Załącznik 5

Cykl publikacji stanowiących osiągnięcie habilitacyjne

SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2007; **37**:65–91
Published online 8 August 2006 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe.746

Simple fast and adaptive lossless image compression algorithm

Roman Starosolski*,†

Silesian University of Technology, Institute of Computer Science, Akademicka 16, 44–100 Gliwice, Poland

SUMMARY

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 large 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 large images the speed is over 60 MB/s, i.e. the algorithm needs less than 50 CPU cycles per byte of image. Copyright © 2006 John Wiley & Sons, Ltd.

Received 9 March 2005; Revised 24 November 2005; Accepted 1 March 2006

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, which is not aware of the images' use; for example, 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 computed tomography (CT) scanner

*Correspondence to: Roman Starosolski, Silesian University of Technology, Institute of Computer Science, Akademicka 16, 44–100 Gliwice, Poland.
†E-mail: roman.starosolski@polsl.pl

Contract/grant sponsor: Ministry of Education and Science of the Republic of Poland; contract/grant number: 4 T11C 032 24

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 non-negative 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 image two-dimensional data characterized by a specific smoothness. These algorithms are also used for volumetric three-dimensional data. Sometimes such data, as a set of two-dimensional images, is compressed using regular image compression algorithms. Other possibilities include preprocessing volumetric data before compressing it as a set of two-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 two-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 algorithms 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 the intensities of a 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. The prediction error distribution is close to Laplacian, i.e. symmetrically exponential [6–8]. Therefore, the entropy of prediction errors is significantly smaller than the 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 are analyzed and the data model is built. During the second pass the data are 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 reconstruction of the model by the decompression algorithm. In 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 a 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 the symbol's probability,

DOI: 10.1002/spe

then it is a 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 two-dimensional neighborhood. In context determination we use pixel intensities, prediction errors of neighboring pixels or some other function of pixel intensity. A 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 the 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 $-\log_2(\text{prob}(s))$ bits, where $\text{prob}(s)$ 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 the probability assigned by the data model to the actual symbol is high. In image compression, as with 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. the Golomb or Golomb–Rice (GR) family [13,14].

The algorithms used for comparisons in this paper employ two more methods to improve compression ratios 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 also for bilevel images encoded as grayscale.

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 the 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

DOI: 10.1002/spe

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 that, 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, independent from the prediction, is to use two-dimensional image transforms, such as discrete-cosine or wavelet transforms. 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 GR 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 compared with 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 approximately 200% 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 million pixels) and various classes of medical images. The following modalities of medical images were used: computed radiography (CR), computed tomography (CT), magnetic resonance (MR) and ultrasonography (US). Nowadays, consumer acquisition devices, such as cameras or scanners, produce images of evergrowing 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-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.

DOI: 10.1002/spe

Table I. Predictors used in the research.

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

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 nine simple linear predictors. Prediction errors are reordered to obtain the 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 GR 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 three neighboring pixels: the left-hand neighbor (A), the upper neighbor (B) and the upper-left neighbor (C). We use eight predictors of the lossless JPEG algorithm (Table I, Pred0–Pred7) [9], and slightly more complex predictor Pred8, which 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 ($\mathsf{Pred}(X)$) 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

DOI: 10.1002/spe

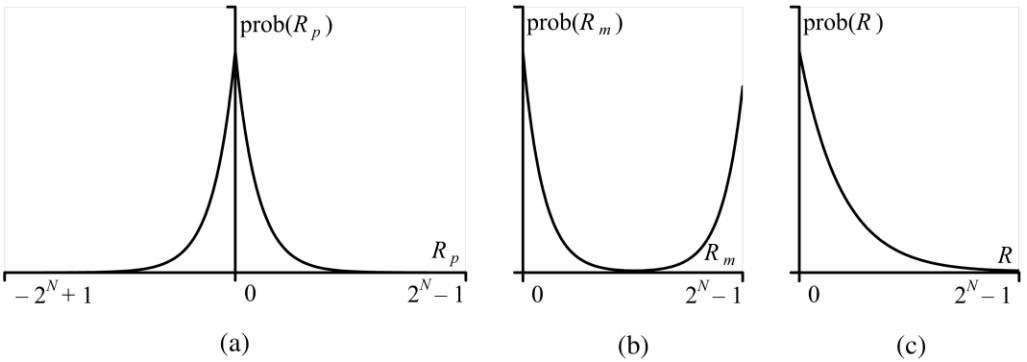

Figure 1. Probability distribution of prediction errors: (a) before modulo reduction; (b) after modulo reduction; (c) after reordering.

the decompression algorithm prior to processing the pixel X. Instead of the above-mentioned formula we use $R_m = (X - \mathsf{Pred}(X)) \bmod 2^N$. For decompression we use $X = (R_m + \mathsf{Pred}(X)) \bmod 2^N$.

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 (Figure 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 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 non-negative integer rank k. In order to encode the non-negative 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 $i \bmod 2^k$ 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 a 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 $\pi_k = \min((l_{\max} - N)2^k, 2^N - 2^k)$. We encode the non-negative 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

DOI: 10.1002/spe

Table II. The code family for integers in the 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	1110•	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	1110•1	10•11	0•111
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	110•11	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

opposite case we output a fixed prefix, $\pi_k/2^k$ ones, and then a 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 II. 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 it 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 a data model, a code of improper (too small) rank to encode a symbol of high value in the data model—the coding of images we deal with here involves alphabet sizes of up to 2^{16} and the code of rank $k=0$ assigns a $(i+1)$-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 the prediction error of the above

DOI: 10.1002/spe

pixel is used as a context; for prediction errors of the remaining pixels the preceding residuum symbol, i.e. the prediction error of the 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 only one symbol 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 the first few 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 with 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 code 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 are even invariable. In order to adapt to gradual changes, we may sample the image, i.e. update the data model, less frequently then every time when the pixel gets coded. Instead we update the model after the coding of selected pixels only. We could simply pick every ith symbol to update the model, but such a constant period could interfere with the image structure. Therefore, after updating the model with some symbol, we select randomly a number of symbols to skip before the next update of the model (Figure 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 its context (the `delay` variable in the Figure 2 is a global variable). In order to permit the decoder to select the same number we use a pseudo-random number generator. To just avoid the interference with an image structure, even the simplest pseudo-random number generator

DOI: 10.1002/spe

```
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.

should suffice. We use the fixed pseudo-random number generator seed—in this way we avoid storing the seed along with the compressed image and we 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. In 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 the case of the algorithm, from which the described algorithm originates, the method increased the compression speed by about 250% at the cost of worsening the ratio by about 1% [20]. To a certain extent, a similar approach was used in the EDP algorithm where the predictor coefficients are determined in an adaptive way by means of relatively complex least-square 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) depend on the image contents.

2.6. Differences from the predecessor algorithm

The most significant differences between the described algorithm and the algorithm 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 and ordering of codes in the family was altered compared to the original Golomb family. Generating codewords from that family was not as simple as in the 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 the whole family may be precomputed and stored in the array of a reasonable size.

DOI: 10.1002/spe

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 a variable number of collective context buckets. The compression started with a single bucket (containing all the contexts), which was subsequently divided into smaller sections. By using this method the compression ratio for the smallest images was improved by over 1%. For the described algorithm, the use of a variable number of buckets resulted in worsening of the average compression ratio (by about 0.2%); only in the case of some small images were the ratios 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 code of the greatest rank. The above feature, along with using a 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

2.7.1. Time complexity

The fast adaptive compression algorithms are of 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 is the coding, c_m is the modeling, c_u is the single model update and p is 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. By updating the model less frequently we accelerate the slowest part of the compression process.

2.7.2. 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, where $c \approx 1$. To perform a 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/. The 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

DOI: 10.1002/spe

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 non-negative integer. Therefore, the 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%, etc. 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 also be selected. We tested the three 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 a specific context may be carried out using single table-lookup to increase the compression speed; buffers for image rows may be allocated for 8-bit pixels instead of 16-bit pixels 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 the 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 the Intel C++ Compiler, version 8.1. To minimize the 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 five 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 an 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 and n is the 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 for preparing the new set was that, to the best of our knowledge, there was no publicly available set of test images containing large high-quality images that were originally acquired with an actual 16-bit depth. The set contains

DOI: 10.1002/spe

Figure 3. Sample *natural* images.

natural continuous tone grayscale images of various bit depths (up to 16 bits), various sizes (up to about 4 million 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 also contains 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 100 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] that contained over 3600 images but, on the other hand, the moderate size of the set allowed it to be made publicly available—it may be downloaded from http://sun.iinf.polsl.gliwice.pl/∼rstaros/mednat/.

The group of natural continuous tone images, i.e. a group of images acquired from scenes available for the human eye (photographic images), was constructed as follows. Four images were acquired from a 36 mm high-quality diapositive film (Fuji Provia/Velvia) using a Minolta Dimage 5400 scanner (Figure 3). In order to minimize the noise, the acquisition was first performed at the device's maximum depth of 16 bits, an optical resolution of 5400 dpi and using multiple sampling of each pixel (16 times

DOI: 10.1002/spe

or four times in the case of the branches image). One image (flower) was softened by setting the scanner focus too close. Then the images' resolution was reduced three times. These images formed a group of 16-bit large images, and were then subject to further resolution reduction (three and nine times) and to bit depth reduction (to 12 and to 8 bits). The set contains the 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 4 000 000 pixels;
- *medium*—12 natural images of various bit depths and size approximately 440 000 pixels;
- *small*—12 natural images of various bit depths and size approximately 49 000 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.

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 of 10 to 16 bits, average size approximately 3 500 000 pixels;
- *ct*—12 medical CT images, nominal depth of 12 to 16 bits, average size approximately 260 000 pixels;
- *mr*—12 medical MR images, nominal depth of 16 bits, average size approximately 200 000 pixels;
- *us*—12 medical US images, 8-bit depth, average size approximately 300 000 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.

The other groups contained in the set were the non-typical images.

- *noise*—nine images with added noise, created using the branches image of various bit depths (8, 12 and 16 bits) and medium size (approximately 440 000 pixels). Noise was added using $v_1 = v_0(1 - a) + ra$, where v_0 denotes original pixel intensity, v_1 is the intensity after adding noise, r is the random value of the 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*—three pseudo-images, intensity of all pixels equals 0, nominal depth of 8, 12 and 16 bits, size approximately 440 000 pixels,
- *random*—three pseudo-images, random intensities of pixels (uniform distribution), bit depth of 8, 12 and 16 bits, size approximately 440 000 pixels.

DOI: 10.1002/spe

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, which triggers the division of 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 that, compared to some other combination, resulted in a 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 being 20% less than the fastest obtained and the compression ratio worsening 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. Figure 4 presents the compression speed and the compression ratio obtained using various update frequencies. Using the reduced update frequency we may get approximately 200% speed improvement at the cost of worsening the compression ratio by about 0.5%. Note that decreasing the update frequency below a certain point prevents further improvement of 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 an 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 a 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):

DOI: 10.1002/spe

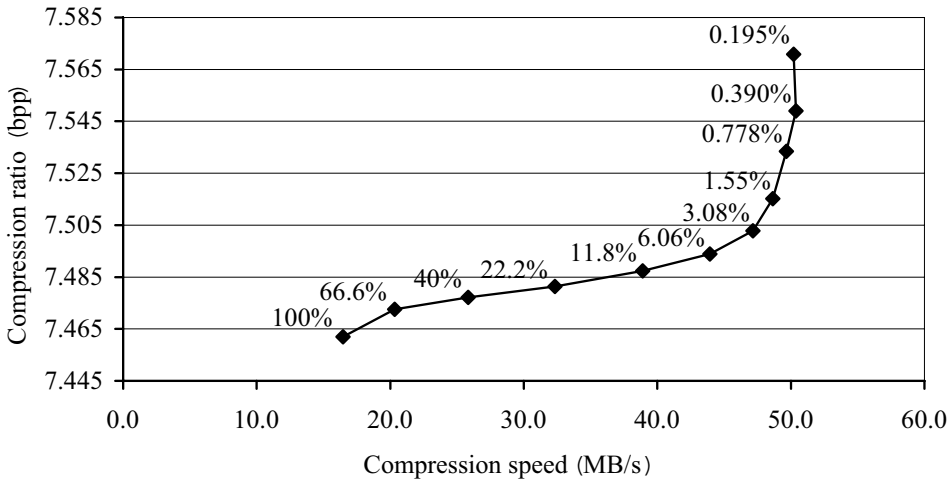

Figure 4. Compression results for various update frequencies (*normal* images).

- 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.

Gradually decreasing the update frequency each time 2048 pixels get coded, compared with compressing the whole image using the same frequency, only affects the results for the smallest images. In this way, for the *small* group, we get a 1.0% compression ratio improvement and a lower speed 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.

Considering results for the data model structures described earlier, (a), (b) and (c) are almost identical. By selecting a model structure other than the default 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 resulting in an expansion of 16-bit images and also to compression with the 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 III and IV 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

DOI: 10.1002/spe

Table III. 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

Table IV. 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

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 the University of Waterloo. The results are reported for the following algorithms.

- CALIC-A—the relatively complex predictive and adaptive image compression algorithm using an 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 seven neighboring pixels, both to determine context and as arguments of the nonlinear

DOI: 10.1002/spe

predictor function. When pixels in the neighborhood are, at most, of two intensity levels, CALIC enters into the so-called binary mode. In the binary mode, for consecutive pixels, we encode information as to 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 an implementation by Wu and Memon [25]. Since this implementation is a binary executable for UltraSparc processors, the compression speed of the 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 the 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 GR family. The algorithm is based on the LOCO-I algorithm [26,27]. In the JPEG-LS algorithm, we use three neighboring pixels for nonlinear prediction, and four 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 two variants: one for images of depths up to 8 bits and the other for image depths of 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 GR 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 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: GR 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 the UNM implementation [29]. It was optimized for the default block size of 16 symbols. Since the largest images (*big* and *cr*) required greater block size, we used a block size of 20 symbols for all the images. For smaller images, compared to the reported results, by using the default block size we obtain a higher compression speed by about 10–20% and the compression ratio from 0.5% worse to 1.2% better, depending on the image group. A 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, this is 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

DOI: 10.1002/spe

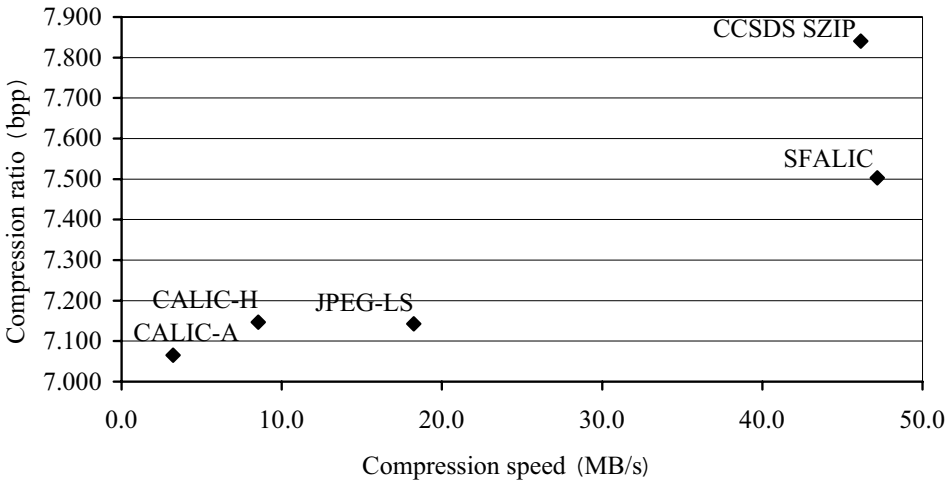

Figure 5. The compression results for various algorithms (*normal* images).

ratios, depending on image or image class, a little better or a little worse than JPEG-LS [24,31]. Compared with SFALIC, the algorithms Lossless JPEG Huffman, PNG and FELICS obtain worse compression ratios for almost all the groups (except for the *us* group for PNG and FELICS, and the *empty* group for PNG) and lower compression speeds for all the groups. JPEG2000 obtains an average ratio 2.3% better than SFALIC and compression speeds of about 15 times lower. SFALIC was also compared to the algorithm from which it originates. For *8bpp* images, SFALIC's predecessor obtained a compression ratio worse by 1.3% and a compression speed lower by 34%.

The SFALIC algorithm is clearly the fastest algorithm among algorithms that use an adaptive data model. The compression speed of the SFALIC algorithm for *normal* images (Figure 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). The 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 a speed only slightly 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 a block size of 20 symbols or by optimizing it for low image bit depths, SFALIC by integrating prediction into loop, which performs a coding and a modeling. The highest compression speed is achieved for the largest images (*big* and *cr*). A compression speed for these groups is over 60 MB/s, i.e. for the largest 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 algorithms examined. The 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

DOI: 10.1002/spe

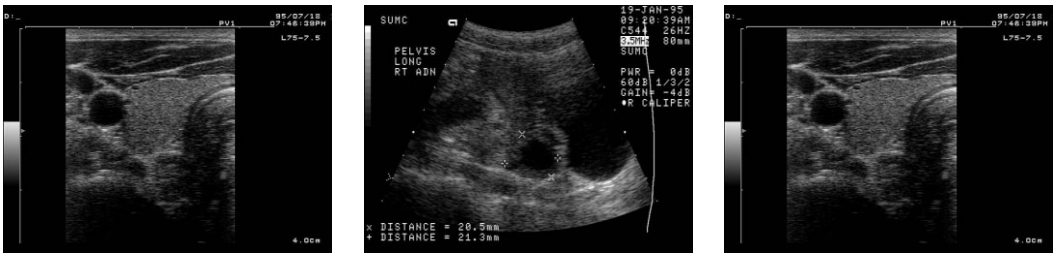

Figure 6. Sample medical *us* images.

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 the much lower complexity of the run mode employed by JPEG-LS for smooth areas found in some images, compared with the complexity of the 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 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 for 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 (Figure 6).

In Figure 7 we compare ratios of SFALIC and JPEG-LS obtained for individual images. The absolute differences of ratios are moderate; the greatest is about 1 bpp. Note that larger differences occur for smaller compression ratios so the relative differences of ratios may be, for highly compressible images, practically important. Therefore, in Figure 8, instead of an absolute ratio we present the relative compression ratio of JPEG-LS expressed as a percentage of the ratio that SFALIC obtained for a specific image. We also mark, by the gray background, images that contain a significant amount of smooth areas. Here, the image is considered to contain a significant amount of smooth areas if at least 15% of its pixels are 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 images). 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 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%. the above observation confirms the significance of using the method of efficient encoding of smooth image areas in the compression algorithm.

DOI: 10.1002/spe

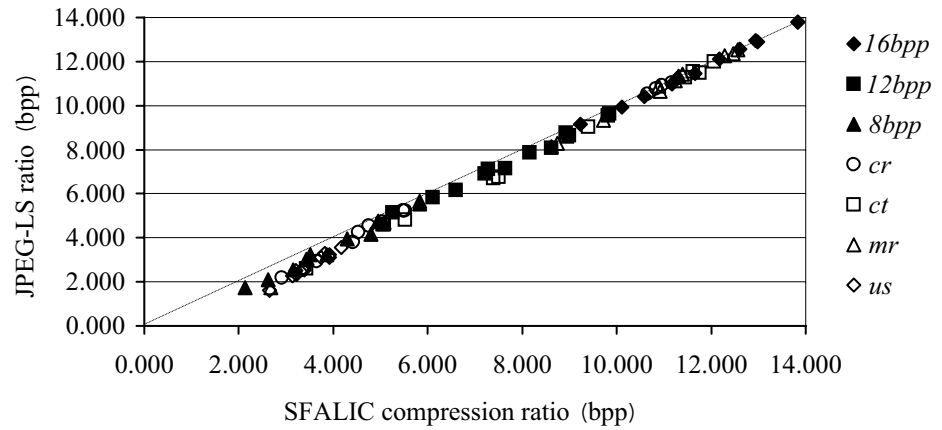

Figure 7. SFALIC and JPEG-LS ratios for individual images.

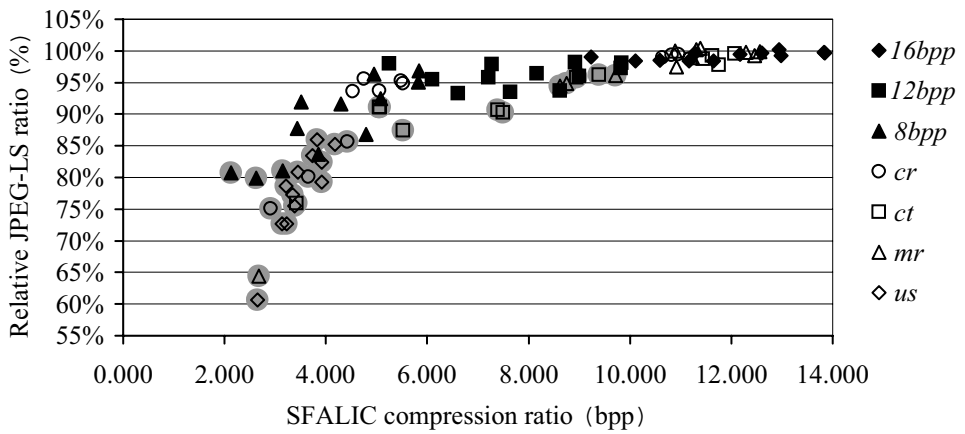

Figure 8. SFALIC ratio and relative JPEG-LS ratio for individual images.

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%. In general 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 fewer than 2^{16} levels. Among 24 such images 21 use below 4000 levels and only three *cr* images

DOI: 10.1002/spe

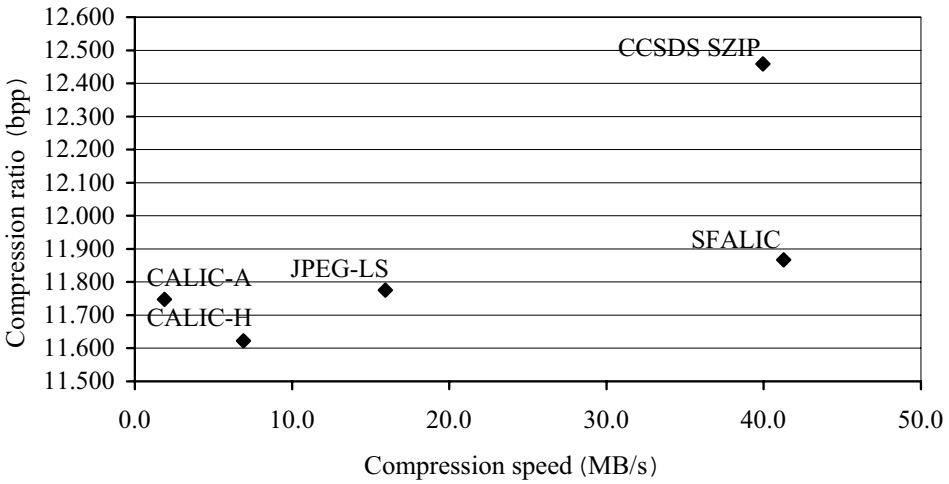

Figure 9. The compression results for various algorithms (*16bpp* images).

use about 25 000 levels. For these three images, the ratio of the 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 are the actual numbers of levels found in these images much smaller than the nominal number of levels, but the levels are distributed throughout almost all the entire nominal intensity range as well. Such characteristics are clearly different from what is expected by a lossless image compression algorithm, both in the case of predictive and of transform coding. In [32] we reported efficient methods of compressing these images. Employing the so-called histogram packing technique we may vastly improve the 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 (Figure 9). For this group the compression ratio of the 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. the group containing 16-bit images) and for *mr* images (of 16-bit nominal depth). The above observations suggest that the small difference in compression ratio between SFALIC and CALIC for *16bpp* images should be attributed to imperfections of other algorithms, rather than to the especially good performance of SFALIC. There is probably still a possibility of improving the compression ratio of CALIC for high bit depth images.

The *empty* pseudo-images (Tables V and VI) 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 for any other group; the greatest speedup is observed for JPEG-LS.

DOI: 10.1002/spe

Table V. 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

Table VI. 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

For non-typical noisy images the compression speed of all algorithms is similar to the average speed for the *medium* group that contains images of similar size. Compression of these images, in the case of some algorithms even with individual images with 50% noise added, still results in compression ratios smaller than the image bit depth, although not by much. The *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. a pseudo-image of characteristics opposite to what is expected in prediction or modeling. The best method of processing incompressible image data is to simply 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 the case of the 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 VII and VIII). 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 images. The france image is computer generated and the library is a compound image. The washsat is an aerial photo image. Compared to other GreySet2 images, the latter three images contain a significantly greater amount of smooth areas (Figure 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

DOI: 10.1002/spe

Table VII. The compression speed, BragZone GreySet2 images (MB/s).

Image	Pixels	CALIC-A	CALIC-H	JPEG-LS	CCSDS SZIP	SFALIC
barb	262 144	1.8	4.2	10.3	28.7	34.3
boat	262 144	1.9	4.4	10.6	28.7	36.3
france	333 312	4.3	10.7	23.1	45.5	40.0
frog	309 258	2.0	5.3	10.3	30.4	35.8
goldhill	262 144	1.7	4.1	10.3	29.0	35.4
lena	262 144	1.9	4.4	10.5	28.8	36.6
library	163 328	1.6	4.2	11.0	24.1	27.4
mandrill	262 144	1.6	4.1	9.5	27.9	33.7
mountain	307 200	1.9	5.1	10.1	28.8	33.6
peppers	262 144	1.8	4.1	10.4	28.5	34.8
washsat	262 144	2.1	4.8	11.5	30.2	36.3
zelda	262 144	2.1	4.6	11.0	29.5	37.1
Average GreySet2	267 521	2.1	5.0	11.5	30.0	35.1
Average *medium8bpp*	440 746	3.1	7.7	12.1	36.0	44.1

Table VIII. The compression ratio, BragZone GreySet2 images (bpp).

Image	Pixels	CALIC-A	CALIC-H	JPEG-LS	CCSDS SZIP	SFALIC
barb	262 144	4.453	4.569	4.733	5.775	5.315
boat	262 144	4.151	4.233	4.250	5.153	4.632
france	333 312	0.823	1.684	1.411	2.425	3.736
frog	309 258	5.853	6.232	6.049	6.657	6.536
goldhill	262 144	4.629	4.719	4.712	5.280	4.870
lena	262 144	4.110	4.184	4.244	5.046	4.567
library	163 328	5.012	5.228	5.101	5.858	6.025
mandrill	262 144	5.875	6.031	6.036	6.374	6.256
mountain	307 200	6.265	6.538	6.422	6.717	6.840
peppers	262 144	4.378	4.488	4.489	5.167	4.933
washsat	262 144	3.670	4.107	4.129	4.825	4.526
zelda	262 144	3.862	3.973	4.005	4.838	4.289
Average GreySet2	267 521	4.424	4.665	4.632	5.343	5.210
Average *medium8bpp*	440 746	3.630	3.826	3.701	4.400	4.153

DOI: 10.1002/spe

Figure 10. BragZone GreySet2 images with smooth areas.

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.

In general, these results adhere to the results obtained for the new set. Average compression speeds for GreySet2 images are slightly lower than the speeds obtained for *medium8bpp* images, which are of slightly greater size. The compression speed for all the photographic images, for a specific algorithm, does not vary significantly. An increased speed is observed for the france image, which contains smooth regions and, by all the algorithms, is compressed faster than other images of similar size. In the case of CALIC and JPEG-LS, the speed is more than doubled compared with other images of similar size. The library image also contains smooth regions, but this image is smaller than the others—as could be expected SFALIC compresses this image more slowly compared with the larger images. For some other algorithms, the presence of smooth regions seems to have a 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 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 a little worse than in the 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 a significant amount of smooth areas. If we exclude these images from the comparison, then the CALIC-A ratio becomes smaller than SFALIC's by only 10.5%. 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 for large differences among CALIC-A and other algorithms are the CALIC's binary mode (capable of encoding sequences of symbols of two intensity levels), the arithmetic coder used (which as opposed to GR codes is capable of efficiently encoding any probability distribution), the predictors

DOI: 10.1002/spe

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 only uses the prediction error of a single neighbor of the current pixel to determine the 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, which in the 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; in this way the CALIC compression ratio for washsat, frog and mountain may be improved by 44.5%, 16.8% and 18.9%, respectively (a similar ratio improvement is reported for the JPEG-LS algorithm) [34]. We also notice that for all the algorithms the average ratios for photographic images containing visible dithering patterns are noticeably worse than the 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 a simple linear predictor are encoded using codes adaptively selected from the modified GR code family. As opposed to the unmodified GR 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 the FELICS algorithm. Since updating the data model, although fast when compared with 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 approximately 200% at the cost of worsening the compression ratio by about 0.5%. This method could probably be used for improving the speed of other algorithms, in which data modeling is a considerable factor in the overall algorithm time complexity. The memory complexity is low—the algorithm's data structures fit into contemporary CPUs' cache memory.

The algorithm presented was compared experimentally to several others. For continuous tone natural and medical images, on average, its compression ratio is 5.8% worse, compared with 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 an adaptive data model, the presented algorithm obtains similar compression speed, and a 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 a significant amount of highly compressible smooth areas, such as medical US images. For compound and computer-generated images, more sophisticated algorithms may improve ratios even further. For images having sparse histograms, such as MR and CT medical images, significant ratio improvement is possible both in the 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

DOI: 10.1002/spe

the three-dimensional characteristics of the data, which is an interesting field of further research, might be the use of three-dimensional prediction functions.

The described algorithm is especially good for:

- large images, since it compresses them with the very high speed—over 60 MB/s on a 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 very good compression ratio for them—its ratio differs by a couple of percent from the ratio of the CALIC algorithm;
- noisy images, as when compared with 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, which produce images of high bit depths, large sizes and usually containing a certain amount of noise. The algorithm presented is an alternative for compressing images to be transmitted over a 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

This research was fully supported by grant No. 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 2003 and 2004. The author would like to thank Sebastian Deorowicz and the anonymous referees for reviewing the manuscript and suggesting significant improvements.

REFERENCES

1. Memon ND, Sayood K. Lossless image compression: A comparative study. *Proceedings of SPIE* 1995; **2418**:8–20.
2. Wu X, Memon N. Context-based, adaptive, lossless image codec. *IEEE Transactions on Communications* 1997; **45**(4):437–444.
3. Wu X. Efficient lossless compression of continuous-tone images via context selection and quantization. *IEEE Transactions on Image Processing* 1997; **IP-6**:656–664.
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; **40**(2–3):127–136.
7. Carpentieri B, Weinberger MJ, Seroussi G. Lossless compression of continuous-tone images. *Proceedings of the IEEE* 2000; **88**(11):1797–1809.
8. Merhav N, Seroussi G, Weinberger MJ. Optimal prefix codes for sources with two-sided geometric distributions. *IEEE Transactions on Information Theory* 2000; **IT-46**(1):121–135.
9. Langdon G, Gulati A, Seiler E. On the JPEG model for lossless image compression. *Proceedings of the 1992 Data Compression Conference (DCC '92)*. IEEE Computer Society Press: Los Alamitos, CA, 1992; 172–180.
10. Shannon CE. A mathematical theory of communication. *Bell System Technical Journal* 1948; **27**:379–423, 623–656.
11. Moffat A, Neal RM, Witten IH. Arithmetic coding revisited. *ACM Transactions on Information Systems* 1998; **16**(3):256–294.
12. Huffman DA. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers* 1952; **40**(9):1098–1101.

DOI: 10.1002/spe

13. Golomb SW. Run-length encodings. *IEEE Transactions on Information Theory* 1966; **IT-12**:399–401.
14. Rice RF. Some practical universal noiseless coding techniques—part III. *Technical Report JPL-79-22*, Jet Propulsion Laboratory, 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, October 1999, vol. 4. IEEE Computer Society: Los Alamitos, CA, 1999; 63–67.
16. Li X, Orchard MT. Edge-directed prediction for lossless compression of natural images. *IEEE Transactions on Image Processing* 2001; **10**(6):813–817.
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* 2000; **46**(4):1103–1127.
19. Howard PG, Vitter JS. Fast and efficient lossless image compression. *Proceedings of the 1993 Data Compression Conference (DCC '93)*. IEEE Computer Society Press: Los Alamitos, CA, 1993; 351–360.
20. Starosolski R. Fast, robust and adaptive lossless image compression. *Machine Graphics and Vision* 1999; **8**(1):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. Academic Computer Centre Cyfronet: Cracow, Poland, 2003; 423–437.
22. Starosolski R. Reversing the order of codes in the Rice family. *Studia Informatica* 2002; **23**(4(51)):7–16.
23. Starosolski R. Performance evaluation of lossless medical and natural continuous tone image compression algorithms. *Proceedings of SPIE* 2005; **5959**:116–127.
24. Clunie DA. Lossless compression of grayscale medical images—effectiveness of traditional and state of the art approaches. *Proceedings of SPIE* 2000; **3980**:74–84.
25. Wu X, Memon N. Implementation of context-based, adaptive, lossless image coder (CALIC). ftp://ftp.csd.uwo.ca/pub/from_wu/ [20 November 1998].
26. Weinberger MJ, Seroussi G, Sapiro G. LOCO-I: A low complexity, context-based, lossless image compression algorithm. *Proceedings of the 1996 Data Compression Conference (DCC'96)*. IEEE Computer Society Press: Los Alamitos, CA, 1996; 140–149.
27. Weinberger MJ, Seroussi G, Sapiro G. The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS. *IEEE Transactions on Image Processing* 2000; **9**(8):1309–1324.
28. Signal Processing and Multimedia Group, University of British Columbia: SPMG/JPEG-LS Implementation, version 2.2. ftp://ftp.se.netbsd.org/pub/NetBSD/packages/distfiles/jpeg_ls_v2.2.tar.gz [1 November 2004].
29. University of New Mexico, Microelectronics Research Center: SZIP science data lossless compression program, combined version: 1.5. ftp://ftp.ncsa.uiuc.edu/HDF/lib-external/szip/ [22 September 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, September 2000, vol. 2. EURASIP European Association for Signal, Speech, and Image Processing: Lausanne, Switzerland, 2000; 673–676.
32. Starosolski R. Compressing images of sparse histograms. *Proceedings of SPIE* 2005; **5959**:209–217.
33. Pinho AJ. 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. IEEE Computer Society: Los Alamitos, CA, 2002.
34. Pinho AJ. 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. IEEE Computer Society: Los Alamitos, CA, 2002; 673–676.

DOI: 10.1002/spe

Theoretical and Applied Informatics
ISSN 1896–5334
Vol. 19 (2007), no. 2
pp. 103–115

Parallelization of an adaptive compression algorithm using the reduced model update frequency method

Roman Starosolski [a]

[a]Institute of Computer Science,
Silesian University of Technology
Akademicka 16, 44-100 Gliwice, Poland
roman.starosolski@polsl.pl

Received 22 September 2007. Revised 10 October 2007. Accepted 23 October 2007.

Abstract: Modeling and coding is the most complex part of many adaptive compression algorithms; it is also an inherently serial process. The method of reduced model update frequency is a modification of the typical adaptive scheme, which was originally employed in order to improve the speed of modeling at the cost of a negligible worsening of the modeling quality. In this paper, we notice that the above method permits to parallelize the compression process. We find that for the Itanium 2 processor the speed of coding and modeling in a SFALIC image compression algorithm may be improved by about 50% through exploiting the fine-grained parallelism, also the medium-grained parallelism may be exploited in a significantly larger extent.

Keywords: Parallel processing; Fine- and medium-grained parallelism; Adaptive modelling; Data compression; Image compression

1. Introduction

Designing compression algorithms, we aim at encoding the given input sequence of symbols using the smallest number of bits. In order to encode the symbol s optimally, we should use -$\log_2$(prob(s)) bits, where prob(s) is the probability of the symbol s occurrence [12]. Employing an arithmetic coder [10] we may get arbitrarily close to the above optimum, however practical implementations of the arithmetic coding are relatively slow. We may also use prefix codes, such as Huffman codes [7]. In this case we encode each symbol from a sequence with a binary codeword of integer length. Prefix codes may be inefficient if the probabilities are high, but in practice are much faster.

Huffman codes may be used for any probability distribution; there are also prefix codes optimal for certain probability distributions, like Golomb [5] or Golomb-Rice [11] families optimal for exponential distribution. Knowing the type of the distribution permits faster coding and does not require estimating of the whole unknown distribution – instead we estimate the parameter of a known distribution. The probability distribution is estimated by 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. In the adaptive modeling, the distribution for specific symbol is estimated based on statistics gathered from the already processed symbols. This way, as opposed to two-pass schemes, we do not have to transmit explicitly the data model since the decompression algorithm is able to reconstruct the model on-line from already decoded symbols. After encoding of the symbol we update the data model (Fig. 1). The coding and modeling process is inherently serial; before estimating the probability distribution for the next symbol, and before encoding the next symbol, we have to update the model with information on the previous symbol.

```
loop
  read symbol s
  estimate coding parameter r for symbol s
  encode symbol s using r
  update model with symbol s
endloop
```

Fig. 1. Adaptive modeling and coding

In the SFALIC lossless image compression algorithm [14] a method of reduced model update frequency (RUF) was employed. The motivation for introducing the RUF 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 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 symbol, we select randomly a number of symbols to skip before the next update of the model (Fig. 2). In order to permit the decoder to select the same number we use pseudo-random number generator. 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 the compression, the data model should adapt to the image data characteristics as quickly as possible, so we start the compression using all symbols for updating the model (i.e., we use the update

frequency of 100%) and then we gradually decrease the model update frequency, until it reaches some minimal value. The RUF method allowed significant improvements of compression speed at the expense of worsening the compression ratio insignificantly. Using the update frequency of 3.08% (i.e., the default update frequency of the SFALIC algorithm) we got 3 times greater compression speed and 0.5% worse compression ratio compared to the update frequency of 100%.

```
delay := random(range)
loop
  read symbol s
  estimate coding parameter r for symbol s
  encode symbol s using r
  if delay = 0 then
    update model with symbol s
    delay := random(range)
  else
    delay := delay-1
  endif
endloop
```

Fig. 2. The reduced update frequency (RUF) method

Many lossless image compression algorithms, including SFALIC, are predictive. In a predictive algorithm we do not encode pixels explicitly, instead we use a predictor function to guess the pixel intensities and then we calculate prediction errors, i.e., differences between actual and predicted pixel intensities. Next, we encode the sequence of symbols, which are prediction errors. To calculate the predictor for a specific pixel we usually use intensities of small number of already processed pixels neighboring it. For typical grayscale continuous-tone images, the pixel intensity distribution is close to uniform. Prediction error distribution is close to Laplacian, i.e., symmetrically exponential [3], making prediction errors easier and faster to compress.

The detailed description of the SFALIC algorithm exceeds the scope of this paper, hence we just briefly characterize it here and refer the reader to [14] for details. It is a predictive algorithm; for prediction we use a couple of already-processed neighbors of a specific pixel. A sequence of prediction errors is encoded using a modified Golomb-Rice family of limited codeword length. A context data model, based on a model of the FELICS algorithm [6], is used to adaptively estimate the coding parameter.

The RUF method was introduced in order to improve the compression speed by means of reducing the time spent, by the compression algorithm, on updating the data model. However, we found that this method permits to parallelize the compression process. In the fine-grained parallelism or instruction-level parallelism single CPU performs in parallel several operations of a single process, which is possible since contemporary

CPUs contain multiple execution units capable of parallel processing. By an analogy to multiprocessor systems (for introduction and classification of parallel processing architectures see e.g. [4]), we may recognize two types of fine-grained parallelism: MIMD and SIMD. In the Multiple-Instruction-Multiple-Data (MIMD) fine-grained parallelism several execution units of a single CPU operate in parallel – each of them executes its own instruction. In the Single-Instruction-Multiple-Data (SIMD) parallelism we execute in parallel the same instruction for multiple arguments. Contemporary CPUs contain special SIMD execution units designed exclusively for operating on multiple arguments, actually packed into special, long registers. Obvious candidates for parallelization are inner loops. In the case of the SIMD processing several iterations of the loop body are performed in parallel, this technique is called loop vectorization. We may employ SIMD processing provided that there are no inter-iteration dependencies and that the same sequence of operations is executed for each iteration, e.g. that there are no operations performed conditionally. In the MIMD processing the parallelization is possible even when both the above conditions are not met – since for consecutive iterations we may perform in parallel separate parts of the loop body (these parts have to be free of inter-iteration dependencies; the technique is called software pipelining [2]). In the medium-grained parallelism we execute multiple threads using multiprocessor system.

The reminder of this paper is organized as follows. The idea of parallelizing the adaptive image compression algorithm is introduced in section 2. In section 3 we analyze experimentally effects of the fine-grained parallelization of coding and modeling in the SFALIC algorithm and then we estimate effects of medium-grained parallelization of the whole compression process. Section 4 summarizes the research.

2. Idea of parallelization

In the RUF method the model is updated after coding of some symbols only (recall Fig. 2); we will call the sequence of symbols encoded between two consecutive model updates the block of symbols. Since for a block of symbols the model is constant, operations performed using the data model for symbols from the block may run in parallel. Therefore, in parallel we may find coding parameters using the data model as well as generate codewords. Provided that the length of the block of symbols is sufficient, parallel processing of pixels within the block should result in a speedup of coding and modeling. The parallelizable variant of the RUF method (RUF-P) is presented on the Fig. 3. The only operation that still has to be performed sequentially for each of the block symbols is outputting the variable length binary codewords. Storing the generated codewords might be a problem, because the codeword lengths are variable. In the SFALIC algorithm, however, the codeword length is limited to 31 bits, so the codewords may be stored as pairs of integers <codeword_length, codeword_bits>.

For RUF-P method in the SFALIC algorithm the MIMD processing may be effective. The steps of coding parameter determination and codeword generation require just several simple integer operations (table lookup, AND, OR, add, subtract, shift) feasible for typical CPUs' execution units, including SIMD units. Unfortunately they contain alternative paths of symbol processing selected dynamically based on the symbol value which makes the body of the `parfor` loop in Fig. 3 not implementable as a sequence of SIMD operations (although some operations inside this loop might be implemented as SIMD operations). The fine-grained MIMD parallelization of this loop may be effective, provided that the CPU contains sufficient resources (non-SIMD execution units and general-purpose integer registers).

```
delay := random(range)
loop
  read symbols s0 … sdelay
  parfor i:=0 to delay
    estimate coding parameter ri for symbol si
    generate codeword cwi of symbol si using ri
  endparfor
  output codewords cw0 ... cwdelay
  update model with symbol sdelay
  delay := random(range)
endloop
```

Fig. 3. The idea of fine-grained parallelization using the RUF method (RUF-P)

The RUF method permits to parallelize operations performed using the data model. The outputting of the already-generated variable length codewords cannot be parallelized itself; it must be performed sequentially for all the symbols we compress. Fortunately it may be performed in parallel to updating of the data model or, provided that we use two arrays of generated codewords, in parallel to determination of coding parameters and generation of codewords for the next group of symbols. So in this part of the compression process may exploit the medium-grained parallelism.

As mentioned above, the model may be updated while outputting the codewords. Actually almost all of the model update operations may also be performed in parallel to determination of coding parameters and generation of codewords. To start updating the model we need to know the symbol, after encoding of which, the model should be updated – this information is available as soon as the number of symbols to be encoded before the next model update is known and the symbols are read. The data model of the adaptive compression algorithm stores data on characteristics of the already processed symbols, which is needed to select a coding parameter. In the context model we store such data for each context separately. Knowing a symbol, along with its context, that

after encoding of the block of symbols should be used to update the model, we can calculate new data describing characteristics for the context in parallel to encoding of the block. After encoding, the context data in the model may be updated by just overwriting context data with updated information. Alternatively, for memoryless model, we may have two models and switch between them. Yet another possibility is natural to the RUF method. We use the model to find the coding parameter, but since we do not update the model each time we retrieve the parameter, it's better to find the parameter after updating the model and store it, than to find it each time the parameter is needed. In the SFALIC's context data model, an additional structure stores coding parameters selected for contexts (single integer per context). Only the updating of this additional structure, i.e., single assignment operation per model update, has to be performed after determination of parameters for block of symbols.

3. Experimental results and discussion

For experiments we used the unmodified implementation of the SFALIC algorithm, version 04, as well as its variant modified as in Fig. 3. Actual differences between these two implementations were smaller, than it appears from Figures 2 and 3. Unmodified implementation (RUF) is optimized and also processes symbols in blocks (Fig. 4). On the other hand the RUF-P variant is implemented using the `for` keyword, not the `parfor` (recall RUF-P variant in Fig. 3), so it's the compiler's responsibility to parallelize this loop exploiting CPU's resources. Among other things, in order not to favor the RUF-P variant, we relied solely on compiler abilities to recognize and take advantage of the parallelizable code, i.e., we did not use explicit parallel programming tools like e.g. MPI. Both implementations process images in rows: each row of image pixels is read from the input file, then the prediction for the whole row is performed and a resulting row of residuum symbols is stored in a memory buffer. Then modeling and coding takes place resulting in row of residuum symbols being encoded to another memory buffer, which is written to the output file afterwards.

Most of experiments were performed using an Itanium 2 processor because of its large number of resources (execution units and registers) [9]. This CPU contains 9 integer arithmetic execution units (6 of them being general-purpose ones), 10 multimedia execution units (capable of SIMD integer processing, 6 of them general-purpose), and 128 general-purpose 64-bit integer registers. For experiments we employed a computer equipped with two Itanium 2 processors (1.4 GHz, 256 kB L2 cache, 3 MB L3 cache) running Linux (kernel 2.4). However, in order to evaluate the fine-grained parallelism of a single CPU, the application executables were compiled as single-threaded. We used Intel optimizing C++ Compiler 9.0. The implementations used permit to skip certain stages of the compression process (writing output file, modeling and coding, etc.) so we

were able to measure precisely real execution time of selected stages of the compression process by subtracting results obtained for compression with certain stage skipped or left. For experiments we used for a set of 12 big natural continuous-tone grayscale images, of depths from 8 to 16 bits per pixel and approximate size of 4 millions pixels; images are from a set described in [13] (group "*big*"). As mentioned in the introduction, the update frequency is variable; compression process starts with updating the model after encoding of each symbol, and then the frequency is gradually decreased until it reaches specified destination frequency value. In order to obtain results for a certain constant update frequency only, we subtracted results obtained for upper half of the image from the results for whole image. We also checked that performing measurements just for whole image would change results by about a percent or less (which was expected since modeling frequency stops decreasing after algorithm processes first 12 thousands of pixels – 0.3% of the whole image). The execution time was measured by running the executables several times. The time of the first run was ignored and the collective time of other runs (executed for at least two seconds) was measured and then averaged. The time measured was a sum of user and system time, as reported by the operating system after application execution. The results are expressed in average number of ticks (CPU clock cycles) per image pixel, or in percents relative to the time of whole compression process.

```
delay := random(range)
loop
  read symbols s0 … sdelay
  for i:=0 to delay
    estimate coding parameter r for symbol si
    generate codeword cw of symbol si using r
    output codeword cw
  endfor
  update model with symbol sdelay
  delay := random(range)
endloop
```

Fig. 4. Actual implementation of the RUF method in the unmodified SFALIC implementation

We have compared speeds for the default SFALIC model update frequency (3.08%). The parallelized version RUF-P requires on average 22.3 ticks per pixel to find the coding parameter, generate the codeword, output the codeword and for some pixels only to update the model. The unmodified version requires 32.9 ticks, so the speedup is 1.47. The coding and modeling speed increase by about 50% may be practically useful, justifying the effort to alter the implementation sources, especially if we notice that it is the speedup of the whole modeling and coding process, not of just the parallelizable part of it.

On the Fig. 5 we report speedups for various update frequencies available in the implementations. As could be expected, for update frequencies greater than a certain threshold (22.2%) the RUF-P variant is slower than the unmodified version and the RUF-P variant speedup increases as the update frequency decreases. Apparently, for update frequencies greater than 22.2% (which implies very small sizes of blocks of symbols that are processed in parallel – on average 2.5 symbols or less) the cost of parallelization (e.g. of storing generated codewords in a memory buffer and of retrieving them) is greater than the savings due to parallel processing of symbols within the block. Therefore the speedup would be greater if we used update frequency smaller then the default one, however, the performance of the lossless image compression algorithm is not judged based solely on the compression speed; the usual primary criterion is the compression ratio. For the whole set from which test images were taken, using the default model update frequency worsened the average compression ratio by 0.5% compared to updating the data model each time a pixel gets coded; below the default update frequency ratios start to worsen much faster so using smaller frequencies for this algorithm may in practice be not justified.

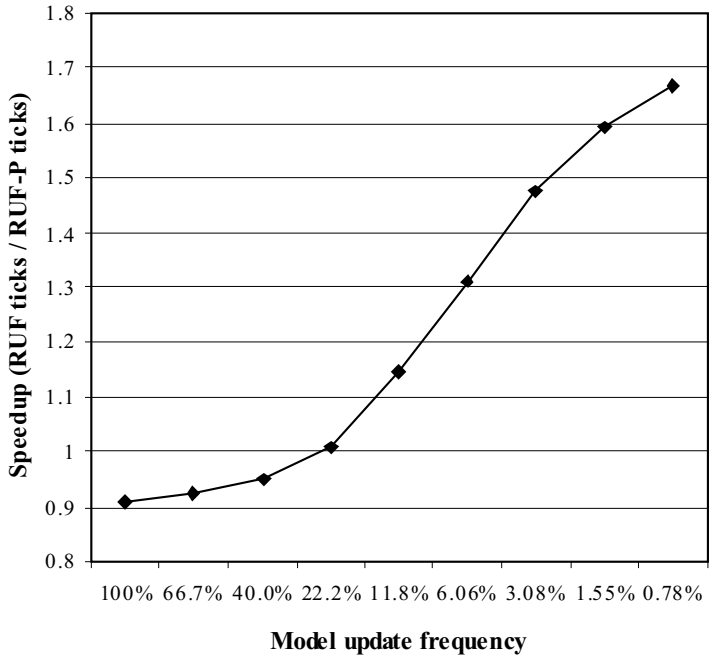

Fig. 5. Fine-grained coding and modeling speedups for various update frequencies

Similar experiments were performed for a couple of other systems described below.

- Sun UltraSparc IIIi (1.06 GHz, 1 MB L2 cache, Solaris 9, Sun C 5.7 compiler). This CPU has 3 integer arithmetic execution units, 2 SIMD execution units, and 160 general-purpose 64-bit integer registers (of which 32 are accessible to a given function) [15].
- AMD Athlon64 3200 in 64-bit mode (2 GHz, 512 kB L2 cache, Linux kernel 2.6, Intel C++ 9.0 and GCC 4.0.0 compilers). This CPU has 4 integer arithmetic execution units, 3 SIMD execution units, 16 general-purpose 64-bit integer registers, 8 multimedia 64-bit registers (for SIMD MMX extensions), and 16 multimedia 128-bit registers (for SIMD SSE2 extensions) [1].
- Intel Xeon DP 3.06 (3.06GHz, 512kB L2 cache, Linux kernel 2.6, Intel C++ 9.0 and GCC 3.3.3 compilers). This CPU has 3 integer arithmetic execution units, 3 SIMD execution units, 8 general-purpose 32-bit integer registers, 8 multimedia 64-bit registers (for SIMD MMX extensions), and 8 multimedia 128-bit registers (for SIMD SSE2 extensions) [8].
- Intel Itanium 2 (1.4 GHz, 256 kB L2 cache, 3 MB L3 cache, Linux kernel 2.4) – as already described, except that we used also GCC 3.2.2 compiler. This CPU has 9 integer arithmetic execution units of which 6 are general-purpose ones, 10 multimedia execution units (capable of SIMD integer processing, 6 of them general-purpose), and 128 general-purpose 64-bit integer registers.

It is worth mentioning that UltraSparc, Athlon64, and Xeon contain smaller number of resources, especially of non-SIMD integer arithmetic execution units potentially useful for fine-grained parallelization, then Itanium 2 (3-4 vs. 9). The Intel and Sun compilers are able to optimize efficiently code placed in different source files; GCC hasn't got such functionality, but it can be simulated by preparing a module that includes all the required files; results reported for GCC are obtained this way. In case of the Itanium we did not analyze whether the speedup is due to MIMD processing or due to SIMD processing. Although all the examined architectures support the SIMD processing, for Itanium and UltraSparc we'd have to reverse engineer binaries to check whether SIMD instructions are actually used. For Athlon64 and Xeon processors, in case of the code compiled using Intel compiler we have a choice whether to use SIMD extensions, gaining access to additional execution units and registers (rows marked "SIMD" in table 1), or to use basic architecture.

Our idea of fine-grained parallelization improved the coding and modeling speed only in the case of Itanium 2 CPU and only for one of the two compilers tested. Clearly, the older version of GCC, as opposed to recent Intel compiler, did not exploit the possibility of fine-grained parallelization. For all the other systems, if the speeds of RUF

and RUF-P variants differ by more than a couple of percents, then the RUF-P variant is slower. As mentioned in section 2 the fine-grained parallelization requires sufficient CPU resources. Probably the simpler CPUs' resources are already fully exploited by the RUF variant. Results obtained on Athlon64 and Xeon by non-SIMD and SIMD are as expected; access to SIMD execution units does not improve the speed of the RUF-P variant.

System CPU	Compiler	Coding and modeling time		
		RUF	RUF-P	speedup
Intel Itanium 2	Intel 9.0	32.9	22.3	1.47
Intel Itanium 2	GCC 3.2.2	39.6	45.2	0.88
Sun UltraSparc IIIi	Sun C 5.7	32.9	38.1	0.86
AMD Athlon64	Intel 9.0	28.0	33.3	0.84
AMD Athlon64	Intel 9.0 SIMD	26.1	33.6	0.78
AMD Athlon64	GCC 4.0.0	27.8	31.5	0.88
Intel Xeon	Intel 9.0	38.4	39.7	0.97
Intel Xeon	Intel 9.0 SIMD	38.2	42.2	0.91
Intel Xeon	GCC 3.3.3	49.5	48.7	1.02

Tab. 1. Coding and modeling time for various systems [ticks per symbol], default update frequency (3.06%)

Stage	RUF	RUF-P
prediction	21%	27%
coding parameter determination and codeword generation	53%	20%
outputting codewords		20%
model updating	12%	16%
reminder (file i/o, etc.)	14%	17%

Tab. 2. Execution time of stages of the compression process (time relative to the time of a whole compression process)

As mentioned in section 2, the RUF-P method also permits certain medium-grained parallelizations. We measured the execution time of various stages of the compression process. In Table 2 we report the time expressed in percents of the time required to perform the complete compression process including the file i/o and the prediction. Results were obtained on the Itanium 2 processor using executables compiled by Intel compiler and for the default 3.08% model update frequency. Except for the model update time, results were calculated based on measurements of compression process with skipping of certain stages. Model update time was estimated based on comparing results for differ-

ent model update frequencies; the update time may include overhead of initializing the processing of a block of symbols and thus may be overstated at the expense of time of coding parameters determination and codewords generation.

For the RUF variant we assume that the model update may be performed in parallel to coding parameter determination and codeword generation and to outputting of codewords since actual implementation is as good for it (Fig. 4) as the RUF-P. From results presented in Table 2, we can expect for the RUF variant the medium-grained parallelization speedup of no more than about 2.

For the RUF-P variant the figures in Table 2 are relative to the speed already increased by the fine-grained parallelization. Let's estimate what further speedup can be achieved by employing medium-grained parallelism. The prediction stage itself may be parallelized, e.g., it may be performed in parallel for halves of a row of image pixels, so it does not limit attainable speedups. The speedup is now limited by outputting of codewords – the slowest inherently serial part of the compression process. It is also limited by the coding parameter determination and codeword generation, i.e., the stage that in the RUF-P variant may run in parallel to outputting of codewords. Note, that this stage can be performed independently for two halves of a block of symbols, but in practice the thread synchronization costs may thwart parallelization effects since the block size is small and variable (for the update frequency used, it is in the range [1, 64], 32.5 on average). Now, the slowest stages that limit the speedup attainable through medium-grained parallelization require about 20% of the time of a whole compression process. Hence for the RUF-P variant we can expect the medium-grained parallelization speedup of no more than 5.

4. Conclusions

Modeling and coding is the most complex part of many adaptive compression algorithms; it is also an inherently serial process. The method of reduced model update frequency is a modification of the typical adaptive scheme, which was originally employed in the SFALIC lossless image compression algorithm in order to improve the speed of modeling at the cost of a negligible worsening of the modeling quality.

In this paper, we notice that the above method permits to parallelize the compression process. We experimentally evaluate the fine-grained parallelization speedups and estimate the medium-grained parallelization effects. For the Itanium 2 CPU and a recent Intel compiler, the fine-grained parallelization improved the coding and modeling speed of SFALIC by about 50%. For other architectures and for various compilers no significant speedups were observed indicating that the speedup is conditioned on both the CPU architecture and the compiler. Reduced update frequency method also allows greater extent of medium-grained parallelism since it allows splitting, into two threads, the slowest

part of the compression process (and accelerates it through fine-grained parallelism as well). The presented method may be used for parallelizing some other adaptive algorithms in which it is applicable, provided that we may accept worsening of the modeling quality.

Acknowledgments

This research was supported by the grant BK-239/Rau-2/2005 from the Institute of Computer Science, Silesian University of Technology.

References

[1] Advanced Micro Devices: AMD64 Architecture Programmer's Manual Volume 1: Application Programming. Rev. 3.11, December 2005.

[2] Allan V.H., Jones R.B., Lee R.M., Allan S.J.: Software Pipelining. ACM Computing Surveys, 27(3), 1995, pp. 367-432.

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

[4] Flynn M.J.: Very high-speed computing systems. Proceedings of the IEEE, 54(12), 1966, pp. 1901-9.

[5] Golomb S.W.: Run-Length Encodings. IEEE Trans. on Information Theory, IT-12, 1966, pp. 399-401.

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

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

[8] Intel: IA-32 Intel Architecture Optimization Reference Manual. December 2003.

[9] Intel: Intel Itanium 2 Processor Reference Manual For Software Development and Optimization. April 2003.

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

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

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

[13] Starosolski R.: Performance evaluation of lossless medical and natural continuous tone image compression algorithms. Proc. SPIE, 5959, 2005, pp. 116-27.

[14] Starosolski R.: Simple Fast and Adaptive Lossless Image Compression Algorithm. Software-Practice and Experience, 37(1), 2007, pp. 65-91.

[15] SUN Microsystems: UltraSPARC III Cu User's Manual. Version 2.2.1, January 2004.

Zrównoleglenie adaptacyjnego algorytmu kompresji z użyciem metody zmniejszonej częstości aktualizacji modelu danych

Streszczenie

Modelowanie i kodowanie to najbardziej złożone elementy wielu adaptacyjnych algorytmów kompresji, przy czym sam proces kompresji oparty o modelowanie i kodowanie musi być realizowany w sposób sekwencyjny. Metoda zmniejszonej częstości aktualizacji modelu danych to modyfikacja zastosowana do adaptacyjnego algorytmu kompresji, aby poprawić prędkość modelowania kosztem nieznacznego, z praktycznego punktu widzenia, pogorszenia jakości modelowania. W niniejszej pracy zauważono, iż zastosowanie tej metody umożliwia zrównoleglenie algorytmu kompresji. Badania eksperymentalne wykazały, że dzięki wykorzystaniu równoległości drobnoziarnistej dla procesora Itanium 2 i algorytmu SFALIC metoda zmniejszonej częstości aktualizacji modelu danych pozwala na zwiększenie prędkości kodowania i modelowania o około 50%. Przeprowadzone szacunki pokazały, że metoda ta pozwala również na wykorzystanie równoległości średnioziarnistej znacznie większym stopniu.

Compressing High Bit Depth Images of Sparse Histograms

Roman Starosolski

Institute of Computer Science, Silesian University of Technology, Akademicka 16, 44-100 Gliwice, Poland

Abstract. To improve the lossless compression ratios for images having sparse histograms, a method of histogram packing was introduced. The method was found to be effective for low bit depth images. We investigate effects of packing histograms of high bit depth images—medical CR, CT, and MR images as well as various natural 16-bit ones. We analyze an off-line packing method, which requires encoding the original histogram along with the compressed image. We present several methods of histogram encoding and analyze their usefulness. One of them (RLE+LZ77) obtains the shortest encoded histogram length for nearly all tested images and in practice is sufficiently good for encoding histograms of wide range of images. A simpler method (MT) may be useful for medical images. For these images, its use results in improvements of the compression ratio little worse compared to RLE+LZ77, but decoding of images with histograms encoded using the MT method is already supported by JPEG-LS and JPEG2000 (part 2) standards. Effects of histogram packing are examined for the CALIC, JPEG2000, and JPEG-LS algorithms. Histogram packing improves significantly lossless compression ratios for high bit depth sparse histogram images. The ratio improvement may exceed a factor of two, as in the case of MR medical images.

Keywords: image coding, lossless image compression, high bit depth images, sparse histogram, histogram packing.
PACS: 07.05.Pj; 42.30.Va; 42.68.Sq; 95.75.Mn

INTRODUCTION

Most single-frame single-band medical images, like CR, CT, and MR, are of a high nominal bit depth, which usually varies from 12 to 16 bits per pixel. The number of active levels, i.e., intensity levels actually used by image pixels, may be smaller, than implied by the nominal bit depth, by an order of magnitude or even more. Furthermore, active levels are distributed throughout almost all the entire nominal intensity range, i.e., the images have sparse histograms of intensity levels. Also, the continuous tone natural (photographic) images of high bit depths may have sparse histograms due to acquisition device characteristics or to processing applied to images (like gamma correction or contrast adjustment). Histograms of some images are inherently sparse. Although this observation probably won't lead to improving the compression ratios for such images, we note the obvious fact: regardless of the nominal bit depth, the number of active levels cannot be greater, than the number of pixels. All in all, sparse histograms occur frequently in high bit depth images.

Image compression algorithms are based on sophisticated assumptions about images they process. Sparse histogram is clearly different from what is expected by most lossless image compression algorithms, both in the case of predictive and of transform coding. The impact of histogram sparseness on compression ratios of low bit depth sparse histogram images is well known—applying to such images a histogram packing may lead to significant ratio improvement [1,2]. An off-line histogram packing simply maps all the active levels to the lowest part of the nominal intensity range (order-preserving one-to-one mapping). The off-line packing requires the information, describing how to expand the histogram, to be encoded along with the compressed image; i.e., we have to encode the original histogram. There are also other methods targeted at sparse histogram images (for overview see [3]), however, the off-line packing has significant practical advantages over others. Several algorithms along with the compressed image store the "image palette" (PNG [4]) or the level "mapping table" (JPEG-LS [5]). For these algorithms, if we use off-line packing prior to compression, then the decompression reconstructs image and its original histogram solely by means of the algorithm (i.e., no additional step of histogram expanding is required after decompression). To our best knowledge, except for the first stage of the research reported herein [3], the compression of high bit depth images having sparse histograms has not been investigated. High bit depth images require the histogram to be encoded efficiently—in this paper we analyze methods of encoding histograms of high bit depth images as well as effects of histogram packing on compression ratio obtained using CALIC, JPEG2000, and JPEG-LS algorithms.

EFFICIENT ENCODING OF THE HISTOGRAM

The off-line histogram packing method actually is an image transform; we apply it to an image before the compression. It transforms sparse histogram image into the packed histogram image. The transform is reversible if, along with the compressed image, we encode the original histogram. For the histogram expanding, it is enough to encode which of intensity levels are active—we do not need to know how many times the active level was used.

In the case of 8-bit images, we may simply encode binary all the active levels. Following the JPEG-LS terminology, we call this method of histogram encoding the Mapping Table. For encoding a histogram of an N-bit image containing L active levels we need $(L + 1)\ N$ bits. For 16-bit images, in the worst case (all levels active), we'd need 128 kilobytes.

Instead of encoding the intensity level of each active level, we may encode, for all nominally available levels, the information whether the specific level is active. Therefore, we need 2^N bits to encode the histogram of an N-bit image, regardless of the number of active levels. This method of histogram encoding was used for 8-bit images in the EIDAC algorithm starting from its first version [1]. We call a histogram encoded in this manner the Bit-Array of the histogram. For a 16-bit image, the Bit-array requires 8 kilobytes; for an 8-bit image, 32 bytes only.

Some images, like MR images used for experiments in this paper, use below 1% of all the nominally possible levels. A histogram of such image, encoded using the Bit-Array method, contains long runs of 0's separated by single 1's. Such histogram could be represented more compactly if we encoded lengths of runs of 0's. If, on the other hand, the histogram is not sparse, then it contains long runs (or just one long run) of 1's. Therefore we encode the Bit-Array of the histogram using the Run Length Encoding (RLE) variant described in the Table 1. Encoding the histogram using the RLE method is most efficient when the number of levels is close to 0 or close to 2^N. In the worst case, i.e., when every second level is used, we need 2^{N+2} bits for the RLE encoded histogram—32 kilobytes for the worst case histogram of a 16-bit image.

TABLE 1. Run Length Encoding of histograms of images of bit depths up to 16 bits.

RLE codeword	Sequence
$0\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$	run of $r + 1$ 0's followed by single 1, $r = b_6 \ldots b_0$, $r < 126$
$0\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ b_7 \ldots b_0$	run of $r + 127$ 0's followed by single 1, $r = b_7 \ldots b_0$
$0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ b_{15} \ldots b_0$	run of $r + 383$ 0's followed by single 1, $r = b_{15} \ldots b_0$
$1\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$	run of $r + 1$ 1's followed by single 0, $r = b_6 \ldots b_0$, $r < 126$
$1\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ b_7 \ldots b_0$	run of $r + 127$ 1's followed by single 0, $r = b_7 \ldots b_0$
$1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ b_{15} \ldots b_0$	run of $r + 383$ 1's followed by single 0, $r = b_{15} \ldots b_0$

The Bit-Array is inefficient when the number of active levels is low; the RLE may be inefficient for certain numbers of intensity levels. Fortunately, both the Bit-Array of the histogram and the RLE encoded histogram may be further compressed. In the cases, when the above methods are most inefficient, the histograms encoded using them are likely to contain multiple repetitions of long sequences of symbols (bits or RLE codewords). For compressing such data we may use a universal compression algorithm capable of capturing long contexts, like the LZ77 universal dictionary compression algorithm [6].

EXPERIMENTAL RESULTS

In order to evaluate the impact of histogram sparseness on compression ratio for typical medical image of a certain modality, we used all the CR, CT, and MR medical images from a test image set described in another study [7]. There were 12 images of each of the modalities; not all the medical images are of 16-bit depth and not every medical image has sparse histogram. Obviously, for the 10- or 12-bit images the method of histogram encoding gets less important for the overall compression ratio. Natural continuous tone grayscale images of 16-bit depth were included in experiments to evaluate effects of histogram packing on various non-medical images. These images included unprocessed images of various sizes as well as processed ones—for gamma and contrast adjustment we used Adobe Photoshop 9.0. Following groups of non-medical images were evaluated, each containing 4 images: natural (photographic) images of 16-bit depth classified in [7] as medium-sized (Medium), Medium images with contrast increased by 25% (Contrast), Medium images with gamma (value 1.25) correction applied (Gamma), and small images containing below 2^{16} pixels, which are reduced size Medium images (Small).

The characteristics of images and the results of encoding histograms are reported in the Table 2 (for brevity we report averaged results only). To characterize numerically image sparseness, we define the image level utilization $U = L / (1 + l_{hi} - l_{lo})$, where l_{lo} and l_{hi} are respectively the lowest and the highest active level, and L is number of

active levels. In the tables, images are characterized by the image name, size (number of pixels), nominal depth (N), nominal (2^N) and actual (L) number of intensity levels, and by the level utilization (U). Sizes (in bytes) of encoded histograms are reported in the Table 2 for the following methods: Mapping Table (MT), Bit-Array (BA), Bit-Array compressed using LZ77 (BA+LZ77), RLE, and RLE method followed by LZ77 (RLE+LZ77); for the LZ77 compression we used the gzip compression utility (version 1.2.4).

TABLE 2. Comparison of histogram encoding methods (averages for groups).

Images						Encoded histogram size [B]				
Name	**Pixels**	N	2^N	L	U	**MT**	**BA**	**BA+ LZ77**	**RLE**	**RLE+ LZ77**
CR	3527076	12.5	23296	7878	59.5%	15184	2912	285	7071	179
CT	257569	14.7	45056	1951	17.3%	3592	5632	541	1852	219
MR	196608	16.0	65536	1104	1.7%	2210	8192	550	1127	102
Medium	440746	16.0	65536	55839	87.1%	111681	8192	4358	6231	3528
Contrast	440746	16.0	65536	23737	36.4%	47475	8192	1251	23737	909
Gamma	440746	16.0	65536	28076	44.4%	56154	8192	1546	28080	1314
Small	48776	16.0	65536	25174	39.7%	50350	8192	8447	13195	7288

The RLE+LZ77 method appears to be the most efficient, therefore for further evaluating effects of histogram packing on compression ratios of popular algorithms we use the RLE+LZ77 method. The compression ratios obtained for images before histogram packing (Norm.), after packing (Pack.), and the ratio improvements due to histogram packing are reported in the Table 3. The compression ratio is expressed in bits per pixel [bpp]: $8\,e / n$, where n is the number of pixels in the image, e—the size in bytes of the compressed image (including the size of the histogram encoded using the RLE+LZ77 method in the case of ratio after packing). We performed experiments for the following image compression algorithms: CALIC [8] (implementation by Wu and Memon), JPEG-LS [5] (SPMG/UBC implementation version 2.2), and JPEG2000 [9] (JasPer implementation by Adams version 1.700.0).

TABLE 3. Effects of histogram packing on compression ratios of CALIC, JPEG-LS, and JPEG2000; results obtained for histograms encoded using RLE+LZ77 method (averages for groups).

Images		CALIC			JPEG-LS			JPEG2000		
Name	U	**Norm. [bpp]**	**Pack. [bpp]**	**Impro-vement**	**Norm. [bpp]**	**Pack. [bpp]**	**Impro-vement**	**Norm. [bpp]**	**Pack. [bpp]**	**Impro-vement**
CR	59.5%	6.229	5.287	15.1%	6.343	5.398	14.9%	6.394	5.426	15.1%
CT	17.3%	7.759	4.485	42.2%	7.838	4.557	41.9%	8.044	4.630	42.4%
MR	1.7%	9.975	4.811	51.8%	10.009	4.944	50.6%	10.024	4.849	51.6%
Medium	87.1%	11.735	11.760	-0.2%	11.829	11.844	-0.1%	12.058	12.082	-0.2%
Contrast	36.4%	11.330	10.010	11.6%	11.416	9.992	12.5%	11.951	10.558	11.7%
Gamma	44.4%	11.850	10.646	10.2%	11.950	10.676	10.7%	12.183	10.965	10.0%
Small	39.7%	12.547	12.939	-3.1%	12.414	12.813	-3.2%	12.712	13.180	-3.7%

We notice, that effects of packing histograms on the compression ratios of tested algorithms are, for all algorithms, very similar (see also Fig. 1.a). As expected, the histogram packing does not improve compression ratios for Small images. Also the average compression ratio of Medium images, most of which have non-sparse histograms, gets negligibly worse if we employ histogram packing. Except for the above cases, the histogram packing improves average compression ratios for high bit depth sparse histogram images. The improvement varies depending on the image level utilization U, which we use as a measure of the histogram sparseness (see Fig. 1.b). For $U < ¼$ the compression ratio improvement is roughly 50%, i.e., the size of the compressed image gets halved by applying the histogram packing method. For $U \approx ½$ we get the compression ratio improvement of about 10–20%; this level of improvement is not negligible for lossless image compression algorithm—the difference in compression ratio between algorithms obtaining best ratios and algorithms obtaining best speeds usually does not exceed 10% for the images used [7]. For $U > ¾$ the histogram packing improves ratios for some images only, however, it does not deteriorate ratios for the remaining ones.

The greatest improvements are obtained for CT and MR images, yet for these images the use of another histogram encoding method, namely MT, may be a practical alternative. This way, at the cost of loosing of small fraction of the ratio improvement obtained (compare Tables 2 and 3) we get possibility to decompress image within standard algorithms like JPEG-LS (which is included in the DICOM standard [10]) or JPEG2000 (2nd part [11]). Except that ratio improvement for CR images is smaller than for CT and MR, above conclusion applies to medical

CR images also. Using the off-line histogram packing and the MT method of histogram encoding we may significantly improve compression ratios of medical images while maintaining compatibility with current standards.

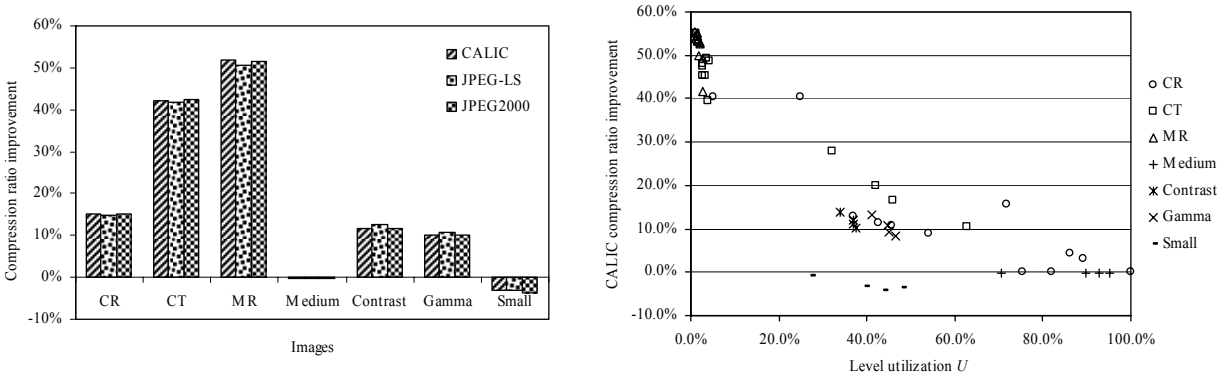

FIGURE 1. a) Average compression ratio improvement due to histogram packing (RLE+LZ77); **b)** CALIC compression ratio improvement of individual images due to histogram packing (RLE+LZ77).

CONCLUSIONS

We introduced and analyzed experimentally a couple of methods of encoding histograms of high bit depth images. One of the methods (RLE+LZ77) obtains the shortest encoded histogram length for nearly all tested images and in practice is sufficiently good for encoding histograms of wide range of images. A simpler method (MT) may be useful for medical images. For these images, its use results in improvements of the compression ratio little worse compared to RLE+LZ77, but decoding of images with packed histograms encoded using the MT method is already supported by JPEG-LS and JPEG2000 (part 2) standards. The effects of packing histograms on the compression ratios of CALIC, JPEG2000, and JPEG-LS are, for all tested algorithms, very similar—histogram packing improves significantly lossless compression ratios for high bit depth sparse histogram images. The ratio improvement due to histogram packing may exceed a factor of two, as in the case of MR medical images.

ACKNOWLEDGMENTS

This research was supported by the grant BK-239/Rau-2/2005 from the Institute of Computer Science, Silesian University of Technology.

REFERENCES

1 Y. Yoo, Y. G. Kwon, A. Ortega, *Embedded image-domain compression of simple images*, Proc. of the 32nd Asilomar Conf. on Signals, Systems, and Computers, 2 (1998) 1256-1260.
2. A. J. Pinho, *On the impact of histogram sparseness on some lossless image compression techniques*, Proc. ICIP-2001, III (2001) 442-445.
3. R. Starosolski, *Compressing images of sparse histograms*, Proc. SPIE Medical Imaging, 5959 (2005) 595912(209-217).
4. WWW Consortium, *W3C Recommendation: PNG (Portable Network Graphics) Specification*, Version 1.0, 1996.
5. ISO/IEC, ITU-T, *Information technology – Lossless and near-lossless compression of continuous-tone still images – Baseline*, ISO/IEC International Standard 14495-1 and ITU-T Recommendation T.87, June 1999.
6. J. Ziv, A. Lempel, *A universal algorithm for sequential data compression*, IEEE Trans. on Inf. Theory, 32(3) (1977) 337-343.
7. R. Starosolski, *Performance evaluation of lossless medical and natural continuous tone image compression algorithms*, Proc. SPIE Medical Imaging, 5959 (2005) 59590L(116-127).
8. X. Wu, N. Memon, *Context-based, Adaptive, Lossless Image Codec*, IEEE Trans. Comm., 45(4) (1997) 437-444.
9. C. Christopoulos, A. Skodras, T. Ebrahimi, *The JPEG2000 Still Image Coding System an Overview*, IEEE Trans. on Consumer Electronics, 46(4) (2000) 1103-1127.
10. National Electrical Manufacturers Association, *Digital Imaging and Communications in Medicine (DICOM) standard*, 2004.
11. ISO/IEC, ITU-T, *Information technology – JPEG 2000 image coding system: Extensions*, ISO/IEC International Standard 15444-2 and ITU-T Recommendation T.801, August 2002.

COMPUTER METHODS AND PROGRAMS IN BIOMEDICINE 108 (2012) 511–523

journal homepage: www.intl.elsevierhealth.com/journals/cmpb

Application of detector precision characteristics and histogram packing for compression of biological fluorescence micrographs

Tytus Bernas[a,b], *Roman Starosolski*[c,*], *J. Paul Robinson*[d], *Bartlomiej Rajwa*[d]

[a] *Department of Physiology and Medical Physics, RCSI, 123 St. Stephens Green, Dublin 2, Ireland*
[b] *Department of Biochemistry, Biophysics and Biotechnology, Jagiellonian Univesity, Gronostajowa 7, 30-387 Krakow, Poland*
[c] *Institute of Computer Science, Silesian University of Technology, Akademicka 16, 44-100 Gliwice, Poland*
[d] *Purdue University Cytometry Laboratories, 1203 West State St., West Lafayette, IN, USA*

ARTICLE INFO

Article history:
Received 24 January 2011
Received in revised form
16 March 2011
Accepted 26 March 2011

Keywords:
Microscope imaging
Intensity resolution
Histogram packing
Image compression
Image coding

ABSTRACT

Modern applications of biological microscopy such as high-content screening (HCS), 4D imaging, and multispectral imaging may involve collection of thousands of images in every experiment making efficient image-compression techniques necessary. Reversible compression algorithms, when used with biological micrographs, provide only a moderate compression ratio, while irreversible techniques obtain better ratios at the cost of removing some information from images and introducing artifacts. We construct a model of noise, which is a function of signal in the imaging system. In the next step insignificant intensity levels are discarded using intensity binning. The resultant images, characterized by sparse intensity histograms, are coded reversibly. We evaluate compression efficiency of combined reversible coding and intensity depth-reduction using single-channel 12-bit light micrographs of several subcellular structures. We apply local and global measures of intensity distribution to estimate maximum distortions introduced by the proposed algorithm. We demonstrate that the algorithm provides efficient compression and does not introduce significant changes to biological micrographs. The algorithm preserves information content of these images and therefore offers better fidelity than standard irreversible compression method JPEG2000.

1. Background

Digital imaging based on light microscopy has become an established technique in basic and applied biological sciences. Modern applications such as high-content screening (HCS), 4D imaging, and multispectral imaging may involve collection of thousands of images in one experiment. On the other hand, storage space and network transmission bandwidth which may accommodate these data is often limited. Therefore efficient image-compression techniques may be necessary to mitigate this problem. Several compression routines developed for digital photography and film, may be used for this purpose. Reversible compression algorithms (Deflate, LZW, RLE, Huffman encoding, etc.) neither introduce distortion to images [1] nor remove any information from images and therefore preserve the data integrity, as defined by 21 CFR part 11 [2]. However, reversible ("lossless") techniques when used with

* *Corresponding author*. Tel.: +48 322372151.
E-mail addresses: rstarosolski@polsl.pl, rstaros@gmail.com (R. Starosolski).
0169-2607/$ – see front matter © 2011 Elsevier Ireland Ltd. All rights reserved.
doi:10.1016/j.cmpb.2011.03.012

biological micrographs usually provide only a moderate compression ratio, which does not exceed 3:1. This is caused by the large dynamic range (4096–65,536 intensity levels per 'color' component) of biological micrographs, and the fact that in a typical case the full extent of the dynamic range is used and all the intensity levels are populated. Furthermore, no typical intensity distribution may be constructed for these images.

More efficient compression can be obtained with irreversible ("lossy") techniques, which use vector quantization [3,4], discrete cosine transform [5], wavelet transform [6,7] or fractal coding [3,8]. However, these algorithms by definition remove some information from images and introduce artifacts. In the field of digital photography these distortions are considered acceptable as long as the essential perceptual image quality is not affected [9–12]. In other words, all intensity differences between pixels are regarded as significant if they are detectable by human observers [9]. This approach is established (sometimes mandatory) in medical applications of brightfield microscopy (including pathology, cytology and hematology) owing to massive amounts of digitized data. For instance, the whole-slide imaging (WSI), where the entire micrograph area is digitized, may generate daily terabytes of uncompressed data.

However, procedure of verification of compression integrity by a panel of experts is time-consuming and may be specific for different applications (biomedical assays). Thus, the procedure difficult to standardize and to apply systematically across a range of specimens and imaging regimes. This problem hinders use of automated data analysis/interpretation methods in the context of medical imaging. Therefore, one might adopt a different approach and focus on the information content of the input image data in order to establish acceptable compression limits and criteria. Owing to the presence of noise in the microscope images, not all intensity differences can be considered significant from a statistical standpoint. Thus, the number of meaningful intensity levels may be lower than the nominal dynamic range (corresponding to 12- or 16-bit precision) provided by the AD converters of the cameras. Therefore, to establish an irreversible yet information-preserving compression mechanism that does not use perceptual quality as a criterion, one should first construct a model of noise that is a function of signal in an imaging system. Then, intensity binning can be applied to discard insignificant intensity levels. The processed images are characterized by sparse intensity histograms; therefore reversible coding and intensity depth-reduction algorithms should be employed to provide efficient compression.

Herein we report the design and implementation of a compression pre-processing scheme which takes advantage of the noisy nature of microscope images by preserving only statistically significant levels of intensity. We provide results of local and global measures of intensity distribution to demonstrate that the alterations introduced by this algorithm to biological micrographs range from none to minor. We also demonstrate that the algorithm offers better fidelity than JPEG2000 (a standard "lossy" compression routine) at the same compression ratio.

2. Materials and methods

2.1. Cells and fluorescence labeling

FluoCells prepared slide #2 (Molecular Probes) was used in all experiments. The slide contained fixed bovine pulmonary artery endothelial cells in which microtubules were labeled using mouse anti-bovine α-tubulin monoclonal antibodies in conjunction with BODIPY FL goat anti-mouse IgG antibody; the cell nuclei were labeled with DAPI.

2.2. Microscope imaging

Images of the endothelial cells were collected using a Nikon E1000 wide-field fluorescence microscope equipped with a Nikon 40× Fluor oil-immersion objective lens (NA 1.3) and a 100-W Hg arc lamp. The BODIPY FL fluorescence was registered using a 475–495-nm excitation filter (band-pass), a 505-nm long-pass dichroic mirror and a 525–565-nm emission filter (band-pass). A monochrome CCD camera (Retiga 4000R, Qimaging, Burnaby, Canada) was used for image acquisition. A 16× neutral density (ND) filter was used to attenuate the flux of excitation light. The microscope aperture diaphragm was fully open, whereas the field diaphragm was adjusted to match the field of view of the objective. Image collection was carried out at room temperature. The camera was cooled to 25 °C below ambient.

A time series of 128 images of stationary (fixed) cells were collected using full frame (no binning) at 5-s intervals. The series was registered at three gain settings and for 0.25-s or 0.75-s acquisition times. Image acquisition was controlled using ImagePro Plus v 5.1 (Media Cybernetics, Silver Spring, MD, USA).

2.3. Calculation of noise levels and background signal

Levels of background (dark) and fluorescence signals together with their respective variances (corresponding to total noise) were calculated as described elsewhere [13]. Briefly, for every image pixel the fluorescence intensity changes in time were modeled with three components: a systematic trend (related to photobleaching), a periodic component (associated with fluctuation of the excitation light source), and an irregular component (representing noise). Following [14] we studied our system using a simple univariate version of the unobserved components (UC) model

$$y_t = T_t + S_t + e_t \quad (1)$$

where t denotes the value of the associated pixel intensity at the tth time point, y is the observed value, T is the trend (or low-frequency component), S is the periodic (or "seasonal" component), and e is the irregular component. All the calculations were executed on a pixel-by-pixel basis utilizing the CAPTAIN modeling toolkit operating within the environment

of Matlab [14]. First, the stochastic trend component was estimated using the integrated random walk (IRW) model:

$$I_t = T_t + e_t$$
$$T_t = 2T_{t-1} - T_{t-2} + \eta_t \quad (2)$$

where I_t is registered fluorescence intensity (at the tth time point), T_t is the smoothed intensity at the tth time point, T_{t-1} and T_{t-2} are values of T_t at two previous time points, e_t is measurement noise (zero mean, variance σ_e^2), and η_t is the system disturbance (zero mean, variance σ_η^2).

The trend was subtracted from the observed intensity (I_t). The de-trended data were used to isolate periodic components of intensity changes with the dynamic harmonic regression (DHR). Subsequently, the IRW and optimal order DHR were used jointly to fit the trend and periodic component to the initial fluorescence intensity data (I_t). One should note that the noise variance ratio (σ_η^2/σ_e^2) was optimized in this step as well to minimize residual variance globally. The sum of the trend and periodic components represented the true instantaneous fluorescence intensity (signal, S_t^i) at every time point. Hence, the instrumental noise (for a pixel at a given time point) and its variance (for a signal level) were estimated as:

$$N_t^i = \left|S_t^i - I_t^i\right|$$
$$V_{S=F} = \frac{\sum_{i,t}\delta_{SF}(N_t^i)^2}{\sum_{i,t}\delta_{SF}} \quad (3)$$

where N is the noise, S is the signal, I is the fluorescence intensity registered at the ith and tth points of the image time.

Estimates of the other two components of a time series (periodic component and trend) can be further used to characterize the stability of the light source and the photobleaching rate of the fluorochromes used in the experiment. However, they were utilized here only to provide an estimate of total signal level (fluorescence and background) and thus to calculate the corresponding level of total noise.

In order to estimate the background signal, uniform dark image regions were identified for each time series. These regions (represented using binary masks) were comprised of pixels characterized by fluorescence intensity and local fluorescence heterogeneity that were smaller than 10% of the respective maxima. The heterogeneity was measured using the algorithm described in [15]. Average intensity (I_b) calculated in dim and homogenous regions was taken as the background (i.e., the pixel value of an image registered in the absence of fluorescence). The noise variance (V, Eq. (3)) was plotted against the signal corrected for background ($S_c = S - I_b$). A quadratic function was fitted to these data in order to characterize signal-noise dependency:

$$V = A + PS_c + MS_c^2, \quad (4)$$

where M, P, and A are estimators of the signal variance associated with the multiplicative, Poisson (photonic), and additive noise components. The standard deviation of I_b ($\sqrt{B}$) was calculated to estimate the background noise.

2.4. Calculation of significant intensity levels and histogram binning (HB)

Owing to the presence of noise in the images, not all intensity differences can be considered significant. Thus, the number of meaningful intensity levels is lower than the nominal dynamic range provided by the cameras (12 bits, 4096 levels). Hence, the significant levels were calculated iteratively using the following algorithm:

1. Input I_b, A, P, M
2. Set $k = 0$
3. Do:
   4. Set $k = k + 1$
   5. Set $I_{med}^k = I_b$
   6. Set $\sigma(I_{med}^k) = \sqrt{(A + I_{med}^k * P + I_{med}^k * I_{med}^k * M)}$
   7. Set $I_{high}^k = I_{med}^k + 1.96 * \sigma(I_{med}^k)$
   8. Set $I_{low}^{k+1} = I_{high}^k$
   9. Calculate I_{med}^{k+1} so that $I_{med}^{k+1} - I_{low}^{k+1} = 1.96 * \sigma(I_{med}^{k+1})$
4. Loop while $I_{med}^{k+1} < 4095$
5. Terminate. Output scalar $s = k$ and vector $I = [I_{med}^1, I_{med}^2, \ldots, I_{med}^s]$.

The algorithm produces a set of I_{med}^k for which $I_{med}^k - I_{med}^{k-1} = 1.96(\sigma(I_{med}^k) + \sigma(I_{med}^{k-1}))$. I_{med}^k values smaller than I_{max} represent intensity levels significantly different from one another with 95% probability (confidence) in the sense of Student's t test (hence the factor of 1.96). The choice of confidence interval was arbitrary. However, similar calculations can be performed for every confidence level. The set of values was used to segment the representative images by setting all pixel intensities (I_r) to the nearest significant level. This operation is hereinafter referred as histogram binning (HB).

2.5. Simple depth reduction and histogram packing

Reversible image-compression algorithms used on data characterized by sparse intensity histograms (such as those generated by HB) may exhibit poor performance [16]. To alleviate this problem a simple depth reduction (SDR) transform, first introduced in [17] was used. SDR maps pixel intensity levels in the following way:

$$I_{SDR} = \frac{I_0 s}{I_{max}}, \quad (5)$$

where I_{SDR} denotes intensity after SDR transform, I_0 is the original intensity, s is the number of significant levels, and I_{max} is the intensity of the highest significant level.

To perform the inverse SDR transform one needs to know s and I_{max}. The compression ratios were calculated taking into consideration this necessary overhead (4 bytes).

The performance of reversible image-compression algorithms may also be improved using histogram packing (HP) [16]. HP maps all the significant levels to the lowest part of the nominal intensity range using order-preserving one-to-one mapping. The transform is reversible provided that information that permits histogram expansion after decompression is encoded with compressed images. It is sufficient to encode significant intensity levels for histogram expansion, and there

is no need to specify how many times a level was used. An array of histogram-encoding methods is available [18]. When the number of significant levels is low, a simple mapping table (MT) may be used. Briefly, if n significant intensity levels are found in the image the MT represents sorted intensity values as contiguous indexes:

$$MP(I_n) = (I_0 \mapsto 0, \quad I_1 \mapsto 1, \quad \ldots, \quad I_{n-1} \mapsto n-1) \tag{6}$$

The compression ratios reported in this work were calculated taking into consideration the size of MTs encoded according to the JPEG-LS standard. Therefore the MTs were encoded with $5+2s$ bytes, where s is the number of significant levels. This form of histogram packing used together with histogram binning (HB) is hereinafter referred as histogram binning/packing (HBP). Note, that instead of storing original histogram with each compressed image, one could store it only once for all images that share the same detector characteristic, or just store the detector parameters used to perform Histogram Binning. An implementation illustrating this possibility and allowing to experiment with the HB and HBP methods has been prepared (http://sun.aei.polsl.pl/~rstaros/hbp/index.html).

2.6. Image-compression algorithms

We estimated compression efficiency of two still-image coding methods: JPEG-LS [19] and JPEG2000 [6], developed by the JPEG committee. These algorithms are recent coding standards of ISO/IEC and ITU organizations, and are incorporated in the DICOM standard of NEMA [20]. JPEG-LS describes a low-complexity predictive compression algorithm with entropy coding using a modified Golomb-Rice [21,22] code. The technique is based on the LOCO-I algorithm [23]. We used the SPMG/UBC implementation (version 2.2, ftp://ftp.netbsd.org/pub/NetBSD/packages/distfiles/jpeg_ls_v2.2.tar.gz). JPEG2000 is based on wavelet-transform image decomposition and arithmetic coding [24]. This algorithm provides progressive transmission and region-of-interest coding [25]. We used JasPer implementation by Adams (version 1.700.0, http://www.ece.uvic.ca/~mdadams/jasper/). We also tested the performance of a universal data-compression algorithm Deflate (RFC 1951 [26]). The algorithm is based on the Ziv-Lempel dictionary coding LZ77 [27] and is incorporated in the DICOM standard. We used the gzip implementation (version 1.2.4, http://www.gzip.org/).

All the compression methods were used with default coding parameters. Compression ratio is defined as U/C, where C is the size, in bytes, of a compressed image (including header); U is the size of the original image, defined as $U = bn/8$, where b is the image bit depth (12 in our case), and n is number of pixels in the image. One should note that in practice space savings may be greater than implied by the compression ratio since uncompressed image file formats store image pixels on whole bytes, so each 12-bit pixel actually occupies 16 bits.

One should note that both JPEG2000 and JPEG-LS may be used for reversible and irreversible coding. While the reversible compression ratio depends on the image contents, it may be set in arbitrary manner when the irreversible mode is used. However, high compression ratio is obtained in that case at the expense of the fidelity of decompressed image (micrograph). Typical artifacts include blurring and generation of spurious image details (Fig. 1).

2.7. Estimation of global changes in fluorescence intensity distributions

We used the earth mover's distance (EMD) algorithm in order to establish whether compression introduced changes in the total fluorescence intensity distributions [28]. We compared image histograms of raw images with their counterparts processed with SDR, HB, or irreversible JPEG2000. The minimal average intensity change (per pixel) needed to transform histograms of a compressed image into the respective histogram of an uncompressed (reference) image was computed for every such image pair.

2.8. Alterations of local fluorescence distributions

To verify whether reduction in the number of intensity levels altered spatial fluorescence intensity distributions, raw images were compared with their counterparts subjected to HB. The distributions were compared using texture parameters (features): Haralick features based on the grey-level co-occurrence matrix (GLCM), gradient-based features, run-length matrix parameters, and wavelet energy. Detailed descriptions of these parameters are provided in [29,30]. The GLCMs were calculated at distances from 1 to 9. The gradient-based features and run-length matrix parameters were calculated at 0° (horizontal), 45°, 90° (vertical), and 135°. The wavelet energy was calculated at first, second, and third decomposition levels. Calculations were performed for the areas where the fluorescence intensity was higher than background. The texture parameters of images subjected to histogram binning were divided by the respective values for their raw counterparts. The total number of 137 normalized texture parameters was subjected to step-wise linear discriminant analysis. The Mahalanobis distance was used to establish the parameters characterized by the highest discriminant power. The parameters were added to and removed from the analysis set using probability of F (0.05 for entry and 0.10 for removal). An identical set of texture parameters was calculated for images processed with SDR and compressed with irreversible JPEG2000.

3. Results

3.1. CCD noise and background

Square roots of Poisson and additive noise coefficients (Eq. (4)) were analyzed as a function of gain for the monochrome (single-channel) images registered Retiga 4000R CCD camera at 12-bit (4096 levels) precision, as described in detail elsewhere [13]. Briefly, the amount of both types of noise depended linearly on the gain for the CDD, but was not affected by acquisition time (Table 1). On the other hand, the background signal (see Section 2) increased linearly with acquisition time and gain [13]. There was no significant difference between the additive noise ($\sqrt{A}$) and background noise ($\sqrt{B}$). Hence both

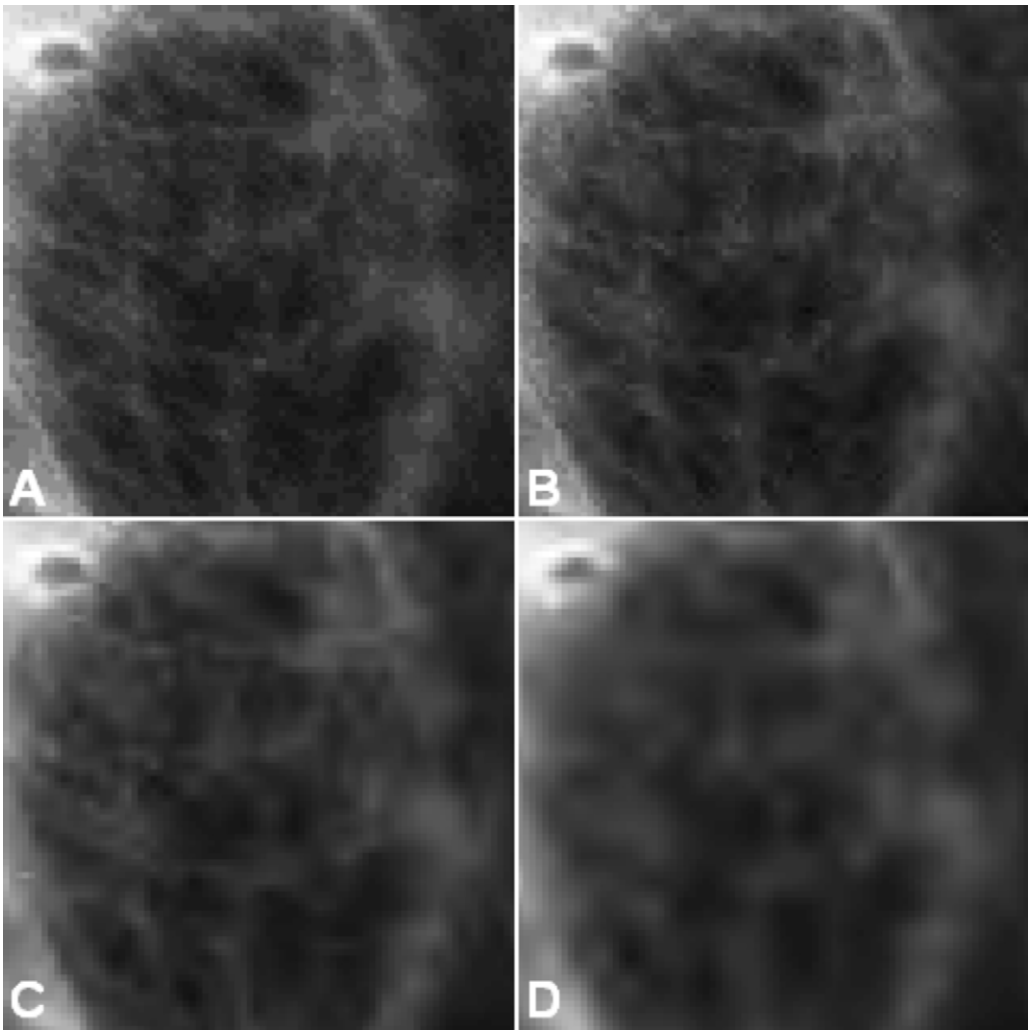

Fig. 1 – JPEG2000 compression at different ratios. Enlarged fragment of micrograph: raw image (A), for which the reversible JPEG-LS ratio is 1.75, and images compressed with irreversible JPEG2000 at ratios 10 (B), 25 (C) and 50 (D).

these parameters may be regarded as estimators of the dark noise. Consequently, dependence between signal and noise for the CCD may be accurately expressed using Eq. (4) [13].

3.2. *Calculation of significant intensity levels and histogram binning*

The CCD cameras registered images with 4096 nominal intensity levels (12-bit digitization). However, owing to the presence of noise not all differences in intensity between pixels may be considered significant (see Section 2), and consequently practical dynamic resolution is lower than nominal. Hence, the number of significant intensity levels (with 0.95 probabilities) was calculated at several values of CCD settings (gain, acquisition time and, offset) and are presented in Table 2. These numbers of levels were used to perform SDR (Eq. (5)). Furthermore, the respective vectors of intensity values corresponding to significant levels were used to segment series of images registered at given values of gain, acquisition time, and offset (see Section 2) by setting all pixel intensities (I_r) to the nearest level (histogram binning, HB). Result of these transformations is illustrated by a representative image in the Fig. 2. One may note that no gross image distortions were introduced by this operation.

Table 1 – Dependence of Poisson and additive noise on CCD gain for the set of 12-bit single-channel images used in experiments. The respective fit coefficients ($\sqrt{P}$, $\sqrt{A}$, Eq. (2)) calculated for the variance (V, Eq. (2)) as the function of gain are given with their standard errors (for details see [13]).

	Acq. time [s]	Slope (SL_p)	Intercept (IN_p)	Correlation (r^2)
Poisson noise	0.250	0.0948 ± 0.007	−0.028 ± 0.005	0.98
	0.750	0.0986 ± 0.006	−0.018 ± 0.002	0.99
Additive noise	0.250	2.030 ± 0.150	−1.07 ± 0.10	0.98
	0.750	2.087 ± 0.371	−0.37 ± 0.22	0.99

Table 2 – Total number of significant ($p = 0.95$) intensity levels of the CCD calculated with histogram binning (HB) for the detector and acquisition parameters used in experiments. The highest numbers of levels are indicated with asterisks, the lowest with crosses.

		Offset [AU]					
		0		150		400	
Acq. time [s]	Gain	HB	SDR	HB	SDR	HB	SDR
0.250	5	74	66	72	61	70	52
	10	36	32	36	32	34	26
	15	24	22	24	21	23^{+}	19^{+}
0.750	2	190^{*}	165^{*}	186	146	178	124
	5	74	65	72	60	69	55
	10	36	32	36	26	34	28

3.3. *Coding efficiency of sparse histogram images*

Fluorescence microscopy images characterized by various levels of noise and background were coded reversibly using JPEG2000, JPEG-LS, and Deflate compressors (Table 3). Only moderate ratios of compression were obtained by digital photography dedicated algorithms such as JPEG2000 and JPEG-LS. Nonetheless, these ratios were higher than those provided by the Deflate algorithm (which is not optimized for image data). Reduction in the number of intensity levels achieved

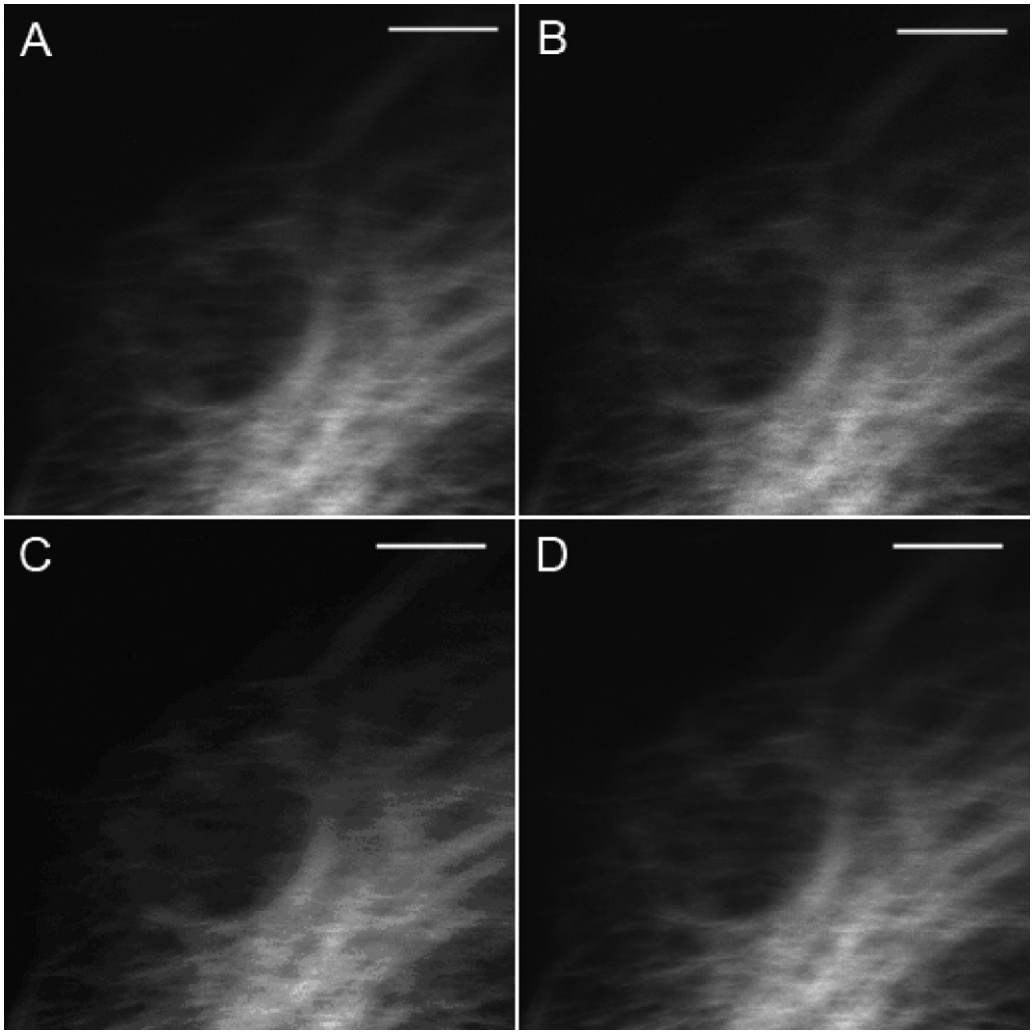

Fig. 2 – Micrograph of fluorescently (BODIPY) immunostained tubulin in a fibroblast, registered at gain of 10, acquisition time of 0.75 s, and black level of 150 units. The image is shown using nominal (A) and reduced number of intensity levels, calculated with histogram binning, HB (B), and simple depth reduction, SDR (C). For comparison a raw image compressed using irreversible JPEG2000 (with the same ratio as obtained with HBP and reversible JPEG2000 coding) is shown (D). Image segmentation was performed using HB with $p = 0.95$. Scale bar 10 μm.

Table 3 – Compression efficiency of micrographs with full and reduced number of intensity levels. The images were reversibly coded with JPEG2000, JPEG-LS, and Deflate. Compression ratios are expressed as the average ± standard deviation.

Intensity depth reduction method	Coding algorithm		
	JPEG 2000	JPEG LS	Deflate
None	1.80 ± 0.21	1.79 ± 0.20	1.13 ± 0.08
Histogram binning (HB)	1.94 ± 0.21	2.87 ± 0.26	6.46 ± 0.52
Histogram binning + packing (HBP)	9.45 ± 0.47	9.55 ± 0.50	6.53 ± 0.51
Simple reduction (SDR)	11.44 ± 1.79	11.48 ± 1.53	8.29 ± 1.08

by HB operation resulted in a slight increase in compression ratio for JPEG-LS, and a large improvement for the Deflate algorithm. These results indicate that HB improved compressibility of microscopic images. However, HB caused deviation from typical image data characteristics necessary for optimal performance of standard image compression algorithms. As expected, application of HBP operation markedly increased compression ratios of JPEG2000 and JPEG-LS, but did not significantly affect the performance of Deflate. The increase in compression was similar for JPEG2000 and JPEG-LS. The resultant ratios were better by about 45% comparing with the Deflate algorithm. Even greater improvement in compression efficiency was obtained when SDR was used. However, the standard deviation of the mean compression ratio was greater in SDR than in HBP, indicating that only some specific images were compressed more efficiently with SDR than with HBP.

3.4. *Conservation of image information in histogram binning and simple depth reduction*

3.4.1. *Alteration of global intensity distribution in compression*

The intensity histograms of the processed images were compared quantitatively to the histograms of their raw counterparts. Average EMD values for HB-processed images with histogram binning (HB) and their raw counterparts (see Table 4) were small comparing to the average distance between nearest significant intensity levels (the nominal intensity range divided by the number of significant levels, see Table 2). This notion indicates that HB did not introduce gross changes to global intensity distributions. Greater alterations were introduced by simple depth reduction (SDR). On the other hand, compression of raw images with irreversible JPEG2000 (compression ratio was adjusted to match efficiency of HBP with reversible JPEG2000) produced much smaller histogram alterations than HB did.

Table 4 – Alterations of fluorescence intensity distributions (histograms) in irreversible compression, as measured using EMD. Data are expressed as the average ± standard deviation.

Method	EMD
Histogram binning (HB)	19.73 ± 8.81
Simple depth reduction (SDR)	59.62 ± 27.64
JPEG2000	1.14 ± 0.60

3.4.2. *Alteration of local intensity distribution in compression*

Local intensity distribution in the images was characterized using Haralick texture parameters based on GLCM, wavelet energy, and gray-level runlength (see Section 2). The parameters which provided strongest discrimination between raw and HB-processed images were identified using linear discriminant analysis (LDA). As a result of LDA the following parameters were studied in detail: wavelet energy (LH and HH bands), runlength (fraction and short run emphasis), and Haralick features (contrast, correlation, inverse difference moment, difference average).

3.4.3. *Wavelet energy*

Application of HB resulted in a modest increase of wavelet energy in five out of six decomposition bands (Fig. 3A–E). No change of energy was observed in the LH band at the third decomposition level (Fig. 3F). This band corresponded to details 8–16 pixels in size. A larger increase in the energy in the former five bands was detectable when SDR was used (Fig. 3A–E). One should note that this method produced some distortions detectable at the highest (third) decomposition level (Fig. 3E and F). Furthermore, the dispersion of the relative energy values was larger with SDR than with HB, indicating that the effect of image content on the magnitude of the distortion was greater with SDR than with HB. Compression with irreversible JPEG2000 (with the same ratio as obtained with HBP and reversible JPEG2000) generated no significant change of the wavelet energy in the bands at the third decomposition level corresponding to 8-pixel (Fig. 3E) and 8–16 pixel (Fig. 3F) details. However, at the second decomposition level (the HH [4-pixel details] and LH bands [4–8 pixel details]) the energy change was similar in magnitude to that produced by HBP and SDR, but opposite in sign (Fig. 3C and D). Furthermore, at the first decomposition level (the HH [2-pixel details] and LH bands [2–4 pixel details]) irreversible JPEG2000 generated larger distortions than HPB (Fig. 3A and B). One should note that significant dispersion (manifested at the 2-pixel detail level, Fig. 3A) indicates that the distortion was affected by the image content and was not uniform within images (as describer further).

3.4.4. *Haralick features*

Reduction of the number of intensity levels and coding with HBP resulted in a moderate increase of GLCM contrast at the 1-pixel distance (Fig. 4A). HB did not produce changes of this parameter at 4-pixel (Fig. 4C) and 7-pixel (Fig. 4E) distances. When compression of raw images (with ratio identical

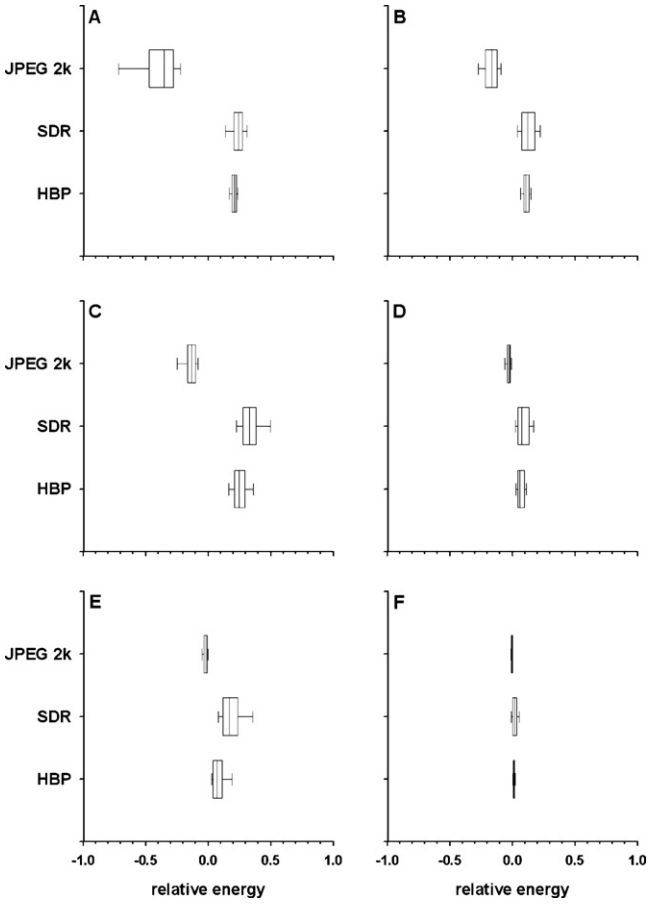

Fig. 3 – Change of relative wavelet energy in compression with HBP, SDR, and irreversible JPEG 2000 (at the same compression ratio as obtained with HBP). The energy was calculated in HH (A, C and E) and LH (B, D and F) bands at first (A and B), second (C and D), and third (E and F) levels of decomposition. The data boxes represent medians with 75th percentiles, while error bars indicate corresponding 90th percentiles.

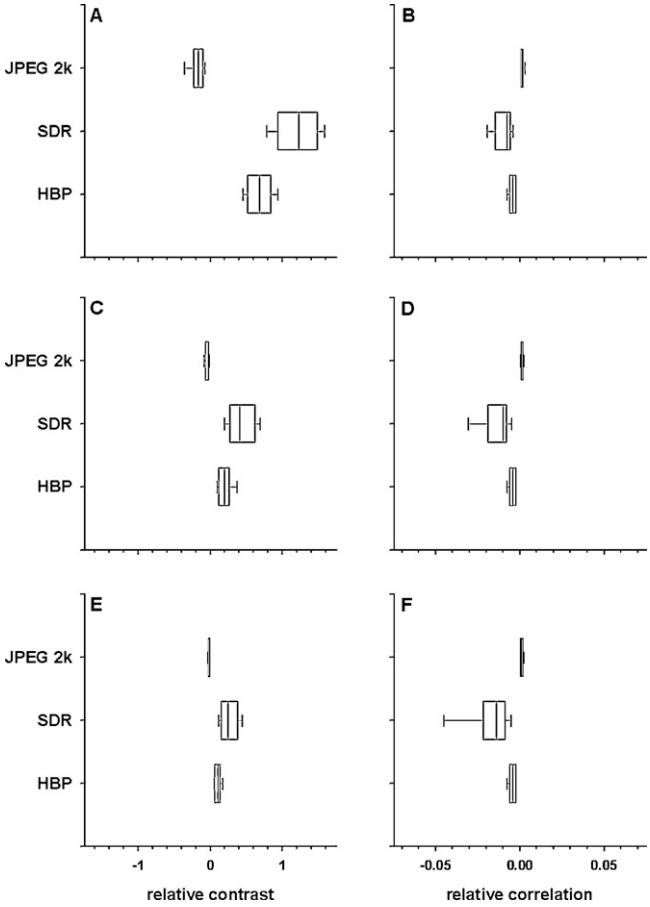

Fig. 4 – Change of relative GLCM contrast (A, C and E) and correlation (B, D and F) in compression with HBP, SDR, and irreversible JPEG 2000 (at the same compression ratio as obtained with HBP). The GLCM parameters were calculated at 1-pixel (A and B), 4-pixel (C and D) and 7-pixel (E and F) distances. The data boxes represent medians with 75th percentiles, while error bars indicate corresponding 90th percentiles.

to that obtained with HBP) was performed using irreversible JPEG2000, a decrease in the contrast was detected at the 1-pixel distance, whereas no changes were observed at larger distances. One should note that the decrease was similar in magnitude to the change produced by HBP at the same distance. Application of SDR resulted in an increase of the contrast detectable at all pixel distances (Fig. 4A, C and E). The increase was larger than that generated by HBP. Furthermore, dispersion of the correlation was greater with SDR than with HBP. One should note that the former algorithm altered GLCM correlation at all pixel distances (Fig. 4B, D and F). No such effect was observed when HBP or irreversible JPEG2000 was used.

Application of HBP produced a marked increase in inverse difference moment at distances of 1 pixel (Fig. 5A), 4 pixel (Fig. 5C) and 7 pixels (Fig. 5E). The magnitude of this increase was similar at all distances. Only small dispersion of GLCM inverse difference moment values was observed (at all distances) when HBP was used. Larger median increase and dispersion were detected when SDR was applied (Fig. 5A, C and E). Both these parameters were dependent on pixel distance when this algorithm was used. On the other hand, application of irreversible JPEG2000 resulted in increase of the inverse difference moment only at the 1-pixel distance. Furthermore, the magnitude of this change was smaller (by a factor of 10) than that generated by HBP. The opposite situation could be observed when texture alterations were studied using GLCM difference average. Application of HBP did not generate any significant changes of this parameter (Fig. 5B, D and F), whereas irreversible JPEG2000 produced a marked decrease at the 1-pixel distance and a smaller decrease at the 4-pixel distance. One should note that large dispersion of this parameter was detectable at the 1-pixel distance. This indicates that the distortion was affected by the image content (as described further). As with HBP, use of SDR did not result in a significant median change of the correlation.

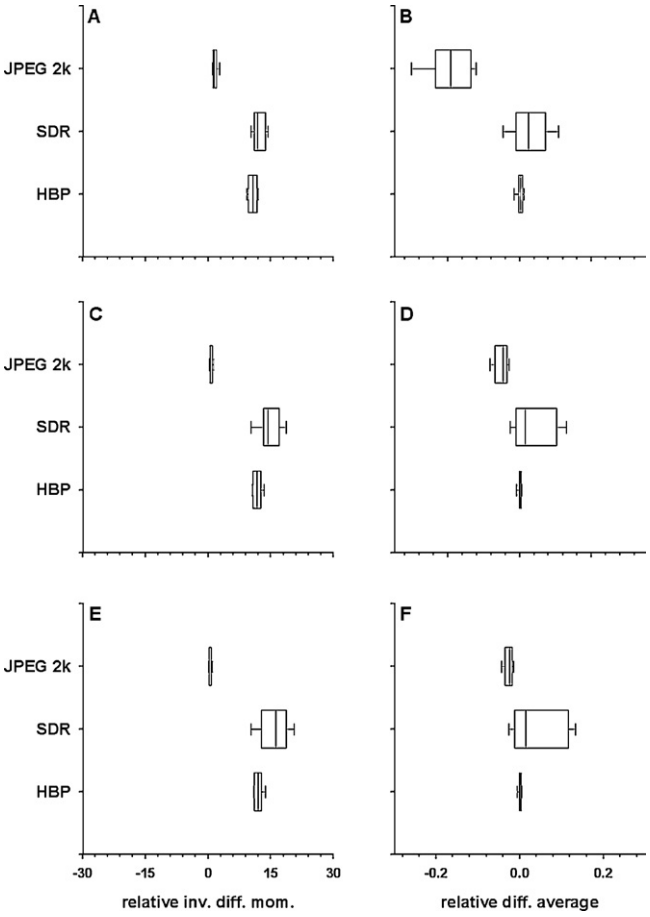

Fig. 5 – Change of relative GLCM inverse difference moment (A, C and E) and difference average (B, D and F) in compression with HBP, SDR, and irreversible JPEG 2000 (at the same compression ratio as obtained with HBP). The GLCM parameters were calculated at 1-pixel (A and B), 4-pixel (C and D) and 7-pixel (E and F) distances. The data boxes represent medians with 75th percentiles, while error bars indicate corresponding 90th percentiles.

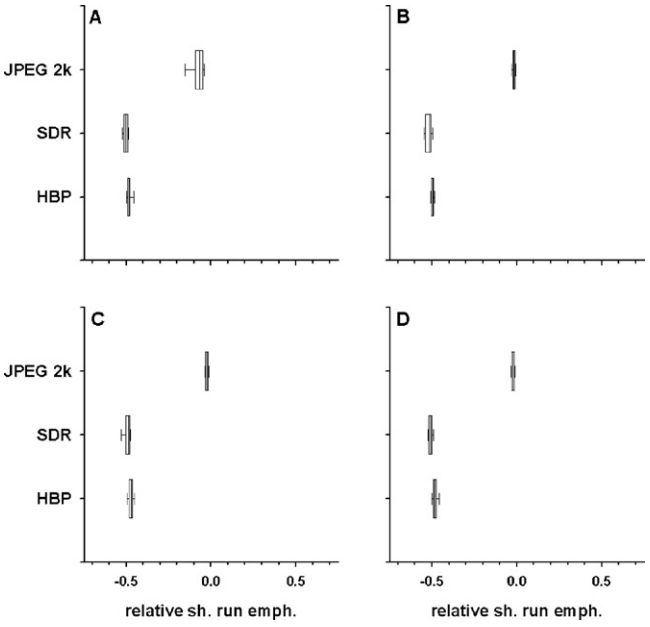

Fig. 6 – Change of relative runlength short-run emphasis in images compressed with HBP, SDR, and irreversible JPEG 2000 (at the same compression ratio as obtained with HBP and reversible JPEG2000 coding). The parameter was calculated at directions corresponding to 0° (A), 45° (B), 90° (C), and 135° (D). The data boxes represent medians with 75th percentiles, while error bars indicate corresponding 90th percentiles.

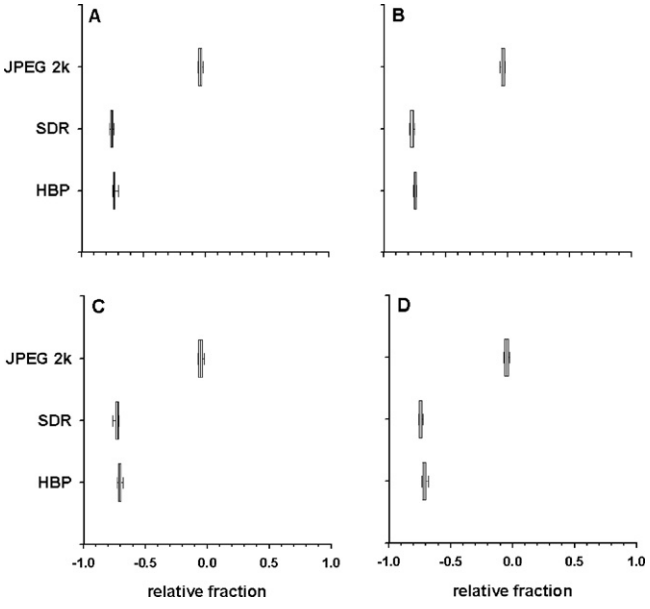

Fig. 7 – Change of relative runlength fraction in images compressed with HBP, SDR, and irreversible JPEG 2000 (at the same compression ratio as obtained with HBP). The parameter was calculated at directions corresponding to 0° (A), 45° (B), 90° (C), and 135° (D). The data boxes represent medians with 75th percentiles, while error bars indicate corresponding 90th percentiles.

However, dispersion of this parameter was large at all pixel distances.

3.4.5. *Runlength parameters*

Application of HB and SDR resulted in a decrease of runlength short-length emphasis (Fig. 6). The median decrease was similar in all directions (compare panels A–D in Fig. 6) and the dispersion was small in all cases. Compression with reversible JPEG2000 did not alter median the value of this parameter. However, dispersion short-length emphasis indicated that changes in texture could be present in some of the images. Furthermore, the dispersion was detectable only at 0° (Fig. 6A), and not at 45°, 90°, or 135° (Fig. 6B–D). This notion indicates directional character of these alterations. Like the previous parameter the runlength fraction was decreased when images were compressed using HB and SDR (Fig. 7). Again the magnitude of the decrease was similar in all directions and the values exhibited small dispersion. No changes of

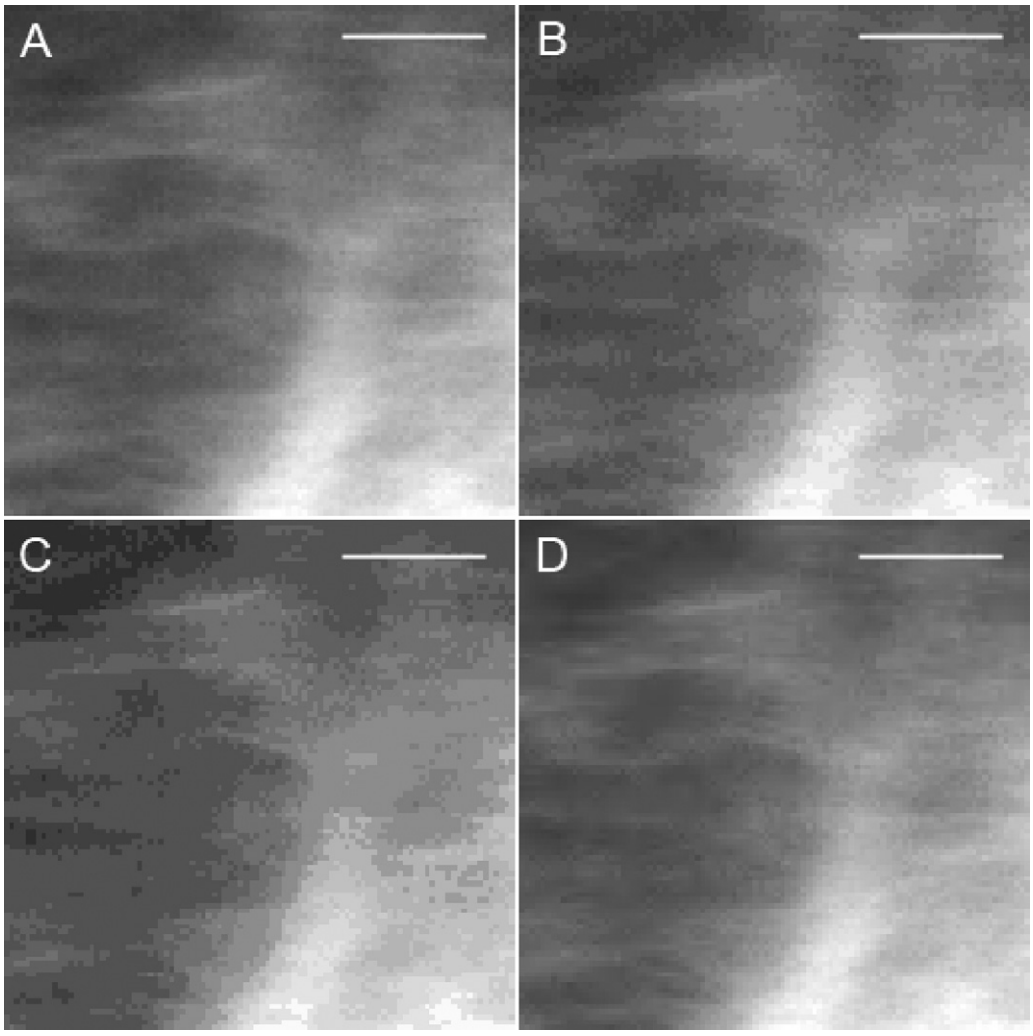

Fig. 8 – Enlarged central region of micrograph of fluorescently immunostained fibroblast (presented in the Fig. 2). The image is shown with gamma of 1.3 and using nominal (A) and reduced number of intensity levels, calculated with histogram binning, HB (B), and simple depth reduction, SDR (C). For comparison a raw image compressed using irreversible JPEG2000 (with the same ratio as obtained with HBP and reversible JPEG2000 coding) is shown (D). Image segmentation was performed using HB with $p = 0.95$. Scale bar 5 μm.

this parameter were detected in the images compressed with irreversible JPEG2000.

3.4.6. *Local vs. global image distortion*

The presented measures of image distortion deliver one estimate which takes into account all pixels in a studied image. However, one may hypothesize that the distortions are not distributed uniformly in images. Closer investigation of images reveals no such effect in the images compressed with HBP (Fig. 8B) and SDR (Fig. 8C), where dim and bright image regions are altered in similar manner. However, in the images compressed with irreversible JPEG2000 pronounced smoothing is detectable in the dim but not in the bright image regions (compare Fig. 8A and D). Furthermore, texture artifacts (in the form of crosses) are introduced in the dim image regions when this algorithm is used. This notion indicates that compression with irreversible JPEG2000 resulted in both removal of original and introduction of new texture elements.

4. Discussion

Owing to the heterogeneity of imaged specimens and the presence of detector noise, typical biological micrographs are characterized by a large number of intensity levels. Using characteristics of noise as a function of signal one may reduce the number of levels with intensity-binning (HB) procedure. The resultant images have sparse histograms, i.e., a small number of intensity levels distributed throughout the nominal intensity range. One should note that image-compression algorithms are not optimized for this type of histograms. Indeed, the presence of sparse histograms has been reported to worsen compression ratios of images coded with reversible algorithms [16]. Therefore, to take advantage of a small number of intensity levels, histogram-packing techniques should be used with reversible coding.

This idea is in agreement with the observation that only a small increase in compression ratios was obtained when

images with sparse histograms were coded using reversible algorithms optimized for image data (JPEG-LS and JPEG2000). On the other hand, compression efficiency was significantly improved when a universal coding algorithm (such as Deflate) was used. As expected, the highest compression ratios for biological fluorescence micrographs were obtained when histogram packing (employing HBP or SDR) was combined with reversible image coding (JPEG-LS and JPEG2000). The ratios were nearly an order of magnitude higher than those typically obtained for the reversible coding used with raw data. One should note that histogram packing did not improve compression efficiency for universal coding. Usually, higher ratios were obtained with SDR than with HBP. However, this notion was true for images characterized by high background (i.e., registered at high offset values), so it may not be regarded as a general rule.

Compression pre-processing with HBP does not markedly affect the global intensity distribution (image histogram) as estimated with EMD. The average change in pixel intensity was 0.48% of the nominal intensity range (4096 levels). Greater distortion was observed when SDR was used (1.46% of the intensity range). However, both these values were smaller than the average distance between nearest intensity levels in the images processed with HBP and SDR (2.57%). Histogram alterations generated by irreversible JPEG2000 were negligible when compared to SDR and HBP (0.03% respectively).

Processing images with HBP and SDR does significantly alter local intensity distribution measured with wavelet energy, regardless of decomposition band. On the other hand, distortions detectable in the decomposition bands corresponding to fine details were produced when irreversible JPEG2000 was used. One should note that these distortions were dependent on image content and were not uniform within images. Irreversible JPEG2000 compression and HBP/SDR produced changes in Haralick contrast and correlation similar in magnitude but opposite in sign. Larger dispersion of these values was noted only when SDR was used, indicating some dependence on image content. Irreversible JPEG2000 generated no texture changes detectable with GLCM inverse difference moment, whereas such changes were observed for HBP and SDR. Conversely, JPEG2000 generated changes in GLCM correlation at small distances (corresponding to fine details), whereas HBP did not influence this value. Large dispersion of the correlation values suggests dependence of JPEG2000-generated distortion on image content. Similar large dispersion was noted for SDR, albeit with smaller median compared to the respective value for JPEG2000.

HBP and SDR produced large decreases in runlength fraction and short-run emphasis. However, this change was similar in all the images (as indicated by small dispersion) and independent of direction. No significant change of median values of these parameters was introduced by irreversible JPEG2000. A large dispersion fraction values indicated directional character of the changes.

It should be emphasized that the texture parameters were specifically picked to detect changes introduced to the images by reduction in the number of intensity levels. Therefore the presented results may be considered to be upper estimates of image distortions introduced by SDR and HBP. Therefore, this specific set of texture parameters might not detect all distortions introduced by irreversible JPEG2000. Nonetheless, performance of HPB was superior to irreversible JPEG2000 in the sense of many texture parameters. What is more, the effects produced by the former were similar within the image set and uniform within single images. Hence, possible distortions were introduced in a consistent and non-arbitrary manner. This is not surprising since reduction of the number of intensity levels was performed using the detector characteristics but not image-content information. JPEG2000 and other popular irreversible image-compression algorithms do not offer these advantages as they are optimized to preserve perceptual image quality.

Although objective metrics of image fidelity are used in this work, one may note also that the changes introduced to images by tested methods could be detected on visual inspection. These changes were clearly manifested on the large scale SDR method (Figs. 2 and 8). Similar effects of HBP or JPEG2000 (at ratio identical to HBP) could be noticed when pixels become distinguishable with naked eye (small scale, Figs. 1 and 8). Since these differences may not always be evident in print or pdf file (as creating these is an operation of image processing) we provide original (uncompressed bitmap) figures upon request.

Intensity binning presented in this work was executed so that the levels were different from one another with 95% probability (confidence). The choice of confidence interval was arbitrary. Similar calculations can be performed for every confidence level. Therefore, one may optimize the statistical significance of intensity differences to obtain a desired compression ratio. In other words limited storage space or network transmission bandwidth may be used optimally so as to accommodate maximum amount of scientific information. Since such optimization depends only on detector characteristics and is not affected by image content management of the storage and bandwidth resources may be facilitated by intensity binning. This advantage may be especially important in the storage systems consisting of separate volumes connected by a network. Furthermore, image data corresponding to various levels of statistical significance may be encoded sequentially providing progressive image transmission over a network. One should note that data may be transmitted in this scheme in the order of statistical significance (as opposed to ("visual importance"). This may be an important advantage in the fields of telemicroscopy [31,32] and telemedicine [32,33].

The proposed HBP algorithm can be used directly with JPEG2000 and JPEG-LS compressors/decompressors. In fact, direct support for histogram packing, designed initially for palette images, is part of the JPEG-LS baseline standard [19]. Moreover, the 2nd part of the JPEG2000 standard [7] describes two generic non-linear transformations that may be applied to decoded pixel intensity levels: the piece-wise linear function and the gamma-style function. Encoding packed histogram as a mapping table is a special case of the latter. Therefore these algorithms are capable of reconstructing an image and its original histogram employing their standard functionality (i.e., no additional step of histogram expansion is required). Furthermore, the JPEG-LS and JPEG2000 standards define only the decoding parts of the

respective algorithms and the format of the compressed data stream. Provided that histograms are encoded in the required format, the whole process of applying HBP and core JPEG-LS/JPEG2000 image-coding routines is compliant with these standards.

The SDR and HP algorithms, applied in this study for monochrome images, might also be used with color data. It should be noted that color channels are usually transformed (Multi-Component Transform) to remove their mutual correlation. Therefore, if such operation was applied directly to multi-channel images processed with HB an increase of the number of levels would be expected. Consequently, the compression results could be worsened. This problem may be avoided by applying SDR/HP to individual channels of color image after subjecting them to HB and prior to Multi-Component Transform.

The presented algorithms were used with commercial package for modeling of camera noise. In the next stage open-source implementation of SDR and HP together with the noise analysis routine is envisaged.

5. Conclusions

The results presented show that HBP pre-processing provides significant enhancement in compression efficiency and does not introduce significant changes to biological micrographs. The algorithm has better fidelity than irreversible JPEG2000 (at the same compression rate) and can be easily implemented within the context of the current image-compression standards. This makes the proposed procedure an attractive enhancement to data-acquisition packages developed for biological imaging systems.

Acknowledgement

This work was supported by the Polish Ministry for Science and Higher Education (MNiSW) Grant no. N N301 463834 (TB).

REFERENCES

[1] T. Bernas, J.P. Robinson, Z. Darzynkiewicz, W. Hyun, A. Orfao, P. Rabinovitch, in: J.P. Robinson (Ed.), Basics of Digital Microscopy, Wiley-Liss, New York, 2005, pp. 1–14.
[2] US Federal Government, 21CFR11 Code of Federal Regulations Title 21 Chapter I Part 11 – Electronic Records; Electronic Signatures, USA, 2004.
[3] C.C. Chen, On the selection of image compression algorithms, in: Proceedings of the 14th Int'l Conference on Pattern Recognition, Brisbane, Australia, 1998, pp. 1500–1504.
[4] P.C. Cosman, R.M. Gray, R.A. Olshen, Evaluating quality of compressed medical images: SNR, subjective rating, and diagnostic accuracy, Proc. IEEE 82 (1994) 6919–6932.
[5] ISO/IEC, The basic JPEG standard, ISO/IEC International Standard 10918-1 and ITU-T recommendation T.81 part 1, 1994.
[6] ISO/IEC, Information Technology – JPEG2000 image coding system: core coding system, ISO/IEC International Standard 15444-1 and ITU-T recommendation T.800, 2004.
[7] ISO/IEC, Information Technology – JPEG2000 image coding system: extensions, ISO/IEC International Standard 15444-2 and ITU-T recommendation T.801, 2004.
[8] B. Wohlberg, G. de Jager, A review of the fractal image coding literature, IEEE Trans. Image Process. 8 (no. 12) (2002) 1716–1729.
[9] F. Ebrahimi, M. Chamik, S. Winkler, JPEG vs. JPEG2000: an objective comparison of image encoding quality, Proc. SPIE Appl. Digital Image Process. 5558 (2004) 300–308.
[10] S. Grgic, M. Mrak, M. Grgic, B. Zovko-Cihlar, Comparative study of JPEG and JPEG2000 image coders, Proc. 17th Conf. Appl. Electromagn. Commun. (2003) 109–112.
[11] J. Paz, M. Perez, I. Miranda, Diagnostic quality of high resolution JPEG 2000 compressed CT and MR brain images, IFMBE Proc. 25 (5) (2009) 334–337.
[12] F. Zanca, J. Jacobs, H. Bosmans, Evaluation of clinical image processing algorithms used in digital mammography, Med. Phys. 36 (2009) 765–776.
[13] T. Bernas, E.K. Asem, J.P. Robinson, B. Rajwa, Estimation of imaging precision for microscope calibration and image compression, J. Microsc. 226 (Pt. 2) (2007) 163–174.
[14] P. Young, Environmental modelling and software, Data-based Mech. Model. 13 (2) (1998) 105–122.
[15] A. Amer, E. Dubois, A. Mitiche, Reliable and fast structure-oriented video noise estimation, Proc. IEEE Int. Conf. Image Process 1 (2002) 840–843.
[16] A.J. Pinho, On the impact of histogram sparseness on some lossless image compression techniques, Int. Conf. Image Process. (2001) 442–445.
[17] T. Bernas, B. Rajwa, E.K. Asem, J.P. Robinson, Compression of fluorescence microscopy images based on the signal-to-noise estimation, Microsc. Res. Tech. 69 (1) (2006) 1–9.
[18] R. Starosolski, Compressing images of sparse histograms, Proc. SPIE Med. Imaging 5959 (2005) 209–217.
[19] ISO/IEC, Information Technology – lossless and near-lossless compression of continuous-tone still images – baseline, ISO/IEC International Standard 14495-1 and ITU-T recommendation T.87, 1999.
[20] National Electrical Manufacturers Association (NEMA), Digital imaging and communications in medicine (DICOM) part 5: data structures and encoding, NEMA Standard PS 3.5-2009, Rosslyn, USA, 2009.
[21] S.W. Golomb, Run-length encodings, IEEE Trans. Inform. Theor. IT-12 (1966) 399–401.
[22] R.F. Rice, Some practical universal noiseless coding techniques – part III, Jet Propul. Lab. 9 (1-3) (1979).
[23] M.J. Weinberger, G. Seroussi, G. Sapiro, A low complexity, context based, lossless image compression algorithm, Proc. IEEE Data Compr. Conf. (1996) 140–149.
[24] A. Moffat, R.M. Neal, I.H. Witten, Arithmetic coding revisited, ACM Trans. Inform. Syst. 16 (3) (1998) 256–294.
[25] C. Christopoulos, A. Skodras, T. Ebrahimi, The JPEG2000 still image coding system an overview, IEEE Trans. Consum. Electron. 46 (4) (2000) 1103–1127.
[26] P. Deutsch, Network working group, request for comments: 1951 – Deflate compressed data format specification version 1.3, 1996.
[27] J. Ziv, A. Lempel, A universal algorithm for sequential data compression, IEEE Trans. Inform. Theor. 32 (3) (1977) 337–343.
[28] Y. Rumner, C. Tomassi, J.G. Leonidas, Earth mover's distance as a metric for image retrieval, Int. J. Comp. Vis. 40 (2000) 99–121.
[29] A. Materka, M. Strzelecki, Texture analysis methods – a review, COST B11 report (1998) 1–33.

[30] M. Tuceryan, A.K. Jain, Texture analysis, in: C.H. Chen, L.F. Pau, P.S.P. Wang (Eds.), Handbook of Pattern Recognition and Computer Vision, 2nd ed., World Scientific Publishing Co., 1998, pp. 207–248.
[31] P.J. Hines, Telemicroscopy – A Review Australian Conference on Microscopy and Microanalysis, ACMM19, Sydney, Australia, 2006.
[32] H.K. Huang, Telemedicine and Teleradiology in PACS and Imaging Informatics: Basic Principles and Applications, 2nd ed., Wiley-Blackwell, 2010, pp. 451–468.
[33] M.G. Strintzis, A review of compression methods for medical images in PACS, International Journal on Medical Informatics 52 (1998) 159–165.

J. Vis. Commun. Image R. 25 (2014) 1056–1063

Contents lists available at ScienceDirect

J. Vis. Commun. Image R.

journal homepage: www.elsevier.com/locate/jvci

New simple and efficient color space transformations for lossless image compression

Roman Starosolski

Institute of Computer Science, Silesian University of Technology, Akademicka 16, 44-100 Gliwice, Poland

ARTICLE INFO

Article history:
Received 17 December 2013
Accepted 5 March 2014
Available online 15 March 2014

Keywords:
Reversible color space transformation
Lossless image compression
Human vision system
Image coding
RCT
YCoCg-R
Karhunen–Loève transformation
JPEG-LS
JPEG2000
JPEG XR

ABSTRACT

We present simple color space transformations for lossless image compression and compare them with established transformations including RCT, YCoCg-R and with the optimal KLT for 3 sets of test images and for significantly different compression algorithms: JPEG-LS, JPEG2000 and JPEG XR. One of the transformations, RDgDb, which requires just 2 integer subtractions per image pixel, on average results in the best ratios for JPEG2000 and JPEG XR, while for a specific set or in case of JPEG-LS its compression ratios are either the best or within 0.1 bpp from the best. The overall best ratios were obtained with JPEG-LS and the modular-arithmetic variant of RDgDb (mRDgDb). Another transformation (LDgEb), based on analog transformations in human vision system, is with respect to complexity and average ratios better than RCT and YCoCg-R, although worse than RDgDb; for one of the sets it obtains the best ratios.

1. Introduction

It is known, that red, green, and blue primary color components of the RGB color space are highly correlated for natural images. The high correlation indicates that more than one image component contains the same information, e.g., image area which is bright in green component usually is also bright in red and blue. Above usually is true also for computer generated images since artificial images mostly are made to resemble natural ones, however it depends on the actual objective of the image's creator. The most common approach to RGB color image compression is to compress independently the image components obtained using a transformation from RGB to some less correlated color space. Without the transformation we would unnecessarily compress the same information more than once.

For a specific image, using on it the Principal Component Analysis (PCA) we may obtain the image-dependent Karhunen–Loève transformation (KLT), which optimally decorrelates the image [1]. Since PCA/KLT is practically too time complex to be computed each time an image gets compressed, fixed transformations are constructed by performing PCA on a representative set of images. Then it is assumed that the obtained KLT transformation will match individual images from and outside of the used set. However note, that optimal decorrelation of color space of the set of images may not lead to the best compression ratios of individual images — since, among other things, actual inter-component dependencies may be different in various images or even in various regions of the same image; also the transformation while removing inter-component correlation may transfer incompressible noise from one component to another. Many transformations were constructed based on KLT; recently different approaches allowing adaptation of the color space transformation to a given image were proposed. In [2] an adaptive selection of transformation, from a large family of simple transformations, is done at the cost of slight increase of the color image transformation process complexity. Significantly more complex, yet simpler than computing PCA/KLT for the whole image, Singular Value Decomposition based, image adaptive method of constructing color space transformation for the lossy compression is presented in [3]. Decades ago a PCA/KLT transformation constructed for video data with additional requirement to obtain one component that approximates the intensity perception of the human vision system, was used to construct the YCbCr color space [4]. The YCbCr color space is up to today used in various television systems and in lossy compression algorithms. Several variants of the space and of transformations from RGB to YCbCr exist. One of them (ICT), used in JPEG2000 [5] for lossy compression, is presented below with its inverse (Eq. (1)).

E-mail addresses: rstarosolski@polsl.pl, rstaros@gmail.com

http://dx.doi.org/10.1016/j.jvcir.2014.03.003
1047-3203/

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.16875 & -0.33126 & 0.5 \\ 0.5 & -0.41869 & -0.08131 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \iff \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.402 \\ 1 & -0.34413 & -0.71414 \\ 1 & 1.772 & 0 \end{bmatrix} \begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} \quad (1)$$

Following [4], to distinguish between actual perception and it's computer representation, we use the term luma for the color space component representing image intensity perception (actual luminance), and term chroma for remaining components responsible for image chrominance.

It is an interesting fact, that analog color space transformation resulting in single luminance and 2 chrominance components is performed by the human vision system. Three types of cone cells in our retinas are most sensitive to three light wavelengths, these are L-cones (long wavelength with sensitivity peak in yellow), M-cones (middle, peak in green) and S-cones (short, peak in violet). Note, that the popular opinion, according to which cones simply respond to red (L-), green (M-) and blue (S-cones) light, is wrong — not only because cone sensitivity peaks are outside of red and blue wavelengths, but also since M- and L-cones are sensitive to the full visible spectrum; S-cones to colors ranging from violet to green. However, the highest reaction to blue color, among all cone types, is indeed shown by S-cones, to green by M-cones, and to red by L-cones. The cone response is then transformed and three calculated components are transmitted to the brain via the optic nerve:

- the luminance being a sum of L- and M-cones response,
- the red minus green color component (a difference between responses of L- and M-cones),
- and the blue minus yellow color component (a difference between response of S-cones and a sum of L- and M-cones responses; it may also be seen as difference between response of S-cones and the luminance).

We mentioned only certain aspects of human color vision reduced to essentials, for thorough description the Reader is referred to [6].

In case of lossless color image compression, the color space transformation has to be reversible considering that transformed components are stored using integers (it has to be integer-reversible). The transformation to the YCbCr color space could be used for that purpose at the cost of a dynamic range expansion of all the transformed color space components by 2 bits [7]. Here, the dynamic range of a component is defined as a number of bits required to store pixel intensities of this component. Since transformations designed for the lossless compression result in better lossless ratios as well as in smaller dynamic range expansion and are of smaller computational complexities, they are used instead. There are several established and standard such transformations, usually being variants of an irreversible transformation. In JPEG2000 for lossless coding the reversible RCT transformation is used [5], it is defined as a series of integer-reversible steps:

$$\begin{aligned} Cv &= R - G & G &= Y - \lfloor (Cu + Cv)/4 \rfloor \\ Cu &= B - G & \iff R &= Cv + G \\ Y &= G + \lfloor (Cu + Cv)/4 \rfloor & B &= Cu + G \end{aligned} \quad (2)$$

where the floor symbol $\lfloor x \rfloor$ denotes the greatest integer not exceeding x.

The RCT transformation is computationally simple. Floor of division by integer power of 2 may be calculated using single bit-shift, so both forward and inverse transformations require 5 simple integer operations (add, subtract, bit-shift) per image pixel. The dynamic range of the luma component Y is the same as of RGB components, chroma Cu and Cv components are 1 bit greater.

The RCT transformation was obtained using a lifting scheme [8] for factorization of the below transformation matrix (Eq. (3)) into lifting steps. Hence the below matrix, without additional assumptions, is a close approximation of the RCT transformation only.

$$\begin{bmatrix} Y \\ Cu \\ Cv \end{bmatrix} \approx \begin{bmatrix} 1/4 & 1/2 & 1/4 \\ 0 & -1 & 1 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \iff \begin{bmatrix} R \\ G \\ B \end{bmatrix} \approx \begin{bmatrix} 1 & -1/4 & 3/4 \\ 1 & -1/4 & -1/4 \\ 1 & 3/4 & -1/4 \end{bmatrix} \begin{bmatrix} Y \\ Cu \\ Cv \end{bmatrix} \quad (3)$$

The necessary and sufficient condition for such factorization is that the determinant of the transformation matrix is 1 or −1 [9]. Therefore linear transformations may be made reversible using the lifting scheme with additional scaling of transformation matrix rows if necessary. Notice, that the forward RCT transformation matrix is an approximation of the ICT matrix with scaled chroma rows.

Another such transformation (YCoCg-R) is included in the JPEG XR recent standard [10]. It is a variant of the irreversible transformation (YCoCg), which was obtained based on the KLT transformation constructed for a Kodak set of images (the set is described in Section 3.2) [4]. YCoCg-R is performed in following steps:

$$\begin{aligned} Co &= R - B & t &= Y - \lfloor Cg/2 \rfloor \\ t &= B + \lfloor Co/2 \rfloor & G &= Cg + t \\ Cg &= G - t & \iff B &= t - \lfloor Co/2 \rfloor \\ Y &= t + \lfloor Cg/2 \rfloor & R &= B + Co \end{aligned} \quad (4)$$

Both forward and inverse transformations require 6 simple integer operations per image pixel. The dynamic range of the luma component Y is the same as of RGB components, chroma Co and Cg components are 1 bit greater. The YCoCg-R transformation approximate forward and inverse matrix equivalents are presented in below Eq. (5).

$$\begin{bmatrix} Y \\ Co \\ Cg \end{bmatrix} \approx \begin{bmatrix} 1/4 & 1/2 & 1/4 \\ 1 & 0 & -1 \\ -1/2 & 1 & -1/2 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \iff \begin{bmatrix} R \\ G \\ B \end{bmatrix} \approx \begin{bmatrix} 1 & 1/2 & -1/2 \\ 1 & 0 & 1/2 \\ 1 & -1/2 & -1/2 \end{bmatrix} \begin{bmatrix} Y \\ Co \\ Cg \end{bmatrix} \quad (5)$$

Sometimes the dynamic range expansion is not allowed or undesirable. Expansion may be not allowed if implementation we use limits the bit-depth of processed data — some implementations allow bit depths not exceeding 8 or 16 bits. Such expansion is undesirable if it involves extra cost, e.g., certain implementations are optimized for 8-bit components, which are processed faster and requiring less memory, than while compressing components of 9–16 bit depth. Expansion may be avoided by means of the modular-arithmetic, the transformation included in the JPEG-LS extended standard (mRCT) is a modular-arithmetic version of the RCT [11]:

$$\begin{aligned} mCv &= (R - G) \bmod 2^N & G &= (mY - \lfloor (mCu + mCv)/4 \rfloor) \bmod 2^N \\ mCu &= (B - G) \bmod 2^N & \iff R &= (mCv + G) \bmod 2^N \\ mY &= (G + \lfloor (mCu + mCv)/4 \rfloor) \bmod 2^N & B &= (mCu + G) \bmod 2^N \end{aligned} \quad (6)$$

where N is the dynamic range of components of RGB image expressed in bits per pixel, e.g., for N bit dynamic range the allowable R, G, and B pixel values are in the range $[0 \ldots 2^N - 1]$; $a \bmod 2^N$ is the positive reminder of the division of a by 2^N, or practically the N least significant bits of a, which is in range $[0 \ldots 2^N - 1]$; $a \operatorname{smod} b$ is the symmetrical modulo in the range $[-b/2 \ldots b/2-1]$, $a \operatorname{smod} b = ((a + b/2) \bmod b) - b/2$. We assume that both mod and smod are of a similar complexity as, e.g., integer addition. Actually the mod may be simpler (especially for 8 and 16 bit data, where it is just a 1 or 2 Byte memory read). The smod requires two additions, or less, e.g., if pixels are stored as unsigned integers (we may skip subtracting the half of the range). Notice, that not all operations are performed using modular arithmetic, $(mCu + mCv)/4$ is computed in regular arithmetic. The smod is used for chroma components since chroma pixel value, being a difference between primary RGB colors, is most often close to 0. Due to modulo clipping the regular mod would result in creation of greater number of sharp edges in the transformed chroma components. Such edges result from mod on neighboring chroma component pixels close to 0 (between ones smaller than 0 and ones greater or equal 0), while after smod two less frequent cases introduce edges: pixels close to 2^{N-1} and close to $-(2^{N-1})$. Since luma (mY) is calculated using transformed chroma components, also this component contains extra edges (see Fig. 1). The dynamic range of all the transformed components is the same as of RGB components. Both forward and inverse mRCT transformations require 8 simple integer operations per image pixel.

Using the lifting technique the integer-reversible variants of other linear transformations may be obtained, including the KLT. Method presented in [9,12] decomposes the KLT transformation matrix into a series of Triangular or Single-row Elementary Reversible Matrixes, which are then used for integer-reversible KLT transformation (RKLT). In a general case, for 3 component image pixel both forward and inverse RKLT transformations require 8 additions, 8 multiplications and 4 to 5 roundings. Note, that the above complexity does not include the process of computing the KLT transformation matrix and of decomposing it. The KLT dynamic range expansion is image-dependant, for typical RGB images the first component gets expanded by up to 2 bits, while remaining ones are not expanded.

In this paper we present a couple of new color space transformations for lossless image compression. Our primary aim was to find transformations simpler than the established ones, while not being practically inferior in terms of obtainable compression ratios or the dynamic range expansion. New transformations are constructed based on observations of compression effects of individual transformed color components rather than on approximations of KLT transformation; some of them are inspired by the analog calculations taking place in the human vision system.

The reminder of this paper is organized as follows. In Section 2, supported by preliminary experiments, we propose new transformations. In Section 3 we compare experimentally new transformations with others, including the reversible variant of the optimal KLT. Experiments are performed for 3 sets of test images and for 3 significantly different standard compression algorithms: JPEG-LS, JPEG2000 and JPEG XR. We analyze complexities of transformations, compression ratios obtained for the transformed images and correlation of transformed components. Section 5 summarizes the research.

2. Proposed transformations

2.1. Giving the luminance up

Luma calculated from all components is used in lossy compression as it is a good representation of perceptual luminance, to which human vision is more sensitive than to chrominance; luma typically is encoded with higher quality than chroma, a common practice in lossy compression is to subsample chroma components. In lossless compression we usually compress an image in order just to then decompress the entire image. In such a case a good direct approximation of the luminance is not necessary in the transformed image color space. Replacing the luma component with something else may simplify the color space transformation — since, e.g., calculation of the Y component in RCT is more complex than calculation of remaining components. Furthermore, luma calculated based on all components contains a fraction of the noise from all components. As opposed to lossy algorithms that filter out the noise, lossless algorithms preserve all the information contained in the component, including the incompressible noise. Most reversible transformations for lossless compression are based on irreversible transformations for lossy compression, but maybe a different criteria should be applied when constructing transformation for lossless and for lossy compression? Preliminary experiments (Table 1) showed, that for the Waterloo set of test images and the JPEG2000 algorithm in the lossless mode (sets, algorithms and experimental procedure are described in Section 3.2) the untransformed R component, indeed, is more compressible than Y component of the RCT color space, which in turn compresses better, than untransformed components G and B. Note that only certain combinations of three transformed single component formulas from Table 1 constitute the reversible color space transformation (the transformation matrix determinant should be 1 or -1), and that certain transformations are defined using

Fig. 1. Luma components of the "peppers" image after RCT (left) and mRCT (right) transformations.

Table 1
Average JPEG2000 lossless compression ratios of individual transformed components of images from Waterloo set (algorithms and test images described in Section 3.2).

Description	Formula	Ratio (bpp)	Formula	Ratio (bpp)	Formula	Ratio (bpp)
Untransformed	R	4.2562	G	4.3954	B	4.3456
RCT Y variants	$(2R+G+B)/4$	4.2786	$(R+2G+B)/4$	4.3205	$(R+G+2B)/4$	4.3114
Two primary color difference	$R-G$	3.4148	$G-B$	3.4718	$B-R$	3.6532
Negated difference	$G-R$	3.4043	$B-G$	3.4784	$R-B$	3.6506
Average of two primary colors	$(R+G)/2$	4.3112	$(G+B)/2$	4.3533	$(B+R)/2$	4.2867
$B-L$ variants	$R-(G+B)/2$	3.4515	$G-(R+B)/2$	3.2453	$B-(R+G)/2$	3.4937

lifting steps, so the actual ratio of Y of RCT may slightly differ from the ratio reported for component formula $(R+2G+B)/4$.

RCT chroma components seem to be a good choice, since their calculation is done in only one integer subtraction per component pixel and they compress significantly better than Y or untransformed primary color components. It is better to subtract primary colors of closer wavelengths ($R-G$ and $G-B$), then the most distant ones ($B-R$). Therefore in the color space named RDgDb (Eqs. (7) and (8)) we use the R component of the RGB space instead of luminance and, similarly to RCT, use differences of primary colors as chroma components:

$$\begin{bmatrix} R \\ Dg \\ Db \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \iff \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 0 \\ 1 & -1 & -1 \end{bmatrix} \begin{bmatrix} R \\ Dg \\ Db \end{bmatrix} \tag{7}$$

Forward and inverse transformations require 2 integer subtractions only:

$$\begin{aligned} R &= R & R &= R \\ Dg &= R - G \iff & G &= R - Dg \\ Db &= G - B & B &= G - Db \end{aligned} \tag{8}$$

The dynamic range of Dg and Db chroma components is by 1 bit greater, than the dynamic range of RGB components. As opposed to most established color space transformations, where matrix representation is an approximation of actual transformation performed in a series of lifting steps, in case of the RDgDb the matrix is an equivalent definition of the transformation.

Our Db component is a negation of RCT Cu component. We subtract shorter wavelength primary color from one of longer wavelength, while in RCT always the green is subtracted from the other one. The difference is practically not important (negating chroma components would change average ratios by about 0.01 bpp or less), may be of opposite sign for other images or for other algorithms; the formula used clearly suggests how to extend the RDgDb color space to multispectral images.

Transformations close to RDgDb have been recently investigated in [2] — among others, 3 transformations were proposed (A2, A6 and A7), in which a primary color is used instead of luminance while both chroma components are differences in which the primary color replacing luminance is used as subtrahend. Below we present forward transformations A2 (Eq. (9)), A6 (Eq. (10)) and A7 (Eq. (11)).

$$\begin{bmatrix} Y_{A2} \\ U_{A2} \\ V_{A2} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{9}$$

$$\begin{bmatrix} Y_{A6} \\ U_{A6} \\ V_{A6} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 0 & 1 \\ -1 & 1 & 0 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{10}$$

$$\begin{bmatrix} Y_{A7} \\ U_{A7} \\ V_{A7} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & -1 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{11}$$

Among them the best results were reported for A2, where G is used instead of luminance. However, results presented in the Table 1 indicate, that it is better to use R instead of luminance, while out of two chroma components, both being primary color differences, only in one the component R replacing luminance should be used. Complexity and dynamic range expansion of A2, A5 and A7 is the same as of RDgDb.

2.2. Human vision inspired transformations

If we expect that the luma component may be decompressed without decompressing image chroma components, then a reasonably good approximation of luminance is needed. May the luminance approximation be better than a single primary color, but simpler to compute then the luma component in RCT? The affirmative answer to the above question is in our (human) vision system, where luminance is a sum of responses of two cone cell types: L- and M-cones. Therefore in LDgEb space the luma is, as luminance in humans, a sum of two longer wavelength components, but multiplied by the smallest factor that permits transformation reversibility $L=(R+G)/2$. Following the human vision also for chroma components leads to chroma formulas $Dg=R-G$ and $Eb=B-L$ (Eb is the excess of blue over the luminance). The approximate matrix definition of the LDgEb transformation is:

$$\begin{bmatrix} L \\ Dg \\ Eb \end{bmatrix} \approx \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1 & -1 & 0 \\ -1/2 & -1/2 & 1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \iff \begin{bmatrix} R \\ G \\ B \end{bmatrix} \approx \begin{bmatrix} 1 & 1/2 & 0 \\ 1 & -1/2 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} L \\ Dg \\ Eb \end{bmatrix} \tag{12}$$

The integer reversible transformation is defined as a following sequence of steps:

$$\begin{aligned} Dg &= R - G & R &= L + \lfloor Dg/2 \rfloor \\ L &= R - \lfloor Dg/2 \rfloor \iff & G &= R - Dg \\ Eb &= B - L & B &= Eb + L \end{aligned} \tag{13}$$

Alternatively we may use "human vision" luma, but define chroma components as in RDgDb — obtaining transformation named LDgDb (Eq. (15)). Based on the Table 1 we may expect good compression ratios from both LDgEb and LDgDb transformations. LDgDb actually differs from LDgEb in calculation of one component only:

$$\begin{bmatrix} L \\ Dg \\ Db \end{bmatrix} \approx \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \iff \begin{bmatrix} R \\ G \\ B \end{bmatrix} \approx \begin{bmatrix} 1 & 1/2 & 0 \\ 1 & -1/2 & 0 \\ 1 & -1/2 & -1 \end{bmatrix} \begin{bmatrix} L \\ Dg \\ Db \end{bmatrix} \tag{14}$$

$$\begin{aligned} Dg &= R - G & R &= L + \lfloor Dg/2 \rfloor \\ L &= R - \lfloor Dg/2 \rfloor \iff & G &= R - Dg \\ Db &= G - B & B &= G - Db \end{aligned} \tag{15}$$

Both LDgEb and LDgDb forward and inverse transformations require 4 simple integer operations only. The dynamic range of chroma components is by 1 bit greater, than the dynamic range of RGB components, luma component is not expanded.

2.3. Modular arithmetic transformations

Since avoiding the expansion may be required in practical applications, we define modular-arithmetic versions of RDgDb (mRDgDb), LDgEb (mLDgEb), and of LDgDb (mLDgDb); mRDgDb is performed in following reversible steps:

$$\begin{aligned} R &= R & R &= R \\ mDg &= (R - G) \bmod 2^N \iff & G &= (R - mDg) \bmod 2^N \\ mDb &= (G - B) \bmod 2^N & B &= (G - mDb) \bmod 2^N \end{aligned} \tag{16}$$

where N is a dynamic range of RGB and of transformed mRDgDb components. Forward and inverse mRDgDb transformations require 4 simple integer operations only, while mLDgEb and mLDgDb (also forward and inverse) avoid expansion at a cost of 7 such operations; mLDgEb is performed in following steps:

$$\begin{aligned} mDg &= (R - G) \bmod 2^N & R &= (mL + \lfloor mDg/2 \rfloor) \bmod 2^N \\ mL &= (R - \lfloor mDg/2 \rfloor) \bmod 2^N \iff & G &= (R - mDg) \bmod 2^N \\ mEb &= (B - mL) \bmod 2^N & B &= (mEb + mL) \bmod 2^N \end{aligned} \tag{17}$$

3. Experimental results and discussion

3.1. Examined transformations

Properties of the examined transformations are presented in the Table 2. In the comparison we included RCT, YCoCg-R, and mRCT transformations being both simple and standard. We do not include other transformations, that may approximate YCbCr or KLT better, than the above mentioned ones. Instead, as a benchmark for simple transformations and to check how the optimal decorrelation of color space affects compression ratios we report results obtained using the irreversible KLT (not listed in Table 2) and the reversible KLT (RKLT) — in both cases the transformation was constructed for each image individually. Software developed by Janczur within his MSc thesis [13] was used to construct and to apply the RKLT to the tested images. Other transformations where applied using a prepared free software, which may be downloaded from http://sun.aei.polsl.pl/~rstaros/imgtransf/index.html. We report results for the A2, RDgDb, LDgEb, LDgDb as well as their modular-arithmetic variants. mA2 — the modular arithmetic variant of A2 was constructed analogically to mRDgDb, mLDgDb variant of LDgDb analogically to mLDgEb. We also check compression performance of variants of A2, RDgDb, LDgEb, and LDgDb obtained by the use of different primary colors for luma and if necessary for chroma calculation.

3.2. Procedure

Based on preliminary estimation, performed for the popular Waterloo set of color test images and the JPEG2000 algorithm in the lossless mode, we proposed RDgDb and LDgEb reversible color space transformations as well as a couple of their variants. The evaluation of transformations proposed was performed for the following sets of 8-bit RGB test images:

Table 2
Properties of the reversible color space transformations. Complexities reported in simple integer operations per image pixel, except for RKLT (floating-point operations, not including the cost of computing KLT matrix and decomposing it into lifting steps); component dynamic range expansion is the maximum possible, except for RKLT (typical for natural images).

Transformation	Complexity	Component range expansion		Remarks
		Luma/first	Chroma/others	
RKLT	20	2	0	See Refs. [7,10]
RCT	5	0	1	Eq. (2), from JPEG2000 standard [3]
YCoCg-R	6	0	1	Eq. (4), from JPEG XR standard [8]
A2	2	0	1	Eq. (9), see Ref. [11]
RDgDb	2	0	1	Eq. (7) and (8)
LDgEb	4	0	1	Eq. (13)
LDgDb	4	0	1	Eq. (15)
mRCT	8	0	0	Eq. (6), from JPEG-LS Extended standard [9]
mA2	4	0	0	Modular arithmetic variant of A2
mRDgDb	4	0	0	Eq. (16)
mLDgEb	7	0	0	Eq. (17)
mLDgDb	7	0	0	Modular arithmetic variant of LDgDb

- Waterloo — the already mentioned set of color images from the University of Waterloo, Fractal Coding and Analysis Group repository, used for a long time in image processing research. The set contains 8 natural photographic and artificial images, among them the well-known "lena" and "peppers", image sizes vary from 512 × 512 to 1118 × 1105. It is available from http://links.uwaterloo.ca/Repository.html.
- Kodak — a set of 24 photographic images released by the Kodak corporation, the set is frequently used in color image compression research. All images are of size 768 × 512, downloaded from http://www.cipr.rpi.edu/resource/stills/kodak.html.
- EPFL — a recent set of 10 high resolution images used at the École polytechnique fédérale de Lausanne for evaluation of subjective quality of JPEG XR [14]. Image sizes from 1280 × 1506 to 1280 × 1600, downloaded from http://documents.epfl.ch/groups/g/gr/gr-eb-unit/www/IQA/Original.zip.

Lossless compression ratios obtained for the transformed images were analyzed for the following standard algorithms:

- JPEG-LS — a standard of the JPEG committee for primarily lossless compression of still images. The baseline standard describes low-complexity predictive image compression algorithm with entropy coding using modified Golomb-Rice code family, standard extensions include the mRCT transformation [11,15].
- JPEG2000 — a JPEG committee image compression standard describing algorithm based on discrete wavelet transformation image decomposition and arithmetic coding, the standard includes the RCT color space transformation [5]. Apart from lossy and lossless compressing and decompressing of whole images JPEG2000 delivers many interesting features (progressive transmission, region of interest coding, etc.).
- JPEG XR — a recent JPEG committee standard describing algorithm designed primarily for high quality, high dynamic range photographic images; it is based on discrete cosine transformation image decomposition and adaptive Huffman coding; it defines the YCoCg-R color space transformation [10]. The standard supports lossy and lossless coding and certain additional features, like random access to parts of the encoded image.

Above algorithms process images in significantly different ways. First step of color image compression is similar: using the color space transformation we obtain 3 components, that are then compressed independently. In a predictive algorithm the next step for each component is to use the predictor function to guess the pixel intensities and then the sequence of prediction errors, being differences between actual and predicted pixel intensities, is encoded. Even using extremely simple predictors, such as one that predicts that pixel intensity is identical to the one on its left-hand side, results in a much better compression ratio, than without the prediction. JPEG-LS employs nonlinear edge-detecting predictor calculated using 4 neighbors of given pixel. In transformation algorithms, instead of pixel intensities, we encode a matrix of transformation coefficients. The transformation is applied to a whole image component (optional in JPEG2000), or to the component split into fragments, which may be big (default fragment size for JPEG2000 is 256×256 pixels), or even very small — JPEG XR applies transformation to blocks of size 4×4 pixels (2×2 in case of lossy chroma component coding). Transformation for lossless coding has to be integer-reversible, which for JPEG2000 and JPEG XR is accomplished with use of the lifting scheme.

All algorithms were used to compress individual transformed components, one component at a time. Due to requirements of employed file formats and standard implementations, all transformed components were stored using nonnegative integers obtained by subtracting the minimum nominally possible value from the pixel component. For example, if primary colors were in nominal range $[0,255]$, then $Dg = R - G$, which would normally be in the range $[-255,255]$, was actually stored as $Dg' = R - G + 255$. The compression ratio, expressed in bits per pixel (bpp), is calculated as $8e/n$, where n is the number of pixels in the image, e is the total size in Bytes of the individually and independently compressed 3 components of the transformed image, including compressed file format headers; hence smaller ratios mean better compression. We also report average absolute correlation of image components after transformation, which is calculated as $(|r(C_1,C_2)| + |r(C_2,C_3)| + |r(C_3,C_1)|)/3$, where C_1, C_2, and C_3 are components of the transformed image, r is Pearson product moment correlation coefficient, and $|x|$ is the absolute value of x.

3.3. Lossless compression ratios

Lossless compression ratios for transformed images are reported in Tables 3 (for JPEG-LS), 4 (JPEG2000) and 5 (JPEG XR). The best transformation result, for a given set and for average of 3 sets, is marked in bold; underlined are results not worse by more than 0.1 bpp then the best. Except for the transformations listed in the Table 2, we include results for irreversible KLT (irrevKLT).

Looking at the results of RKLT and irrevKLT transformations we can see, that optimal decorrelation of the image color space, even in its irreversible variant, does not lead to the best lossless ratios. Compared to the oldest reversible transformation RCT, for all the algorithms and all the sets, the RKLT and irrevKLT transformations resulted in ratios worse by at least about 0.5 bpp and 0.3 bpp respectively.

Among modular-arithmetic transformations, the best ratios for all the sets and for all the algorithms were obtained using the mRDgDb, except for the Kodak set in case of JPEG-LS (where the mLDgEb is by 0.06 bpp better). Modular-arithmetic transformations, as compared to their regular variants, may improve the compression ratios only in the case of JPEG-LS — ratios are improved for mRDgDb and mA2 in case of all the sets, and for all the transformations in case of the Kodak set; for JPEG2000 and JPEG XR ratios are consistently worsened. The improvement over regular variant always is the greatest in case of mRDgDb and mA2, also for these transformations the ratio worsening due to modulo arithmetic for JPEG2000 and JPEG XR is the smallest. Probable reason of the above advantage of the simplest among modular transformations over others is that all image components after other transformations contain edges introduced by modulo clipping, while after mRDgDb and mA2 only chroma components contain extra edges. Since JPEG-LS obtains better lossless ratios than other two algorithms, the overall best average ratios were obtained using mRDgDb and JPEG-LS.

The best ratios average for 3 sets, among all the transformations in case of JPEG2000 and JPEG XR as well as among non-modular transformations in case of JPEG-LS, are obtained using the simplest RDgDb. Non-modular transformations using human vision inspired luma component formula obtain ratios worse by up to 0.04 bpp, A2 is worse by 0.08 to 0.09 bpp, standard transformations RCT and YCoCg-R are worse by 0.05 to 0.15 bpp. RDgDb is most often the best transformation for a specific set. However, in case of the JPEG XR algorithm the YCoCg-R is better for Waterloo and Kodak sets (for Waterloo it obtains the best ratio). For the Kodak set in case of all the algorithms either LDgEb or LDgDb is the best non-modular transformation, the other one is the second best — notice that worse ratios were obtained by YCoCg-R based on KLT constructed for this set, the worst by KLT variants constructed for each image individually.

Since the proposed color spaces were constructed based on estimation of ratios for Waterloo set and JPEG2000 algorithm only, for all the sets and algorithms we checked their additional variants as well as variants of A2 transformation. For transformations which replace luminance with a primary color and require only 2 simple integer operations per pixel: RDgDb and A2, there are 9 possible variants not counting cases when given component is calculated as difference or negated difference. There are 3 variants of A2 (having following components: a primary color as luma and two differences between another primary colors and luma) and 6 variants of RDgDb (components: a primary color as luma, difference between luma and another primary color, difference between two primary colors other than luma). Out of these 9 variants the best for a specific set and on average was RDgDb, except for the Waterloo set, where for all the algorithms ratios better by less than 0.02 bpp were obtained when green color replaced luminance. Among variants of transformations requiring 4 operations: LDgEb (3 variants) and LDgDb (6 variants) either LDgEb or LDgDb was the best, except for the Waterloo set in case of JPEG2000 and JPEG XR, where LDgEb variant using R and B primary colors in luma calculation was better than LDgEb by about 0.04 bpp.

In practice, small differences in average compression ratios may not be the most important property of the color space transformation. Properties like dynamic range expansion, computational complexity, quality of luminance approximation, and existence of the standard describing the transformation may play practical role. In Tables 3–5 the ratios within 0.1 bpp from the best one are underlined. The selection of the 0.1 threshold was arbitrary, but observing when the ratios are within 0.1 bpp from the best generally confirms earlier observations: consistently outside of that range on average and for all the sets are modular-arithmetic transformations in case of JPEG2000 and JPEG XR, as well as KLT variants in all cases. For JPEG-LS the non-modular RCT and YCoCg-R are always outside. For all the sets the 3 proposed non-modular transformations are in 0.1 bpp range in case of JPEG2000 and JPEG XR compression. Other cases when for all the sets given transformation is within 0.1 bpp from the best one are RCT for JPEG XR, and mRDgDb for JPEG-LS. Looking at the average ratios, the proposed non-modular transformations are in the range for all the algorithms, while other transformations are for some only or for none.

Table 3
Average JPEG-LS compression ratios (bpp). Best ratio for a set and on average is marked in bold, ratios within 0.1 bpp from the best are underlined.

Transformation	Set			
	Waterloo	Kodak	EPFL	Average
No (RGB)	10.3764	13.0948	12.3369	11.9360
RKLT	9.5987	10.4365	11.0722	10.3691
irrevKLT	9.4989	10.1976	10.8014	10.1659
RCT	8.9625	9.5734	10.4660	9.6673
YCoCg-R	9.0232	9.6044	10.6125	9.7467
A2	8.9914	9.5502	10.4930	9.6782
RDgDb	8.8653	9.5673	10.3383	9.5903
LDgEb	8.9589	9.4335	10.4421	9.6115
LDgDb	8.9309	9.5476	10.4181	9.6322
mRCT	9.0017	9.4805	10.5079	9.6633
mA2	8.9546	9.4387	10.4719	9.6217
mRDgDb	**8.8285**	9.4559	**10.3172**	**9.5338**
mLDgEb	9.1277	**9.3985**	10.4992	9.6751
mLDgDb	8.9880	9.4580	10.4338	9.6266

Table 4
Average JPEG2000 compression ratios (bpp). Best ratio for a set and on average is marked in bold, ratios within 0.1 bpp from the best are underlined.

Transformation	Set			
	Waterloo	Kodak	EPFL	Average
No (RGB)	12.9972	13.4256	12.8244	13.0824
RKLT	11.8353	10.5579	11.3968	11.2634
irrevKLT	11.7252	10.3329	11.1456	11.0679
RCT	11.2141	9.5062	10.8356	10.5186
YCoCg-R	11.2125	9.4876	10.9776	10.5593
A2	11.2819	9.4686	10.8655	10.5387
RDgDb	**11.1428**	9.4754	**10.7240**	**10.4474**
LDgEb	11.2178	**9.4231**	10.8054	10.4821
LDgDb	11.1969	9.4590	10.7839	10.4799
mRCT	11.6473	9.6107	11.0823	10.7801
mA2	11.5023	9.5375	10.9883	10.6760
mRDgDb	11.3631	9.5443	10.8468	10.5848
mLDgEb	11.8572	9.5766	11.0409	10.8249
mLDgDb	11.6436	9.5652	10.9711	10.7266

Table 5
Average JPEG XR compression ratios (bpp). Best ratio for a set and on average is marked in bold, ratios within 0.1 bpp from the best are underlined.

Transformation	Set			
	Waterloo	Kodak	EPFL	Average
No (RGB)	14.9604	14.1331	13.6662	14.2532
RKLT	13.8199	11.6542	12.3010	12.5917
irrevKLT	13.7459	11.5089	12.1505	12.4684
RCT	13.3211	10.9196	11.7629	12.0012
YCoCg-R	**13.2655**	10.8552	11.8987	12.0065
A2	13.4243	10.8565	11.7911	12.0240
RDgDb	13.2978	10.8725	**11.6723**	**11.9476**
LDgEb	13.2982	10.8531	11.7402	11.9638
LDgDb	13.3108	**10.8484**	11.7154	11.9582
mRCT	14.2161	11.1797	12.2143	12.5367
mA2	13.9917	11.0834	12.0583	12.3778
mRDgDb	13.8652	11.0995	11.9395	12.3014
mLDgEb	14.5268	11.1971	12.1715	12.6318
mLDgDb	14.2450	11.1296	12.0795	12.4847

3.4. Component correlation

The average absolute correlation of image components, in RGB color space and after the examined transformations, is reported in Table 6. We can see, that both the irreversible KLT and its integer-reversible lifting approximation decorrelate image color space almost perfectly, which confirms observation that decorrelation of the color space is not a proper objective of the color space transformation for lossless compression. The opposite may be true — greater correlation of image components may potentially allow for better compression in algorithms that during compression of given component exploit other, already processed ones. There are algorithms, which instead of component transformation, exploit during coding the inter-component dependencies, e.g., the relatively time complex Interband CALIC [16]; low complexity algorithms were also proposed, e.g., SICLIC [17] or recent LMMIC [18]. Interband CALIC uses the inter-component predictor instead of regular (intra-component) one if in the neighborhood of the pixel being predicted the correlation between components is high. For such an algorithm, out of two transformations which would result in equal ratios for independently compressed components, probably better is the transformation decorrelating worse. However checking whether is it better than not to transform and leave all the correalation in components requires further investigation.

Table 6
Average absolute correlation of transformed image components.

Transformation	Set			
	Waterloo	Kodak	EPFL	Average
No (RGB)	0.6311	0.8435	0.8002	0.7583
RKLT	0.0005	0.0019	0.0009	0.0011
irrevKLT	0.0024	0.0053	0.0046	0.0041
RCT	0.3187	0.3100	0.3058	0.3115
YCoCg-R	0.2616	0.3374	0.3507	0.3165
A2	0.3978	0.3451	0.3095	0.3508
RDgDb	0.4006	0.3496	0.3455	0.3652
LDgEb	0.3746	0.3899	0.3972	0.3872
LDgDb	0.3633	0.3269	0.2980	0.3294
mRCT	0.1974	0.2870	0.2944	0.2596
mA2	0.1932	0.2972	0.2741	0.2548
mRDgDb	0.2288	0.3074	0.3120	0.2827
mLDgEb	0.2894	0.3587	0.3443	0.3308
mLDgDb	0.2075	0.2953	0.2715	0.2581

Interestingly, while the compression ratio of irrevKLT is better than of RKLT, the correlation of components is little greater in case of irrevKLT. Probably irrevKLT due to rounding removes fraction of noise contained in the image. Presence of noise on the one hand worsens compression ratios, on the other one decreases correlation of components.

Among non-modular transformations, ones being approximations of KLT (i.e., RCT and YCoCg-R) on average decorrelate better, than transformations constructed based on different criteria (A2, RDgDb, LDgEb, and LDgDb), however for a specific set it may not be true. In all cases modular-arithmetic transformations decrease correlation more, than corresponding regular variants. On average LDgEb decorrelates worstly, for a specific set — LDgEb or RDgDb.

4. Conclusions

We have proposed RDgDb and LDgEb simple color space transformations for lossless image compression and a couple of their variants. We departed from a traditional method of constructing transformation for lossless image compression based on transformation for lossy compression, which in turn is based on PCA/KLT for specific image set. RDgDb was proposed based on observation of actual lossless ratios of individual image components obtained with simple transformations or untransformed, while LDgEb originates from the human vision system. These transformations were evaluated and compared with established transformations including RCT, YCoCg-R and the optimal KLT for 3 sets of test images and for significantly different compression algorithms: predictive JPEG-LS, Discrete Wavelet Transformation based JPEG2000 and Discrete Cosine Transformation based JPEG XR.

The RDgDb transformation has the minimum computational complexity, equal to complexity of the simplest known so far A2 transformation, it requires just 2 integer subtractions per image pixel. RDgDb and A2 have certain disadvantages: the dynamic range of chroma components is expanded by 1 bit, the luminance is replaced by a primary color. However in a typical case of lossless compression, when an image is compressed just to be then decompressed as a whole, RDgDb seems to be the most universal — despite of being so simple, on average it results in the best ratios for JPEG2000 and JPEG XR, while for a specific set or in case of JPEG-LS its compression ratios are either the best or within 0.1 bpp from the best.

The overall best lossless ratios were obtained using the JPEG-LS algorithm and the modular-arithmetic variant of RDgDb, named mRDgDb, which requires 2 integer subtractions and 2 symmetrical modulo operations. Here RDgDb resulted in second best average ratio, worse by about 0.06 bpp. In practice such a little ratio improvement may not justify the increased complexity of the transformation. As all modular arithmetic transformations the mRDgDb avoids the dynamic range expansion of image components; compared to them it is of the lowest complexity and results in the best average ratios for all the algorithms. It will be practically useful in cases where the dynamic range expansion is not allowed or undesirable.

Sometimes a reasonably good perceptual luminance approximation in a transformed image may be useful, since it allows retrieving the luminance from compressed image by decompressing of one component only. We proposed LDgEb and LDgDb transformations inspired by analog calculations performed in human vision system, which in the above case may be useful for natural images. Compared to RDgDb these transformations are more complex and result in little worse average ratios, but compared to RCT and YCoCg they are simpler and result in better ratios. The most interesting is LDgEb, since it is closest to transformations in human vision system and for one of the test image sets it obtains the best ratios (its modular arithmetic variant in case of JPEG-LS).

We also notice that optimal color space decorrelation performed with KLT, despite of constructing the KLT for each image individually and even giving up the integer reversibility, does not lead to good lossless ratios. Color space decorrelation is not a proper aim of transformation for lossless compression, controversially poor decorrelation may allow for better compression in algorithms exploiting inter-plane correlation. In the case of tested transformations, the greatest correlation of transformed image components is observed for LDgEb and RDgDb.

Color space components after the LDgEb transformation are closer to components transmitted to the human brain via the optic nerve, than components of spaces traditionally used in color image digital transmission, like YCbCr or untransformed RGB. On the other hand, YCbCr and RGB spaces are used directly in various algorithms that generally are aimed at mimicking effects of image processing and analysis conducted by the human vision system (e.g., image retrieval and recognition). Checking whether by employing LDgEb the results of such algorithms will get closer to results we expect from experience with our own visual system is an interesting field of future research. Naturally, for above applications we do not need integer reversibility of the transformation, so LDgEb chroma components may be scaled down by a factor of 2 making the transformed color space components dynamic range equal to the range of untransformed components. Other potential fields of further research are: extending RDgDb to multispectral data and applying LDgEb, or it's above-mentionedvariant with scaled chroma rows, to lossy compression.

Conflict of interest statement

None.

Role of the funding source

The study sponsors were not involved in the study design, in the collection, analysis and interpretation of data, in the writing of the manuscript, or in the decision to submit the manuscript for publication.

Acknowledgments

This work was supported by BK-219/RAu2/2012 grant from the Institute of Computer Science, Silesian University of Technology and by POIG.02.03.01-24-099/13 grant: GCONiI – Upper-Silesian Center for Scientific Computations.

References

[1] W.K. Pratt, Digital Image Processing, Wiley, New York, 1978.
[2] T. Strutz, Adaptive selection of colour transformations for reversible image compression, in: Proc. of the 20th European Signal Processing Conference (EUSIPCO), 2012, pp. 1204–1208.
[3] S.K. Singh, S. Kumar, Novel adaptive color space transform and application to image compression, Signal Process.: Image Commun. 26 (10) (2011) 662–672, http://dx.doi.org/10.1016/j.image.2011.08.001.
[4] H.S. Malvar, G.J. Sullivan, S. Srinivasan, Lifting-based reversible color transformations for image compression, Proc. SPIE 7073 (2008) 707307, http://dx.doi.org/10.1117/12.797091.
[5] ISO/IEC and ITU-T, Information Technology – JPEG2000 image coding system: core coding system, ISO/IEC International Standard 15444-1 and ITU-T recommendation T.800, 2004.
[6] K.R. Gegenfurtner, D.C. Kiper, Color vision, Annu. Rev. Neurosci. 26 (2003) 181–206, http://dx.doi.org/10.1146/annurev.neuro.26.041002.131116.
[7] M. Domański, K. Rakowski, Lossless and near-lossless image compression with color transformations, Proc. ICIP 3 (2001) (2001) 454–457, http://dx.doi.org/10.1109/ICIP.2001.958149.
[8] I. Daubechies, W. Sweldens, Factoring wavelet transforms into lifting steps, J. Fourier Anal. Appl. 4 (1998) 247–269.
[9] P. Hao, Q. Shi, Matrix factorizations for reversible integer mapping, IEEE Trans. Signal Process. 49 (10) (2001) 2314–2324, http://dx.doi.org/10.1109/78.950787.
[10] ISO/IEC and ITU-T, JPEG XR Image Coding Specification, ISO/IEC International Standard 29199-2 and ITU-T Recommendation T.832, 2009.
[11] ISO/IEC and ITU-T, Information technology – Lossless and near-lossless compression of continuous-tone still images: Extensions, ISO/IEC International Standard 14495-2 and ITU-T Recommendation T.870, 2003.
[12] P. Hao, Q. Shi, Reversible integer KLT for progressive-to-lossless compression of multiple component images, in: Proc IEEE Int. Conf. Image Process, 2003, pp. 633–636.
[13] P. Janczur, Reversible PCA/KLT transformation in color image compression, M.Sc. thesis under supervision of R. Starosolski, Institute of Computer Science, Faculty of Automatic Control, Electronics and Computer Science, Silesian University of Technology, Gliwice, 2011.
[14] F. De Simone, L. Goldmann, V. Baroncini, T. Ebrahimi, Subjective evaluation of JPEG XR image compression, Proc. SPIE 7443 (2009) 74430L, http://dx.doi.org/10.1117/12.830714.
[15] ISO/IEC and ITU-T, Information technology – Lossless and near-lossless compression of continuous-tone still images – Baseline, ISO/IEC International Standard 14495-1 and ITU-T Recommendation T.87, 2000.
[16] X. Wu, N. Memon, Context-based lossless interband compression – extending CALIC, IEEE Trans. Image Process. 9 (6) (2000) 994–1001.
[17] R. Barequet, M. Feder, SICLIC: a simple inter-color lossless image coder, in: Proc. Data Compression Conference DCC'99, 1999, pp. 501–510.
[18] X. Chen, N. Canagarajah, J.L. Nunez-Yanez, Lossless multi-mode interband image compression and its hardware architecture. Algorithm-architecture matching for signal and image processing, Lect. Notes Electr. Eng. 73 (2011) 3–26, http://dx.doi.org/10.1007/978-90-481-9965-5_1.

Application of Reversible Denoising and Lifting Steps to LDgEb and RCT Color Space Transforms for Improved Lossless Compression

Roman Starosolski(✉)

Institute of Informatics, Silesian University of Technology,
Akademicka 16, 44-100 Gliwice, Poland
roman.starosolski@polsl.pl

Abstract. The lifting step of a reversible color space transform employed during image compression may increase the total amount of noise that has to be encoded. Previously, to alleviate this problem in the case of a simple color space transform RDgDb, we replaced transform lifting steps with reversible denoising and lifting steps (RDLS), which are lifting steps integrated with denoising filters. In this study, we apply RDLS to more complex color space transforms LDgEb and RCT and evaluate RDLS effects on bitrates of lossless JPEG-LS, JPEG 2000, and JPEG XR coding for a diverse image test-set. We find that RDLS effects differ among transforms, yet are similar for different algorithms; for the employed denoising filter selection method, on average the bitrate improvements of RDLS-modified LDgEb and RCT are not as high as of the simpler transform. The RDLS applicability reaches beyond image data storage; due to its general nature it may be exploited in other lifting-based transforms, e.g., during the image analysis for data mining.

Keywords: Image processing · Lossless image compression · Lifting technique · Denoising · Reversible denoising and lifting step · RDgDb · LDgEb · RCT · JPEG-LS · JPEG 2000 · JPEG XR

1 Introduction

Most color image compression algorithms independently compress the image components; since components in the RGB space are correlated, the compression is performed after transforming image data to a less correlated color space. For the lossless compression, the reversible color space transforms are employed, that are built using lifting steps [6]. In [9], we noticed that such step may increase the total amount of noise that must be encoded during compression. We replaced lifting steps with reversible denoising and lifting steps (RDLS), which are lifting steps integrated with denoising filters. We applied RDLS to a simple RDgDb [11] transform (also known as $A_{2,1}$ [14]) and found that RDLS improved bitrates of images in optical resolutions of acquisition devices. Experiments were performed for 3 significantly different standard image compression algorithms in lossless

S. Kozielski et al. (Eds.): BDAS 2016, CCIS 613, pp. 623–632, 2016.
DOI: 10.1007/978-3-319-34099-9_48

mode (LOCO-I/JPEG-LS[1] [16], JPEG 2000[2] [15], and HD-Photo/JPEG XR[3] [4,8]) and for a simple denoising filter. We found that the memoryless entropy of the component prediction error obtained with the nonlinear edge-detecting predictor MED [7,16] was a very efficient estimator of image component transform effects, suitable for selecting a filter for given image component. In this study, we apply RDLS to more complex color space transforms LDgEb [11] (denoted $A_{4,10}$ in [14]) and RCT (among others, used in JPEG 2000 standard) and evaluate RDLS effects using the same denoising filters, compression algorithms, and test images, as previously. Entropy estimation employing MED is used for selecting the denoising filter and deciding whether to exploit denoising.

The remainder of this paper is organized as follows. In Sect. 2 we briefly characterize RDLS, present the proposed RDLS-modified transforms along with the filter selection method and the denoising filters used, and describe the implementations and test data. Results are presented and discussed in Sect. 3; Sect. 4 summarizes the research.

2 Materials and Methods

2.1 RDLS

In [9] we proposed to replace color space transform lifting steps (Eq. 1) with RDLS (Eq. 2):

$$C_x \leftarrow C_x \oplus \mathrm{f}(C_1, \ldots, C_{x-1}, C_{x+1}, \ldots, C_n) \tag{1}$$

$$C_x \leftarrow C_x \oplus \mathrm{f}(C_1^d, \ldots, C_{x-1}^d, C_{x+1}^d, \ldots, C_n^d) \tag{2}$$

where C_i is the i-th component of the pixel, C_x is the component which is modified by the step, C_i^d is the denoised i-th component of the pixel, n is the number of components, and the operation $\oplus$ is reversible. Different denoising filters may be used for different components. For denoising of arguments of function f we may use any component of any pixel, but the C_x of the pixel to which the RDLS is being applied; in this study for denoising of C_i of specific pixel we use C_i of this pixel and of its neighbors. RDLS, like the lifting step it originates from, is trivially and perfectly invertible and may be computed in-place. However note, that denoising exploited in RDLS is irreversible and it is not an in-place operation—computing the function f argument C_i^d does not alter C_i. For more detailed characteristics of RDLS and examples of its application we refer the Reader to [9,12] and to the next subsection.

[1] Information technology–Lossless and near-lossless compression of continuous-tone still images–Baseline, ISO/IEC International Standard 14495-1 and ITU-T Recommendation T.87 (2006).

[2] Information technology–JPEG 2000 image coding system: Core coding system, ISO/IEC International Standard 15444-1 and ITU-T Recommendation T.800 (2004).

[3] Information technology–JPEG XR image coding system–Image coding specification, ISO/IEC International Standard 29199-2 and ITU-T Recommendation T.832 (2012).

2.2 RDLS-Modified Transforms

Here we show the application of RDLS to RDgDb and, for brevity, present only the RDLS-modified variants of LDgEb and RCT. Equation 3 shows the RDgDb transform, as it was presented in [11], where the definitions of other unmodified transforms may also be found. To obtain RDLS-modified RDgDb (RDLS-RDgDb) we first rewrite RDgDb using the notation as in Eqs. 1 and 2, where the same symbol denotes the pixel's component both before and after modifying its value by the lifting step or by the RDLS. Equation 4 persents RDgDb (both forward [left-hand side] and inverse) as a sequence of lifting steps; generally, the steps must be performed in a specified order. C_1, C_2, and C_3 denote R, G, and B components of the untransformed image, respectively, and R, Dg, and Db components of the transformed image, respectively. Next, we simply replace lifting steps (Eq. 1) in Eq. 4 with RDLS (Eq. 2) constructed based on them and obtain RDLS-RDgDb (Eq. 5). The employed denoising filters and method of selecting the filter for a given image component is described in the following subsection.

$$\begin{aligned} R &= R \\ Dg &= R - G \\ Db &= G - B \end{aligned} \iff \begin{aligned} R &= R \\ G &= R - Dg \\ B &= G - Db \end{aligned} \tag{3}$$

$$\begin{aligned} &\text{step 1: } C_3 \leftarrow -C_3 + C_2 \\ &\text{step 2: } C_2 \leftarrow -C_2 + C_1 \\ &\text{step 3: } C_1 \leftarrow C_1 \end{aligned} \iff \begin{aligned} &\text{step 1: } C_1 \leftarrow C_1 \\ &\text{step 2: } C_2 \leftarrow -C_2 + C_1 \\ &\text{step 3: } C_3 \leftarrow -C_3 + C_2 \end{aligned} \tag{4}$$

$$\begin{aligned} &\text{step 1: } C_3 \leftarrow -C_3 + C_2^d \\ &\text{step 2: } C_2 \leftarrow -C_2 + C_1^d \\ &\text{step 3: } C_1 \leftarrow C_1 \end{aligned} \iff \begin{aligned} &\text{step 1: } C_1 \leftarrow C_1 \\ &\text{step 2: } C_2 \leftarrow -C_2 + C_1^d \\ &\text{step 3: } C_3 \leftarrow -C_3 + C_2^d \end{aligned} \tag{5}$$

The RDLS-RDgDb transform (Eq. 5) may be presented using component symbols like in standard definitions of color space transforms (Eq. 6). Note, that the same symbols denote components of regular transform and its RDLS-modified variant.

$$\begin{aligned} &\text{step 1: } Db = -B + G^d \\ &\text{step 2: } Dg = -G + R^d \\ &\text{step 3: } R = R \end{aligned} \iff \begin{aligned} &\text{step 1: } R = R \\ &\text{step 2: } G = -Dg + R^d \\ &\text{step 3: } B = -Db + G^d \end{aligned} \tag{6}$$

In Eq. 7 we present the RDLS-LDgEb transform and in Eq. 8 the RDLS-RCT transform, that were obtained from LDgEb and RCT, respectively. The floor of division by integer power of 2 is computed using the arithmetic right shift.

$$\begin{aligned} &\text{step 1: } Dg = -G + R^d \\ &\text{step 2: } L = R - \lfloor Dg^d/2 \rfloor \\ &\text{step 3: } Eb = B - L^d \end{aligned} \iff \begin{aligned} &\text{step 1: } B = Eb + L^d \\ &\text{step 2: } R = L + \lfloor Dg^d/2 \rfloor \\ &\text{step 3: } G = -Dg + R^d \end{aligned} \tag{7}$$

$$\begin{array}{ll} \text{step 1:} & Cv = R - G^d \\ \text{step 2:} & Cu = B - G^d \\ \text{step 3:} & Y = G + \lfloor (Cv^d + Cu^d)/4 \rfloor \end{array} \iff \begin{array}{ll} \text{step 1:} & G = Y - \lfloor (Cv^d + Cu^d)/4 \rfloor \\ \text{step 2:} & B = Cu + G^d \\ \text{step 3:} & R = Cv + G^d \end{array} \tag{8}$$

It is worth noting, that components of RDLS-LDgEb and RDLS-RCT, as opposed to RDLS-RDgDb, may require greater bit depths, than their non-RDLS equivalents. In research reported herein, we used increased component depth only when component pixels actually exceeded original depth; in all such cases extending the depth by 1 bit was sufficient.

2.3 Denoising Filters and Filter Selection

For denoising we employed 11 low-pass linear averaging filters (smoothing filters) with 3×3 pixel windows. The filtered pixel component C_i^d was calculated as a weighted arithmetic mean of the C_i components of pixels from the window. The weight of the window center point was different for different filters—from 1 to 1024 (integer powers of 2 only), while its neighbors' weights were fixed to 1.

In each RDLS-modified forward transform step s for all pixels of a given image a filter was selected individually for each component requiring denoising. E.g., in step 3 of forward RDLS-RCT (Eq. 8), 2 filters were selected for denoising of Cv and Cu and applied to all image pixels. All filter combinations were tested and we also allowed to not use the filtering. The combination resulting in the best estimated bitrate of the component being modified by the step s was used for actual compression. Thus we performed an exhaustive search of filters in a given step, however only step 3 of RDLS-RCT requires denoising of 2 components. Filter(s) selection for a given step was not revised based on compression effects of components transformed in other steps, therefore even assuming the perfect estimation of compression algorithm bitrate, modifying this way transform with RDLS may result in bitrate worsening. Filter selection must be passed to the decoder along with the compressed data, but its cost is negligible.

The memoryless entropy of the component prediction error obtained using MED predictor was used as an estimator of compressed component bitrate, overall 3-component image bitrate was estimated as sum of computed this way entropies of 3 components. We denote this estimation method as H0_pMED. The compression ratio or bitrate r, expressed in bits per pixel (bpp), is calculated as $r = 8l/m$, where m is the number of pixels in the image component, l is the length in Bytes of the file containing the compressed component, including compressed file header; smaller r denotes better compression. The memoryless entropy of the image component $H_0 = -\sum_{i=0}^{N-1} p_i \log_2 p_i$, where N is the alphabet size and p_i is the probability of occurrence of pixel component value i in the image component, is also expressed in bpp.

2.4 Implementations and Test Data Used

We used the following sets of images:

- Waterloo—Set ("Colour set") of images from the University of Waterloo[4];
- Kodak—Image set from the Kodak corporation[5];
- EPFL—Image set from the École polytechnique fédérale de Lausanne[6] [3];
- A1, A2, and A3—Image sets from the Silesian University of Technology[7];
- A1-red.3, A2-red.3, and A3-red.3—reduced size (3×) sets A1, A2, and A3.

Sets A1, A2, and A3 contain unprocessed photographic images in optical resolutions of acquisition devices, or (A3) as close to such resolution, as possible without interpolation of all components. Except for Waterloo, all images may be characterized as continuous-tone photographic. The most widely-known Waterloo set contains both photographic and artificial images; some of them are dithered, sharpened, have sparse histograms of intensity levels [10], are computer-generated or composed of others. The same image sets were used for experiments in the previous study, their detailed characteristics may be found in [9].

We used the Signal Processing and Multimedia Group, Univ. of British Columbia JPEG-LS implementation, version 2.2[8], JasPer implementation of JPEG 2000 by M. Adams, version 1.900[9] [1], and JPEG XR standard reference software[10].

3 Results and Discussion

In Tables 1, 2, and 3, for RDLS-RDgDb, RDLS-LDgEb, and RDLS-RCT, respectively, we report average entropies obtained using H0_pMED estimator and average bitrates for JPEG-LS, JPEG 2000, and JPEG XR compression algorithms in lossless mode, summed for all 3 image components. Entropy and bitrate changes due to RDLS with respect to non-RDLS transform variants are also reported—in columns labeled ΔH_0 for H0_pMED and Δr for compression algorithms. Figure 1 for the examined transforms presents average entropy and bitrate changes due to RDLS of individual transformed components and of the 3-component image; in electronic version of the paper components are presented using colors of RGB components from which they were transformed.

We examined RDLS effects for several transforms, compression algorithms, and test image sets. In majority of cases, the RDLS-RDgDb obtains the best bitrates among all RDLS-modified transforms. For all these transforms, the

[4] http://links.uwaterloo.ca/Repository.html.
[5] http://www.cipr.rpi.edu/resource/stills/kodak.html.
[6] http://documents.epfl.ch/groups/g/gr/gr-eb-unit/www/IQA/Original.zip.
[7] http://sun.aei.polsl.pl/~rstaros/optres/.
[8] http://www.stat.columbia.edu/~jakulin/jpeg-ls/mirror.htm.
[9] http://www.ece.uvic.ca/~mdadams/jasper/.
[10] Information technology–JPEG XR image coding system–Reference software, ISO/IEC International Standard 29199-5 and ITU-T Recommendation T.835 (2012).

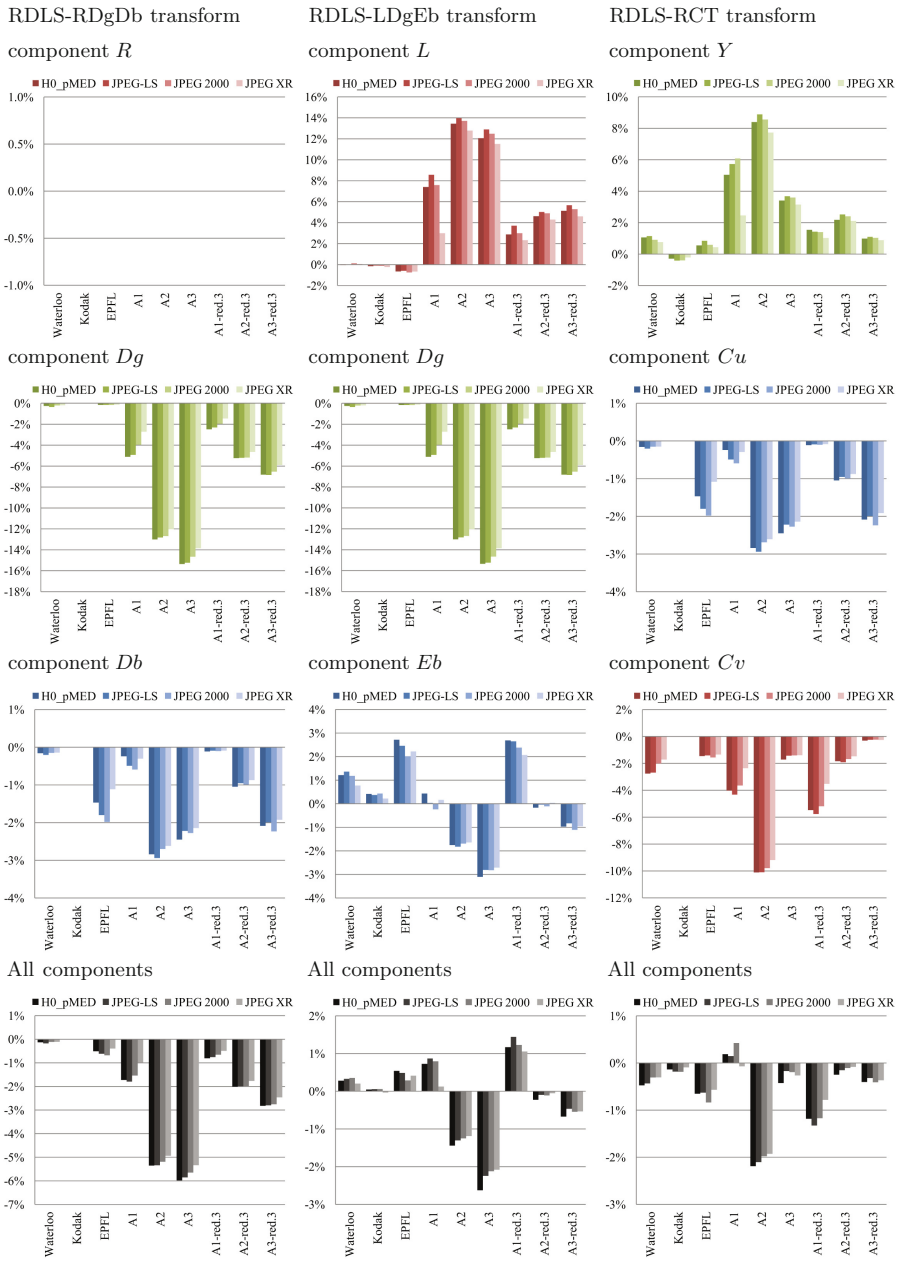

Fig. 1. Average entropy and bitrate changes due to RDLS for RDgDb (left-hand panels), LDgEb (middle panels), and RCT (right-hand panels) (Colour figure online).

Table 1. Effects of the RDLS-RDgDb transform and comparison to RDgDb

Images	H0_pMED		JPEG-LS		JPEG 2000		JPEG XR	
	H_0	ΔH_0	r	Δr	r	Δr	r	Δr
Waterloo	8.9014	−0.13 %	8.8498	−0.18 %	11.1299	−0.12 %	13.2834	−0.11 %
Kodak	10.2876	0.00 %	9.5676	0.00 %	9.4755	0.00 %	10.8725	0.00 %
EPFL	11.1806	−0.51 %	10.2746	−0.62 %	10.6515	−0.68 %	11.6264	−0.39 %
A1	6.7779	−1.73 %	6.2722	−1.80 %	6.2938	−1.54 %	8.1445	−0.99 %
A2	14.6469	−5.35 %	14.0353	−5.34 %	14.2550	−5.19 %	14.6526	−4.94 %
A3	15.1662	−5.98 %	14.5854	−5.85 %	14.7919	−5.65 %	15.3186	−5.34 %
A1-red.3	9.0112	−0.81 %	8.2770	−0.76 %	8.4625	−0.65 %	9.5931	−0.49 %
A2-red.3	14.0530	−2.01 %	13.4235	−1.99 %	13.8597	−1.98 %	14.5830	−1.77 %
A3-red.3	13.3956	−2.82 %	12.7130	−2.80 %	13.0802	−2.76 %	14.0082	−2.46 %

Table 2. Effects of the RDLS-LDgEb transform and comparison to LDgEb

Images	H0_pMED		JPEG-LS		JPEG 2000		JPEG XR	
	H_0	ΔH_0	r	Δr	r	Δr	r	Δr
Waterloo	9.0756	0.28 %	8.9888	0.33 %	11.2588	0.35 %	13.3249	0.21 %
Kodak	10.1512	0.05 %	9.4370	0.06 %	9.4282	0.06 %	10.8499	−0.04 %
EPFL	11.3739	0.54 %	10.4858	0.49 %	10.8297	0.29 %	11.7821	0.41 %
A1	6.9416	0.73 %	6.4348	0.87 %	6.4421	0.80 %	8.2329	0.13 %
A2	14.6249	−1.44 %	14.0454	−1.30 %	14.2524	−1.25 %	14.6502	−1.19 %
A3	15.3025	−2.63 %	14.7217	−2.24 %	14.9125	−2.12 %	15.4212	−2.08 %
A1-red.3	9.2698	1.17 %	8.5435	1.44 %	8.7113	1.23 %	9.8088	1.05 %
A2-red.3	14.1779	−0.23 %	13.5386	−0.10 %	13.9815	−0.11 %	14.7211	−0.05 %
A3-red.3	13.6814	−0.67 %	12.9811	−0.46 %	13.3597	−0.54 %	14.2952	−0.53 %

Table 3. Effects of the RDLS-RCT transform and comparison to RCT

Images	H0_pMED		JPEG-LS		JPEG 2000		JPEG XR	
	H_0	ΔH_0	r	Δr	r	Δr	r	Δr
Waterloo	8.9756	−0.48 %	8.9236	−0.43 %	11.1792	−0.31 %	13.2790	−0.31 %
Kodak	10.2758	−0.14 %	9.5577	−0.18 %	9.4893	−0.19 %	10.9110	−0.09 %
EPFL	11.2681	−0.65 %	10.3995	−0.63 %	10.7430	−0.84 %	11.6942	−0.57 %
A1	6.9397	0.19 %	6.4536	0.15 %	6.4821	0.42 %	8.2503	−0.07 %
A2	14.6337	−2.19 %	14.0611	−2.10 %	14.2587	−1.98 %	14.6532	−1.93 %
A3	15.2361	−0.43 %	14.6042	−0.17 %	14.7557	−0.19 %	15.2918	−0.27 %
A1-red.3	9.1002	−1.18 %	8.4063	−1.33 %	8.5820	−1.17 %	9.6624	−0.78 %
A2-red.3	14.0771	−0.25 %	13.4646	−0.15 %	13.8917	−0.11 %	14.6175	−0.08 %
A3-red.3	13.2992	−0.40 %	12.5899	−0.32 %	12.9520	−0.41 %	13.9179	−0.37 %

obtained bitrates significantly differ among the compression algorithms. Constantly, except for the Kodak set, JPEG-LS obtains the best bitrates and JPEG-XR the worst; for Kodak the JPEG 2000 is the best. However, the bitrate improvements due to RDLS are similar for different compression algorithms and entropy estimated bitrate.

On average, RDLS improves bitrates of all the examined transforms. The bitrate improvements of RDLS-modified LDgEb and RCT are not as high as in the case of the simpler RDgDb transform and for the former transforms RDLS sometimes results in bitrate worsening of 3-component image and, to a greater extent, of individual components. For these transforms we did not find any objective image feature allowing to predict RDLS effectiveness—as opposed to RDgDb, that is the most effective for images in optical resolutions of acquisition devices.

LDgEb and RCT transforms contain steps, during which a given component C_i is modified based on another one C_a which has already been modified based on C_i (see step 2 of LDgEb and step 3 of RCT). For example, let's look at steps 1 and 2 of forward LDgEb (its RDLS version is presented in Eq. 7). Step 1 modifies the G component, that from this moment is denoted as Dg, step 2 modifies R, that is then denoted as L. Step 2 of the regular lifting transform may decrease in L the amount of information which was originally present in R, that is of both noise and of the noise-free signal, and insert to L a fraction of information originally present in G (also consisting of noise and noise-free signal). Actually, as the transform is reversible, the information from R is beforehand (in step 1) propagated to Dg, and then in step 2 a part of it (as a fraction of Dg) is subtracted from L. In step 1 components are subtracted—the use of the noise-free signal from R is supposed to result in decreasing the correlation between R and Dg whereas noise from R adds to noise in Dg. When we employ RDLS, step 2 applied to R reduces the noise-free signal originally contained in it, but not the noise (noise from R is not present in Dg because filtering during step 1 prohibited propagating it); on the other hand RDLS avoids transferring to L the noise originally present in G. In the case of our images the former effect, complemented by component distortions introduced by imperfect denoising we use, has greater impact on the component bitrate, than the later—see component L (and Y for RDLS-RCT) in Fig. 1. Also the Eb component bitrate is worsened for some sets, which may be attributed to using for its calculation the denoised component L, bitrates of which are for some sets worsened by RDLS.

The filter selection method we use may be responsible for worse RDLS effects in the case of more complicated transforms. Since we select the best filters for a given step by analyzing filtering effects on bitrate of the component C_i being modified by this step only, the selection may not be optimal if C_i is then used in calculation of components modified in further steps. Better bitrates of such components and of overall 3-component image could be obtained by selecting filters based on overall 3-component image bitrate. Checking all filter combinations for all the steps would be too complex in practice, but employing a heuristics that would search for the best filters in a given step based on compression effects estimated for the entire image (similarly to [12]) seems worthwhile. We also note, that the selection process

complexity might be substantially reduced by using only a certain number of image pixels for transform effect estimation—in [13,14] such optimization applied to a similar problem resulted in a close to optimum estimation.

4 Conclusions

RDLS effects differ among transforms, yet are similar for JPEG-LS, JPEG 2000, and JPEG XR algorithms as well as for entropy of the component prediction errors obtained using MED. On average, RDLS improves bitrates of all the examined transforms. The bitrate improvements of RDLS-modified LDgEb and RCT are not as high as in the case of the simpler RDgDb transform and for the former transforms RDLS sometimes results in bitrate worsening; both effects may be attributed to the employed method of selecting the denoising filters. As opposed to RDgDb, we did not identify an objective image feature to which the RDLS bitrate improvement could be linked. As the RDLS effects clearly depend on denoising filters used, we expect that the application of better filters may further improve bitrates of the RDLS-modified color space transforms. For some images the denoising filter parameters might be determined directly from the acquisition process parameters [2]. The bitrate improvements we obtained for some of the test-sets are useful from practical standpoint. In the ongoing research, we investigate other filters and filter selection methods as well as simplified compression effect estimators, that jointly are expected to result in greater bitrate improvements obtained at a significantly lower filter selection cost. RDLS recently was found effective for a much more complex, involving more interdependent steps, multi-level 2D DWT transform in lossless JPEG 2000 compression [12], however, its applicability reaches beyond image data storage. Due to its general nature it may be exploited in other lifting-based transforms, e.g., during the image analysis for data mining and skin segmentation [5].

Acknowledgments. This work was supported by BK-263/RAU2/2015 grant from the Institute of Informatics, Silesian University of Technology.

References

1. Adams, M.D., Ward, R.K.: JasPer: a portable flexible open-source software tool kit for image coding/processing. In: 2004 Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2004), vol. 5, pp. 241–244 (2004). doi:10.1109/ICASSP.2004.1327092
2. Bernas, T., Starosolski, R., Robinson, J.P., Rajwa, B.: Application of detector precision characteristics and histogram packing for compression of biological fluorescence micrographs. Comput. Methods Programs Biomed. **108**(2), 511–523 (2012). doi:10.1016/j.cmpb.2011.03.012
3. De Simone, F., Goldmann, L., Baroncini, V., Ebrahimi, T.: Subjective evaluation of JPEG XR image compression. In: Proceedings of the SPIE, Applications of Digital Image Processing XXXII, vol. 7443, p. 74430L (2009). doi:10.1117/12.830714

4. Dufaux, F., Sullivan, G.J., Ebrahimi, T.: The JPEG XR image coding standard. IEEE Sig. Process. Mag. **26**(6), 195–199, 204 (2009). doi:10.1109/MSP.2009.934187
5. Kawulok, M., Kawulok, J., Nalepa, J.: Spatial-based skin detection using discriminative skin-presence features. Pattern Recogn. Lett. **41**, 3–13 (2014). doi:10.1016/j.patrec.2013.08.028
6. Malvar, H.S., Sullivan, G.J., Srinivasan, S.: Lifting-based reversible color transformations for image compression. In: Proceedings of the SPIE, Applications of Digital Image Processing XXXI, vol. 7073, p. 707307 (2008). doi:10.1117/12.797091
7. Martucci, S.A.: Reversible compression of HDTV images using median adaptive prediction and arithmetic coding. In: Proceedings of the IEEE International Symposium on Circuits and Systems, pp. 1310–1313 (1990)
8. Srinivasan, S., Tu, C., Regunathan, S.L., Sullivan, G.J.: HD Photo: a new image coding technology for digital photography. In: Proceedings of the SPIE, Applications of Digital Image Processing XXX, vol. 6696, p. 66960A (2007). doi:10.1117/12.767840
9. Starosolski, R.: Reversible denoising and lifting based color component transformation for lossless image compression (2015). arXiv:1508.06106 [cs.MM]
10. Starosolski, R.: Compressing high bit depth images of sparse histograms. In: Simos, T.E., Psihoyios, G. (eds.) International Electronic Conference on Computer Science. AIP Conference Proceedings, vol. 1060, pp. 269–272. American Institute of Physics, USA (2008). doi:10.1063/1.3037069
11. Starosolski, R.: New simple and efficient color space transformations for lossless image compression. J. Vis. Commun. Image Represent. **25**(5), 1056–1063 (2014). doi:10.1016/j.jvcir.2014.03.003
12. Starosolski, R.: Application of reversible denoising and lifting steps to DWT in lossless JPEG 2000 for improved bitrates. Sig. Process. Image Commun. **39**(A), 249–263 (2015). doi:10.1016/j.image.2015.09.013
13. Strutz, T.: Adaptive selection of colour transformations for reversible image compression. In: Proceedings of the 20th European Signal Processing Conference (EUSIPCO 2012), pp. 1204–1208 (2012)
14. Strutz, T.: Multiplierless reversible colour transforms and their automatic selection for image data compression. IEEE Trans. Circuits Syst. Video Technol. **23**(7), 1249–1259 (2013). doi:10.1109/TCSVT.2013.2242612
15. Taubman, D.S., Marcellin, M.W.: JPEG2000 Image Compression Fundamentals, Standards and Practice. The Springer International Series in Engineering and Computer Science, vol. 642. Springer, New York (2004). doi:10.1007/978-1-4615-0799-4
16. Weinberger, M.J., Seroussi, G., Sapiro, G.: The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS. IEEE Trans. Image Process. **9**(8), 1309–1324 (2000). doi:10.1109/83.855427

Journal of Electronic Imaging

JElectronicImaging.org

Application of reversible denoising and lifting steps with step skipping to color space transforms for improved lossless compression

Roman Starosolski

SPIE. IS&T imaging.org

Roman Starosolski, "Application of reversible denoising and lifting steps with step skipping to color space transforms for improved lossless compression," *J. Electron. Imaging* **25**(4), 043025 (2016), doi: 10.1117/1.JEI.25.4.043025.

Application of reversible denoising and lifting steps with step skipping to color space transforms for improved lossless compression

Roman Starosolski*
Silesian University of Technology, Institute of Informatics, Akademicka 16, 44-100 Gliwice, Poland

Abstract. Reversible denoising and lifting steps (RDLS) are lifting steps integrated with denoising filters in such a way that, despite the inherently irreversible nature of denoising, they are perfectly reversible. We investigated the application of RDLS to reversible color space transforms: RCT, YCoCg-R, RDgDb, and LDgEb. In order to improve RDLS effects, we propose a heuristic for image-adaptive denoising filter selection, a fast estimator of the compressed image bitrate, and a special filter that may result in skipping of the steps. We analyzed the properties of the presented methods, paying special attention to their usefulness from a practical standpoint. For a diverse image test-set and lossless JPEG-LS, JPEG 2000, and JPEG XR algorithms, RDLS improves the bitrates of all the examined transforms. The most interesting results were obtained for an estimation-based heuristic filter selection out of a set of seven filters; the cost of this variant was similar to or lower than the transform cost, and it improved the average lossless JPEG 2000 bitrates by 2.65% for RDgDb and by over 1% for other transforms; bitrates of certain images were improved to a significantly greater extent. [DOI: 10.1117/1.JEI.25.4.043025]

Keywords: lossless image compression; lifting technique; reversible color space transform; denoising; reversible denoising and lifting step.

Paper 16307 received Apr. 14, 2016; accepted for publication Jul. 22, 2016; published online Aug. 12, 2016.

1 Introduction

Most color image compression algorithms independently compress the image components; since components in the RGB space are correlated, the compression is performed after transforming image data to a less correlated color space. For the lossless compression, reversible color space transforms are employed, which are built using lifting steps.[1,2] In Ref. 3, it was noticed that such a step might increase the total amount of noise that had to be encoded during compression. To remove correlation without increasing noise, lifting steps were replaced with reversible denoising and lifting steps (RDLS), which are lifting steps integrated with denoising filters. The step is modified in such a way that, despite involving the inherently irreversible denoising, it is perfectly reversible. RDLS was applied to a simple RDgDb[4] transform (also known as $A_{2,1}$[5]) and it was found that RDLS improved bitrates of images in optical resolutions of acquisition devices. Experiments were performed for three significantly different standard image compression algorithms in the lossless mode (JPEG-LS,[6] JPEG 2000,[7] and JPEG XR[8]) and for simple linear denoising filters ("smoothing" filters). The memoryless entropy of the component prediction error obtained with the MED predictor[9] was a very efficient estimator of image component transform effects that was found suitable for selecting a filter for a given image component independently of the image compression algorithm.

*Address all correspondence to: Roman Starosolski, E-mail: roman.starosolski@polsl.pl

An intermediate stage of the research reported herein was presented in Ref. 10, where RDLS was applied to more complex color space transforms LDgEb[4] (denoted $A_{4,10}$ in Ref. 5) and RCT (among others, used in JPEG 2000). RDLS effects were evaluated using the same denoising filters, compression algorithms, and test images, as in Ref. 3. Entropy estimation employing MED was used for selecting the denoising filter and deciding whether to exploit denoising. The selection of filters for a given transform step was based only on the estimated filtering effects on a bitrate of component modified by this step. As a result, the filtering might, for RCT and LDgEb, result in worsening of the overall image bitrate even if assuming the perfect estimation.

In Ref. 3, it was also observed that, although RDgDb or its RDLS-modified variant improves bitrates in the average case, for some color images, the best ratios were obtained when untransformed components were compressed. In Ref. 11, RDLS was applied to discrete wavelet transform (DWT) in lossless JPEG 2000 compression of grayscale images. The noise filtering was the most effective when applied only to some steps. Some images were compressed better when the DWT stage of this algorithm was skipped, although in the average case, RDLS improved bitrates. Thus, it was suspected that the optimum might be in-between skipping and applying the transform, i.e., that better bitrates may be obtained by skipping only some of the steps of the transform.

The new contributions of this study are mainly motivated by conclusions from earlier works and are aimed at properties of RDLS-modified color space transforms that are worthwhile from a practical standpoint. Since sometimes it is better to compress an untransformed image, we propose

employing a special filter, named "null," which may make the RDLS-modified color space transform skip all or some of its steps. We also propose an image-adaptive denoising filter selection heuristic that, as compared to the heuristic exploited in Ref. 10, avoids worsening of the image bitrate and for more complex transforms is faster. As the heuristic cost (i.e., its computational time complexity) may still be too high for practical applications, we propose the limited complexity estimator of compression effects H0_pMED (10k:100), which is based on a fast estimator from Ref. 5. The cost of the latter, however, significantly increased if RDLS was employed. The evaluation of the effects of the above contributions is performed using the same transforms, compression algorithms, and test images, as in Ref. 10. Additionally, we apply RDLS to the YCoCg-R transform,[1] test the heuristic against an exhaustive filter search, and the estimators against the actual bitrates of lossless image compression algorithms.

The remainder of this paper is organized as follows. In Sec. 2, first, the reversible color space transforms and the RDLS method are briefly described. Next, in Sec. 2.3, we present the RDLS-modified transforms including the new RDLS-YCoCg-R and compare their dynamic ranges and bit-depths to the non-RDLS counterparts. In Sec. 2.4, we demonstrate the transform reversibility in a step-by-step example and present sample effects that RDLS may have on the transformed components of a noisy image. In Sec. 2.5, we describe the denoising filters used in the research including the proposed null filter. Then we introduce the filter selection heuristic (Sec. 2.6) along with the compression effect estimators including the proposed H0_pMED(10k:100) (Sec. 2.7) and report their complexities for various transforms. We also describe the compression algorithms, implementations, and test data (Sec. 2.8). Results are presented and discussed in Sec. 3. We start from analyzing the effects of contributions that are aimed at the bitrate improvement (i.e., the new heuristic and the null filter) and comparing them to the previously known methods. Next (Sec. 3.2), we reduce the cost of the bitrate improvement by limiting the number of denoising filters and iterations of the heuristic and by applying H0_pMED (10k:100). We also perform certain additional experiments (Sec. 3.3), among others, to check how far from optimal are our heuristic and estimation method. Finally, the extensive summary is followed by a brief conclusion.

2 Materials and Methods

2.1 *Lifting-Based Reversible Color Space Transforms*

Reversible color space transforms investigated in this study are sequences of lifting steps. Below, we characterize them briefly and refer the reader to Refs. 1, 4, and 5 for more detailed descriptions and comparisons among them. In each lifting step of a transform, a single pixel component is modified by adding to it a linear combination of other components of the same pixel; the sum may be negated. Up to three steps are needed to transform an RGB pixel to any of the spaces discussed in this section. A transform realized as a sequence of lifting steps has advantageous properties: it may be computed in-place, it is reversible when transformed components are stored using integers (it is integer-reversible), and it is easily and perfectly invertible. To obtain an inverse transform, the inverses of lifting steps should be applied in an order exactly opposite to the order employed by the forward transform. However, transforms are usually presented using different symbols for components before and after applying lifting steps to them, lifting equations are simplified, or a lifting sequence is transformed in order to present an optimized method of transform implementation. Thus, it may not be obvious which component is modified in a given step or how to inverse the step. In this section, we follow the usual way of presentation; for transforms presented as sequences of lifting steps, see Sec. 2.3.

Probably the most frequently used reversible color space transform is the RCT transform employed in JPEG 2000 for lossless compression, which is an approximation of an irreversible ICT transform used in JPEG 2000 for lossy compression, that in turn may be seen as an approximation of an irreversible YCbCr color space transform.[7] Equation (1) presents forward (left-hand side) and inverse RCT:

$$\begin{aligned} Yr &= \lfloor (R+2G+B)/4 \rfloor \\ Ur &= R-G \\ Vr &= B-G \end{aligned} \quad \Leftrightarrow \quad \begin{aligned} G &= Yr - \lfloor (Ur+Vr)/4 \rfloor \\ R &= Ur+G \\ B &= Vr+G \end{aligned}, \tag{1}$$

where the $\lfloor i/2^q \rfloor$ is the floor of $i/2^q$; i.e., the greatest integer not exceeding $i/2^q$, that for integer i and positive integer q may be simply computed as the arithmetic right shift of i by q bits. RCT transforms RGB primary color components R, G, and B to component Yr representing pixel luminance and two chrominance components Ur and Vr. Note that compared to primary color and luminance components, the dynamic range of chrominance components and consequently their bit-depths are greater—which is also true for other transforms presented in this section.

Another standard transform, among others used in JPEG XR, is YCoCg-R. In Eq. (2), it is presented as it was originally proposed in Ref. 12, i.e., as a sequence of steps involving storing an intermediate result in a temporary variable t; Y represents pixel luminance, whereas Co and Cg are chrominance components:

$$\begin{aligned} Co &= R-B \\ t &= B+\lfloor Co/2 \rfloor \\ Cg &= G-t \\ Y &= t+\lfloor Cg/2 \rfloor \end{aligned} \quad \Leftrightarrow \quad \begin{aligned} t &= Y-\lfloor Cg/2 \rfloor \\ G &= Cg+t \\ B &= t-\lfloor Co/2 \rfloor \\ R &= B+Co \end{aligned}. \tag{2}$$

In Ref. 3, RDLS was applied to the RDgDb transform [Eq. (3)],[4] which is also known as $A_{2,1}$.[5] RDgDb was chosen because of its simplicity and good performance. It requires only two simple integer operations (add or subtract) per pixel, which for the three-component RGB color space is possible because we do not transform all the components. There are two transformed chrominance components Dg and Db, but instead of luminance, the untransformed primary color R is used:

$$\begin{aligned} R &= R \\ Dg &= R-G \\ Db &= G-B \end{aligned} \quad \Leftrightarrow \quad \begin{aligned} R &= R \\ G &= R-Dg \\ B &= G-Db \end{aligned}. \tag{3}$$

In this research, we also include the LDgEb transform [Eq. (4)][4] (denoted $A_{4,10}$ in Ref. 5). Like typical transforms,

it results in a luminance (L) and two chrominance (Dg and Eb) components; interestingly, the component formulas are based on actual analog transforms from the human visual system:

$$\begin{array}{l} Dg = R - G \\ L = R - \lfloor Dg/2 \rfloor \\ Eb = B - L \end{array} \quad \Leftrightarrow \quad \begin{array}{l} R = L + \lfloor Dg/2 \rfloor \\ G = R - Dg \\ B = Eb + L \end{array} . \tag{4}$$

2.2 *Reversible Denoising and Lifting Step*

A lifting step in a reversible color space transform may propagate the noise to the component it modifies from other components. In Ref. 3, integrating denoising into lifting steps was proposed in order to avoid noise propagation while preserving other transform properties (i.e., reversibility, in-place operation, and removing correlation). The method was based on the generalized lifting step of a color space transform:

$$C_x \leftarrow C_x \oplus \mathrm{f}(C_1, \ldots, C_{x-1}, C_{x+1}, \ldots, C_m), \tag{5}$$

where C_n is the n'th component of the pixel, C_x is the component which is modified by the step, m is the number of components, f is a deterministic function, and the operation $\oplus$ is reversible, i.e., an inverse operation $\otimes$ exists, such that $c = a \oplus b \Leftrightarrow a = c \otimes b$.

By denoising of arguments of function f in the generalized lifting step [Eq. (5)], a reversible denoising and lifting step [RDLS, Eq. (6)] was constructed:

$$C_x \leftarrow C_x \oplus \mathrm{f}(C_1^d, \ldots, C_{x-1}^d, C_{x+1}^d, \ldots, C_m^d), \tag{6}$$

where C_n^d is the denoised n'th component of the pixel. Different denoising filters may be employed for different components and in different steps. For denoising of arguments of function f, any component of any pixel may be used except the C_x of the pixel to which the RDLS is being applied. Denoising is not an in-place operation, computing the function f argument C_n^d does not alter C_n. In this study, for denoising of C_n of a specific pixel, we use C_n of this pixel and of its neighbors.

Despite the inherently lossy nature of denoising, RDLS exploiting denoising is perfectly and easily invertible. An inverse of an RDLS-modified color space transform is obtained by applying inverses of RDLS:

$$C_x \leftarrow C_x \otimes \mathrm{f}(C_1^d, \ldots, C_{x-1}^d, C_{x+1}^d, \ldots, C_m^d), \tag{7}$$

in an order opposite to one employed by the forward transform—see examples in the following two sections. Naturally, the same denoising filters must be used for the same components in inverse RDLS, as they were applied during forward RDLS.

2.3 *Reversible Denoising and Lifting Steps-Modified Transforms*

In Eq. (8), the RCT transform is defined as a sequence of lifting steps—both the forward RCT transform (left-hand side) and inverse:

$$\begin{array}{ll} \text{step 1:} & C_1 \leftarrow C_1 - C_2 \\ \text{step 2:} & C_3 \leftarrow C_3 - C_2 \\ \text{step 3:} & C_2 \leftarrow C_2 + \lfloor (C_1 + C_3)/4 \rfloor \end{array} \quad \Leftrightarrow \quad \begin{array}{ll} \text{step 1:} & C_2 \leftarrow C_2 - \lfloor (C_1 + C_3)/4 \rfloor \\ \text{step 2:} & C_3 \leftarrow C_3 + C_2 \\ \text{step 3:} & C_1 \leftarrow C_1 + C_2 \end{array} . \tag{8}$$

We use the notation as in Eqs. (5)–(7), where the same symbol denotes the pixel's component before and after modifying it by the lifting step or the RDLS. For all the transforms presented in this section, C_1, C_2, and C_3 denote the R, G, and B components of the untransformed image, respectively. For RCT, the C_1, C_2, and C_3 denote also the Ur, Yr, and Vr components of the transformed image, respectively. Generally, the steps must be performed in a specified order.

The RDLS-modified RCT [RDLS-RCT, Eq. (9)] is obtained by simply replacing the RCT [Eq. (8)] lifting steps [Eq. (5)] with the RDLS [Eq. (6)] constructed based on them:

$$\begin{array}{ll} \text{step 1:} & C_1 \leftarrow C_1 - C_2^d \\ \text{step 2:} & C_3 \leftarrow C_3 - C_2^d \\ \text{step 3:} & C_2 \leftarrow C_2 + \lfloor (C_1^d + C_3^d)/4 \rfloor \end{array} \quad \Leftrightarrow \quad \begin{array}{ll} \text{step 1:} & C_2 \leftarrow C_2 - \lfloor (C_1^d + C_3^d)/4 \rfloor \\ \text{step 2:} & C_3 \leftarrow C_3 + C_2^d \\ \text{step 3:} & C_1 \leftarrow C_1 + C_2^d \end{array} . \tag{9}$$

We use the same symbols to denote components of regular transforms and of their RDLS-modified counterparts; thus, C_1, C_2, and C_3 denote the Ur, Yr, and Vr components of the RDLS-RCT transformed image, respectively. The regular lifting transform is a special case of the RDLS-modified transform; the lifting transform may be obtained by using a denoising filter, for which $C_n^d = C_n$. We call such a filter the "none" filter.

The dynamic range of RDLS-RCT components differs from RCT components' range in the case of the C_2 component. We assume that denoising of the pixel component C_n may result in any integer within the dynamic range of the image component C_n. Note that the "component" term may refer to a pixel and to an image; in the latter case, the image component C_n is an image consisting of C_n components of all pixels of a color image. In RDLS-RCT, for the $[0, 2^b - 1]$ range of untransformed RGB components, the range of C_1 and C_3 before performing the forward step 3 is $[-2^b + 1, 2^b - 1]$. Due to denoising, step 3 of RDLS-RCT adds to C_2, a floor of a quarter of a sum of two values, each of which may be any integer from the $[-2^b + 1, 2^b - 1]$ range. Thus, the range of the RDLS-RCT transformed C_2 is $[-2^{b-1}, 3 \cdot 2^{b-1} - 2]$.

As noted in Ref. 3, a lifting-based color space transform may be performed for each pixel independently of others. The RDLS sequence, constructed based on a color space transform, is a transform of the whole image components.

It is not a color space transform, since denoising of a specific pixel's component C_n requires to access the C_n of (at least) neighboring pixels. The lifting-based color space transform of an image may be performed in a pixel-by-pixel regime, i.e., we apply all transform steps to a pixel, then we proceed to the next pixel, or step-by-step, i.e., we apply a lifting step to all pixels, and then we proceed to the next step; these regimes are equivalent. Also for the RDLS-modified transform, both regimes may be exploited, but they are not equivalent as filters depend on the regime. In this study, we employ the step-by-step regime. In this regime, for each image component, except for the component being modified in the current step, either all pixels are transformed or all are untransformed. Assuming that each untransformed and each transformed image component has invariant characteristics, the same filter may be selected for all image pixels in a given RDLS-modified transform step for denoising of a given component.

Presented below are the RDLS-modified variants of YCoCg-R [RDLS-YCoCg-R, Eq. (10)], RDgDb [RDLS-RDgDb, Eq. (11)], and LDgEb [RDLS-LDgEb, Eq. (12)]:

$$\begin{array}{ll} \text{step 1:} & C_1 \leftarrow C_1 - C_3^d \\ \text{step 2:} & C_3 \leftarrow -C_3 - \lfloor C_1^d/2 \rfloor + C_2^d \\ \text{step 3:} & C_2 \leftarrow C_2 - \lceil C_3^d/2 \rceil \end{array} \Leftrightarrow \begin{array}{ll} \text{step 1:} & C_2 \leftarrow C_2 + \lceil C_3^d/2 \rceil \\ \text{step 2:} & C_3 \leftarrow -C_3 - \lfloor C_1^d/2 \rfloor + C_2^d, \\ \text{step 3:} & C_1 \leftarrow C_1 + C_3^d \end{array} \tag{10}$$

$$\begin{array}{ll} \text{step 1:} & C_3 \leftarrow -C_3 + C_2^d \\ \text{step 2:} & C_2 \leftarrow -C_2 + C_1^d \\ \text{step 3:} & C_1 \leftarrow C_1 \end{array} \Leftrightarrow \begin{array}{ll} \text{step 1:} & C_1 \leftarrow C_1 \\ \text{step 2:} & C_2 \leftarrow -C_2 + C_1^d, \\ \text{step 3:} & C_3 \leftarrow -C_3 + C_2^d \end{array} \tag{11}$$

$$\begin{array}{ll} \text{step 1:} & C_2 \leftarrow -C_2 + C_1^d \\ \text{step 2:} & C_1 \leftarrow C_1 - \lfloor C_2^d/2 \rfloor \\ \text{step 3:} & C_3 \leftarrow C_3 - C_1^d \end{array} \Leftrightarrow \begin{array}{ll} \text{step 1:} & C_3 \leftarrow C_3 + C_1^d \\ \text{step 2:} & C_1 \leftarrow C_1 + \lfloor C_2^d/2 \rfloor. \\ \text{step 3:} & C_2 \leftarrow -C_2 + C_1^d \end{array} \tag{12}$$

By applying to the RDLS-modified transforms the none filter, the non-RDLS variants may be obtained in a form of sequences of lifting steps. Note that in Eq. (10), the $\lceil C_3^d/2 \rceil$ is the smallest integer greater or equal to $C_3^d/2$. Names of the transformed components are, for each transform, listed in Table 1.

Table 1 also presents dynamic ranges and bit-depths of components of all transforms investigated in this study. As opposed to RDLS-RDgDb, some components of RDLS-RCT (Yr), RDLS-YCoCg-R (Y and Cg), and RDLS-LDgEb (L and Eb) may require greater bit-depths than their non-RDLS counterparts. In such cases, the range of the non-RDLS transformed component is placed approximately in the center of the corresponding RDLS transformed component range. The dynamic range expansion with respect to the non-RDLS transform is the greatest for the RDLS-YCoCg-R transform, which needs 2 bits more than YCoCg-R for encoding the Y component.

2.4 *Example of a Reversible Denoising and Lifting Steps-Modified Color Space Transform*

In this section, using the RDLS-RCT as an example, we demonstrate how an RDLS-modified transform processes an image and how the transform reversibility is maintained despite involving the inherently irreversible denoising. Next, we present sample effects that RDLS may have on the transformed components.

The diagram in Fig. 1 presents operations performed by consecutive steps of forward and inverse RDLS-RCT. Effects of these operations on components of a sample noisy image are presented in Fig. 2; letters surrounded by dashed lines in Fig. 1 denote the panels in Fig. 2 that contain the current

Table 1 Names, dynamic ranges and bit-depths of components of lifting transforms and their RDLS counterparts; b, bit-depth of the untransformed RGB components, $b > 1$.

	C_1			C_2			C_3		
Transform	Name	range	depth	name	range	depth	name	range	depth
RGB	R	$[0, 2^b - 1]$	b	G	$[0, 2^b - 1]$	b	B	$[0, 2^b - 1]$	b
RCT	Ur	$[-2^b + 1, 2^b - 1]$	$b+1$	Yr	$[0, 2^b - 1]$	b	Vr	$[-2^b + 1, 2^b - 1]$	$b+1$
RDLS-RCT	Ur	$[-2^b + 1, 2^b - 1]$	$b+1$	Yr	$[-2^{b-1}, 3 \cdot 2^{b-1} - 2]$	$b+1$	Vr	$[-2^b + 1, 2^b - 1]$	$b+1$
YCoCg-R	Co	$[-2^b + 1, 2^b - 1]$	$b+1$	Y	$[0, 2^b - 1]$	b	Cg	$[-2^b + 1, 2^b - 1]$	$b+1$
RDLS-YCoCg-R	Co	$[-2^b + 1, 2^b - 1]$	$b+1$	Y	$[-3 \cdot 2^{b-2}, 7 \cdot 2^{b-2} - 2]$	$b+2$	Cg	$[-3 \cdot 2^{b-1} + 2, 3 \cdot 2^{b-1} - 1]$	$b+2$
RDgDb	R	$[0, 2^b - 1]$	b	Dg	$[-2^b + 1, 2^b - 1]$	$b+1$	Db	$[-2^b + 1, 2^b - 1]$	$b+1$
RDLS-RDgDb	R	$[0, 2^b - 1]$	b	Dg	$[-2^b + 1, 2^b - 1]$	$b+1$	Db	$[-2^b + 1, 2^b - 1]$	$b+1$
LDgEb	L	$[0, 2^b - 1]$	b	Dg	$[-2^b + 1, 2^b - 1]$	$b+1$	Eb	$[-2^b + 1, 2^b - 1]$	$b+1$
RDLS-LDgEb	L	$[-2^{b-1} + 1, 3 \cdot 2^{b-1} - 1]$	$b+1$	Dg	$[-2^b + 1, 2^b - 1]$	$b+1$	Eb	$[-3 \cdot 2^{b-1} + 1, 3 \cdot 2^{b-1} - 2]$	$b+2$

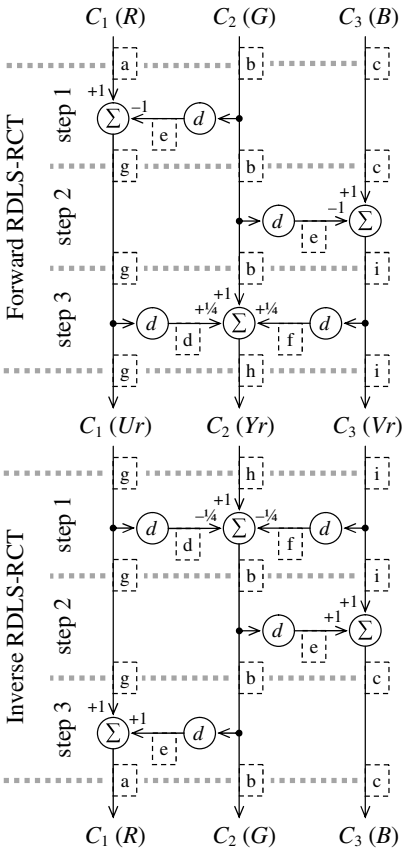

Fig. 1 Example of forward and inverse RDLS-RCT. $\sum$, weighted arithmetic mean of components; *d*, denoising of a component; dashed lines surround labels of Fig. 2 panels with the transformed component of a sample image.

appearance of the component. In this example, we use the same denoising filter in all transform steps for all components requiring denoising and employ the step-by-step regime.

Step 1 of forward RDLS-RCT applied to all image pixels transforms C_1, C_2, and C_3 components of the untransformed image (i.e., R, G, and B primary color components, respectively) by modifying only the C_1 component. The latter is modified by subtracting from it the temporarily created component obtained by denoising of the C_2 component. In step 2, we modify in an analogical way the C_3 component. Step 3 ($C_2 \leftarrow C_2 + \lfloor (C_1^d + C_3^d)/4 \rfloor$) is more complicated. For brevity in Fig. 1, we ignore computing of the floor. In this step, we add to the C_2 component the quarter of the sum of the temporary denoised components C_1 and C_3, which are obtained based on the components already transformed in earlier steps. To C_2 of each pixel, we add the quarter of the sum of the temporary denoised components C_1 and C_3 of the same pixel. After step 3, we have the RDLS-RCT transformed C_1, C_2, and C_3 components, i.e., Ur, Yr, and Vr, respectively.

Inverse transform simply applies inverses of the forward transform steps in a reversed order. Step 1 of the inverse transform ($C_2 \leftarrow C_2 - \lfloor (C_1 + C_3)/4 \rfloor$) is an inverse of the forward step 3. These two steps modify the C_2 component only. Other components are not changed by them but are used in a read-only manner to obtain temporary denoised components. Thus, components C_1 and C_3 before inverse transform step 1 are exactly the same as before forward step 3. Inverse transform step 1 subtracts from C_2 the quarter of the sum of the denoised components C_1 and C_3—subtracts from C_2 of each pixel exactly the same value that was added to it in forward step 3. The reversibility is maintained because, based on the same C_1 and C_3 as in forward step 3, we obtained the same temporary denoised components. Therefore, the denoising filter must be deterministic but does not have to be reversible, or invertible; we perform "forward" denoising in both the forward and the inverse RDLS-modified color space transform. In this example, we use the same denoising filter for all the components. However, in a general case, for denoising of a given component in the inverse of a certain forward step, we must use the same denoising filter that was used for this component in the forward step. As a result of inverse step 1, we obtain the untransformed C_2, i.e., the G primary color component, that is then used in inverse transform steps 2 and 3 to obtain the temporary denoised untransformed C_2. The latter was in forward transform subtracted from the untransformed C_1 and C_3; in inverse transform, we add it to the transformed C_3 and C_1, reconstructing the untransformed C_3 and C_1—the primary color components B and R.

In Fig. 2, we compare effects of RCT and RDLS-RCT for a noisy image. Components of an original untransformed image [Figs. 2(a)–2(c)] are contaminated with impulse noise (10% of white pixels were replaced by black ones). Impulse noise is not a typical distortion found in real-life images; we use it because it is easy to observe and, in most cases, may be efficiently removed by using a simple median denoising filter. Hence, in this example, for denoising, within all RDLS steps of RDLS-RCT we employ the median denoising filter with 3×3 pixels window. This filter may fail to remove noise from components of our image, or introduce distortions, only at the edges between areas of different brightness [in Fig. 2, compare panels (d), (e), and (f) with (g), (b), and (i), respectively]. Figure 2 also reports the component bit-depths and bitrates of a lossless image compression algorithm (estimated using the H0_pMED estimator, which is described in Sec. 2.7).

Looking at the effects of RCT [Figs. 2(j)–2(l)], we see that the transformed components contain noise from all the components used to calculate them. When we compress these components independently, then we encode the information on noise from the untransformed components twice (noise from the C_1 and C_3 components) or three times (noise from C_2). The most pronounced effect of RDLS [Figs. 2(g)–2(i)] is that the transformed components contain noise only from their untransformed counterparts.

A more subtle difference between RDLS-based and lifting-based transform effects may be noticed for components that, during transform, are modified based on themselves. For example, the component C_2 in step 3 is modified based on components C_1 and C_3 that have already been modified based on C_2 in earlier steps 1 and 2, respectively. Basically, step 3 of the lifting-based RCT transform makes C_2 contain a weighted arithmetic mean of the untransformed C_1, C_2, and C_3 components. Therefore, it decreases in C_2 the amplitude of the signal originally present in this component, that is, of both the ideal noise-free image and noise contaminating it. On the other hand, assuming the perfect denoising, the RDLS modifies a component based only on the ideal

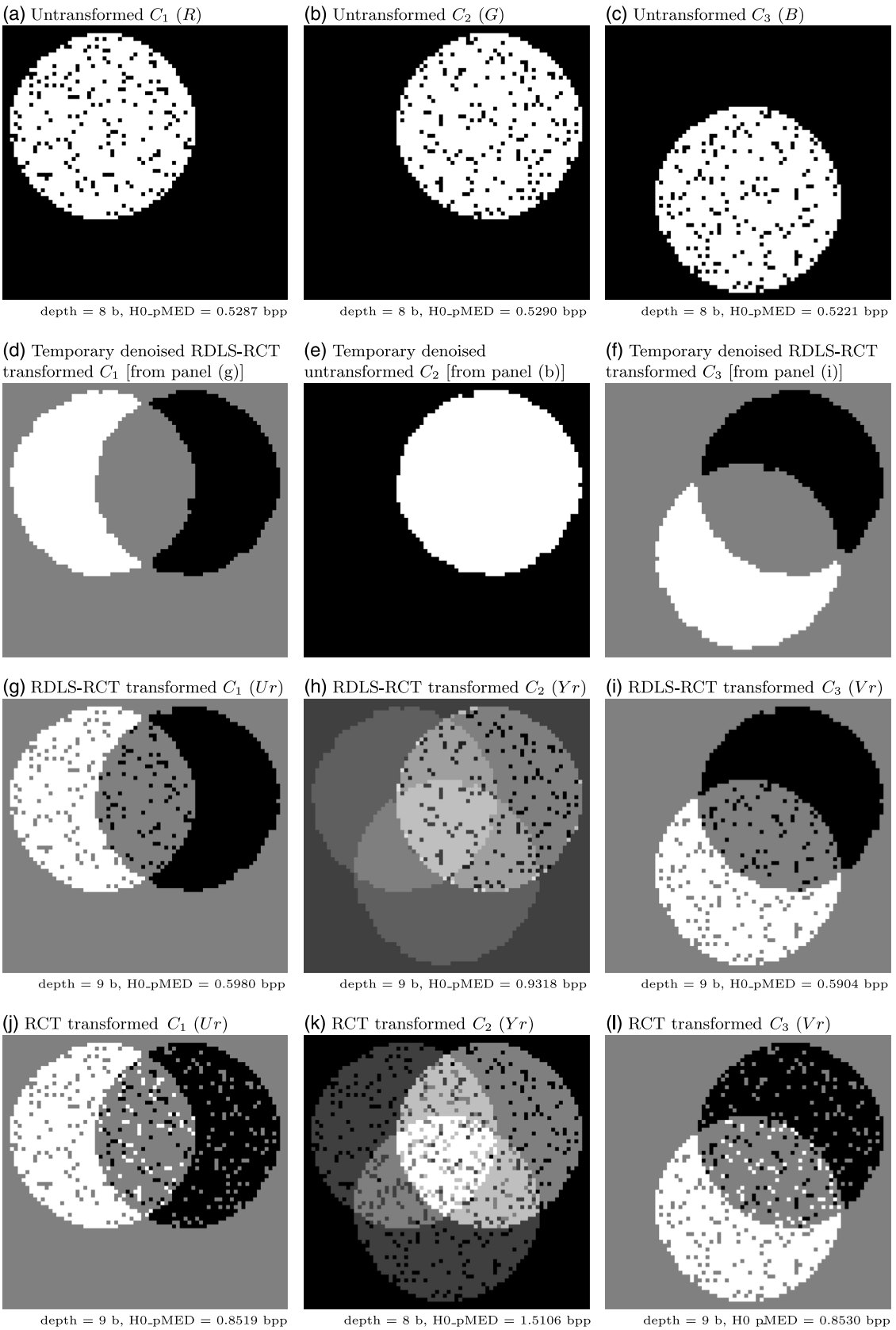

Fig. 2 Effects of RCT and RDLS-RCT on components of a noisy image. (a–c) Untransformed components of the original image, (d–f) temporary denoised components created while computing RDLS-RCT, (g–i) RDLS-RCT transformed components, and (j–l) RCT transformed components; image sizes are 74×71 pixels, transformed components are presented normalized to the dynamic range of original ones, for denoising the median filter with 3×3 pixels window was used; H0_pMED, estimated bitrate of the component (see Sec. 2.7).

noise-free images contained in other components. Thus, the ideal noise-free image is transformed differently than noise. The former is transformed as in a regular lifting-based transform, while the latter does not get propagated to other components and cannot be "weighted" using its copy from other components. The RDLS-RCT transformed C_2 contains a weighted arithmetic mean of ideal noise-free images from the C_1, C_2, and C_3 components and unmodified noise originally present in C_2. Therefore, the amplitude of noise from unmodified C_2 in transformed C_2 is greater after RDLS-RCT than after RCT. The above effect may be hard to notice on Fig. 2. The pixels affected by noise from untransformed C_2 appear similarly dark in C_2 transformed by RCT [Fig. 2(k)] and RDLS-RCT [Fig. 2(h)] because all components in Fig. 2 are presented normalized to the dynamic range of untransformed primary colors, so the actual dynamic range of C_2 in the case of RDLS-RCT was reduced twice, whereas for RCT, it was not changed.

It is worth noting that the actual denoising is not perfect, which may affect bitrates of RDLS-modified color space transforms. For our example image, the effects of the imperfect denoising we applied are rough edges between areas of different brightness (noticeable in all RDLS-RCT transformed components) and additional noisy pixels in transformed C_2 (of intensity different to pixels that were noisy in untransformed C_2).

The estimated bitrates of RDLS-RCT components are significantly lower than bitrates of RCT components. Interestingly, bitrates of untransformed components are even better, which suggests that for some images, the untransformed components should be compressed. In the next section, we propose a special case of a denoising filter for RDLS, which may result in skipping of the RDLS-modified color space transform.

2.5 *Denoising Filters*

For denoising, we employed simple low-pass linear averaging filters (smoothing filters) with 3×3 pixels windows; these filters were previously found effective for RDLS-RDgDb, RDLS-LDgEb, and RDLS-RCT.[3,10] The filtered pixel component C_n^d was calculated as a weighted arithmetic mean of the C_n components of pixels from the window. The weight of the window center point varied for different filters, while its neighbors' weights were fixed to 1. We tested 11 smoothing filters with window center point weights from 1 to 1024 (integer powers of 2 only).

The filter set contains the none filter for which $C_n^d = C_n$. The none filter turns RDLS into a regular lifting step if it is applied to all arguments of function f [see Eqs. (5) and (6)]. The heuristic we employ for an image-adaptive filter selection (see the next section) requires this filter to be present in the filter set.

We also used the null special filter case, for which $C_n^d = 0$. For the examined RDLS-modified color space transforms, the null filter may result in step skipping. If it is applied to all arguments of function f, then RDLS becomes $C_x \leftarrow C_x$ or $C_x \leftarrow -C_x$ and negating the image component does not change its entropy and has virtually no effect on its compression ratio.

It is noteworthy that the none and null filters may turn the RDLS-modified transform into a lifting transform or cause skipping it as a whole or partially—because different filters may be selected for different steps by the filter selection heuristic we employ. On the other hand, the specific RDLS may be only partially affected if its function f has more than one argument.

2.6 *Filter Selection Heuristic*

We used a simple and greedy denoising filter selection heuristic based on one applied in Ref. 11 for the RDLS-modified DWT transform. It consists of the below described steps: A and B.

A. Transform the image using the none filter in all steps for all arguments of function f. Store the estimated bitrates of the transformed components and, for each component in each step, assign the none filter.

B. In each transform step s (starting from step 1) for each function f argument C_n (analyzed in the $C_1, \ldots, C_3$ order), try to find a better denoising filter by checking for each filter (except for the one already selected), the overall estimated bitrate obtained by using in step s this filter for denoising of component C_n for all image pixels, while the filters selected so far are used for other components and in other steps.

Step B of the heuristic may be repeated for a given number of iterations. This step of the heuristic in each RDLS-modified forward transform step for each component requiring denoising greedily selects the denoising filter to be applied to all image pixels. For example, in step 3 of forward RDLS-RCT [Eq. (9)], first, a filter for C_1 is selected for denoising of all the image pixels and then a filter for C_3 is also selected for all pixels.

To obtain the estimated overall image bitrate after changing a filter, it may be sufficient to estimate bitrates of some components only—depending on the transform and the step, the changing of a denoising filter may not affect all components. Table 2 presents computational time complexities of the filter selection heuristic that take into account the above-mentioned property. Complexities are, for the investigated RDLS-modified transforms, expressed using the cost of operations on a single image component. For comparison, the complexities of compression using these transforms with the already selected filters and compression with the non-RDLS transforms are also reported. We assumed that all image pixels are used in bitrate estimation. Using for this purpose only some of the pixels allows lowering the complexity, which is discussed in the next section.

These estimations are rough, among others because the complexity of denoising may differ significantly for different filters and because different lifting steps are not equally complex (we took into account that step 3 of forward RDgDb is done at no cost and that no actual denoising is applied in step A of the heuristic). However, they allow making certain general observations. The complexity of the heuristic depends linearly on the number of iterations of step B, the number of filters, and on complexities of bitrate estimation, lifting, and denoising. It also depends on the RDLS-modified transform it selects filters for; it is the smallest for RDLS-RDgDb, whereas for others, it is about three (for RDLS-RCT and RDLS-LDgEb) or four (RDLS-YCoCg-R) times greater.

Table 2 Complexities of the heuristic and the compression exploiting transforms.

Transform	Heuristic	Compression (RDLS transform)	Compression (non-RDLS transform)
RDLS-RCT	$6h(f-1)(c_e+c_l+c_d)+3c_e+3c_l$	$3c_c+3c_l+4c_d$	$3c_c+3c_l$
RDLS-YCoCg-R	$8h(f-1)(c_e+c_l+c_d)+3c_e+3c_l$	$3c_c+3c_l+4c_d$	$3c_c+3c_l$
RDLS-RDgDb	$2h(f-1)(c_e+c_l+c_d)+3c_e+2c_l$	$3c_c+2c_l+2c_d$	$3c_c+2c_l$
RDLS-LDgEb	$6h(f-1)(c_e+c_l+c_d)+3c_e+3c_l$	$3c_c+3c_l+3c_d$	$3c_c+3c_l$

Note: c_d, cost of denoising of an image component; c_l, cost of modifying an image component by applying a lifting step to all image pixels; c_e, cost of estimating the bitrate for the component; c_c, cost of actual compression of a component; f, number of denoising filters; h, number of iterations of step B of the heuristic.

Assuming the perfect bitrate estimation, the step-by-step regime, and that for denoising of C_n of a specific pixel only C_n of this and of other pixels are used, the results obtained after a single iteration of step B of the heuristic are optimal in the case of the RDLS-RDgDb transform. In this transform, a component modified in a given step is modified based on only one other component and is not used in the next steps. Therefore, the filter selected for this step affects the bitrate of the component modified by it only.

The selected filters have to be passed to the decoder along with the compressed data. In this research, we initially used up to 13 filters (described in Sec. 2.5) and from two to four filters must be selected for an image depending on the applied transform. The cost of encoding the filter selection using a fixed-length binary code is at most 20 bits per image—it is negligible.

2.7 *Estimation of Component Compression Effects*

As the primary estimator of the image component compression effects, denoted H0_pMED, we used the memoryless entropy of the component residual image, i.e., of a single-component image consisting of prediction errors calculated as differences between actual and predicted component pixels. The bitrate of the three-component image was estimated as a sum of the estimated bitrates of its three components. The memoryless entropy of a single-component image (a residual image in this case) is $H_0=-\sum_{i=0}^{N-1}p_i\log_2 p_i$, where N is the alphabet size and p_i is the probability of occurrence of pixel value i in the image. For prediction, we used the nonlinear edge-detecting predictor MED [Eq. (13)],[9] which originates from the median adaptive prediction coding of video data:[13,14]

$$\mathrm{MED}(C_n^{[a,b]})=\begin{cases}\min(C_n^{[a-1,b]},C_n^{[a,b-1]}) & \text{if } C_n^{[a-1,b-1]}\geq\max(C_n^{[a-1,b]},C_n^{[a,b-1]})\\ \max(C_n^{[a-1,b]},C_n^{[a,b-1]}) & \text{if } C_n^{[a-1,b-1]}\leq\min(C_n^{[a-1,b]},C_n^{[a,b-1]})\\ C_n^{[a-1,b]}+C_n^{[a,b-1]}-C_n^{[a-1,b-1]} & \text{otherwise}\end{cases}, \tag{13}$$

where $C_n^{[a,b]}$ is the component C_n of the image pixel in column a and row b and MED $(C_n^{[a,b]})$ is its predicted value. For the top image row, we used $C_n^{a-1,b}$ as a predictor; for the leftmost column, we used $C_n^{a,b-1}$; and 0 was a predictor for the top left image pixel.

We also examined limited-complexity estimation methods, where for each transformed image component, instead of entropy of prediction errors of all the pixels, we used:

- the memoryless entropy of 10,000 pseudorandomly selected pixels' component prediction errors, this estimator is denoted H0_pMED(10k:1), and
- the memoryless entropy of 10,000 component prediction errors from 100 pseudorandomly selected nonoverlapping 10×10 pixels rectangles, denoted H0_pMED(10k:100).

For the selection of pixels in H0_pMED(10k:1) and H0_pMED(10k:100), we reinitialized the pseudorandom number generator each time the image component bitrate was estimated. Thus, from all components of an image, in all steps and iterations of the heuristic, the same pixels were used for estimation.

H0_pMED(10k:1) was found by Strutz to be a sufficient estimator for a close to the optimum color space transform selection.[5] For typical image sizes, compared to using for all image pixels, a simpler predictor or to computing the entropy of the image component instead of the component prediction error, it allows a greater reduction of the computational time complexity of the compression effects estimation for the lifting-based transforms.[15] Computing H0_pMED(10k:1) for the three-component image transformed with non-RDLS transform is of low complexity; in order to obtain 30,000 component prediction errors (10,000 in each component), we have to transform 40,000 pixels and compute the MED predictor 30,000 times, whereas the smallest image used in this study contained 262,144 pixels. In the case of the RDLS-modified transform, however, there appears the large extra cost of denoising operations necessary to obtain the prediction errors of individual pixels; denoising operations are the most important factor of the complexity of the heuristic employing H0_pMED(10k:1).

For example, 72 pixel components must be denoised in order to obtain the prediction errors of the single pixel components computed by RDLS-YCoCg-R, which was calculated as follows. We assumed that neighborhoods of

individual pixels whose prediction errors are selected for bitrate estimation are not overlapping or an accidental overlapping is not exploited to reduce the estimation cost and that we use denoising filters with 3×3 pixels windows. To obtain the MED prediction error of a transformed C_2 component of a pixel, which is computed in step 3 of RDLS-YCoCg-R [Eq. (10)], we need C_2 of this pixel and of its neighbors (upper, left-hand, and upper-left)—a 2×2 pixels rectangle area; these pixels are computed in step 3 based on a 2×2 pixels rectangle of C_3^d components (requiring four denoising operations). To obtain a 2×2 pixels rectangle of C_3^d components, we use a 4×4 pixels rectangle of transformed C_3 components that are computed in step 2. In step 2, to obtain a 4×4 pixels rectangle of C_3, we use 4×4 pixels rectangles of C_1^d and C_2^d (32 denoisings); C_2 in this step is an untransformed G primary color component, but C_1 is computed in step 1. To obtain a 4×4 pixels rectangle of C_1^d, we need a 6×6 pixels rectangle of transformed C_1, that, in step 1, is computed using a 6×6 pixels rectangle of C_3^d (36 denoisings). While computing the C_2 prediction error, we obtained data sufficient for computing prediction errors of the remaining components. All in all, we have to perform the pixel's component denoising 72 times per pixel and 720,000 times to get the H0_pMED(10k:1) estimation of the image compression ratio.

For RDLS-YCoCg-R and the smallest images used in this research, the number of denoising operations required by H0_pMED(10k:1) is not much smaller compared to H0_pMED, which requires four denoising operations per image pixel. To reduce the estimation cost, we proposed the H0_pMED(10k:100) estimator that requires 68,400 denoising operations to obtain the bitrate estimation of an RDLS-YCoCg-R transformed image (assuming the 3×3 pixels window of denoising filter and that the neighborhoods of the 10×10 pixels rectangles selected for bitrate estimation are not overlapping or that an accidental overlapping is not exploited to reduce the estimation cost).

As already noted, the heuristic does not require performing all of the transform steps after each filter change and not all components' bitrates must be estimated each time an overall image bitrate is estimated, which allows certain optimizations. Taking them into account, we present in Table 3 the cost of the heuristic for various RDLS-modified transforms, expressed as a number of denoisings of a component of a single pixel. The cost is calculated assuming that in step A of the heuristic, no actual denoising is used, but each time we estimate the bitrate in step B, we do it as if all filters used a 3×3 pixels window. In the cases when the heuristic cost for H0_pMED(10k:1) is the highest (i.e., for RDLS-YCoCg-R and RDLS-LDgEb), employing H0_pMED(10k:100) decreases it over eight times.

We also note that the filtering operation may be optimized. For example, computing the smoothing filter with a 3×3 pixels window and center point weight 1 for an individual pixel [which is needed for the H0_pMED(10k:1) estimator] requires nine arithmetic operations; for pixels inside a contiguous rectangular area [for H0_pMED and H0_pMED (10k:100)], the cost of this filter drops to five arithmetic operations. Having computed the smoothing filter with a certain center point weight, computing it for some other weight requires just three arithmetic operations.

2.8 *Test Data, Implementations, and Procedure*

The evaluation was performed for the sets of 8-bit RGB test images listed below.

- Waterloo—a set of eight color images from the University of Waterloo, image sizes range from 512×512 to 1118×1105 pixels;[16]
- Kodak—a set of 23 images released by the Kodak corporation, all images are of size 768×512 pixels;[17]
- EPFL—a set of 10 images used at the École polytechnique fédérale de Lausanne for subjective JPEG XR quality evaluation,[18] images sizes: 1280×1506 to 1280×1600 pixels;[19]
- A1—a set of three large images scanned from a 36-mm high quality diapositive film, image sizes range from 7376×4832 to 7424×4864 pixels;[20]
- A2—a set of 17 images acquired from 36 mm negatives, image sizes are from 1620×1128 to 1740×1164 pixels;[20]
- A3—a set of 116 images acquired using a camera equipped with a Bayer-pattern RGGB color filter array, all images are of size 1992×1493 pixels;[20]
- A1-red.3, A2-red.3, and A3-red.3—sets of reduced size (3×) images from sets A1, A2, and A3, respectively.

The sets A1, A2, and A3 contain unprocessed photographic images in optical resolutions of acquisition devices, or (A3) as close to such resolution as possible without

Table 3 The cost of the heuristic for various bitrate estimation methods and the cost of the actual RDLS transforms, expressed in number of denoising operations of a pixel's component. The heuristic complexity is reported for denoising filters using up to 3×3 pixels windows.

Transform	Heuristic cost			Transform cost
	H0_pMED	H0_pMED(10k:1)	H0_pMED(10k:100)	
RDLS-RCT	$6t(f-1)h$	$480000(f-1)h$	$82200(f-1)h$	$4\,t$
RDLS-YCoCg-R	$8t(f-1)h$	$1000000(f-1)h$	$121600(f-1)h$	$4\,t$
RDLS-RDgDb	$2t(f-1)h$	$80000(f-1)h$	$24200(f-1)h$	$2\,t$
RDLS-LDgEb	$6t(f-1)h$	$800000(f-1)h$	$92600(f-1)h$	$3\,t$

Note: t, number of pixels in the image; f, number of denoising filters; h, number of iterations of step B of the heuristic.

interpolation of all components. Except for Waterloo, all images may be characterized as continuous-tone photographic. The most widely known Waterloo set contains both photographic and artificial images; some of them are dithered, sharpened, computer-generated, composed of others, or have globally or locally highly sparse histograms of intensity levels.[21,22] The same image sets were used for experiments in Ref. 3, where their more detailed characteristics may be found.

RDLS effects on bitrates were analyzed for three significantly different standard image compression algorithms in the lossless mode: the predictive JPEG-LS,[6,9] the DWT-based JPEG 2000,[7,23] and the JPEG XR employing the discrete cosine transform.[8,24,25] We used the Signal Processing and Multimedia Group, Univ. of British Columbia JPEG-LS implementation, version 2.2,[26] the JasPer implementation of JPEG 2000 by M. Adams, version 1.900,[27,28] and the JPEG XR reference software.[29]

All algorithms were used to compress individual transformed components, one component at a time. Due to requirements of employed file formats and implementations, all components were stored using non-negative integers. Components transformed with the lifting transforms were stored using the nominal component bit-depths. See Table 1 for nominal depths of components of all the examined, lifting-based and RDLS-modified transforms. Since in initial tests, the greater nominal depth of the RDLS-modified transform was rarely needed, for these transforms, we used the bit-depth of the lifting counterpart or, only if pixels of an actual transformed image component exceeded this depth, we increased the component bit-depth up to the nominal depth of the RDLS-modified transform component. The implementation used for applying transforms and the heuristic is freely available.[30] The compression ratio or bitrate r, expressed in bits per pixel (bpp), is calculated based on the total size in bytes of the individually and independently compressed three components of the transformed image, including compressed file format headers; smaller r denotes better compression.

3 Results and Discussion

3.1 *Reversible Denoising and Lifting Steps Effects on Color Space Transforms*

In Fig. 3, we present the RDLS effects on the bitrates of individual RDLS-RCT transformed components and on the overall image bitrates of each of the examined transforms. Bitrates for non-RDLS and RDLS-modified transforms, obtained using denoising filters selected based on H0_pMED estimator in three iterations of step B of the heuristic, were averaged for each set. For easier comparison of RDLS effects, in figures we show the bitrate changes due to RDLS expressed as the percentages of the bitrates obtained with a non-RDLS transform, whereas the absolute bitrates of selected variants of transforms are presented in tables. We also report an average for all sets, however, calculated using average of set-averages, not the direct average of all images. The A3 and A3-red.3x sets contain many more images (116 in each) than all other sets (81 images); therefore, a simple average would be biased toward the A3 and A3-red.3x.

Figures 3(a)–3(c) show that for RDLS-RCT, the overall bitrate improvement due to RDLS is, in many cases, a result of improved bitrates of chrominance components and worsened bitrates of luminance. The above is also true for RDLS-YCoCg-R and RDLS-LDgEb (not shown in Fig. 3). The overall three-component improvements for the RDLS-RCT transform [Fig. 3(d)] result from both employing the actual denoising filters [Fig. 3(e)] and the step skipping by applying the null filter to RDLS [Fig. 3(f)]. Looking at the RDLS effects for other transforms [Fig. 3(g)–3(i)], we see that the greatest bitrate improvements of over 2% on average for all sets were obtained for RDLS-RDgDb. For this transform, the improvements for chrominance components are not accompanied by a worsened bitrate of the third component (i.e., the unmodified primary color R). Generally, the bitrate improvements due to application of RDLS to a color space transform may significantly differ for different sets and for different transforms in the case of a specific set, but are similar for different compression algorithms. Similarity among compression algorithms is stronger when we do not use the null filter. RDLS effects are less pronounced for the JPEG XR algorithm, especially in the case of Waterloo images.

We examined the RDLS effects on color space transforms using several compression algorithms and test image sets. For brevity, we focus on results of the most popular JPEG 2000 algorithm, averaged for all sets. In Table 4, for various transform variants, we report both the bitrates obtained by the RDLS-modified transforms and the bitrate improvements with respect to the non-RDLS transform. To provide a single measure allowing comparisons of variants, we also report the RDLS bitrate change averaged for all the transforms (column labeled "All"). Employing only the transform step skipping, implemented as a special case of the RDLS [i.e., using a null filter and not using smoothing filters, row "RDLS (no smoothing)"], allows bitrate improvements comparable to those obtained by RDLS with the typical denoising and without the step skipping [row "RDLS (no null)"]—better for RCT and YCoCg-R, worse for RDgDb and LDgEb. Step skipping results are better than results obtained by simply deciding, based on the estimated bitrate, whether to perform unmodified transform or to skip it [row "min(RGB, non-RDLS)"], although for YCoCg-R, the simpler method is better. Finally, employing both the step skipping and typical denoising filters allows significantly larger RDLS bitrate improvements, than those obtained when using only the typical denoising filters (compare the two last rows in Table 4). Compared to the filter selection heuristic from Ref. 10 (row "RDLS (Ref. 10, no null)"), for the same filter set, the heuristic we propose in this study results in greater bitrate improvements [row "RDLS (no null)"] and it is of significantly lower complexity in the case of RCT and YCoCg-R.

The average level of improvement of over 1% that we obtained for RCT, YCoCg-R, and LDgEb by using all the denoising filters described in Sec. 2.5 selected based on the H0_pMED estimator in three iterations of the heuristic step B is not negligible as for lossless image compression. However, the heuristic cost may be too high for practical applications. Significantly larger improvements were obtained for specific sets and in the case of RDgDb. In the next section, we reduce the heuristic cost without sacrificing most of the bitrate improvements.

For the above variant, we checked the actual bit-depth expansion of the transformed components. For each

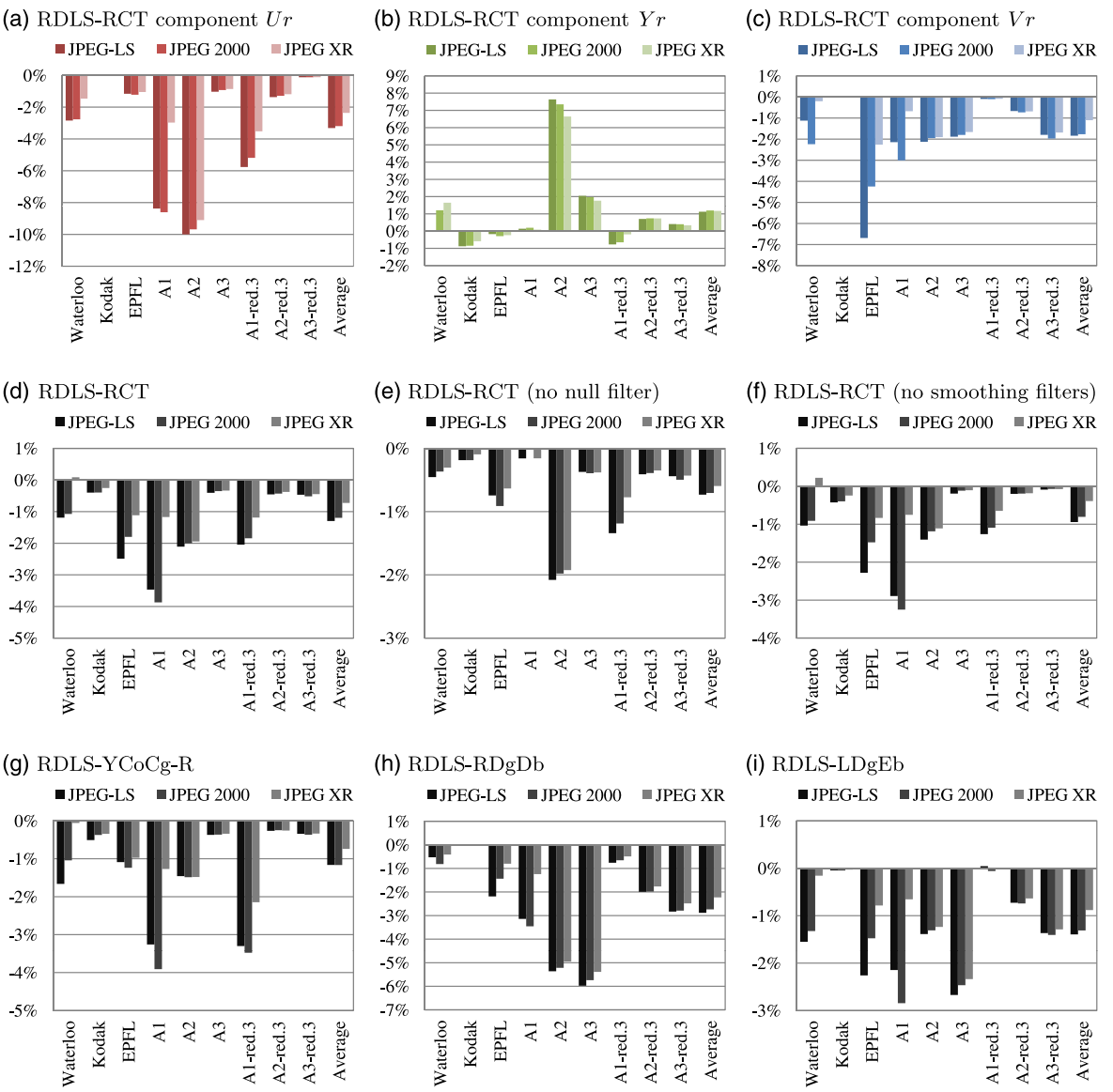

Fig. 3 (a–c) Average JPEG 2000 bitrate changes due to RDLS, for the individual RDLS-RCT components, and (d–i) the overall for examined transforms, obtained using denoising filters described in Sec. 2.5 (all, unless indicated otherwise) selected in three iterations of step B of the heuristic based on bitrates estimated with H0_pMED.

Table 4 Comparison of the RDLS and non-RDLS transform variants. Unless indicated otherwise, denoising filters were selected in three iterations of heuristic step B from all filters described in Sec. 2.5; H0_pMED was employed for the RDLS filter selection and in min(RGB, non-RDLS) non-RDLS variant for choosing between performing unmodified transform or skipping it; RDLS (Ref. 10, no null), filters selected as in Ref. 10.

	RCT		YCoCg-R		RDgDb		LDgEb		All	
Transform variant	r	Δr	r	Δr	r	Δr	r	Δr	r	Δr
non-RDLS transform	11.4374	0.00%	11.4977	0.00%	11.6173	0.00%	11.5039	0.00%	11.5141	0.00%
min(RGB, non-RDLS)	11.3890	−0.42%	11.3921	−0.92%	11.4460	−1.47%	11.4637	−0.35%	11.4227	−0.79%
RDLS (no smoothing)	11.3452	−0.81%	11.4174	−0.70%	11.4067	−1.81%	11.4113	−0.81%	11.3951	−1.03%
RDLS (Ref. 10, no null)	11.3704	−0.59%	11.5184	0.18%	11.3333	−2.44%	11.4640	−0.35%	11.4215	−0.80%
RDLS (no null)	11.3568	−0.71%	11.4345	−0.55%	11.3333	−2.44%	11.4000	−0.90%	11.3812	−1.15%
RDLS	11.3004	−1.20%	11.3635	−1.17%	11.2995	−2.74%	11.3531	−1.31%	11.3291	−1.61%

Note: r—JPEG 2000 bitrate, average for all sets (bpp); Δr—bitrate change with respect to the non-RDLS transform.

transform that may result in a component bit depth greater than the non-RDLS counterpart (RDLS-RCT, RDLS-YCoCg-R, and RDLS-LDgEb), such expansion happened for about two thirds of the images in the case of the luminance component, which each time was expanded by 1 bit.

3.2 *Reducing the Complexity of the Filter Selection*

As shown in Sec. 2.6, the heuristic complexity is proportional to the number of iterations of its step B and to the number of denoising filters. Therefore, we investigated decreasing the iterations number and smaller filter sets. For RDLS-RDgDb, a single iteration of heuristic step B results in the optimal filter selection; bitrate improvements due to RDLS obtained for other transforms in 1, 2, and 3 iterations are presented in Fig. 4. In practice, two iterations seem sufficient; the average bitrates for all sets, obtained in two and three iterations, do not differ noticeably and for individual sets only in three cases the bitrate differs by more than 0.1 percentage points (for the A1.red.3 set in the case of RDLS-RCT and RDLS-YCoCg-R and for A1 in the case of the former transform). On the other hand, by using only one iteration, as compared to two iterations, average bitrates for all sets get over 0.1 percentage point worse for RDLS-YCoCg-R and RDLS-LDgEb, whereas for individual sets, one iteration may be worse by over 1 percentage point. Thus, remembering that in the case of RDLS-RDgDb, the single iteration is optimal, we use two iterations as a starting point for testing other options of the complexity reduction.

In Table 5, we report the JPEG 2000 bitrate changes for a couple of reduced complexity filter selection variants. As previously, we also report the RDLS bitrate change averaged for all the transforms in the column labeled "All." Among others, we examined reducing the number of denoising filters by rejecting of some of the smoothing filters. By using filters with center point weights being even powers of 2 in range from 1 to 256 (row labeled "2 iterations, 7 filters, H0_pMED"), instead of integer powers in range from 1 to 1024, we decrease the complexity of the heuristic about two times (as the filter number drops from 13 to 7) at the acceptable cost of a smaller compression ratio improvement by below 0.05 percentage points on average for all transforms. Further reducing the set by using only three smoothing filters (row "2 iterations, 5 filters, H0_pMED") results in a smaller complexity decrease and a greater cost. Therefore, for the former variant, we applied the simplified compression effect estimation methods.

The differences between effects of H0_pMED and H0_pMED(10k:1) estimators are negligible. On average for all transforms, they are below 0.005 percentage points. Interestingly, the case of the RDLS-LDgEb transform the H0_pMED(10k:1) estimator results in a better average bitrate for all sets than H0_pMED (better by 0.02 percentage

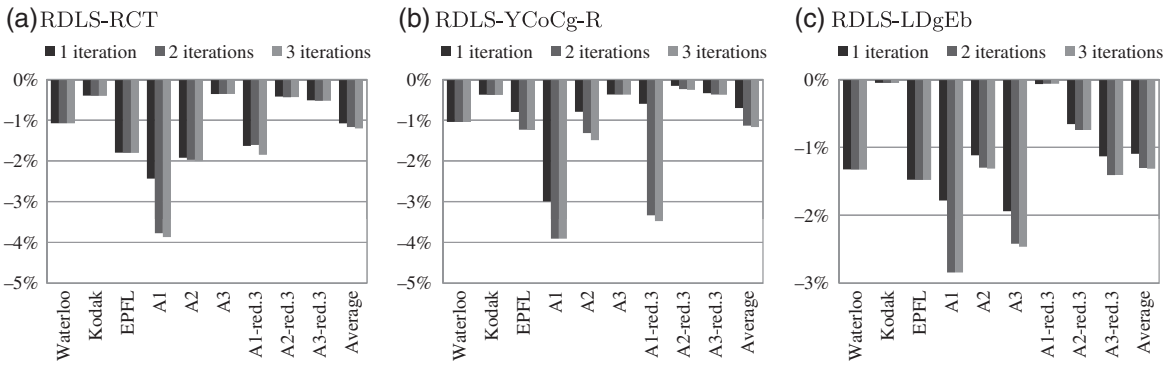

Fig. 4 Average JPEG 2000 bitrate changes due to RDLS obtained using denoising filters selected in 1, 2, and 3 iterations of the heuristic step B out of all filters described in Sec. 2.5. (a) RDLS-RCT, (b) RDLS-YCoCg-R, and (c) RDLS-LDgEb.

Table 5 Reduced complexity RDLS filter selection variants. The JPEG 2000 bitrate changes with respect to non-RDLS transforms are reported. Denoising filters were selected from all or some of the filters described in Sec. 2.5 (13—all; 7—none, null, and smoothing with center point weights 1, 4, 16, 64, and 256; 5—none, null, and smoothing with center point weights 1, 16, and 256).

Filter selection variant	RDLS-RCT	RDLS-YCoCg-R	RDLS-RDgDb	RDLS-LDgEb	All
3 iterations, 13 filters, H0_pMED	−1.20%	−1.17%	−2.74%	−1.31%	−1.61%
2 iterations, 13 filters, H0_pMED	−1.17%	−1.13%	−2.74%	−1.30%	−1.59%
2 iterations, 7 filters, H0_pMED	−1.13%	−1.11%	−2.68%	−1.27%	−1.55%
2 iterations, 5 filters, H0_pMED	−1.08%	−0.98%	−2.57%	−1.19%	−1.46%
2 iterations, 7 filters, H0_pMED(10k:1)	−1.13%	−1.10%	−2.68%	−1.29%	−1.55%
2 iterations, 7 filters, H0_pMED(10k:100)	−1.02%	−1.09%	−2.65%	−1.26%	−1.51%

points). Our results show that the close to the optimum performance of H0_pMED(10k:1), first observed by Strutz for lifting color space transforms, is a more general property of this estimator. Note that by the optimum performance, we mean estimation effects obtained using H0_pMED; in the next section, we check, among others, how good H0_pMED estimation is compared to the actual bitrate of the actual compression algorithm.

The H0_pMED(10k:100) estimator we proposed in order to lower the bitrate estimation cost results in bitrates little worse than H0_pMED(10k:1); on average, for all transforms, it is by 0.04 percentage points worse, for the RDLS-RCT (the worst case) by 0.11 percentage points. The H0_pMED(10k:100) appears to be the most interesting general purpose estimator from a practical standpoint—it allows fast heuristic filter selection that results in bitrates that are close to the bitrates obtained using the most complex variant examined so far (using H0_pMED, three iterations, and all filters described in Sec. 2.5). Compared to the latter, we get bitrates worse by 0.1 percentage points on average for all transforms and sets in the case of the JPEG 2000 coding.

We do not report the actual filter search time of the heuristic or the time of transforming the image with RDLS-modified transforms because our research implementation was not optimized; among others, we did not exploit the possibility of partial estimation of the image bitrate after changing a single filter by the heuristic and each time before outputting a transformed image, the transform reversibility was verified by performing an inverse transform. However, knowing the parameters of the heuristic, its cost may be compared to the cost of the transform it selects filter for. For the smallest images we used (containing 262,144 pixels), with respect to the number of denoising operations (that is to the most expensive part of the heuristic). In Table 6, we report the transform cost and the heuristic cost. The heuristic cost is reported for the H0_pMED(10k:100)-based selection of filters done in two iterations of step B from the set of seven filters using up to 3×3 pixels windows. The heuristic cost is generally close to the transform cost. It is by 6% lower in the case of RDLS-RCT, for RDLS-YCoCg-R and RDLS-LDgEb, it is by about 40% higher, and for RDLS-RDgDb, it is two times lower. Note that for the latter transform, we may use only one iteration as it will not affect the filter selection and further decrease the cost two times. For this transform also, the H0_pMED(10k:1) estimator is practically acceptable, as for one iteration of step B, it requires 480,000 denoisings, i.e., 92% of the number of denoisings required by the RDLS-RDgDb transform. For larger images, the cost of the heuristic exploiting H0_pMED(10k:100) or H0_pMED(10k:1) remains constant, whereas the transform cost grows in direct proportion to the number of pixels in the image.

Table 6 The cost of the heuristic for the H0_pMED(10k:100) estimator and the cost of the actual RDLS transforms, calculated for the smallest images (262144 pixels) and expressed in number of pixel component denoisings. Denoising filters were selected in 2 iterations of the heuristic step B out of the set of 7 filters using up to 3×3 pixels windows.

Transform	Heuristic cost	Transform cost
RDLS-RCT	986,400	1,048,576
RDLS-YCoCg-R	1,459,200	1,048,576
RDLS-RDgDb	288,240	524,288
RDLS-LDgEb	1,111,200	786,432

To verify how the modifications we selected based on the RDLS effects on JPEG 2000 coding affect other algorithms, in Table 7, we report bitrates and bitrate changes obtained for different compression algorithms. Simplifying the estimation method and reducing the size of the filter set and the number of iterations did not change the general way RDLS affects bitrates in the case of different algorithms and color space transforms. RDLS effects for JPEG-LS and JPEG 2000 are close to each other, whereas for the JPEG XR algorithm, the improvements are smaller. The bitrate improvement due to RDLS is the greatest in the case of RDgDb (2.15% for JPEG XR, 2.65% or more for JPEG-LS and JPEG 2000), whereas the bitrates of more complex transforms were improved by over 1% in the case of JPEG-LS and JPEG 2000, and by over 0.5% for JPEG XR. In Fig. 5, the bitrate changes for individual sets are presented and compared to a variant employing H0_pMED estimation, three iterations and a larger set of denoising filters. The greatest differences in effects for these variants may be noticed for RDLS-RCT and RDLS-YCoCg-R in the case of some sets only. In a single case of the former transform for Waterloo images and the JPEG XR algorithm, the simplified filter selection results in about 1.2% worse bitrates than the non-RDLS transform; for this algorithm, images, and transform, the more complex filter selection variant resulted in bitrate worsening (by below 0.1%). For RDLS-RDgDb and RDLS-LDgEb, the results of the simplified filter selection

Table 7 Effects of the H0_pMED(10k:100)-based filter selection variant on JPEG-LS, JPEG 2000, and JPEG XR bitrates. Reported are: the bitrates for the RDLS-modified transforms (r) and the bitrate changes with respect to the non-RDLS transform (Δr), average for all sets and all sets and transforms (All). The denoising filters were selected in two iterations of the heuristic step B out of: none, null, and smoothing filters with center point weights 1, 4, 16, 64, and 256.

	RDLS-RCT		RDLS-YCoCg-R		RDLS-RDgDb		RDLS-LDgEb		All	
Algorithm	r	Δr	r	Δr	r	Δr	r	Δr	r	Δr
JPEG-LS	10.8840	−1.13%	10.9616	−1.10%	10.8621	−2.82%	10.9075	−1.35%	10.9038	−1.60%
JPEG 2000	11.3208	−1.02%	11.3726	−1.09%	11.3089	−2.65%	11.3594	−1.26%	11.3404	−1.51%
JPEG XR	12.4776	−0.51%	12.5078	−0.66%	12.4494	−2.15%	12.5056	−0.83%	12.4851	−1.04%

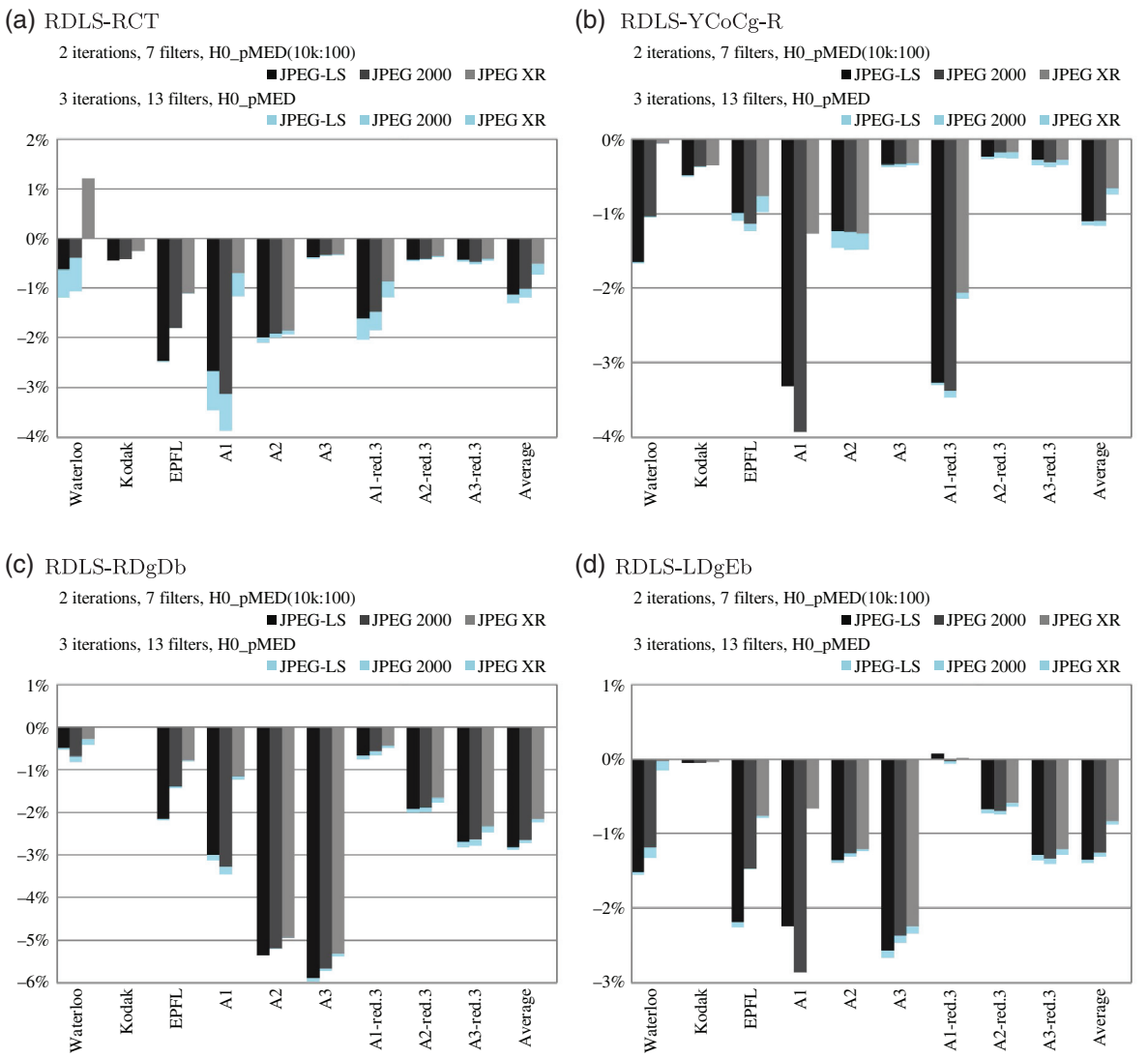

Fig. 5 Average JPEG-LS, JPEG 2000, and JPEG XR bitrate changes due to RDLS with respect to the non-RDLS transform. The denoising filters were selected using the H0_pMED(10k:100) estimator in two iterations of the heuristic step B out of: none, null, and smoothing filters with center point weights 1, 4, 16, 64, and 256, or using H0_pMED in three iterations of the heuristic step B out of all filters described in Sec. 2.5. (a) RDLS-RCT, (b) RDLS-YCoCg-R, (c) RDLS-RDgDb, and (d) RDLS-LDgEb.

for individual sets are similarly close to effects of the more complex variant as on average for all sets.

Looking at the absolute bitrates (Table 7), we notice that differences between compression algorithms for a specific transform are much larger than differences between transforms for a given algorithm. JPEG-LS is consistently the best, JPEG XR the worst. RDLS-RDgDb obtains the best average bitrates for each algorithm which is an effect of the greatest bitrate improvement due to RDLS. In the case of unmodified transforms (recall Table 4), on average for all sets we used in this research, RCT was the best. The bitrate improvements that we attained at a reduced cost appear worthwhile from a practical standpoint. For example, for the lossless JPEG 2000, the bitrates were improved on average for all sets and all transforms by about 1.5%. This improvement is not small if we consider that the best unmodified transform we evaluated (RCT) obtained an average bitrate better than the worst one (RDgDb) by about 1.2% only.

3.3 *Additional Experiments*

We started experiments using the H0_pMED estimator that was found effective for a non-RDLS color space transform selection[5,15] as well as for the selection of denoising filters for some of the RDLS-modified color space transforms investigated in this study.[3,10] We found that by using H0_pMED(10k:100), the estimation complexity may be significantly reduced without sacrificing the RDLS bitrate improvement. But how far from perfect is the H0_pMED or H0_pMED(10k:100) estimation in the case of the investigated RDLS-modified transforms? To check it, we compared the estimation effects to using in the filter selection heuristic the actual image compression algorithm instead of estimating its results (see top three rows in Table 8). The estimation effects are very good from a practical standpoint. Using the actual compressor results in bitrate improvements better than estimation-based by less than 0.1 percentage point on average for all transforms; the greatest difference is for RDLS-RCT, where the H0_pMED

Table 8 RDLS effects for additional filter selection variants. Reported are the average JPEG 2000 bitrate improvements with respect to the non-RDLS transform. The filter set contained the following seven filters: none, null, and smoothing with center point weights 1, 4, 16, 64, and 256. exhaustive, using exhaustive filter search instead of the heuristic; *r*_JPEG_2000, using for filter selection the actual JPEG 2000 bitrate instead of the estimated one.

Filter selection variant	RDLS-RCT	RDLS-YCoCg-R	RDLS-RDgDb	RDLS-LDgEb	All
2 iterations, 7 filters, H0_pMED(10k:100)	−1.02%	−1.09%	−2.65%	−1.26%	−1.51%
2 iterations, 7 filters, H0_pMED	−1.13%	−1.11%	−2.68%	−1.27%	−1.55%
2 iterations, 7 filters, *r*_JPEG_2000	−1.22%	−1.06%	−2.72%	−1.35%	−1.59%
exhaustive, 7 filters, *r*_JPEG_2000	−1.32%	−1.48%	−2.72%	−1.39%	−1.73%

(10k:100)-based bitrate is by 0.2 percentage point worse. Interestingly, for RDLS-YCoCg-R, the use of actual compressor results in bitrates that are worse than estimation-based. Results of the heuristic may not be optimal even when we use the perfect estimation.

The heuristic finds optimal filters for the RDLS-RDgDb transform and for this transform, we obtained the greatest bitrate improvement with respect to the non-RDLS transform. Should the RDLS effects on RDgDb be attributed to the imperfect heuristic filter selection in the case of more complex transforms, or is RDLS the most effective for the simplest transform also when for all transforms we employ optimal filters? We performed the exhaustive filter search and selected for each image and transform the optimal filters out of the set of seven denoising filters. For seven filters, such a search is impractical but realizable, as for the most complex transforms, RDLS-RCT and RDLS-YCoCg-R, it involves testing 2401 filter combinations per image. In Table 8 (row labeled "exhaustive,...") we report the effects of the RDLS-modified transforms for the optimal filter selection based on the actual JPEG 2000 compression bitrate. Let us compare the effects of the heuristic when employing two iterations of step B and actual compression instead of bitrate estimation, to optimal filter selection. The effects of the heuristic significantly vary for different transforms. Indeed, for RDLS-RDgDb, the heuristic filter selection is optimal. For the RDLS-RCT and RDLS-LDgEb, the heuristic is by 0.1 and 0.04 percentage points, respectively, worse than the optimum. However, for RDLS-YCoCg-R, the heuristic result is worse by about 1/4 than the bitrate improvement obtainable for this transform with optimal filter selection. A heuristic based on a different filter search strategy might be better in the case of RDLS-YCoCg-R. Also in the case of the optimal filter selection for all transforms, the largest improvement due to RDLS is for RDgDb—also for the optimal filter selection, RDLS is the most effective for RDgDb.

RDLS recently was found effective for a transform that is much more complex then a color space transform and involves more interdependent steps, i.e., for the multilevel 2-D DWT transform in lossless JPEG 2000 compression.[11] The bitrate improvements exceeding 13% were observed for grayscale images of nonphotographic content when the nonlinear denoising filters were applied. We checked if those filters could be used to further improve the effects of the RDLS-modified color space transforms. We tested all the additional filters from Ref. 11 that were not already used in this study:

- Smoothing filters, with 5×5 pixels windows, employing the same weights of the window center point, as before (11 filters).
- Median—two median filters (3×3 and 5×5 pixels windows), the median 5×5 pixels filter was the strongest (the most harsh) filter used in Ref. 11 and it was found the most effective in the bitrate improvement.
- RCRS-1—two filters (3×3 and 5×5 pixels windows), which belong to a general family of rank-conditioned rank selection (RCRS) filters.[31] RCRS-1 filters replace a sample with the window median if the sample is greater than or smaller than all other samples in the window.
- RCRS-2—two filters (3×3 and 5×5 pixels windows) that replace a sample with the second greatest window sample value if the sample is greater than the median and the greatest; or, if it is smaller than the median and the smallest, they replace a sample with the second smallest window sample value.

Since only the Waterloo set contains nonphotographic images, in Table 9, we report the effects of extending the filter set for RDLS-modified color space transforms with the above filters for both the Waterloo set and average for all the sets. Unfortunately, such a naive approach did not result in practically useful bitrate improvements, especially if we consider the increased complexity of selecting the filters from the set of 30 denoising filters containing filters with larger windows.

RDLS improvements differ for various sets, thus it could be expected that by finding filters better matching the actual image characteristics, greater bitrate improvements due to RDLS could be obtained. Instead of basing on an estimated component bitrate, the noise parameters might be estimated and the denoising filters might be selected based directly on the analysis of the component to be denoised. For a specific acquisition device, the device model may be constructed that allows determining the denoising filters based directly on the acquisition process parameters (e.g., see Refs. 32 and 33). The former approach has an additional advantage. The component that is available as the function f argument for filters selection in RDLS in forward transform [Eq. (6)], is also available for inverse RDLS in inverse transform [Eq. (7)]. Signaling the filter selection (or parameters of a more sophisticated filter) to the decoder might be avoided at the cost of increased decoder complexity, as the same filters, or filter

Table 9 RDLS effects for a larger set of denoising filters. The JPEG 2000 bitrate improvements with respect to the non-RDLS transform are reported, average for all sets and for the Waterloo set. All the filters described in Sec. 2.5 (13) or these filters and the additional filters described in Sec. 3.3 (total 30) were selected using H0_pMED in three iterations of the heuristic step B.

Filters	Sets	RDLS-RCT	RDLS-YCoCg-R	RDLS-RDgDb	RDLS-LDgEb	All
13	All sets	−1.20%	−1.17%	−2.74%	−1.31%	−1.61%
30	All sets	−1.22%	−1.23%	−2.78%	−1.34%	−1.65%
13	Waterloo only	−1.07%	−1.04%	−0.81%	−1.33%	−1.06%
30	Waterloo only	−1.08%	−1.03%	−0.83%	−1.34%	−1.07%

parameters, can be found by the decoder based on analysis of the same data. On the other hand, in Ref. 10, we used a heuristic that performed an exhaustive search of filters in a given step based on the estimated bitrate of the component modified by this step only and in the case of RDLS-RCT and RDLS-LDgEb, it resulted in significant worsening of bitrates of some images. Therefore, it may be expected that selecting a filter for the data to be denoised based on this data may be effective for the simplest RDLS-RDgDb transform and not necessarily for others, which is an interesting topic that we leave for future research.

4 Summary

In this study, we examined the application of RDLS to the RCT, YCoCg-R, RDgDb, and LDgEb color space transforms; the RDLS-modified transforms are named RDLS-RCT, RDLS-YCoCg-R, RDLS-RDgDb, and RDLS-LDgEb, respectively. For the image-adaptive denoising filter selection, we proposed a simple and greedy heuristic consisting of steps A and B, where step B may be performed for a given number of iterations. In the heuristic, we used compression algorithm independent estimators of the filter selection effects on the transformed image bitrate. Initially, we used an estimator based on the memoryless entropy of the transformed image MED prediction errors of all pixels of each component (H0_pMED). We also employed a simplified, limited computational time complexity estimator that uses 10,000 pseudorandomly selected pixels [H0_pMED(10k:1)]. To further decrease the complexity of bitrate estimation, we proposed the H0_pMED(10k:100) estimator that also uses 10,000 pixels, but due to grouping them, in the case of the most complex RDLS-modified color space transforms, its complexity is over eight times lower compared to the H0_pMED(10k:1). Beside typical denoising filters (11 linear smoothing filters with 3×3 pixels windows) and the none filter, that may turn RDLS into the regular lifting step, we proposed the special filter case named the null filter. For the null filter, the denoised sample equals 0, which may result in the step skipping. In the experiments, we used several test image sets and significantly different standard image compression algorithms in the lossless mode: JPEG-LS, JPEG 2000, and JPEG XR.

We found that generally, the RDLS effects significantly differ for different image sets and for different transforms in the case of a specific set. They are similar for different compression algorithms, but they are less pronounced in the case of the JPEG XR algorithm. The largest average bitrate improvements were obtained for the simplest transform RDLS-RDgDb and improvements for others were roughly two times smaller. The overall bitrate improvements due to RDLS result from employing of both the actual denoising filters and the null filter. Although a certain level of bitrate improvement might be obtained by simply checking the estimated effects of skipping the entire color space transform, greater improvements were obtained by employing the null filter that may result in a partial transform skipping. The initial number of smoothing filters could be reduced without sacrificing the bitrate improvements. Assuming the perfect bitrate estimation, due to properties of RDLS-RDgDb, the proposed heuristic in one iteration of step B finds optimal filters for this transform. For other transforms, performing two iterations of step B is justified as using more iterations does not improve RDLS effects noticeably.

The most expensive element of the computational time complexity of the heuristic is the denoising of pixel components. This cost may be limited and reduced by employing simplified compression effect estimators. When using the H0_pMED(10k:100) estimator, the heuristic cost (for two iterations of its step B and seven denoising filters) gets close to the transform cost for RDLS-RCT, RDLS-YCoCg-R, and RDLS-LDgEb transforms. The heuristic cost for RDLS-RDgDb is four times lower than the transform cost (here, one iteration suffices) and for this transform, the cost of the more expensive H0_pMED(10k:1) estimator is still lower than the transform cost. For larger images, the cost of the heuristic exploiting H0_pMED(10k:100) or H0_pMED (10k:1) remains constant, whereas the transform cost (and the cost of the heuristic exploiting H0_pMED) grows in direct proportion to the number of pixels in the image. The H0_pMED(10k:100)-based heuristic filter selection results in at least about three fourths of the bitrate improvement obtainable with RDLS for the optimal filter selection based on the actual bitrate of the compression algorithm.

All in all, the most interesting results from a practical standpoint were obtained for an image-adaptive heuristic filter selection from the set of seven filters (none, null, and smoothing with center point weights 1, 4, 16, 64, and 256) using the simplified estimator of compression effects H0_pMED(10k:100), which is independent of the actually employed compression algorithm. On average, the bitrate improvement due to RDLS is the greatest in the case of RDLS-RDgDb (2.65% or more for JPEG-LS and JPEG 2000, 2.15% for JPEG XR), while the bitrates of more complex transforms were improved by over 1% in the case of JPEG-LS and JPEG 2000, and by over 0.5% for JPEG XR. For some sets, the improvements due to RDLS-RDgDb exceed 5%. Also, with respect to the absolute bitrate, this transform was the best for all the image compression

algorithms investigated in this study. In addition to the better average bitrates and bitrate improvements as well as the lower filter selection cost, another advantage of this transform is that its dynamic range is not increased compared to the non-RDLS counterpart.

We suppose that by finding filters better matching the actual image characteristics greater bitrate improvements due to RDLS could be obtained. For a given acquisition device, the denoising filters may probably be selected based on the acquisition process parameters rather than by using the heuristic employing estimated compression ratio of the denoised component. For RDLS-RDgDb, they may be selected or constructed directly based on the component being denoised, thus avoiding the need to signal to the decoder the filter selection.

5 Conclusion

RDLS applied to color space transforms allows the improvement of the bitrates of lossless image compression algorithms, however, RDLS effects depend on selecting proper denoising filters for the image being compressed. By exploiting the new contributions of this study, i.e., the filter-selection heuristic and the special filter case (the null filter), we attained bitrate improvements that on average are about two times higher than those obtained using the previously reported methods. Another new contribution, the H0_pMED(10k:100) compression effect estimator, reduced the cost of the filter selection process without sacrificing the majority of the bitrate improvement. The filter selection cost gets this way close to or lower than the transform cost, while on average for all the investigated transforms and images, the lossless JPEG 2000 bitrates are improved by about 1.5%; bitrates of certain images are improved to a significantly greater extent. All in all, the RDLS-modified color space transforms appear worthwhile from a practical standpoint.

Acknowledgments

This work was supported by the BK-219/RAU2/2016 grant from the Institute of Informatics, Silesian University of Technology and by the 02/020/RGH15/0060 grant from the Silesian University of Technology. A patent application on the RDLS-modified color space transforms presented in this work was filed to the Polish Patent Office (application number: P.396766).

References

1. H. S. Malvar, G. J. Sullivan, and S. Srinivasan, "Lifting-based reversible color transformations for image compression," *Proc. SPIE* **7073**, 707307 (2008).
2. W. Sweldens, "The lifting scheme: a custom-design construction of biorthogonal wavelets," *Appl. Comput. Harmonic Anal.* **3**, 186–200 (1996).
3. R. Starosolski, "Reversible denoising and lifting based color component transformation for lossless image compression," arXiv:1508.06106 [cs.MM], http://arxiv.org/abs/1508.06106 (2016) (Submitted).
4. R. Starosolski, "New simple and efficient color space transformations for lossless image compression," *J. Vis. Commun. Image Represent.* **25**(5), 1056–1063 (2014).
5. T. Strutz, "Multiplierless reversible colour transforms and their automatic selection for image data compression," *IEEE Trans. Circ. Syst. Video Technol.* **23**(7), 1249–1259 (2013).
6. ISO/IEC and ITU-T, "Information technology–Lossless and near-lossless compression of continuous-tone still images–Baseline," ISO/IEC International Standard 14495-1 and ITU-T Recommendation T.87 (2006).
7. ISO/IEC and ITU-T, "Information technology–JPEG 2000 image coding system: core coding system," ISO/IEC International Standard 15444-1 and ITU-T Recommendation T.800 (2004).
8. ISO/IEC and ITU-T, "Information technology–JPEG XR image coding system - Image coding specification," ISO/IEC International Standard 29199-2 and ITU-T Recommendation T.832 (2012).
9. M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS," *IEEE Trans. Image Process.* **9**(8), 1309–1324 (2000).
10. R. Starosolski, "Application of reversible denoising and lifting steps to LDgEb and RCT color space transforms for improved lossless compression," in *Proc. BDAS 2016, Springer CCIS*, Vol. 613, pp. 623–632 (2016).
11. R. Starosolski, "Application of reversible denoising and lifting steps to DWT in lossless JPEG 2000 for improved bitrates," *Signal Process. Image Commun.* **39**, 249–263 (2015).
12. H. S. Malvar and G. J. Sullivan, "YCoCg-R: a color space with RGB reversibility and low dynamic range," ISO/IEC JTC1/SC29/WG11 and ITU-T SG16 Q.6 Document JVT-I014r3 (2003).
13. H. Murakami, S. Matsumoto, Y. Hatori, and H. Yamamoto, "15/30 Mbit/s universal digital TV codec using a median adaptive predictive coding method," *IEEE Trans. Commun.* **35**(6), 637–645 (1987).
14. S. A. Martucci, "Reversible compression of HDTV images using median adaptive prediction and arithmetic coding," in *Proc. IEEE Int. Symp. Circuits Systems*, Vol. 2, pp. 1310–1313 (1990).
15. T. Strutz, "Adaptive selection of colour transformations for reversible image compression," in *Proc. 20th European Signal Processing Conf.*, pp. 1204–1208 (2012).
16. "Set (Colour Set) of color images from the University of Waterloo, Fractal Coding and Analysis Group," http://links.uwaterloo.ca/Repository.html (21 February 2016).
17. "Set of images from the Kodak corporation," http://www.cipr.rpi.edu/resource/stills/kodak.html (21 February 2016)
18. F. De Simone et al., "Subjective evaluation of JPEG XR image compression," *Proc. SPIE* **7443**, 74430L (2009).
19. "Set of images from École \'Ecole polytechnique f\'ed\'erale de Lausanne," http://documents.epfl.ch/groups/g/gr/gr-eb-unit/www/IQA/Original.zip (21 February 2016)
20. "Sets A1, A2, and A3 of color images in optical resolutions of acquisition devices," http://sun.aei.polsl.pl/~rstaros/optres/ (21 February 2016)
21. R. Starosolski, "Compressing high bit depth images of sparse histograms," in *Int. Electronic Conf. on Computer Science, AIP Conf. Proc.* Vol. 1060, pp. 269–272, American Institute of Physics (2008).
22. A. J. Pinho, "Preprocessing techniques for improving the lossless compression of images with quasi-sparse and locally sparse histograms," in *Proc. ICME*, Vol. 1, pp. 633–636 (2002).
23. D. S. Taubman and M. W. Marcellin, *JPEG2000 Image Compression Fundamentals, Standards and Practice*, Springer Science + Business Media, New York (2002).
24. S. Srinivasan et al., "HD Photo: a new image coding technology for digital photography," *Proc. SPIE* **6696**, 66960A (2007).
25. F. Dufaux, G. J. Sullivan, and T. Ebrahimi, "The JPEG XR image coding standard," *IEEE Signal Proc. Mag.* **26**(6), 195–199 (2009).
26. SPMG/UBC, "Signal Processing and Multimedia Group, Univ. of British Columbia JPEG-LS implementation, version 2.2," http://www.stat.columbia.edu/~jakulin/jpeg-ls/mirror.htm (21 February 2016).
27. M. Adams, "JasPer implementation of JPEG 2000, version 1.900," http://www.ece.uvic.ca/~mdadams/jasper/ (21 February 2016).
28. M. D. Adams and R. K. Ward, "JasPer: a portable flexible open-source software tool kit for image coding/processing," in *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, Vol. 5, pp. 241–244 (2004).
29. ISO/IEC and ITU-T, "Information Technology–JPEG XR Image Coding System - Reference Software," ISO/IEC International Standard 29199-5 and ITU-T Recommendation T.835 (2012).
30. "Implementation of RDLS-modified color space transforms (colortransf-rdls), version 0.9," http://sun.aei.polsl.pl/~rstaros/colortransf-rdls/ (21 February 2016)
31. R. C. Hardie and K. E. Barner, "Rank conditioned rank selection filters for signal restoration," *IEEE Trans. Image Process.* **3**(2), 192–206 (1994).
32. T. Bernas et al., "Application of detector precision characteristics and histogram packing for compression of biological fluorescence micrographs," *Comput. Methods Programs Biomed.* **108**(2), 511–523 (2012).
33. T. Bernas, R. Starosolski, and R. Wójcicki, "Application of detector precision characteristics for the denoising of biological micrographs in the wavelet domain," *Biomed. Signal Process.* **19**, 1–13 (2015).

Roman Starosolski is an assistant professor at the Institute of Informatics, Silesian University of Technology. He received his MSc and PhD degrees in computer science from the Silesian University of Technology in 1995 and 2002, respectively. He is the author of about 30 papers and has written five book chapters. His current research interests include image processing, compression of image, video and volumetric data, color space transforms, medical imaging, and image compression standards.

Signal Processing: *Image Communication* 39 (2015) 249–263

Contents lists available at ScienceDirect

Signal Processing: *Image Communication*

journal homepage: www.elsevier.com/locate/image

Application of reversible denoising and lifting steps to DWT in lossless JPEG 2000 for improved bitrates

Roman Starosolski *

Institute of Computer Science, Silesian University of Technology, Akademicka 16, 44-100 Gliwice, Poland

ARTICLE INFO

Article history:
Received 12 June 2015
Received in revised form
29 August 2015
Accepted 18 September 2015
Available online 9 October 2015

Keywords:
Reversible denoising and lifting step
Discrete wavelet transform
Denoising
Lifting technique
Lossless image compression
JPEG 2000.

ABSTRACT

In a previous study, we noticed that the lifting step of a color space transform might increase the amount of noise that must be encoded during compression of an image. To alleviate this problem, we proposed the replacement of lifting steps with reversible denoising and lifting steps (RDLS), which are basically lifting steps integrated with denoising filters. We found the approach effective for some of the tested images. In this study, we apply RDLS to discrete wavelet transform (DWT) in JPEG 2000 lossless coding. We evaluate RDLS effects on bitrates using various denoising filters and a large number of diverse images. We employ a heuristic for image-adaptive RDLS filter selection; based on its empirical outcomes, we also propose a fixed filter selection variant. We find that RDLS significantly improves bitrates of non-photographic images and of images with impulse noise added, while bitrates of photographic images are improved by below 1% on average. Considering that the DWT stage may worsen bitrates of some images, we propose a couple of practical compression schemes based on JPEG 2000 and RDLS. For non-photographic images, we obtain an average bitrate improvement of about 12% for fixed filter selection and about 14% for image-adaptive selection.

1. Introduction

1.1. Overview

Advantageous properties of a lifting scheme [1] (e.g., perfect reconstruction and in-place operation) made it useful to construct significant transforms for the lossless image compression domain, including discrete wavelet transform (DWT) and various color space transforms [2]. The scheme decomposes a transform into a series of lifting steps that are reversible even if implemented using finite precision integer numbers.

In a previous study [3], we noticed that the lifting step of a color space transform might increase the amount of noise that must be encoded during compression of an image. It is known that Red, Green, and Blue primary color components of the RGB color space are highly correlated for natural images [2]. A common approach to RGB color image compression is to independently compress the image components obtained using a color space transform from the RGB to a less correlated color space. An unwanted side effect of color space transform performed using lifting steps is that removing correlation contaminates the transformed components with noise from other components. Although such transform does not change the overall image information content, the amount of noise in a component considered individually is increased. Because of the independent compression of transformed image components, the overall amount of noise that has to be encoded increases. To remove correlation without increasing noise in components, we proposed the replacement of lifting steps with reversible denoising and lifting steps (RDLS); basically, the new steps are lifting steps integrated with denoising filters. We found the approach to be effective for some of the tested images. In this study, we

* Tel.: +48 322372151.
E-mail addresses: rstarosolski@polsl.pl, rstaros@gmail.com

http://dx.doi.org/10.1016/j.image.2015.09.013
0923-5965/

apply RDLS to DWT in lossless JPEG 2000 coding [4–6], evaluate RDLS effects on bitrates of various types of images, and show practical compression schemes exploiting RDLS.

The remainder of this paper is organized as follows. In Sections 1.2 and 1.3, we briefly characterize the DWT used in lossless JPEG 2000 and the RDLS. The main contribution of this study is described in Section 2, where a new RDLS–DWT transform obtained by the application of RDLS to DWT is introduced along with a heuristic for selecting denoising filters for RDLS–DWT. We evaluate RDLS–DWT on classic test images, images with impulse or Gaussian noise added, and a recent, large, and diverse set of images. Section 3 describes the experimental procedure and the test data used. Section 4 presents the results and discussion. Section 5 summarizes the research.

1.2. Discrete wavelet transform in reversible JPEG 2000 coding

For brevity, we describe here only the lifting-based 5×3 kernel DWT that is used in lossless JPEG 2000 compression of grayscale images, reduced to essentials. For further details as well as for more general characteristics of DWT, JPEG 2000, and the lifting scheme, the reader is referred to [1,4–6].

Using the lifting scheme [1], the one-dimensional DWT (1D DWT) transforms in-place a discrete signal $S=s_0\ s_1\ s_2\ \ldots\ s_{l-1}$ of finite length l into two subbands:

- a low-pass filtered signal L representing the original signal's low-frequency features; and
- a high-pass filtered signal H containing high-frequency features that, along with the low-pass signal, allows the perfect reconstruction of the original signal.

S is transformed in 3 steps. First, in the prediction step, we perform the high-pass filtering of odd samples (hereafter, the parity of sample or pixel is determined by its location and not its value) by applying the lifting step Eq. (1) to each of them:

$$s_x \leftarrow s_x - \lfloor (s_{x-1}+s_{x+1})/2 \rfloor, \tag{1}$$

where the floor symbol $\lfloor v \rfloor$ denotes the greatest integer not exceeding v. Note that the amount by which the value of the odd sample is changed depends only on even samples. Another lifting step is then applied to each even sample (update step):

$$s_x \leftarrow s_x + \lfloor (s_{x-1}+s_{x+1}+2)/4 \rfloor. \tag{2}$$

Here, even samples are updated based directly on the already transformed odd samples. Finally, in the reorder step, we reposition even samples to the lower half of the original signal, preserving their ordering (sample s_x is moved to $s_{\lfloor x/2 \rfloor}$), and odd samples are moved to the upper half. We obtain separate subbands L and H, respectively. The two-dimensional DWT (2D DWT) transform of an image is obtained by first applying 1D DWT to each image column, which results in L and H subbands of the image. Then by applying 1D DWT to each row, we obtain the 1-level DWT transform, consisting of LL and HL subbands (transformed from L subband) and LH and HH subbands (from H subband); see Fig. 1A–C. Higher-level DWT transforms that provide multiresolution image representation are obtained by Mallat decomposition [7]. The $t+1$-level transform is obtained by applying the 1-level transform to the LL subband of the t-level transform (Fig. 1D).

In lossless JPEG 2000, the transformed image is encoded in a complex and flexible manner [4–6]. For the remainder of this study, it is noteworthy that each subband is compressed independently of the others using a context-adaptive entropy coder.

1.3. Reversible denoising and lifting step

The lifting technique is also used in other domains; for example, the reversible color space transforms are built as sequences of lifting steps [2]. Such transforms remove correlations among color image pixel components, but, as a consequence, a transformed component is contaminated with noise from more than one original component. In [3], we integrated denoising into lifting steps to avoid noise propagation while preserving other transform properties (i.e., reversibility and removing correlation). The lifting step that was modified was generalized as the following:

$$c_x \leftarrow c_x \oplus \mathrm{f}(c_1,\ \ldots\ , c_{x-1}, c_{x+1},\ \ldots\ , c_m), \tag{3}$$

where c_n is the n-th component of the pixel and m is the number of components. The step is reversible provided that the function f is deterministic and the operation $\oplus$ is reversible, that is, an inverse operation $\otimes$ exists such that $c = a \oplus b \Leftrightarrow a = c \otimes b$.

Based on the lifting step (Eq. (3)), we constructed a reversible denoising and lifting step (RDLS, Eq. (4)) by simply denoising arguments of function f:

$$c_x \leftarrow c_x \oplus \mathrm{f}(c_1^d,\ \ldots\ , c_{x-1}^d, c_{x+1}^d,\ \ldots\ , c_m^d), \tag{4}$$

where c_n^d is the denoised n-th component of the pixel, which is obtained by using a denoising filter, and different filters may be used for different components. Eq. (3) implies that the function f may use any arguments, except for c_x. In Eq. (4), we denoise these arguments. For denoising them we may use any component of any pixel,

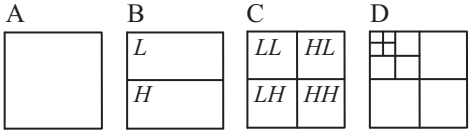

Fig. 1. 1-level 2D DWT of an image (A–C) and 3-level 2D DWT (D).

except for c_x of the pixel to which the RDLS-modified lifting step is being applied; otherwise, this c_x would be indirectly used by f. Obviously, the denoising filter must be deterministic. A color space transform may be performed for each pixel independently of other pixels. The RDLS sequence, constructed based on a color space transform, is a transform of whole image components, not of the color space, since denoising of a pixel's component requires access to the same component of (at least) neighboring pixels. For the RDgDb color space transform [8] (also known as $A_{2,1}$ [9]) and a very simple denoising filter (linear smoothing filter), we found that the proposed method was effective for images in optical resolutions of acquisition devices. As noted in [3], the RDLS method is general in nature and is applicable to other lifting-based transforms; in this study, we apply it to DWT in lossless JPEG 2000. An example of RDLS application to DWT is presented in Section 2.2.

2. Proposed approach

2.1. RDLS-modified DWT

The application of RDLS to the 5×3 kernel 2D DWT is straightforward when we notice the analogy between a color space transform applied to a color image and a DWT of a grayscale image. In the former case, we apply a color space transform as a sequence of lifting steps to pixels consisting of color space components; all pixels constitute a color image. In the latter, we first apply a 1D DWT transform (many equivalent sequences of lifting steps are possible) to image columns consisting of pixels; all columns constitute a grayscale image. Then we repeat the process for the obtained rows. Thus, in order to obtain a RDLS-modified DWT transform (RDLS–DWT), we simply replace the DWT lifting steps with RDLS constructed based on them. The RDLS-modified prediction lifting step (Eq. (1)) becomes

$$s_x \leftarrow s_x - \left\lfloor (s^d_{x-1} + s^d_{x+1})/2 \right\rfloor, \tag{5}$$

where s^d_n is a denoised sample s_n, and the RDLS-update is the following:

$$s_x \leftarrow s_x + \left\lfloor (s^d_{x-1} + s^d_{x+1} + 2)/4 \right\rfloor. \tag{6}$$

But how do we select the denoising filter? Naturally, the filter should be matched to the noise characteristics of the data being filtered. We assume that the noise characteristics of the original image as well as of each of its subbands (including those that are temporarily created and then further transformed – see Fig. 1B), are invariant, i.e., the same denoising filter may be used while computing all samples of a specific subband. For example, when computing *H* subband by applying Eq. (5) for each odd sample in each column of the original image, we use the same filter for denoising all even samples. However, the noise characteristics of different subbands differ, so a different filter may be used when computing the *HL* subband.

In this study, for the denoising of a sample of a specific parity, we use samples of the same parity only. In the above case of computing the *H* subband, for denoising a particular even sample of a particular column, we may use all even samples of all columns. Generally, during computation of a specific subband, we perform the denoising based on samples not belonging to this subband. For the process to be reversible, as noted in Section 1.3, while applying the RDLS to a specific sample s_x, we may use all signal samples for denoising, except the s_x. However, certain practical problems would arise if we used samples from the subband being computed. We assume that the subband characteristics are invariant; this assumption applies to the already computed subbands and to the original signal. Obviously, it cannot be valid for a subband during its computation, as during the computation only some subband samples are already transformed. Thus, for the denoising, we should either only use untransformed samples from this subband, or we should employ a sophisticated denoising filter that is able to take advantage of both transformed and untransformed samples. Second, actual JPEG 2000 implementations apply the lifting steps to a given subband in the same order when performing the inverse transform, as is applied during a forward transform; if we used samples from the subband being computed, then the orders should be exactly the opposite.

A special case of the denoising filter is the filter that does not alter the sample being denoised, denoted here as the None filter. Using the None filter in RDLS makes it a regular lifting step. Since samples may be noise-free, when performing the filter selection, we should be able to select the None filter.

For the *t*-level RDLS–DWT, we need to select denoising filters for $6t$ subbands. One of the possibilities explored in this study is to define a set of filters from which we select the filter for each subband individually, based on actual image compression effects. Subbands are encoded independently but are interdependent. For instance, selecting a filter for some subband that results in a worse bitrate of that subband may help to find a better filter for another subband and therefore may result in an overall better bitrate. Since testing the compression effects of all the combinations of denoising filters would be too complex even for just a few filters considered, we employ a filter selection heuristic, as described in Section 2.3. The resulting filter selection should be transmitted to the decoder as side information along with the compressed image.

In this study, we apply RDLS to the 5×3 kernel DWT because this is the wavelet used in lossless JPEG 2000 coding. DWT with different kernels, as well as other transforms that are implemented using lifting steps, may be modified in a similar way, i.e., by replacing lifting steps (Eq. (3)) with RDLSs (Eq. (4)). However, finding denoising filters for other transforms may be more complicated. For example, for the *t*-level 9×7 kernel transform, we would have to select $12t$ filters, since 1-level 9×7 kernel 1D DWT is done using 4 steps involving lifting (as opposed to 2 such steps in case of 5×3 kernel).

2.2. RDLS–DWT example

Assume that while computing the level 1 of unmodified 2D DWT, we perform the prediction step for image rows by

applying 1D DWT to each of them. The input signal is an image that contains a smoothly changing background and salt-and-pepper noise (Fig. 2A). We start the forward transform from the 3rd sample in the 1st image row $s_{3,1}$, and while reconstructing the image, we apply inverses of the lifting steps in reverse order as compared to forward transform. In contrast, an actual JPEG 2000 codec would start from $s_{1,1}$ and use the same order for forward and inverse transform. The prediction lifting step (Eq. (1)) for $s_{3,1}$ is $s_{3,1} \leftarrow s_{3,1} - \lfloor (s_{2,1}+s_{4,1})/2 \rfloor$, so $s_{3,1}$ becomes 28 (Fig. 2B). When reconstructing $s_{3,1}$, the signal contents will be identical to just after performing a forward lifting step for this sample (Fig. 2B). Lifting steps are trivially invertible, the inverse of Eq. (1) is $s_x \leftarrow s_x + \lfloor (s_{x-1}+s_{x+1})/2 \rfloor$, and in our case $s_{3,1} \leftarrow s_{3,1} + \lfloor (s_{2,1}+s_{4,1})/2 \rfloor$, so $s_{3,1}$ is reconstructed to 55.

For RDLS–DWT, we use a median filter with a 3×3 sample window. The RDLS-modified prediction lifting step (Eq. (5)) applied to $s_{3,1}$ is $s_{3,1} \leftarrow s_{3,1} - \left\lfloor (s^d_{2,1}+s^d_{4,1})/2 \right\rfloor$, and $s_{3,1}$ becomes 1 (Fig. 2C). Note that denoising of a sample is performed using samples of the same parity only, so for denoising $s_{2,1}$ and $s_{4,1}$ during a step performed for image rows, we use only samples from even columns (Fig. 2D) and obtain $s^d_{2,1} = 54$ and $s^d_{4,1} = 55$. During the inverse transform, we apply the inverse of Eq. (5), that is, $s_x \leftarrow s_x + \lfloor (s^d_{x-1}+s^d_{x+1})/2 \rfloor$. To $s_{3,1}$, we apply $s_{3,1} \leftarrow s_{3,1} + \left\lfloor (s^d_{2,1}+s^d_{4,1})/2 \right\rfloor$, where the same $s^d_{2,1}$ and $s^d_{4,1}$ are obtained as during the forward transform, since they are results of denoising of exactly the same samples (Figs. 2C and 2D). Thus, $s_{3,1}$ is reconstructed to 55.

A

	0	1	2	3	4	5	6
0	51	52	0	54	55	56	57
1	100	53	54	55	0	57	100
2	53	54	55	56	57	0	59

B

	0	1	2	3	4	5	6
0	51	52	0	54	55	56	57
1	100	53	54	28	0	57	100
2	53	54	55	56	57	0	59

C

	0	1	2	3	4	5	6
0	51	52	0	54	55	56	57
1	100	53	54	1	0	57	100
2	53	54	55	56	57	0	59

D

	0	1	2	3	4	5	6
0	■		■□		■□		□
1	■		■□		■□		□
2	■		■□		■□		□

Fig. 2. Example of DWT and RDLS–DWT. A – original signal, B – signal after applying the prediction lifting step to sample $s_{3,1}$, C – signal after applying the RDLS-modified prediction lifting step to sample $s_{3,1}$, D – locations of samples used for denoising of $s_{2,1}$(■) and $s_{4,1}$(□).

It is worth noting that in the above example, RDLS reduced the effect of noise on the quality of prediction. Without noise, if $s_{4,1}$ was 56, the regular prediction would result in setting $s_{3,1}$ to 0; contaminating $s_{4,1}$ with impulse noise resulted in a much greater change for unmodified DWT than for RDLS–DWT due to efficient denoising of $s_{4,1}$ exploited by the latter.

2.3. Filter selection heuristic

Our filter selection heuristic consists of the greedy steps described below, in which step B may be repeated certain number of iterations. With a slight abuse of terminology, we also use the term JPEG 2000 compression for JPEG 2000 with DWT replaced by RDLS–DWT, even though it is not compliant with the JPEG 2000 standard.

(A) For each of the filters, perform JPEG 2000 compression of an image, using this filter in RDLS steps for all subbands at all levels. Then for all subbands at all transform levels, select the filter that resulted in the best overall bitrate.

(B) For each transform level a (starting from level 1) and for each subband b (at specific level analyzed in the *H*, *L*, *HL*, *HH*, *LL*, and *LH* order), try to find a better denoising filter by checking for each filter (except for the one already selected) the bitrate obtained using this filter for subband b at level a, while the filters selected so far are used for other subbands.

It is noteworthy that the heuristic selects a filter instead of the one already selected only if such a change improves the overall bitrate. The None filter that turns RDLS into a regular lifting step may be selected in step A for all subbands; thus, the bitrate cannot be worsened by the heuristic compared to unmodified DWT.

The heuristic is quite time-consuming. The RDLS–DWT computational time complexity alone is higher than that of the unmodified DWT due to the noise filtering employed. However, subbands are encoded independently; thus, for example, changing the denoising filter for the level-2 subband neither affects the compression results of level-1 subbands *LH*, *HL*, and *HH* nor requires their repeated denoising. There are two main elements of heuristic computational time complexity: RDLS–DWT subband transforms and encoding of transformed samples. The number of symbols that need to be encoded during a single iteration of step B of the heuristic for the t-level transform is

$$(4-4^{1-t})(f-1)p, \tag{7}$$

where f is the number of denoising filters, and p is the number of pixels in the image. The number of RDLS per single iteration of step B is

$$\left(\frac{86}{9}-\frac{1}{3}2^{5-2t}t-\frac{43}{9}2^{1-2t}\right)(f-1)p. \tag{8}$$

Note that the t-level DWT (or RDLS–DWT if the filters for the latter are already selected) is done using the

following number of lifting steps (or RDLS):

$$\left(\frac{8}{3}-\frac{1}{3}2^{3-2t}\right)p. \tag{9}$$

To allow straightforward comparison of the complexities of various variants of the heuristic and of compression schemes involving the heuristic and the unmodified JPEG 2000, we express the heuristic computational time complexity as relative to unmodified JPEG 2000. A few simplifications are necessary to this aim. Increasing the transform level, except for a couple of lowest levels, does not influence the heuristic or transform complexity significantly. Assuming the infinite transform level, Eq. (7) becomes $4(f-1)p$; thus, a single iteration of step B requires encoding of roughly $4(f-1)$ times more symbols than the unmodified JPEG 2000. By dividing Eq. (8) by Eq. (9), we find that the number of RDLS per iteration of heuristic step B is greater than the number of lifting steps per unmodified DWT $(43/12)(f-1)$ times, which we approximate with $4(f-1)$. All in all, the estimated complexity of the heuristic (step A and i iterations of step B) is roughly

$$f+4(f-1)i \tag{10}$$

times greater than the complexity of the JPEG 2000 compression with additional subband denoising due to replacing lifting steps with RDLS.

3. Materials and methods

3.1. Denoising methods and denoising filters

In the research, the following denoising methods and filters were used:

- None – No denoising. RDLS becomes a regular lifting step.
- Smoothing – We used 12 simple low-pass linear averaging filters with 3×3 and 5×5 sample windows. The filtered sample was calculated as a weighted arithmetic mean of samples from the window. The number of samples used was smaller than the window size at the image edges. The weight of the window center point ranged between 1 and 1024 (even powers of 2 only), while the others' weights were fixed to 1. Smoothing filters (3×3 sample window) were found to be effective for the RDLS-modified color space transform and certain natural images [3].
- Median – Two median filters were used with 3×3 and 5×5 windows. The Median filter and other nonlinear filters described below are effective at removing impulse noise.
- RCRS-1 – Two filters (3×3 and 5×5 windows), which belong to a general family of rank-conditioned rank selection (RCRS) filters [10]. RCRS-1 filters replace a sample with the window median if the sample is greater than or smaller than all other samples in the window.
- RCRS-2 – Two filters (3×3 and 5×5 windows) that replace a sample with the second greatest window sample value if the sample is greater than the median and the greatest; or, if it is smaller than the median and the smallest, they replace a sample with the second smallest window sample value. A practical advantage of this filter, as opposed to the Median and RCRS-1, is that it may be computed without sorting the window samples.

The effects of applying the above filters to a sample image are presented in Fig. 3; for examining filtering methods on the Reader's own images, a free implementation was prepared [11].

3.2. Test data

We used two diverse, publicly available datasets and additional sets of noisy images; all images were of 8-bit precision.

- GS2 – This is the classic and still commonly used set ("Grayscale Set 2") of 12 images from the University of Waterloo Fractal Coding and Analysis Group [12] (Fig. 4). The set contains grayscale, natural photographic images (among others, "lena" and "peppers") as well as others (computer generated, mixed-content, dithered, and satellite); image sizes vary from 464×352 to 672×496.
- CT2g – This set contains images that are green components of images from a "CT2" set [13]. The CT2 is a recent, large set of color images, which was used in the research on reversible color space transforms [9,14]. It contains 746 images taken from different sources; image sizes vary from 180×117 to 6600×5100. The set was divided into subsets: Photo with 499 photographs and No-photo with 247 non-photographic, computer-generated, or mixed-content images. The latter were further divided into images that were better compressed by JPEG 2000 without DWT (No-photo (a) – 81 images) and with the unmodified DWT (No-photo (b) – 166 images). The Photo images were not divided this way since for only one Photo image it would be better to skip the DWT stage of JPEG 2000.
- SP001_GS2, SP002_GS2, SP005_GS2, SP010_GS2, and SP020_GS2 – These are sets of GS2 images with salt-and-pepper impulse noise added; 1%, 2%, 5%, 10%, and 20% of image pixels, respectively, were replaced by black or white pixels (Fig. 5).
- WG002_GS2, WG005_GS2, WG010_GS2, WG020_GS2, and WG050_GS2 – These are sets of GS2 images with added white Gaussian noise with standard deviations 2, 5, 10, 20, and 50, respectively (Fig. 5).

3.3. Experimental procedure and implementations

We used the IRIS-JP3D JPEG 2000 part 10 (JP3D) [15] reference software developed by Tim Bruylants from Vrije Universiteit Brussel (VUB) and the Interdisciplinary Institute for BroadBand Technology (IBBT), version 1.1.1 [16], which is downward compatible with the basic JPEG 2000 standard. In this codec, it was relatively easy to implement RDLS. In the experiments, except for replacing JPEG 2000 lifting steps with RDLS and setting the transform level, we

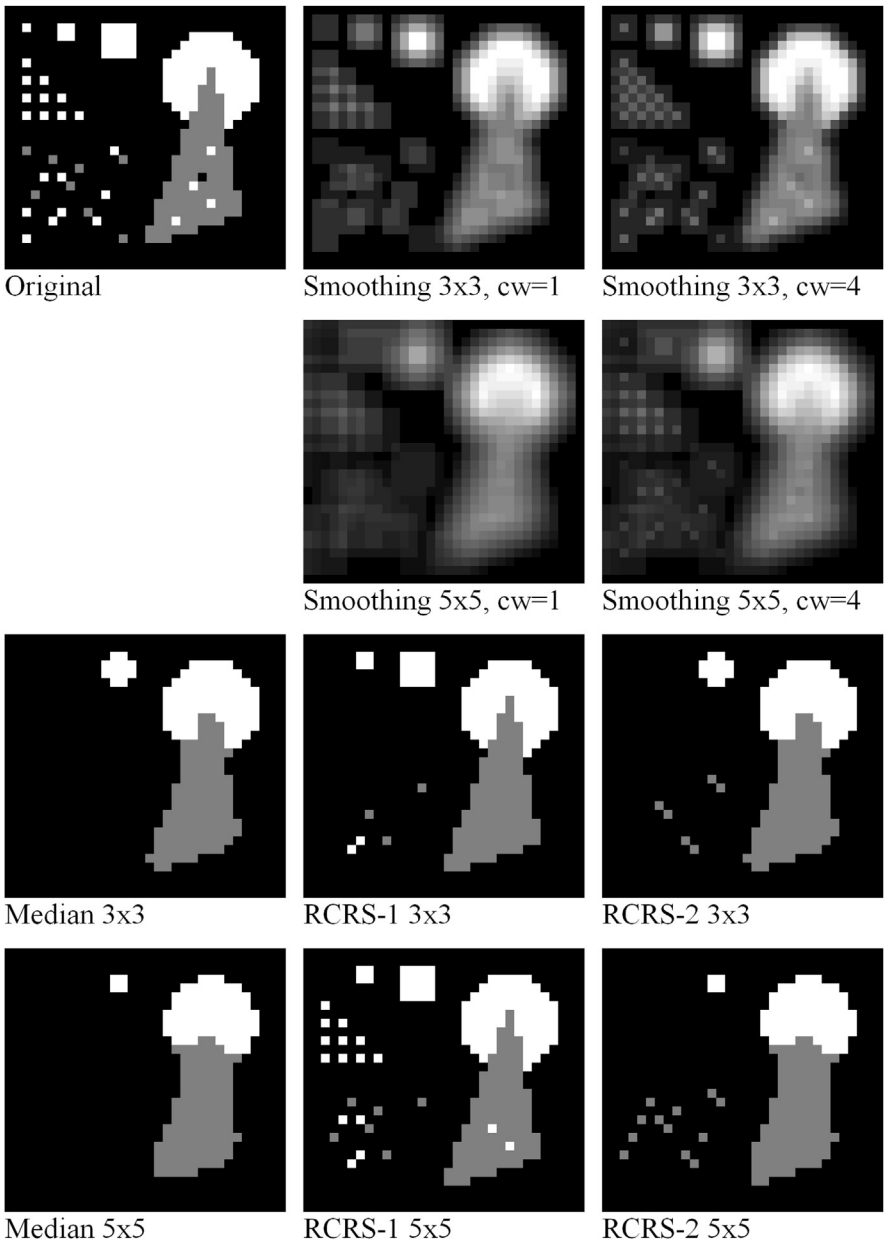

Fig. 3. Effects of filtering a sample image (top left panel); image sizes were 32x30 pixels, and cw is the weight of the window center point of the Smoothing filter.

used the default codec settings. Particularly, a whole image was compressed as a single tile. As the 3-level DWT transform results in bitrates close to the default 5-level, we used the 3-level decomposition. To switch off the DWT transform stage in JPEG 2000 coding, we invoked the codec with the 0-level DWT setting.

For comparison, we also used JPEG-LS [17,18] and HEVC (H.265) [19,20] compression algorithms. JPEG-LS is a standard of the JPEG committee for primarily lossless compression of still images, and it originates from the LOCO-I algorithm. We used the SPMG/UBC JPEG-LS implementation, version 2.2 [21], with default codec settings. HEVC is the most recent video compression standard developed by MPEG and VCEG committees, and it allows lossless compression of individual still images. We used the HEVC Test Model (HM) reference software, version 16.6 [22], with the following set of encoder options: InputChromaFormat=400, FrameRate=1, FramesToBeEncoded=1, Profile=monochrome, IntraPeriod=1, QuadtreeTULog2MaxSize=5, TransquantBypassEnableFlag=1, CUTransquantBypassFlagForce=1, and ConformanceWindowMode=1.

Groups of noisy images were created using pgmtool, version 0.72 [11].

The compression ratio, or bitrate r, expressed in bits per pixel (bpp) is calculated as $r=8e/p$, where p is the number of pixels in the image, and e is the total size in Bytes of the

Fig. 4. The GS2 set.

compressed image, including the compressed file format header. Hence, smaller r means better compression.

We introduced modifications to JPEG 2000, and then we evaluated the modification effects by analyzing bitrate changes with respect to the bitrate of the reference method, that is, of unmodified JPEG 2000, expressed in percentages of the reference method bitrate. Due to the number of test-sets and to the size of the greatest one (CT2g), except for initial observations using GS2 set, we report averaged bitrate changes for a set or for its specific subset rather than results for individual images. Instead of averaged bitrate changes, we could use other common measures. However, the 2 popular ones described below seem inadequate for CT2g images. We could measure how many times the bitrate obtained using the proposed method would be smaller than the bitrate of the reference method. However, the results would be biased toward effects obtained for non-photographic images that were better compressed without the DWT stage, as the bitrates of some of them were 2 to nearly 5 times smaller than the reference method bitrates (which we express as 50% to 80% improvement). Instead of calculating the average of bitrate changes, the average absolute bitrate could be calculated and then compared to the reference method average bitrate. This would result in virtually ignoring the effects obtained for images compressed by the unmodified JPEG 2000 really well and would result in a bias toward effects on poorly compressible images. The unmodified JPEG 2000 bitrate of many non-photographic images was below 1 bpp;

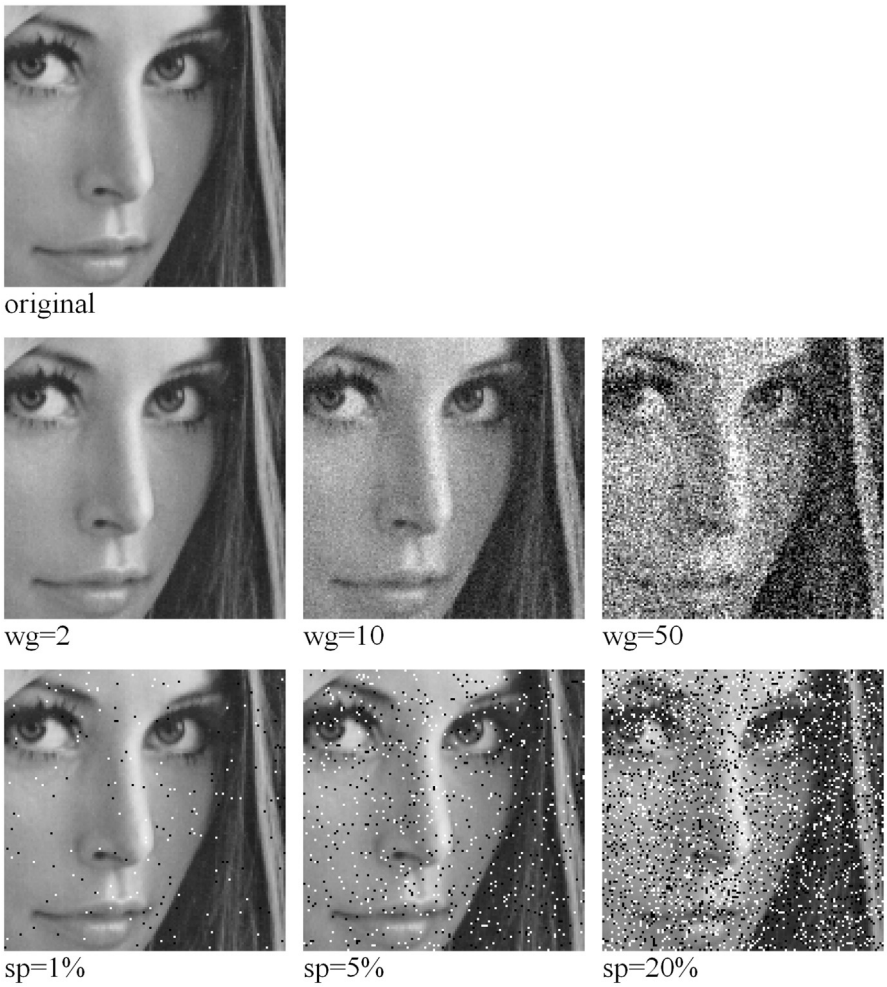

Fig. 5. Fragment (128x128 pixels) of the "lena" image from the GS2 set with added white Gaussian noise (middle row, wg – noise standard deviation) and with added salt-and-pepper impulse noise (bottom row, sp – percentage of image pixels replaced by black or white pixels).

improving this by 50% would affect the averaged bitrate less than improving the bitrate of the image that was compressed to 6 bpp by 10%.

4. Results and discussion

4.1. Initial observations

When the 3 iterations of step B and all the filters presented in Section 3.1 were used, the filter selection heuristic time complexity was roughly 235 times greater than the complexity of JPEG 2000 with subband denoising. Therefore, we initially focused on RDLS effects for the small GS2 set and also analyzed the results for individual images as well as for GS2 images with impulse and Gaussian noise added. Apart from observing RDLS effects on lossless JPEG 2000 bitrates of various types of images, the aim was to check if all the filters are necessary and if the number of iterations may be lowered, thus speeding up further evaluation using the large CT2g set. Results for individual GS2 images as well as averaged results for GS2 set and for sets of noisy images are presented in Table 1. We report lossless JPEG 2000 bitrates obtained for the 3-level unmodified DWT and bitrate changes relative to it obtained for no DWT and for RDLS–DWT with filters selected by up to 3 iterations of step B of the heuristic. For 0 iterations, that is, applying the same filter for all subbands, in case of individual images, we also report the selected filter.

Let's focus on cases when the RDLS bitrate improvement may be practically useful. Since the filter selection heuristic allows only the bitrate improvement, and we start with a large number of filters, a small bitrate improvement might be to some extent accidental. We assume that the bitrate improvement below 1% is unjustified in practical applications when obtained at the cost of a large increase in compression process complexity. Fig. 6 presents cases from Table 1 when the improvement exceeds 1%. For the GS2 images, the improvement exceeds 1% for 4 images (actually, RDLS–DWT is for them at least 3.18% better than DWT). These images are not continuous tone photographic images—"france" is a computer slide containing sharp lines and filled with gradient, "frog" is a dithered photo, "library" is composed of images and text, and finally, "washsat" is a satellite image of a highly sparse histogram [23]. Other images seem

photographic, except for "mountain," which is probably a dithered photo. For images with Gaussian noise added, the improvement is small, which for photographic images and small noise standard deviation could be expected from the GS2 results, since unaltered natural images contain such noise [24]. Interestingly, adding Gaussian noise to images, bitrates of which were significantly improved by RDLS, considerably decreases RDLS improvements—only for "france" for the 2 lowest levels of noise added and for "library" with the lowest level of noise the RDLS improvement exceeds 1%. Probably, the nonlinear denoising filters used ceased to be effective in the presence of Gaussian noise. We see that all groups of impulse noise images (actually, all such images) are compressed significantly better with RDLS. Section 4.3 explains the poor DWT performance on images containing impulse noise or sharp lines. Except for the highest noise level, the RDLS improvement is generally greater when the impulse noise level is higher, while for Gaussian noise, no such dependency can be observed. Moreover, for some GS2 images, it is better not to use DWT; in such cases, RDLS–DWT results in bitrates between bitrates of JPEG 2000 without DWT and with DWT. There are only 3 such images, and the opposite case may be observed for individual images with impulse noise added. If skipping DWT improves the JPEG 2000 bitrate of these images, then in most cases, applying RDLS–DWT results in greater improvement. We take advantage of these observations in Sections 4.4 and 4.5.

4.2. Selection of heuristic parameters

Table 1 shows for individual images, that if a single filter, used for all subbands in RDLS–DWT, significantly improves the GS2 image bitrate, then it is the Median filter. However, greater improvements were obtained when different filters were applied in different RDLS–DWT steps. To reduce the heuristic complexity while preserving high bitrate improvements, for GS2 and noisy images we analyzed which filters were used when the RDLS bitrate improvement was over 1%. The 4 most frequently used were Median 5×5 (for 49.7% of all RDLS–DWT steps), Median 3×3 (21.3%), RCRS-2 3×3 (18.7%), and RCRS-1 3×3 (4.8%). It is noteworthy that the Median 5×5 filter was used in nearly three-fourths of the RDLS–DWT update steps (74.3%). For the remainder of this study, we decided to keep these 4 filters along with the None filter. Even though the None filter was rarely used (0.2%), the heuristic could result in bitrate worsening without it. Smoothing filters were used in 0.0% to 1.1% of the RDLS–DWT steps, RCRS-1 5×5 in 0.2% of steps, RCRS-2 5×5 in 2.6%. We also limited the number of heuristic step B iterations to 2. Although compared to a single iteration, only the "france" image from the GS2 set is noticeably better compressed when we allow 2 iterations, more such cases may happen for the CT2g set with over 700 images. The heuristic complexity for 2 iterations and 5 filters is roughly 37 times greater than the complexity of JPEG 2000 with subband denoising. Table 2 presents the results for the reduced set of filters and 2 iterations.

In cases when the RDLS bitrate improvement obtained using the set of 19 filters is high, the reduction of the filter set to 5 filters has almost no influence on RDLS performance. Both for such individual GS2 images and for averaged improvements for groups of images with impulse noise added, due to simplifying the heuristic, the RDLS improvement is worsened by no more than 0.05 percentage points. The negligible performance drop when reducing the number of filters to 5 suggests that further reduction of the filter set

Table 1
RDLS effects on lossless JPEG 2000 bitrates of individual GS2 images and averaged effects for GS2 set and sets of noisy images.

Image(s)	r_{DWT}	$\Delta r_{NO\text{-}DWT}$ (%)	Step A filter	$\Delta r_{RDLS(19,0)}$ (%)	$\Delta r_{RDLS(19,1)}$ (%)	$\Delta r_{RDLS(19,2)}$ (%)	$\Delta r_{RDLS(19,3)}$ (%)
barb	4.6586	19.59	None	0.00	−0.25	−0.25	−0.25
boat	4.4041	17.39	None	0.00	−0.09	−0.09	−0.10
france	2.0294	41.30	Median 3×3	−7.04	−23.32	−25.30	−25.30
frog	6.2546	−17.35	Median 5×5	−2.87	−3.18	−3.18	−3.18
goldhill	4.8338	12.05	Smoothing 3×3, 2^{10}	−0.02	−0.13	−0.13	−0.13
lena	4.3137	17.65	Smoothing 3×3, 2^{10}	0.00	−0.09	−0.09	−0.09
library	5.6892	−8.90	Median 5×5	−3.87	−5.10	−5.11	−5.11
mandrill	6.1075	3.71	None	0.00	−0.11	−0.12	−0.12
mountain	6.6983	−5.62	Smoothing 3×3, 2^{10}	−0.02	−0.10	−0.10	−0.10
peppers	4.6158	15.74	None	0.00	−0.07	−0.08	−0.08
washsat	4.4308	6.77	Median 5×5	−0.87	−5.89	−5.90	−5.90
zelda	3.9951	26.99	Smoothing 3×3, 2^{10}	−0.01	−0.29	−0.29	−0.29
GS2	4.8359	10.78		−1.22	−3.22	−3.39	−3.39
SP001_GS2	5.4584	−1.31		−9.71	−11.22	−11.26	−11.26
SP002_GS2	5.8882	−6.93		−13.74	−15.03	−15.13	−15.13
SP005_GS2	6.7766	−14.73		−18.95	−20.57	−20.63	−20.63
SP010_GS2	7.5945	−18.44		−21.83	−22.77	−22.78	−22.79
SP020_GS2	8.3557	−18.58		−20.47	−21.11	−21.13	−21.13
WG002_GS2	5.2614	8.43		−0.26	−0.55	−0.55	−0.56
WG005_GS2	5.7360	5.03		−0.09	−0.19	−0.19	−0.19
WG010_GS2	6.3022	2.17		−0.03	−0.12	−0.12	−0.13
WG020_GS2	7.0183	0.39		−0.04	−0.14	−0.15	−0.15
WG050_GS2	8.0824	−1.90		−0.31	−0.35	−0.35	−0.35

r_{DWT} – JPEG 2000 bitrate for 3-level unmodified DWT, $\Delta r_{NO\text{-}DWT}$ – bitrate change obtained by skipping the DWT stage of JPEG 2000, $\Delta r_{RDLS(19,i)}$ – bitrate change obtained by using RDLS–DWT and all denoising filters presented in Section 3.1, adaptively selected in i iterations of heuristic step B. For $\Delta r_{RDLS(19,0)}$, in case of individual images we report the filter selected in step A of the heuristic.

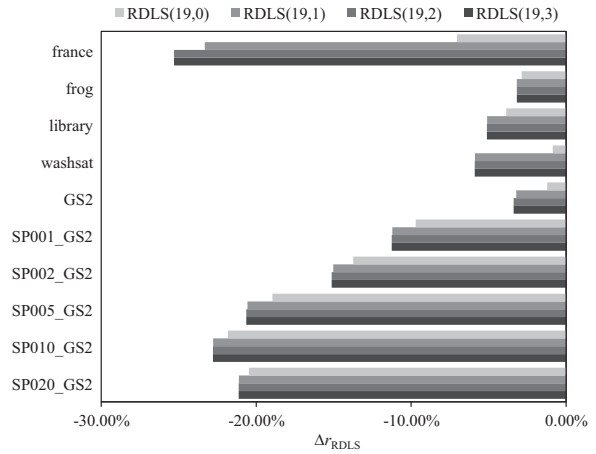

Fig. 6. RDLS effects on lossless JPEG 2000 bitrates of individual GS2 images and averaged effects for the GS2 set and sets of noisy images in cases when the bitrate improvement exceeds 1%. Δr_{RDLS} – bitrate change or average bitrate change due to RDLS as compared to lossless JPEG 2000 bitrate for 3-level unmodified DWT.

may result in still good RDLS bitrate improvements. Therefore, in subsection 4.5, we additionally consider using only the None filter and the Median 5×5 filter. Interestingly, in one case of the SP005_GS2 images, we obtained greater bitrate improvement (by 0.01 percentage points), which shows that heuristic results are not optimal.

4.3. RDLS effect on impulse noise and sharp lines

Fig. 7 shows that in an image with impulse noise added, a single noisy pixel results in the distortion of all subbands of the DWT transformed image. Furthermore, the noisy pixel's neighborhood, originally unaffected by noise, is now altered in all or most subbands, depending on the noisy pixel position in the original image.

The neighborhood is affected since both 1D DWT lifting steps—prediction (Eq. (1)) and update (Eq. (2))—alter the sample based on its neighborhood of opposite parity. Since the update is performed after the prediction, regardless of whether the noisy sample is even or odd (and will end up as part of L or H subband, respectively), both subbands are contaminated. JPEG 2000 encodes subbands independently, so the information on noise appearance has to be encoded 4 times for the 1-level 2D DWT or more for higher levels, increasing the bitrate. Sharp lines affect the DWT similarly, resulting in altered line neighborhoods due to the way 2D DWT is built by using 1D DWTs. For instance, the horizontal line in an image affects the 1D DWT done for the image column the same way as the single impulse noise pixel. The DWT is generally expected to decompose an image into subbands that contain low- and high-frequency image features. For impulse noise and sharp lines, the low- and high-frequency image features are duplicated rather than separated into subbands.

The RDLS–DWT also fails to decompose low- and high-frequency features (Fig. 7, right-hand column). Noisy pixels during prediction and update are just left where they were, and then during the reorder step of transform, they are placed in subbands that correspond to their parities. However, due to denoising, it almost completely avoids generating the unnecessary low-frequency information on noise and propagating the noise into all subbands, allowing improvement of the bitrate.

4.4. RDLS effect on bitrates of typical images

Table 3 reports lossless JPEG 2000 bitrates obtained for the 3-level unmodified DWT and bitrate changes relative to it obtained for no DWT and for RDLS–DWT with a reduced set of 5 filters selected by up to 2 iterations of step B of the heuristic. The results are reported for the CT2g set and its subsets. Fig. 8 presents the results for individual images.

The overall lossless JPEG 2000 bitrate improvement of CT2g images due to RDLS–DWT is significant (roughly 4.8%); it mainly results from RDLS effects on bitrates of non-photographic images, as photographic image bitrates are improved by less than 1% on average. The bitrate improvement for non-photographic images is 13.29% on average, and greater improvements are observed for images that are better compressed with the skipped DWT stage of JPEG 2000. In such cases, RDLS–DWT results in average bitrates that are better than those of JPEG 2000 compression with both unmodified DWT and without DWT.

A single iteration of step B of the parameter selection heuristic results in bitrate improvements close to the improvements obtained with 2 iterations (worse by up to 0.12 percentage points for one of the subsets) and much better than those obtained after step A only (by up to 4.48 percentage points). The increased complexity of parameter selection by using 2 iterations seems practically unjustified: for 2 iterations, it is roughly 37 times greater than the complexity of JPEG 2000 with subband denoising, while for 1 iteration, it is 21 times. The initial selection of denoising filters was arbitrary; the number of filters was reduced based on the results obtained for other images. We could probably obtain larger improvements without such increase in complexity by finding better denoising filters.

The definitely best denoising filter among the examined ones is Median 5×5, which outperformed more sophisticated filters and the Median filter with a smaller window. It is the strongest among the nonlinear filters used, which suggests that a stronger denoising might further improve bitrates. Note also that we did not test larger windows or windows of different shapes. Since when computing H and L subbands during 2D DWT by applying 1D DWT to image columns, we perform denoising based on samples of a specific parity only in a vertical direction and on all samples in a horizontal direction; the actual area corresponding to the denoising filter window in the original image is rectangular. Since the correlation of pixels is proportional to the distance between them, the window should instead be circular or square.

4.5. Practical observations

Up to this point, we evaluated RDLS effects on DWT transform in lossless JPEG 2000 coding. In practice, we would be interested in the bitrate achievable by an algorithm. To improve it, we may consider the observation that for some

Table 2
Effects of RDLS with the reduced set of filters on lossless JPEG 2000 bitrates of individual GS2 images and averaged effects for GS2 set and sets of noisy images.

Image(s)	r_{DWT}	$\Delta r_{RDLS(19,3)}$ (%)	$\Delta r_{RDLS(5,2)}$ (%)
Barb	4.6586	−0.25	−0.21
Boat	4.4041	−0.10	0.00
France	2.0294	−25.30	−25.30
Frog	6.2546	−3.18	−3.18
Goldhill	4.8338	−0.13	−0.02
Lena	4.3137	−0.09	−0.01
Library	5.6892	−5.11	−5.12
Mandrill	6.1075	−0.12	−0.06
Mountain	6.6983	−0.10	−0.02
Peppers	4.6158	−0.08	−0.01
Washsat	4.4308	−5.90	−5.90
Zelda	3.9951	−0.29	−0.13
GS2	4.8359	−3.39	−3.33
SP001_GS2	5.4584	−11.26	−11.21
SP002_GS2	5.8882	−15.13	−15.13
SP005_GS2	6.7766	−20.63	−20.65
SP010_GS2	7.5945	−22.79	−22.79
SP020_GS2	8.3557	−21.13	−21.13
WG002_GS2	5.2614	−0.56	−0.50
WG005_GS2	5.7360	−0.19	−0.12
WG010_GS2	6.3022	−0.13	−0.04
WG020_GS2	7.0183	−0.15	−0.02
WG050_GS2	8.0824	−0.35	−0.09

r_{DWT} – JPEG 2000 bitrate for 3-level unmodified DWT, $\Delta r_{RDLS(f,i)}$ – bitrate change or average bitrate change obtained by using RDLS–DWT and f denoising filters, adaptively selected in i iterations of heuristic step B out of (f=19) all filters from Section 3.1 or (f=5) Median 5 × 5, Median 3 × 3, RCRS-2 3 × 3, RCRS-1 3 × 3, and None.

individual images, it is better to skip the DWT or RDLS–DWT stage of JPEG 2000. The cost of the bitrate improvement is also important. Therefore, we tested a couple more variants of compression, including allowing the DWT stage of JPEG 2000 to be skipped at the cost of a single extra execution of the compression process, using the most frequently selected Median 5 × 5 filter and None filter only, and replacing the heuristic with a fixed assignment of denoising filters to RDLS steps. In the latter case, we used the Median 5 × 5 filter for update steps and the None filter for prediction steps. Table 4 presents the results.

The application of RDLS to DWT transform in lossless JPEG 2000 coding results in significant bitrate improvements for non-photographic images. In most cases, these improvements are obtained at the cost of image-adaptive selecting of denoising filters using the heuristic. The only extra cost during decompression is due to denoising.

First, let's examine the results obtained without using the filter selection heuristic (last 3 rows in Table 4). Using the Median 5 × 5 filter for all RDLS-modified DWT update steps and leaving prediction lifting steps unmodified (row Δr_{RDLS_FIXED}), we improved the bitrates of non-photographic images by 8.64% on average. Since some images were better compressed when the RDLS–DWT stage was skipped, at the cost of an additional execution of the compression process, we may obtain greater improvement of 11.64% for non-photographic images ($\Delta r_{RDLS\_FIXED,\ NO\text{-}DWT}$). However, this improvement should be compared to another improvement, which is obtained at the same cost of a double compression process execution. When for each image we pick a better bitrate out of those obtained by using the unmodified DWT and by skipping the DWT stage ($\Delta r_{NO\text{-}DWT,\ DWT}$), the average improvement is 7.43%. Both of these cases of RDLS application worsen the bitrates of photographic images by about 0.5%. We may avoid the bitrate increase either by employing the filter selection heuristic that allows selecting the None filter or by directly checking the unmodified DWT bitrate. In the latter case, at the cost of 3 executions ($\Delta r_{RDLS\_FIXED,\ NO\text{-}DWT,\ DWT}$), we obtain a compression scheme that is useful for practical applications. It significantly improves the bitrates of non-photographic images by almost 12% on average and does not worsen the bitrate of any image. In [3,9], it was shown that by using the entropy of prediction error of a transformed color image component, we might efficiently select the color space transform or the denoising filter for RDLS-modified color space transform without executing the compression process and independently of the actual image compression algorithm (JPEG 2000, JPEG-LS, and HD Photo/JPEG-XR [25,26] were used in these studies). In [9], only a small subset of image pixels (up to 10000) was found sufficient for close to optimum transform selection. Finding a similar estimator of DWT variants, including RDLS–DWT, unmodified DWT, and skipped DWT is an interesting field of future research.

The image-adaptive selection of denoising filter for each RDLS–DWT step allows greater bitrate improvements at a greater cost. Since for non-photographic images, supplementing the RDLS denoising filter selection heuristic with checking the effect of the compression with the skipped DWT stage results in noticeably greater improvements (by over 1 percentage point), we focus on results obtained this way. Using only the Median 5 × 5 denoising filter and the None filter at the cost of more than two times greater complexity ($\Delta r_{RDLS(2,1),\ NO\text{-}DWT}$), we obtain an average improvement of 14%, i.e., better by over 2 percentage points, than without using the heuristic ($\Delta r_{RDLS\_FIXED,\ NO\text{-}DWT,\ DWT}$). The improvement is mainly due to larger improvements for images that are better compressed with the skipped DWT. Adding 3 adaptively selected denoising filters further increases the complexity three-fold, but the bitrate is further improved by below 0.5 percentage points only ($\Delta r_{RDLS(5,1),\ NO\text{-}DWT}$). Such cost may be too high for practical applications, but a fast estimation of RDLS–DWT effects could make the adaptive application of many filters useful.

4.6. Comparison to other algorithms

Table 5 compares average bitrate changes relative to bitrates of unmodified lossless JPEG 2000 obtained by one of the practical variants that exploit RDLS discussed in the previous subsection (RDLS(2,1), NO-DWT) and obtained by JPEG-LS and HEVC algorithms. It can be seen that JPEG-LS is the best for the CT2g set as well as for its subsets. The recent HEVC is in-between JPEG-LS and our RDLS variant for non-photographic images. By applying RDLS to DWT in lossless JPEG 2000 and by allowing to skip the DWT stage we obtained a compression scheme, results of which for No-photo images are closer to JPEG-LS than to unmodified JPEG 2000. However, JPEG-LS is better by roughly 7.5 percentage points, indicating that the further improvement of lossless JPEG 2000 bitrates should be possible in case of

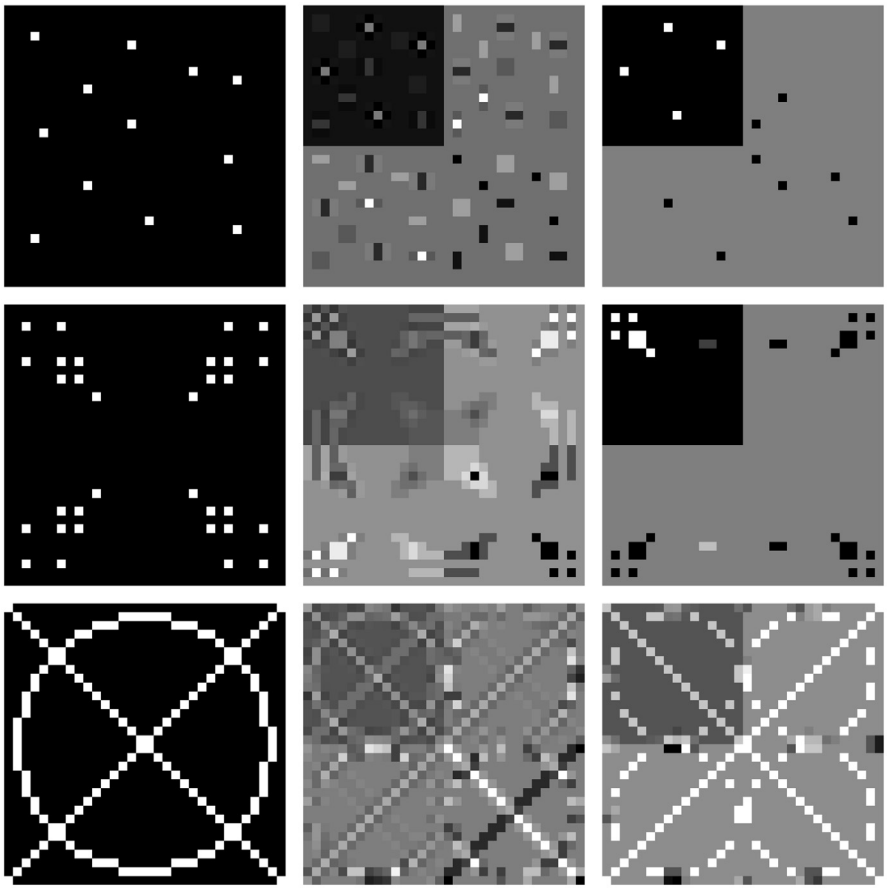

Fig. 7. Effects of impulse noise and sharp lines on 1-level 2D DWT. Left-hand column – original images (32x32 pixels), middle column – DWT transformed images, right-hand column – RDLS–DWT transformed images (Median 3x3 filter used); transformed images are normalized to dynamic range of original ones.

Table 3
Averaged RDLS effects on lossless JPEG 2000 bitrates of CT2g images.

Images	Count	r_{DWT}	$\Delta r_{NO\text{-}DWT}$ (%)	$\Delta r_{RDLS(5,0)}$ (%)	$\Delta r_{RDLS(5,1)}$ (%)	$\Delta r_{RDLS(5,2)}$ (%)
Photo	499	3.9975	25.6	−0.02	−0.62	−0.64
No−photo	247	2.9162	18.8	−9.09	−13.18	−13.29
No−photo (a)	81	3.2882	−22.6	−23.89	−28.37	−28.45
No−photo (b)	166	2.7348	39.0	−1.87	−5.77	−5.89
All	746	3.6395	23.3	−3.02	−4.78	−4.83

r_{DWT} – JPEG 2000 bitrate for 3-level unmodified DWT; $\Delta r_{NO\text{-}DWT}$ – bitrate change obtained by skipping the DWT stage of JPEG 2000; $\Delta r_{RDLS(5,i)}$ – bitrate change obtained by using RDLS–DWT and 5 denoising filters (Median 5×5, Median 3×3, RCRS-2 3×3, RCRS-1 3×3, and None), adaptively selected in i iterations of step B of the heuristic.

non-photographic images. The improvement of the RDLS variant is much greater for No-photo (a) images than for No-photo (b), both as percentage of unmodified JPEG 2000 bitrate and as compared to improvement obtained by JPEG-LS. Interesting observations may be made for Photo images. Here we obtain only a small fraction of improvement possible by use of JPEG-LS. Generally, our method gives only small improvements for photographic images, which probably could be improved, since we were selecting filters for non-photographic and noisy images. It may be also noted, that HEVC bitrates are for Photo images significantly worse than those of unmodified JPEG 2000. The HEVC settings we applied might not be optimal; however, in [27] for lossless coding of video sequences, the HEVC bitrates were worse than bitrates of JPEG-LS by about 11%, which complies with our results.

5. Conclusions and future work

In this study, we applied RDLS, which is basically the integration of lifting steps with denoising filters, to DWT in JPEG 2000 lossless coding and evaluated RDLS effects on bitrates of various types of images. The RDLS application is

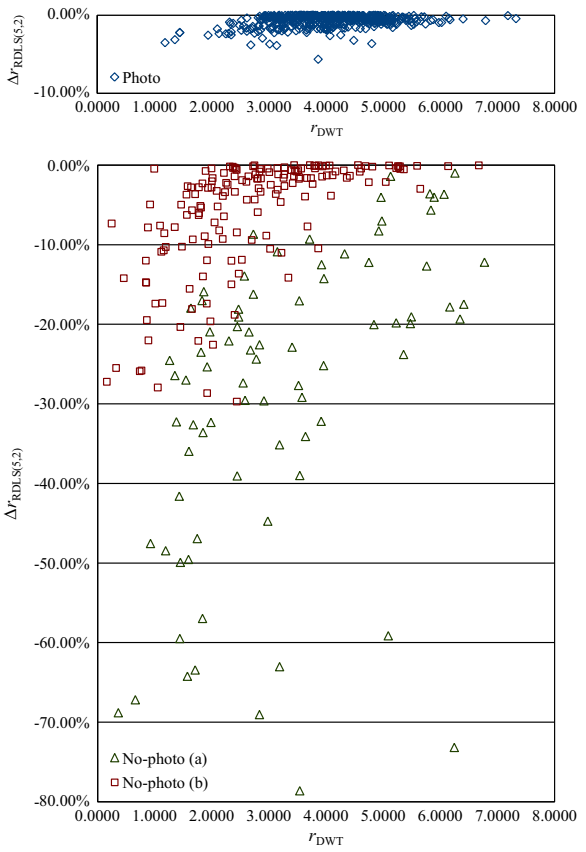

Fig. 8. RDLS effects on lossless JPEG 2000 bitrates of CT2g images. r_{DWT} – JPEG 2000 bitrate for 3-level unmodified DWT, $\Delta r_{RDLS(5,2)}$ – bitrate change obtained by using RDLS–DWT with 5 denoising filters (Median 5x5, Median 3x3, RCRS-2 3x3, RCRS-1 3x3, and None), adaptively selected by 2 iterations of step B of the heuristic. Top panel – photographic images; bottom panel – non-photographic images.

straightforward, as there is an analogy between the DWT of a grayscale image and a transform for which the RDLS technique was originally proposed—the color space transform of a color image. However, assuming that the DWT subbands' characteristics are invariant, we must find proper denoising filters for each RDLS-modified DWT (RDLS–DWT) subband, that is, 6t filters for t decomposition levels. Subbands are interdependent, so we proposed a filter selection heuristic consisting of greedy steps A and B, where step B may be repeated given a number of iterations. The heuristic requires multiple executions of the JPEG 2000 compression process that may involve subband denoising.

In the experiments, we used simple low-pass linear averaging filters (Smoothing filters) that were effective in the RDLS-modified color space transform and nonlinear filters (including the Median filter) that were effective in removing impulse noise; 3×3 and 5×5 sample windows were used for both types of filters. First, for classic grayscale images (Waterloo GraySet2) and for images with impulse or Gaussian noise added, we found that practically useful bitrate improvements with RDLS were obtained for non-photographic images and for images with impulse noise added. In these cases, reducing the set of denoising filters to the 4 filters most frequently selected by the heuristic along with the None filter that turns RDLS into a regular lifting step and using 2 iterations of step B of the heuristic has almost no influence on bitrate improvements due to RDLS as compared to using all 19 filters and a greater number of iterations. All of the 4 most frequently selected filters are nonlinear, and the most frequently selected filter is the Median 5×5 filter. We also observed that some images are better compressed by JPEG 2000 if we skip the DWT stage of the algorithm; the unmodified DWT increases the amount of information that has to be

Table 4
Averaged effects of various compression variants on lossless JPEG 2000 bitrates of CT2g images.

Compression variants	Complexity	Photo	No-photo	No-photo (a)	No-photo (b)	All
r_{DWT}	1	3.9975	2.9162	3.2882	2.7348	3.6395
$\Delta r_{NO\text{-}DWT}$	1	25.59%	18.77%	−22.65%	38.98%	23.33%
$\Delta r_{NO\text{-}DWT,\ DWT}$	2	−0.01%	−7.43%	−22.65%	0.00%	−2.47%
$\Delta r_{RDLS(5,0)}$	5	−0.02%	−9.09%	−23.89%	−1.87%	−3.02%
$\Delta r_{RDLS(5,0),\ NO\text{-}DWT}$	6	−0.03%	−11.11%	−30.05%	−1.87%	−3.70%
$\Delta r_{RDLS(5,1)}$	21	−0.62%	−13.18%	−28.37%	−5.77%	−4.78%
$\Delta r_{RDLS(5,1),\ NO\text{-}DWT}$	22	−0.63%	−14.34%	−31.90%	−5.77%	−5.17%
$\Delta r_{RDLS(5,2)}$	37	−0.64%	−13.29%	−28.45%	−5.89%	−4.83%
$\Delta r_{RDLS(5,2),\ NO\text{-}DWT}$	38	−0.65%	−14.43%	−31.92%	−5.89%	−5.21%
$\Delta r_{RDLS(2,1)}$	6	−0.56%	−12.61%	−27.13%	−5.53%	−4.55%
$\Delta r_{RDLS(2,1),\ NO\text{-}DWT}$	7	−0.57%	−14.00%	−31.37%	−5.53%	−5.02%
Δr_{RDLS_FIXED}	1	0.53%	−8.64%	−17.29%	−4.42%	−2.51%
$\Delta r_{RDLS\_FIXED,\ NO\text{-}DWT}$	2	0.52%	−11.64%	−26.42%	−4.42%	−3.51%
$\Delta r_{RDLS\_FIXED,\ NO\text{-}DWT,\ DWT}$	3	−0.31%	−11.90%	−26.42%	−4.81%	−4.14%

r_{DWT} – lossless JPEG 2000 bitrate for 3-level unmodified DWT; $\Delta r_{variant\_list}$ – bitrate change obtained by picking for each image the best bitrate out of bitrates obtained using the listed variants; DWT – unmodified DWT transform; NO-DWT – skipping the DWT stage of JPEG 2000; RDLS(f,i) – RDLS–DWT using f denoising filters adaptively selected in i iterations of step B of the heuristic out of (f=5) Median 5×5, Median 3×3, RCRS-2 3×3, RCRS-1 3×3, and None or (f=2) Median 5×5 and None; RDLS_FIXED – RDLS–DWT with fixed assignment of denoising filters to RDLS steps (Median 5×5 for update steps and None for prediction steps); Complexity – computational time complexity as roughly compared to the complexity of the JPEG 2000 compression process which may involve subband denoising or DWT stage skipping.

encoded when an image contains impulse noise or sharp lines.

Next, we evaluated RDLS effects on DWT by using a recent, large, and diverse set of photographic and non-photographic images (CT2g). Significant bitrate improvements due to RDLS were also observed for the latter; an average bitrate improvement of over 13% was obtained at the cost roughly 21 times greater than the cost of JPEG 2000 compression process with subband denoising. However, it is noteworthy that for about one-third of these images, the DWT transform worsens the lossless JPEG 2000 bitrate, so the bitrate may be improved by simply skipping the DWT stage. In such cases, RDLS–DWT usually results in average bitrates that are better than those of JPEG 2000 compression with both unmodified DWT and without DWT, but for some individual images, skipping DWT gives the best results. Therefore, in practice, including the possibility of skipping the DWT stage is justified.

Finally, from a practical viewpoint, we evaluated a couple of compression schemes by exploiting RDLS and also the possibility of skipping the DWT stage or using a fixed assignment of denoising filters to RDLS–DWT subbands, instead of time-consuming heuristic. A simple scheme based on a fixed filter assignment, at the cost of a triple compression process execution, significantly improves the bitrates of non-photographic images by almost 12% on average and does not worsen the bitrate of any image. Image-adaptive filter selection by using the heuristic results in an average improvement of 14% and with over two times greater cost. In both cases, only the Median 5×5 and None filters were used. Further bitrate improvements are possible by using more filters but at a cost that seems too high for practical applications. As compared to JPEG-LS, the proposed schemes for non-photographic images allow to obtain bitrates that are closer to JPEG-LS than to unmodified JPEG 2000; however, the better performance of the former indicates that the further improvement of lossless JPEG 2000 bitrates should be possible.

Bitrate improvements were obtained at the cost of multiple executions of the compression process and, in most cases, of multiple denoising of RDLS–DWT subbands. Finding a fast estimator of denoising filter selection effects to be used within or instead of filter selection heuristic is an interesting field of future research. Another field involves finding better denoising filters, as we found reasons for why the filter set used in this study might be not optimal. Some images are better compressed when the DWT/RDLS–DWT compression stage is skipped, but similar to the denoise filtering that is most effective when applied to some subbands only, the optimum may be in-between skipping and applying the transform. Therefore, we are currently investigating the image-adaptive, partial transform skipping.

Conflict of interest statement

None.

Role of the funding source

The study sponsors were not involved in the study design, in the collection, analysis and interpretation of data, in the writing of the manuscript, or in the decision to submit the manuscript for publication.

Acknowledgments

We thank Professor Tilo Strutz for providing the classification of the CT2 set into photographic and non-photographic images and the anonymous reviewers for their constructive comments. This work was supported by the BK-266/RAu2/2014 Grant from the Institute of Informatics, Silesian University of Technology and by the POIG.02.03.01-24-099/13 Grant: GeCONiI – Upper-Silesian Center for Scientific Computations.

References

[1] I. Daubechies, W. Sweldens, Factoring wavelet transforms into lifting steps, J. Fourier Anal. Appl. 4 (1998) 247–269, http://dx.doi.org/10.1007/BF02476026.
[2] H.S. Malvar, G.J. Sullivan, S. Srinivasan, Lifting-based reversible color transformations for image compression, in: Proceedings of SPIE, Applications of Digital Image Processing XXXI, 7073 (2008) 707307, http://dx.doi.org/10.1117/12.797091.
[3] R. Starosolski, Reversible denoising and lifting based color component transformation for lossless image compression, arXiv:1508.06106 [cs.MM], available: ⟨http://arxiv.org/abs/1508.06106⟩, (accessed 28.08.15).
[4] Information Technology – JPEG 2000 image coding system: core coding system, ISO/IEC International Standard 15444-1 and ITU-T recommendation T.800, 2004.
[5] T. Bruylants, A. Munteanu, P. Schelkens, Wavelet based volumetric medical image compression,, Signal Process. Image Commun. 31 (2015) 112–133, http://dx.doi.org/10.1016/j.image.2014.12.007.

Table 5
Comparison of RDLS-modified JPEG 2000 with JPEG-LS and HEVC.

Compression methods	Photo	No-photo	No-photo (a)	No-photo (b)	All
r_{DWT}	3.9975	2.9162	3.2882	2.7348	3.6395
$\Delta r_{\mathrm{RDLS(2,1),\ NO\text{-}DWT}}$	−0.57%	−14.00%	−31.37%	−5.53%	−5.02%
$\Delta r_{\mathrm{JPEG\text{-}LS}}$	−2.72%	−21.46%	−38.27%	−13.26%	−8.92%
Δr_{HEVC}	9.32%	−17.04%	−36.48%	−7.55%	0.59%

r_{DWT} – lossless JPEG 2000 bitrate for 3-level unmodified DWT; $\Delta r_{\mathrm{RDLS(2,1),\ NO\text{-}DWT}}$ – average bitrate change obtained by picking for each image the best bitrate out of bitrates obtained by skipping the DWT stage of JPEG 2000 and by RDLS–DWT using Median 5×5 and None denoising filters adaptively selected in single iteration of heuristic step B; $\Delta r_{\mathrm{JPEG\text{-}LS}}$ – average bitrate change obtained by using JPEG-LS instead of JPEG 2000; Δr_{HEVC} – average bitrate change obtained by using HEVC.

[6] C. Christopoulos, A. Skodras, T. Ebrahimi, The JPEG2000 still image coding system: an overview, IEEE Trans. Consum. Electron. 46 (4) (2000) 1103–1127, http://dx.doi.org/10.1109/30.920468.
[7] S.G. Mallat, A theory for multiresolution signal decomposition: the wavelet representation, IEEE Trans. Pattern Anal. Mach. Intell. 11 (1989) 674–693, http://dx.doi.org/10.1109/34.192463.
[8] R. Starosolski, New simple and efficient color space transformations for lossless image compression, J. Vis. Commun. Image Represent. 25 (5) (2014) 1056–1063, http://dx.doi.org/10.1016/j.jvcir.2014.03.003.
[9] T. Strutz, Multiplierless reversible colour transforms and their automatic selection for image data compression, IEEE Trans. Circuits Syst. Video Technol. 23 (7) (2013) 1249–1259, http://dx.doi.org/10.1109/TCSVT.2013.2242612.
[10] R.C. Hardie, K.E. Barner, Rank conditioned rank selection filters for signal restoration, IEEE Trans. Image Process. 3 (2) (1994) 192–206, http://dx.doi.org/10.1109/83.277900.
[11] Free grayscale image transformation tool "pgmtool", version 0.72, available: ⟨http://sun.aei.polsl.pl/~rstaros/imgtransf/pgmtool/index.html⟩, (accessed 28.07.15).
[12] Set "Grayscale Set 2" of grayscale images from the University of Waterloo, Fractal Coding and Analysis Group, available: ⟨http://links.uwaterloo.ca/Repository.html⟩, (accessed 20.05.15).
[13] Set "CT2" of color images from the Deutsche Telekom, Hochschule für Telekommunikation Leipzig, Institute of Communications Engineering, available: ⟨http://www1.hft-leipzig.de/strutz/Papers/Testimages/CT2/⟩, (accessed 20.05.15).
[14] T. Strutz, A. Leipnitz, Reversible Colour Spaces without Increased Bit Depth and Their Adaptive Selection, IEEE Signal Process. Lett. 22 (9) (2015) 1269–1273, http://dx.doi.org/10.1109/LSP.2015.2397034.
[15] Information Technology – JPEG 2000 image coding system: extensions for three-dimensional data, ISO/IEC International Standard 15444-10 and ITU-T recommendation T.809, 2011.
[16] IRIS-JP3D, JPEG 2000 part 10 reference software, version 1.1.1, available: ⟨http://www.irissoftware.be/⟩, (accessed 20.05.15).
[17] M.J. Weinberger, G. Seroussi, G. Sapiro, The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS, IEEE Trans. Image Process. 9 (8) (2000) 1309–1324, http://dx.doi.org/10.1109/83.855427.
[18] Information technology – Lossless and near-lossless compression of continuous-tone still images – Baseline, ISO/IEC International Standard 14495-1 and ITU-T Recommendation T.87, 2006.
[19] G.J. Sullivan, J.R. Ohm, W.J. Han, T. Wiegand, Overview of the high efficiency video coding (HEVC) standard, IEEE Trans. Circuits Syst. Video Technol. 22 (12) (2012) 1649–1668, http://dx.doi.org/10.1109/TCSVT.2012.2221191.
[20] Information technology – High efficiency coding and media delivery in heterogeneous environments – Part 2: high efficiency video coding, ISO/IEC International Standard 23008-2 and ITU-T Recommendation H.265, 2015.
[21] Signal Processing and Multimedia Group, University of British Columbia JPEG-LS implementation, version 2.2, available: ⟨http://www.stat.columbia.edu/~jakulin/jpeg-ls/mirror.htm⟩, (accessed 28.07.15).
[22] HEVC Test Model (HM) reference software (ISO/IEC International Standard 23008-5 and ITU-T Recommendation H.265.2), version 16.6, revision 4573, available: ⟨https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/⟩, (accessed 28.07.15).
[23] R. Starosolski, Compressing high bit depth images of sparse histograms, in: Proceedings of AIP, International Electronic Conference on Computer Science IeCCS 2007 Part II, 1060, 2008, pp. 269–272, http://dx.doi.org/10.1063/1.3037069.
[24] T. Bernas, R. Starosolski, R. Wójcicki, Application of detector precision characteristics for the denoising of biological micrographs in the wavelet domain, Biomed. Signal Process. Control. 19 (2015) 1–13, http://dx.doi.org/10.1016/j.bspc.2015.02.010.
[25] S. Srinivasan, C. Tu, S.L. Regunathan, G.J. Sullivan, HD Photo: a new image coding technology for digital photography, in: Proceedings of SPIE Applications of Digital Image Processing XXX, 6696, 2007, p. 66960A, http://dx.doi.org/10.1117/12.767840.
[26] Information technology – JPEG XR image coding system – Image coding specification, ISO/IEC International Standard 29199-2 and ITU-T Recommendation T.832, 2012.
[27] Q. Cai, L. Song, G. Li, N. Ling, Lossy and lossless intra coding performance evaluation: HEVC, H.264/AVC, JPEG 2000 and JPEG LS, in: 2012 Asia-Pacific Signal & Information Processing.