some image groups only, by less than 0.1%. The remaining algorithm parameters were selected by examining results of compression using combinations of all p values, all numbers of buckets, all predictors, some values of d, and some code length limits $l_{\max}$. Parameter combinations which, compared to some other combination, resulted in worsening of both the compression speed and the compression ratio, were rejected. One of the remaining combinations was selected as the default parameter set, its use results in the compression speed 20% less than the fastest one obtained and the compression ratio worse by about 0.5%, compared to the best ratio. The default parameters are:

- model update frequency $p = 3.08\%$,
- predictor Pred8 $((3A + 3B - 2C)/4)$,
- decreasing the update frequency each $d = 2048$ pixels,
- code length limit $l_{\max} = 26$,
- doubling model bucket size each bucket (model structure b).

Below we describe how these parameters, considered individually, influence compression results. Fig. 4 presents the compression speed and the compression ratio obtained using various update frequencies. Using the reduced update frequency we may get a couple of hundred percent speed improvement at the cost of worsening the compression

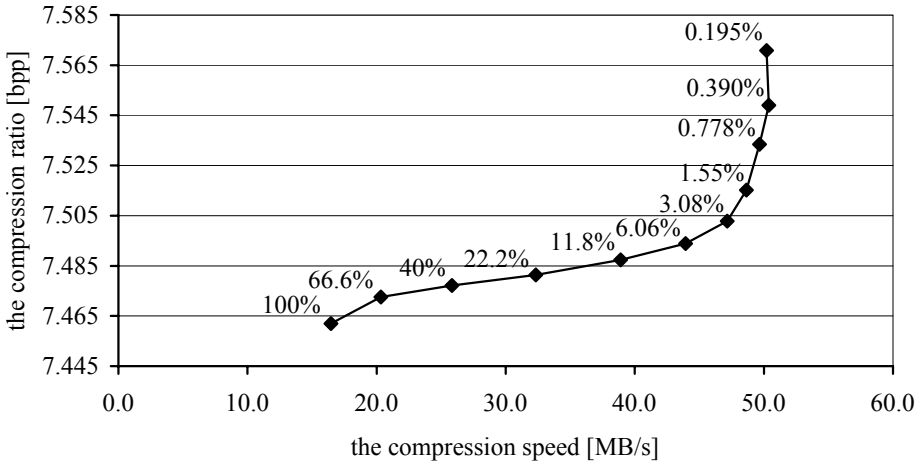

Figure 4: Compression results for various update frequencies (*normal* images).

ratio by about 0.5%. Note that decreasing the update frequency below some point stops improving the modeling speed. The reduced model update frequency method could probably be applied to other adaptive algorithms, in which modeling is a considerable factor in algorithm's overall time complexity. It could be used as a mean of adjusting the algorithm speed versus the quality of modeling or, as in our case, it could be used to improve the speed, to a certain extent, without worsening the modeling quality significantly.

The selected Pred8 predictor, although the most complex, gives the best average compression ratio. Use of this predictor and the selected update frequency proves to be better than the use of any simpler predictor and a greater frequency since the compression ratio improvement is obtained at the relatively small cost. For a few specific image groups, the use of other predictors results in the compression ratio improvement (and in the speed improvement by a few percent):

- for the *cr* and *noise* images Pred7 improves the ratio by 1.0% and 1.6% respectively,
- for the *us* images the use of Pred4 gives a 3.6% improvement of the compression ratio.

Decreasing gradually the update frequency each 2048 pixels, compared to compressing the whole image using the same frequency, only affects the results for the smallest images. This way, for the *small* group, we get a 1.0% compression ratio improvement and the speed lower by a few percent.

The code length limit was selected for the *normal* group. Except for the *us* group, selecting the limit for the specific group may improve the compression ratio by less than 0.1%. For the *us* images we may obtain a 2.1% compression ratio improvement by limiting the codeword length to 14 bits.

Results for the data model structures described earlier: a, b, and c are almost identical. Selecting other than the default model structure we may improve the compression ratio for some groups by less than 0.1%. We also compared the data model that uses collective context buckets to two other model structures. Using collective context buckets proves to be superior to compression with a model of 2^N individual contexts that results in an expansion of 16-bit images and also to compression with memoryless model that, for the *normal* images, results in worsening the average compression ratio by 6.2%.

3.5 Comparison to other techniques

The algorithm described in this paper, denoted here as 'SFALIC', was compared to several other image compression algorithms. In Tables 3 and 4 we report average compression speeds and average ratios obtained by the algorithms described below, for *normal* images. Due to the number of images contained in the set, results for individual images are not included in this paper, they may be downloaded from `http://sun.iinf.polsl.gliwice.pl/~rstaros/sfalic/`. After discussing results for the new set we report results obtained for the well-known images of University of Waterloo. The results are reported for the following algorithms:

- CALIC-A—the relatively complex predictive and adaptive image compression algorithm using arithmetic entropy coder, which because of the very good compression ratios is commonly used as a reference for other image compression algorithms [2, 3]. In the CALIC algorithm we use 7 neighboring pixels, both to determine context and as arguments of the nonlinear predictor function. When pixels in the neighborhood are, at most, of 2 intensity levels CALIC enters, so called, the binary mode. In the binary mode, for consecutive pixels, we encode information whether pixel intensity is equal to brighter neighbors, darker neighbors, or neither of them-in this case we leave the binary mode. CALIC utilizes the bias cancellation method. We used implementation by Wu and Memon [25]. Since this implementation is a binary executable for UltraSparc processors, the compression speed of CALIC algorithm is estimated based on the relative speed of this implementation compared to the SFALIC speed on a different computer system (Sun Fire V440 running Solaris 9, equipped with 1.06 GHz UltraSparc IIIi processors; both implementations were single-threaded).

- CALIC-H—the variant of the CALIC algorithm using Huffman codes (compression speed estimated as in case of CALIC-A).

- JPEG-LS—the standard of the JPEG committee for lossless and near-lossless compression of still images [4]. The standard describes a low-complexity predictive and adaptive image compression algorithm with entropy coding using a modified Golomb–Rice family. The algorithm is based on the LOCO-I algorithm [26, 27]. In the JPEG-LS algorithm, we use 3 neighboring pixels for nonlinear prediction, and 4 pixels for modeling. JPEG-LS utilizes the bias cancellation method, also it detects and encodes in a special way smooth image regions. If the smooth region is detected we enter the, so called, run-mode and instead of encoding each pixel separately we encode, with a single codeword, the number of consecutive pixels of equal intensity. We used the SPMG/UBC implementation [28]. In this implementation, some code parts are implemented in 2 variants: one for images of depths up to 8 bits and the other for image depths 9–16 bits.

- CCSDS SZIP—the standard of the Consultative Committee for Space Data Systems used by space agencies for compressing scientific data transmitted from satellites and other space instruments [5]. CCSDS SZIP is a very fast predictive compression algorithm based on the extended-Rice algorithm, it uses Golomb–Rice codes for entropy coding, and primarily was developed by Rice. CCSDS SZIP is often confused with a general-purpose compression utility by Schindler, which is also called 'SZIP'. CCSDS SZIP does not employ an adaptive data model. The sequence of prediction

Table 3: The compression speed, *normal* images [MB/s].

image group	CALIC-A	CALIC-H	JPEG-LS	CCSDS SZIP	SFALIC
natural	2.6	7.2	15.0	40.3	43.3
big	3.4	9.7	18.1	56.0	61.8
medium	2.8	8.0	16.5	44.8	48.1
small	1.7	3.7	10.3	20.0	20.1
16bpp	1.9	6.9	15.9	40.0	41.3
12bpp	3.0	7.5	17.8	48.4	47.9
8bpp	3.0	7.0	11.1	32.4	40.7
medical	3.6	9.6	20.7	50.6	50.1
cr	4.9	11.5	24.8	73.5	70.5
ct	3.4	9.1	21.6	50.1	49.1
mr	2.7	8.4	20.9	41.6	39.9
us	3.7	9.3	15.4	37.1	40.8
normal	3.2	8.5	18.2	46.1	47.2

errors is divided into blocks. Each block is compressed using a two-pass algorithm. In the first pass, we determine the best coding method for the whole block. In the second pass, we output the marker of the selected coding method as a side information along with prediction errors encoded using this method. The coding methods include: Golomb–Rice codes of a chosen rank; unary code for transformed pairs of prediction errors; fixed length natural binary code if the block is found to be incompressible; signaling to the decoder empty block if all prediction errors are zeroes. We used UNM implementation [29]. It was optimized for the default block size of 16 symbols. Since biggest images (*big* and *cr*) required greater block size, we used block size of 20 symbols for all the images. For smaller images, compared to the reported results, by using the default block size, we get compression speed higher by about 10% to 20%, and the compression ratio from 0.5% worse to 1.2% better, depending on the image group. Higher compression speed for all the images by an average of 8.5% for the *normal* group may be obtained using a block size of 32 symbols, however, at the cost of worsening the compression ratio by 0.9%.

JPEG2000, Lossless JPEG Huffman, PNG [30], and FELICS were also examined [23]. We do not report these results because, for all the image groups, speeds and ratios of these algorithms are worse than obtained by the JPEG-LS. For other test image sets the JPEG2000 algorithm is reported to obtain ratios, depending on image or image class, little better or little worse to JPEG-LS [24, 31]. Compared to SFALIC, the algorithms Lossless JPEG Huffman, PNG, and FELICS, obtain worse compression ratios for almost all the groups (except for *us* group for PNG and FELICS, and *empty* group for PNG) and lower compression speeds for all the groups. JPEG2000 obtains average ratio by 2.3% better than SFALIC and compression speed about 15 times lower. SFALIC was also compared to the algorithm from which it originates. For *8bpp* images SFALIC's predecessor obtained compression ratio worse by 1.3% and the compression speed lower by 34%.

Table 4: The compression ratio, *normal* images [bpp].

image group	CALIC-A	CALIC-H	JPEG-LS	CCSDS SZIP	SFALIC
natural	7.617	7.661	7.687	8.432	7.953
big	6.962	7.059	7.083	7.773	7.274
medium	7.623	7.699	7.710	8.403	8.009
small	8.267	8.227	8.269	9.121	8.576
16bpp	11.748	11.622	11.776	12.458	11.867
12bpp	7.491	7.565	7.571	8.407	7.869
8bpp	3.613	3.797	3.715	4.431	4.123
medical	6.651	6.761	6.734	7.396	7.165
cr	6.229	6.324	6.343	6.883	6.662
ct	7.759	7.840	7.838	8.806	8.266
mr	9.975	9.895	10.009	10.599	10.235
us	2.641	2.985	2.748	3.298	3.497
normal	7.065	7.147	7.143	7.840	7.503

SFALIC algorithm is clearly the fastest algorithm among algorithms that use an adaptive data model. The compression speed of SFALIC algorithm for *normal* images (Fig. 5) is over 2.5 times higher than the speed of the second fastest adaptive model algorithm (JPEG-LS) and about 12 times higher than the speed of an algorithm obtaining the best compression ratios (CALIC-A). Compression speed of SFALIC is almost the same as the speed of the CCSDS SZIP algorithm, which does not employ adaptive modeling. Actually SFALIC obtained speed little higher than CCSDS SZIP. However, the relative speed difference is negligible. Probably both algorithms could be optimized to improve the speed a little—CCSDS SZIP by optimizing it for block size of 20 symbols, or by optimizing it for low image bit depths, SFALIC by integrating prediction into coding and modeling loop. The highest compression speed is achieved for the biggest images (*big* and *cr*). The compression speed for these groups is over 60 MB/s, i.e., for the biggest images we need less than 50 CPU cycles per byte of image. The compression speed significantly lower, than the average, was obtained for *small* images. For these images, the time of initializing the compression implementation executable by the operating system becomes a significant factor in the overall speed of the compression algorithm. To some extent similar behavior may be observed for all the examined algorithms. SFALIC compression speed depends on the image size rather than on the image bit depth. Since for depths over 8 bits the image pixel is stored using 2 bytes, the compression speed for *12bpp* images is greater than the speed for *8bpp* and *16bpp* images. We also notice that, for individual images of similar bit depth and similar size there are no significant differences in SFALIC compression speed. The compression speed of some other algorithms, such as JPEG-LS, depends to a larger extent on the image contents. Here the greater differences are probably due to much lower complexity of the run mode employed by JPEG-LS for smooth areas found in some images, compared to the complexity of regular mode used for non-smooth regions.

The average compression ratio of the CALIC-A, CALIC-H, and JPEG-LS algorithms is better than the ratio of SFALIC by 5.8%, 4.7%, and 4.8% respectively. Such a cost

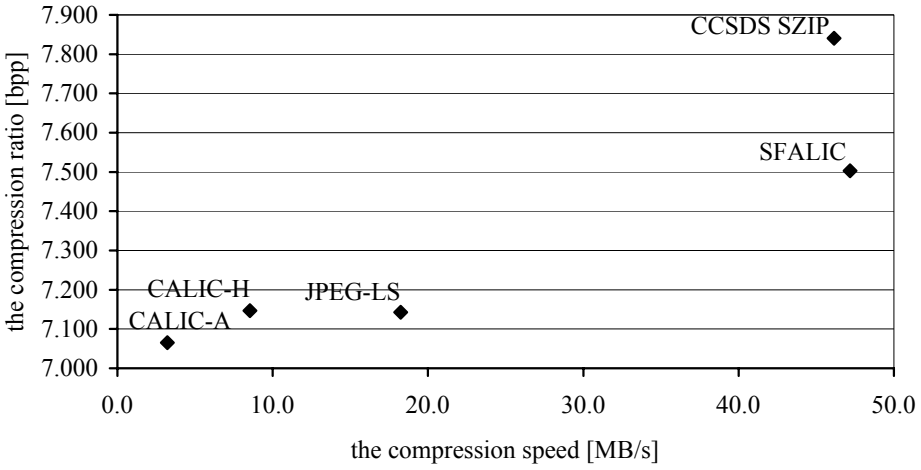

Figure 5: The compression results for various algorithms (*normal* images).

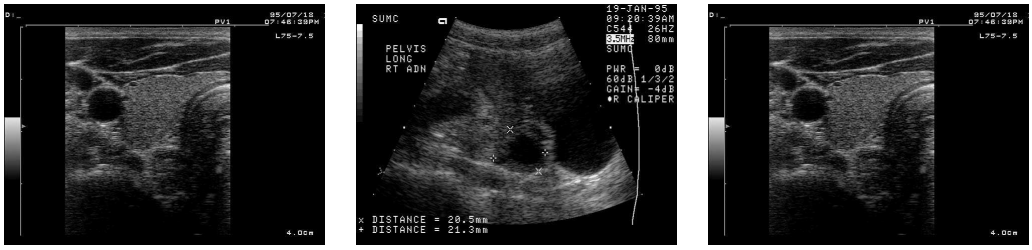

Figure 6: Sample medical *us* images.

of improving the compression speed is not important in many practical image processing systems, especially when we compress images to transmit them or to store them temporarily. Compared to the CCSDS SZIP, which is the only algorithm that obtains speed close to SFALIC, the compression ratio of SFALIC is better by 4.5%.

For compressing some images, other algorithms are much better—the *us* images are compressed 24.5% better by CALIC-A. The predictor function and the codeword length limit were selected for the *normal* group and are not well suited for the *us* images, but the main reason of a worse compression ratio is that SFALIC does not employ any special method of processing smooth image regions—the *us* images contain large uniform intensity areas, i.e., black background for the actual image (Fig. 6).

On the Fig. 7 we compare ratios of SFALIC and JPEG-LS obtained for individual images. The absolute differences of ratios are moderate; the greatest one is about 1 bpp. Note that bigger differences occur for smaller compression ratios so the relative differences of ratios may be, for highly compressible images, practically important. Therefore, in the Fig. 8, instead of an absolute ratio we present the relative compression ratio of JPEG-LS expressed as percents of the ratio that SFALIC obtained for a specific image. We also mark, by the gray background, images that contain significant amount of smooth areas. Here, the image is considered to contain significant amount of smooth areas if at least 15% of its pixels is encoded by the JPEG-LS using the run mode (actually it is at least 17.6% for these images and at most 5.6% for the remaining ones). We can see that the relative ratio differences in favor of JPEG-LS are getting much greater as the image compression ratio decreases. It can also be seen that the JPEG-LS ratio is better than

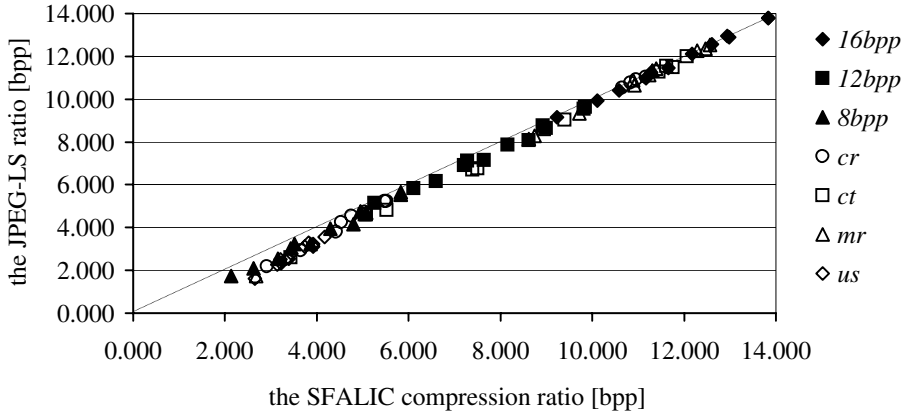

Figure 7: SFALIC and JPEG-LS ratios for individual images.

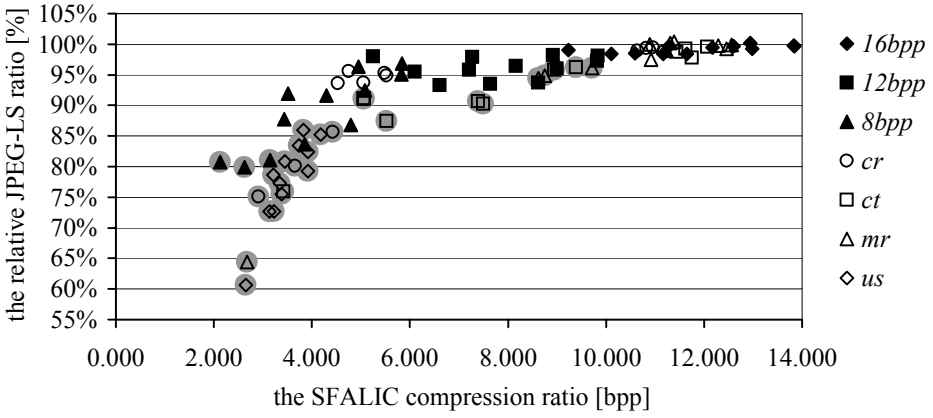

Figure 8: SFALIC ratio and relative JPEG-LS ratio for individual images.

SFALICS's by up to almost 40% for images containing significant amount of smooth areas, whereas for other images the JPEG-LS is better by up to about 16%. Above observation confirms significance of using the method of efficient encoding smooth image areas in the compression algorithm.

For *natural* images the relative compression ratio of algorithms obtaining better ratios compared to SFALIC, does not depend on image size and significantly depends on image bit depth. For *8bpp* images the compression ratio of CALIC-A is 12.4% better, for *16bpp* images it is better by 1.0%. Generally the SFALIC algorithm obtains good compression ratios when the actual number of intensity levels is high. The medical *cr*, *ct*, and *mr* images, which are of 16-bit nominal depth, actually use much smaller than 2^{16} number of levels. Among 24 such images 21 use below 4000 levels and only 3 *cr* images use about 25000 levels. For these 3 images the ratio of CALIC-A algorithm is better than SFALIC's ratio by 1.4%.

Most *ct* and *mr* images and some *cr* images are of sparse histograms. Not only the actual number of levels found in these images is much smaller than the nominal one, but the levels are distributed throughout almost all the entire nominal intensity range as well. Such characteristics is clearly different from what is expected by a lossless image compression algorithm, both in case of predictive and of transform coding. In [32] we re-

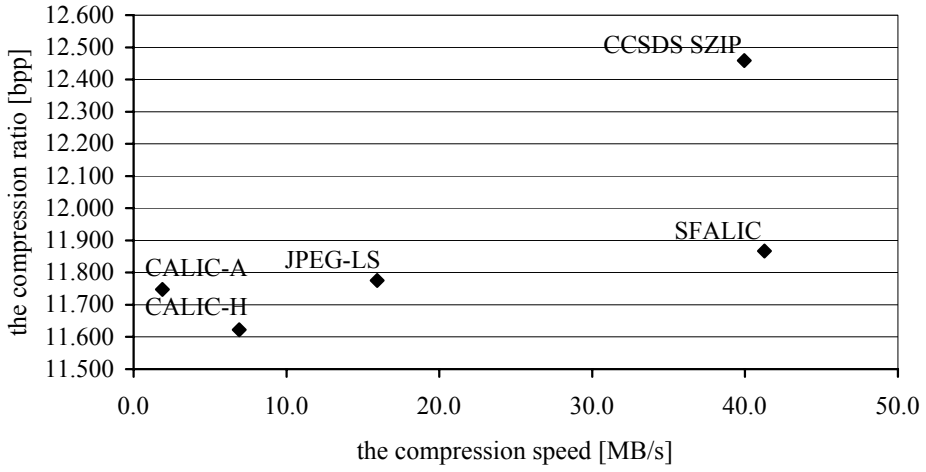

Figure 9: The compression results for various algorithms (*16bpp* images).

Table 5: The compression speed for non-typical images [MB/s].

image group	CALIC-A	CALIC-H	JPEG-LS	CCSDS SZIP	SFALIC
empty	13.2	39.9	156.1	118.2	79.3
noise	2.0	7.4	14.9	41.4	45.6
random	1.3	6.5	14.7	41.5	37.3

ported efficient methods of compressing these images. Employing the so called histogram packing technique we may vastly improve compression ratios of sparse histogram images. This way the CALIC-A average compression ratios were improved to 4.485 bpp for *ct* and to 4.811 bpp for *mr* images (that is by about 42% and 52% respectively). Improvement of the average compression ratio for the *cr* group was about 15%.

Interesting results were obtained for *16bpp* images (Fig. 9). For this group the compression ratio of arithmetic coding version of CALIC is 1.1% worse than the ratio of the Huffman-coder version. CALIC-H obtains ratios better than CALIC-A also for *small* (i.e., group containing 16-bit images) and for *mr* images (of 16 bit nominal depth). Above observations suggest that the small difference in compression ratio between SFALIC and CALIC for *16bpp* images should rather be attributed to imperfections of other algorithms, than to especially good performance of SFALIC. Probably there is still a possibility of improving the compression ratio of CALIC for high bit depth images.

The *empty* pseudo-images (Tables 5 and 6) are the most easily compressible data for the image compression algorithm. As one could expect, for *empty* images the ratio of algorithms that employ a method of efficient encoding smooth image regions is close to 0 bpp. For all the algorithms, the compression speed for the *empty* group is higher than of any other group, the greatest speedup is observed for JPEG-LS.

For non-typical noisy images the compression speed of all algorithms is similar to the average *medium* group speed that contains images of similar size. Compression of these images, in case of some algorithms even of individual images with 50% noise added, still results in compression ratios smaller than the image bit depth, however not by much. The

Table 6: The compression ratio for non-typical images [bpp].

image group	CALIC-A	CALIC-H	JPEG-LS	CCSDS SZIP	SFALIC
empty	0.001	0.045	0.002	0.027	1.000
noise	10.478	10.690	10.693	11.101	10.842
random	12.375	13.008	12.516	12.370	12.009

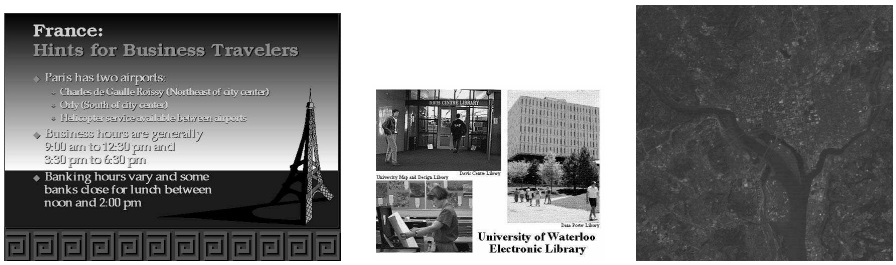

Figure 10: BragZone GreySet2 images with smooth areas.

random pseudo-images are incompressible and may be used for estimating the worst-case algorithm compression ratio, however for a specific image compression algorithm we can prepare data even harder to compress, i.e., pseudo-image of characteristics opposite to what is expected in prediction or modeling. The best method of processing incompressible data is to copy them binary, i.e., to encode pixel intensities using N-bit natural binary code, where N denotes image bit depth—for the *random* group we would get the compression ratio of 12 bpp. The SFALIC algorithm actually acts this way; its code family contains the fixed length natural binary code that is used in case of processing *random* images. All the remaining algorithms cause noticeable data expansion. In case of CCSDS SZIP algorithm, natural binary code is also used, however it is a two-pass scheme. The data expansion of CCSDS SZIP is solely due to including, in the compressed data, side information along with each block.

To verify observations made using the new set, additional experiments were performed with the popular Waterloo BragZone GreySet2 set of 8-bit test images (Tables 7 and 8). In the tables we report results for individual images, average results for the whole BragZone GreySet2 set, and the average results for 8-bit medium size natural images from the new set, i.e., for images belonging to the intersection of groups *8bpp* and *medium* (rows labeled 'Average *medium8bpp*'). Not all the BragZone GreySet2 images are typical photographic continuous-tone ones. The france is a computer generated and the library is a compound image. The washsat is an aerial photo image. Compared to other GreySet2 images, the latter 3 images contain significantly greater amount of smooth areas (Fig. 10). JPEG-LS encodes 46.9%, 22.4%, and 10.7% of pixels of france, library, and washsat respectively, using the run-mode, whereas for the remaining images it is at most 2.9%. In some images dithering-like patterns are visible; these images are probably dithered palette color images converted to grayscale. The patterns are most noticeable in library and frog images, also mandrill, mountain, and peppers seem to be dithered. We also note that the images washsat, frog, and mountain are of sparse histograms—the number of pixel intensity levels actually used in these images is 35, 102, and 110 respectively.

Table 7: The compression speed, BragZone GreySet2 images [MB/s].

image	pixels	CALIC-A	CALIC-H	JPEG-LS	CCSDS SZIP	SFALIC
barb	262144	1.8	4.2	10.3	28.7	34.3
boat	262144	1.9	4.4	10.6	28.7	36.3
france	333312	4.3	10.7	23.1	45.5	40.0
frog	309258	2.0	5.3	10.3	30.4	35.8
goldhill	262144	1.7	4.1	10.3	29.0	35.4
lena	262144	1.9	4.4	10.5	28.8	36.6
library	163328	1.6	4.2	11.0	24.1	27.4
mandrill	262144	1.6	4.1	9.5	27.9	33.7
mountain	307200	1.9	5.1	10.1	28.8	33.6
peppers	262144	1.8	4.1	10.4	28.5	34.8
washsat	262144	2.1	4.8	11.5	30.2	36.3
zelda	262144	2.1	4.6	11.0	29.5	37.1
Average GreySet2	267521	2.1	5.0	11.5	30.0	35.1
Average *medium8bpp*	440746	3.1	7.7	12.1	36.0	44.1

Table 8: The compression ratio, BragZone GreySet2 images [bpp].

image	pixels	CALIC-A	CALIC-H	JPEG-LS	CCSDS SZIP	SFALIC
barb	262144	4.453	4.569	4.733	5.775	5.315
boat	262144	4.151	4.233	4.250	5.153	4.632
france	333312	0.823	1.684	1.411	2.425	3.736
frog	309258	5.853	6.232	6.049	6.657	6.536
goldhill	262144	4.629	4.719	4.712	5.280	4.870
lena	262144	4.110	4.184	4.244	5.046	4.567
library	163328	5.012	5.228	5.101	5.858	6.025
mandrill	262144	5.875	6.031	6.036	6.374	6.256
mountain	307200	6.265	6.538	6.422	6.717	6.840
peppers	262144	4.378	4.488	4.489	5.167	4.933
washsat	262144	3.670	4.107	4.129	4.825	4.526
zelda	262144	3.862	3.973	4.005	4.838	4.289
Average GreySet2	267521	4.424	4.665	4.632	5.343	5.210
Average *medium8bpp*	440746	3.630	3.826	3.701	4.400	4.153

Generally, these results adhere to the results obtained for the new set. Average compression speed for GreySet2 images is little lower than the speed obtained for *medium8bpp* images, that are of little greater size. The compression speed for all the photographic images, for a specific algorithm, does not vary significantly. Increased speed is observed for france image, which contains smooth regions and, by all the algorithms, is compressed faster than other images of similar size. In case of CALIC and JPEG-LS, the speed is increased over 2 times compared to other images of similar size. The library image also contains smooth regions, but this image is smaller than others—as could be expected SFALIC compresses this image slower compared to bigger images. For some other algorithms presence of smooth regions seems to have greater impact on compression speed, than the smaller size of the image. The smooth regions in washsat do not influence noticeably the compression speed. In this image, observed in a raster scan order, runs of pixels (or prediction errors) are much shorter, than in france or in the library (and the overall amount of pixels in smooth regions is smaller). There are no significant differences in compression speed between photographic images with and without dithering patterns.

The average SFALIC compression ratio for GreySet2 images compared to ratios obtained by other algorithms, is little worse than in case of the *medium8bpp* images—for the GreySet2 CALIC-A obtains a ratio better than SFALIC by 15.1%, whereas for *medium8bpp* it is better by 12.6%. The greater differences are due to ratios obtained for two images (france and library) containing significant amount of smooth areas. If we exclude these images from comparison, then the CALIC-A ratio gets smaller than SFALIC's by 10.5% only. For the france image the CALIC-A obtains a ratio of 0.823 bpp, whereas ratios of other algorithms are from 1.411 bpp (JPEG-LS) to 3.736 bpp (SFALIC). Such large differences cannot be attributed to smooth areas alone. The probable reasons of large differences among CALIC-A and other algorithms are: the CALIC's binary mode (capable of encoding sequences of symbols of 2 intensity levels); the arithmetic coder used (which as opposed to Golomb–Rice codes is capable of efficiently encoding of any probability distribution); the predictors used (that in the CALIC and in the JPEG-LS are actually switching between simple predictors in order to detect edges); the more sophisticated data model used (SFALIC uses prediction error of a single neighbor of current pixel only to determine pixel's context) and the algorithms' ability to adapt quickly to rapid changes of the image characteristics (SFALIC uses the reduced model update frequency method; SZIP selects the coding method for a whole block of pixels). Note, however, that france is definitely not a typical continuous tone image and that for such images special algorithms exist. In [33] a reversible image preprocessing method is proposed, that in case of the JPEG-LS algorithm is reported to improve the compression ratio for the france image from 1.411 bpp to 0.556 bpp. For images of sparse histograms we may get significantly better ratios by applying the histogram packing technique, this way the CALIC compression ratio for washsat, frog, and mountain may be improved by respectively 44.5%, 16.8%, and 18.9% (similar ratio improvement is reported for the JPEG-LS algorithm) [34]. We also notice for all the algorithms, that the average ratios for photographic images containing visible dithering patterns are noticeably worse, than average ratios for the remaining photographic images.

4 CONCLUSIONS

The presented predictive and adaptive lossless image compression algorithm was designed to achieve high compression speed. The prediction errors obtained using simple linear predictor are encoded using codes adaptively selected from the modified Golomb–Rice code family. As opposed to the unmodified Golomb–Rice codes, this family limits the codeword length and allows coding of incompressible data without expansion. Code selection is performed using a simple data model based on the model known from FELICS algorithm. Since updating the data model, although fast as compared to many other modeling methods, is the most complex element of the algorithm, we apply the reduced model update frequency method that increases the compression speed by a couple of hundred percent at the cost of worsening the compression ratio by about 0.5%. This method could probably be used for improving speed of other algorithms, in which data modeling is a considerable factor in the overall algorithm time complexity. The memory complexity is low—algorithm's data structures fit into contemporary CPUs' cache memory.

The presented algorithm was compared experimentally to several others. For continuous tone natural and medical images, on average, its compression ratio is by 5.8% worse, compared to the best ratio obtained by CALIC. Its compression speed is over 2.5 times higher than the speed of JPEG-LS. Compared to the CCSDS SZIP, i.e., to the algorithm that does not employ adaptive data model, the presented algorithm obtains similar compression speed, and by 4.5% better compression ratio.

For some images SFALIC compression ratios are significantly worse than ratios of certain other schemes. The ratios worse than CALIC by up to about 1 bpp were obtained for images that contain significant amount of highly compressible smooth areas, such as medical US images. For compound and computer generated images more sophisticated algorithms may obtain ratios better by even more. For images having sparse histograms, such as MR and CT medical images, significant ratio improvement is possible both in case of SFALIC and the remaining algorithms used for comparisons in this paper. Finding a fast and efficient method of processing the above types of data is a potential field of future algorithm improvement.

Another type of data requiring huge amounts of storage, for which a fast algorithm could be practically useful, is volumetric data. A simple method of extending the described algorithm to exploit the 3-dimensional characteristics of the data, which is an interesting field of further research, might be the use of 3-dimensional prediction functions.

The described algorithm is especially good for:

- big images, since it compresses them with the very high speed—over 60 MB/s on 3.06 GHz CPU, i.e., it needs less than 50 CPU cycles per byte of image,
- natural images of 16-bit depth, since it obtains for them very good compression ratio—it's ratio differs by couple percent from the ratio of the CALIC algorithm,
- noisy images, since as opposed to the other algorithms, it causes almost no data expansion even if the image contains nothing, but noise.

Due to the above advantages it is ideally suited for lossless compression of data to be transmitted from modern medical and general purpose image acquisition devices, that produce images of high bit depths, big sizes, usually containing certain amount of noise. Presented algorithm is an alternative for compressing images to be transmitted over the

network—it may improve the transmission throughput when most other algorithms are too slow. The algorithm could also be used for compressing and decompressing, on the fly, large sets of images that are stored in memory for rapid access.

Acknowledgements

The research was fully supported by the Grant Nr 4 T11C 032 24 of the Ministry of Education and Science of the Republic of Poland. It was carried out at the Institute of Computer Science, Silesian University of Technology, in years 2003 and 2004. The author would like to thank Sebastian Deorowicz and anonymous referees for reviewing the manuscript and suggesting significant improvements.

References

[1] Memon, N. D.; Sayood, K.: Lossless image compression: A comparative study. Proceedings of the SPIE, Still-Image Compression, San Jose, California, 1995, Vol. 2418, pp. 8–20.

[2] Wu, X.; Memon, N.: Context-based, Adaptive, Lossless Image Codec. IEEE Transactions on Communications, April 1997, Vol. 45(4), pp. 437–44.

[3] Wu X.: Efficient Lossless Compression of Continuous-tone Images via Context Selection and Quantization. IEEE Transactions on Image Processing, May 1997, Vol. IP-6, pp. 656–64.

[4] ITU-T; ISO/IEC: Information technology—Lossless and near-lossless compression of continuous-tone still images—Baseline. ITU-T Recommendation T.87 and ISO/IEC International Standard 14495-1, June 1999.

[5] Consultative Committee for Space Data Systems: Lossless Data Compression. CCSDS Recommendation for Space System Data Standards, CCSDS 121.0-B-1, Blue Book, May 1997.

[6] Memon, N; Wu, X.: Recent developments in Context-Based Predictive Techniques for Lossless Image Compression. The Computer Journal, 1997, Vol. 40(2–3), pp. 127–36.

[7] Carpentieri, B.; Weinberger, M. J.; Seroussi, G.: Lossless compression of Continuous-Tone Images. Proceedings of the IEEE, November 2000, Vol. 88(11), pp. 1797–809.

[8] Merhav, N.; Seroussi, G.; Weinberger, M. J.: Optimal prefix codes for sources with two-sided geometric distributions. IEEE Transactions on Information Theory, Vol. IT-46(1), January 2000, pp. 121–35.

[9] Langdon, G.; Gulati, A.; Seiler, E.: On the JPEG model for lossless image compression. Proceedings DCC '92, Data Compression Conference, IEEE Comput. Soc. Press, Los Alamitos, California, 1992, pp. 172–80.

[10] Shannon, C.E.: A Mathematical Theory of Communication. Bell System Technical Journal, 1948, Vol. 27, pp. 379–423, 623–56.

[11] Moffat, A.; Neal, R. M.; Witten, I. H.: Arithmetic Coding Revisited. ACM Transactions on Information Systems, 1998, Vol. 16(3), pp. 256–94.

[12] Huffman, D. A.: A method for the construction of minimum-redundancy codes. Proceedings of the Institute of Radio Engineers 40(9), 1952, pp. 1098–101.

[13] Golomb, S. W.: Run-Length Encodings. IEEE Transactions on Information Theory, July 1966, IT-12, pp. 399–401.

[14] Rice, R. F.: Some practical universal noiseless coding techniques—part III. Jet Propulsion Laboratory tech. report JPL-79-22, 1979.

[15] Deng G., Ye H.: Lossless image compression using adaptive predictor combination symbol mapping and context filtering. Proceedings of the IEEE International Conference on Image Processing, Kobe, Japan, Oct. 1999, Vol. 4, pp. 63–7.

[16] Li X., Orchard M. T.: Edge-Directed Prediction for Lossless Compression of Natural Images. IEEE Transactions on Image Processing, June 2001, Vol. 10(6), pp. 813–17.

[17] ITU-T; ISO/IEC: Information technology—JPEG 2000 image coding system: Core coding system. ITU-T Recommendation T.800 and ISO/IEC International Standard 15444-1, August 2002.

[18] Christopoulos C.; Skodras A.; Ebrahimi T.: The JPEG2000 Still Image Coding System an Overview. IEEE Transactions on Consumer Electronics, November 2000, Vol. 46(4), pp. 1103–27.

[19] Howard, P. G.; Vitter, J. S.: Fast and efficient lossless image compression. Proceedings DCC '93 Data Compression Conference, IEEE Comput. Soc. Press, Los Alamitos, California, 1993, pp. 351–60.

[20] Starosolski, R.: Fast, robust and adaptive lossless image compression. Machine Graphics and Vision, 1999, Vol. 8, No. 1, pp. 95-116.

[21] Starosolski, R.; Skarbek, W.: Modified Golomb–Rice Codes for Lossless Compression of Medical Images. Proceedings of International Conference on E-health in Common Europe, Cracow, Poland, June 2003, pp. 423–37.

[22] Starosolski, R.: Reversing the Order of Codes in the Rice Family. Studia Informatica, 2002, Vol. 23, No. 4(51), pp. 7–16.

[23] Starosolski, R.: Performance evaluation of lossless medical and natural continuous tone image compression algorithms. Proceedings of the SPIE, Medical Imaging, Warsaw, Poland, September 2005, Vol. 5959, CID 59590L (pp. 116–27).

[24] Clunie D. A.: Lossless compression of grayscale medical images—effectiveness of traditional and state of the art approaches. Proceedings of the SPIE, Medical Imaging 2000: PACS Design and Evaluation: Engineering and Clinical Issues, San Diego, California, 2000, Vol. 3980, pp. 74–84.

[25] Wu, X.; Memon, N.: Implementation of Context-based, Adaptive, Lossless Image Coder (CALIC), 1995, `ftp://ftp.csd.uwo.ca/pub/from_wu/` (downloaded 20 Nov. 1998).

[26] Weinberger, M. J.; Seroussi G.; Sapiro G.: LOCO-I: A low complexity, context-based, lossless image compression algorithm. Proceedings DCC'96, IEEE Comput. Soc. Press, Los Alamitos, California, 1996, pp. 140–9.

[27] Weinberger M. J., Seroussi G., Sapiro G.: The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS. IEEE Transactions on Image Processing, August 2000, Vol. 9(8), pp. 1309–24.

[28] Signal Processing and Multimedia Group, University of British Columbia: SPMG/JPEG-LS Implementation, version 2.2, 1999, `ftp://ftp.se.netbsd.org/pub/NetBSD/packages/distfiles/jpeg_ls_v2.2.tar.gz` (downloaded 1 Nov. 2004).

[29] University of New Mexico, Microelectronics Research Center: SZIP science data lossless compression program, combined version: 1.5, 2002, `ftp://ftp.ncsa.uiuc.edu/HDF/lib-external/szip/` (downloaded 22 Sept. 2004).

[30] W3C Recommendation: PNG (Portable Network Graphics) Specification, Version 1.0, 1996, `http://www.w3.org/TR/REC-png.html`.

[31] Santa-Cruz, D.; Ebrahimi, T.: A Study of JPEG2000 Still Image Coding Versus Other Standards. Proceedings of X European Signal Processing Conference EUSIPCO, Tampere, Finland, Sept. 2000, Vol. 2, pp. 673–76.

[32] Starosolski, R.: Compressing images of sparse histograms. Proceedings of the SPIE, Medical Imaging, Warsaw, Poland, September 2005, Vol. 5959, CID 595912 (pp. 209–17).

[33] Pinho, A. J.: Preprocessing techniques for improving the lossless compression of images with quasi-sparse and locally sparse histograms. Proceedings of the IEEE International Conference on Multimedia and Expo, ICME-2002, Lausanne, Switzerland, August 2002.

[34] Pinho, A. J.: A comparison of methods for improving the lossless compression of images with sparse histograms. Proceedings of the IEEE International Conference on Image Processing, ICIP-2002, Rochester, NY, September 2002, Vol. 2, pp. 673–6.