Chapter 5

Lossy-to-Lossless Compression of Biomedical Images Based on Image Decomposition

Luís M. O. Matos, António J. R. Neves and Armando J. Pinho

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/60650

Abstract

The use of medical imaging has increased in the last years, especially with magnetic resonance imaging (MRI) and computed tomography (CT). Microarray imaging and images that can be extracted from RNA interference (RNAi) experiments also play an important role for large-scale gene sequence and gene expression analysis, allowing the study of gene function, regulation, and interaction across a large number of genes and even across an entire genome. These types of medical image modalities produce huge amounts of data that, for several reasons, need to be stored or transmitted at the highest possible fidelity between various hospitals, medical organizations, or research units.

In this chapter, we study the performance of several compression methods developed by the authors, as well as of image coding standards, when used to compress medical images (computed radiography, computed tomography, magnetic resonance, and ultrasound), RNAi images, and microarray images. The compression algorithms addressed are based on image decomposition, finite-context modeling, and arithmetic coding. In one of the methods, the input image is split into several bitplanes, and each bitplane is encoded using finite-context models and arithmetic coding. In another approach, the intensity levels of a given image are organized in a binary-tree structure, where each leaf node is associated with an image intensity.

The experimental results presented in this chapter are state of the art regarding the compression of some of these types of images. Moreover, several approaches and preprocessing techniques are presented, giving a good hint about new developments that can be studied further. Also, this chapter intends to be used as a reference for comparison with new compression algorithms that may be developed in the future.

Keywords: Image coding, Lossless coding, Progressive decoding, Biomedical images, Image coding standards, Image decomposition, Arithmetic coding, Finite-context modeling

1. Introduction

Image compression is a very important research field. It is fundamental in many different areas, such as biomedical imaging, consumer electronics, and Internet, among others. The goal of an image compression method is to represent an arbitrary image using the smallest possible number of bits.

The use of medical imaging has increased in the last years, especially with magnetic resonance imaging (MRI) and computed tomography (CT) [1]. These types of medical image modalities produce huge amounts of data that, for several reasons, need to be stored or transmitted with the highest possible fidelity between various hospitals and medical organizations.

Also related to biomedical imaging, microarray images play an important role for large-scale gene sequence and gene expression analysis, allowing the study of gene function, regulation, and interaction across a large number of genes and even across an entire genome [2, 3]. The output of a microarray experiment is a pair of 16-bits-per-pixel images, usually with a very high resolution, sometimes exceeding 13,000×4,000 pixels. Consequently, over 200 MB can be required to store a single microarray image. Due to the development of these digital imaging technologies, some concerns appeared regarding efficient ways of storing and transmitting the images. In order to overcome these problems, sophisticated compression methods are required.

Another type of images that are addressed in this chapter are those that can be extracted from RNA interference (RNAi) experiments. Those experiments involve marking cells with various fluorescent dyes to capture three components of interest, namely, DNA, actin, and PH3 channels [4].

Typically, lossless compression algorithms are recommended for dealing with these types of images. In fact, lossless methods are generally required in applications where cost, legal issues, and value play a decisive role, such as in medical imaging or in image archiving [5]. On one hand, the use of lossless algorithms avoids problems of losing diagnostic information vital to identify life-threatening illnesses in the early stages. On the other hand, if a given image is lossless compressed, it is possible to recompress it in the future, using a more efficient algorithm and without losing any information.

Usually, the performance of image coding standards, such as JPEG, JBIG, JPEG-LS, and JPEG2000, falls short when they are applied to certain types of biomedical images. In order to overcome this lower-performance issue, specific compression algorithms are essential. In the class of lossless compression algorithms, there is a specific type that has progressive decoding capabilities. This type of compression is usually designated as "lossy-to-lossless compression." These algorithms are very flexible, because they allow stopping decoding at a certain point, according to the available resources or user requirements. However, the original image can also be obtained, without any loss, if the decoding process goes until the end.

In this chapter, we study the performance of several specific compression methods developed by the authors, as well as of image coding standards, when applied to medical images (computed radiography, computed tomography, magnetic resonance, and ultrasound), RNAi images, and microarray images. The addressed compression algorithms are based on image decomposition, finite-context models, and arithmetic coding. We can divide these algorithms into two categories. In the first one, the input image is split into several bitplanes and each bitplane is encoded individually. This approach is combined with several preprocessing techniques in order to improve the compression efficiency. In the second approach, the intensity levels of a given image are organized in a binary-tree structure, where each leaf node is associated with an image intensity.

We start by presenting the compression algorithms, namely, bitplane decomposition and binary-tree decomposition. We include a brief explanation about finite-context models and arithmetic coding, the entropy coding block used in both approaches. Then, we present a set of experiments that have been performed using the compression algorithms described in Sect. 4, as well as experimental results using the most important image coding standards (e.g., PNG, JBIG, JPEG-LS, and JPEG2000). At the end, we draw some conclusions.

This chapter intends to be a reference for comparison of new compression algorithms that may be developed in the future, for two main reasons. On one hand, the experimental results presented in Sect. 5.2 are state of the art regarding the compression of some of these types of images. On the other hand, several approaches and preprocessing techniques are presented, giving hints about new developments that can be done.

2. Compression methods

The compression algorithms that we address here can be classified into two different categories. One based on bitplane decomposition and the other one based on binary-tree decomposition. For the first category, we use a sophisticated bitplane decomposition approach that was successfully developed for the compression of microarray images [6]. Furthermore, we include some preprocessing techniques, such as segmentation and histogram reduction, in order to improve the compression results. An alternative bitplane decomposition approach based on *bit modeling by pixel value estimates* is also considered.

The other approach, based on binary-tree decomposition, was developed with success for the compression of medical images [7] and microarray images [8]. In this decomposition approach, the intensity levels of a given image are organized in a binary-tree structure, where each leaf node is associated with an image intensity.

In the following subsections, we address these two decomposition approaches and give a brief explanation regarding finite-context models and arithmetic coding. The implementation of the compression algorithms can be found in [9].

2.1. Finite-context models

Markov modeling is widely used in several fields, including image [7, 8, 10, 11, 12] and DNA [13, 14, 15] compression. Finite-context models rely on the Markov property, since an order- k

finite-context model gives a probability distribution for the next symbol, in a sequence of symbols from an alphabet A, taking into account a recent past of depth k. Hence, the finite-context model assigns probability estimates for each symbol, regarding the next outcome, according to a conditioning context, $c_{k,n}$, computed over a finite and fixed number $k>0$ of past outcomes $c_{k,n}=x_{n-k+1\dots n}=x_{n-k+1}\dots x_n$ (order- k finite-context model [16, 17, 18] with $\mid \mathrm{A} \mid^k$ states). In the example illustrated in Fig. 1, where A0,1 and then $\mid A \mid =2$, an order- k model implies having 2^k conditioning states. In this case, $k=5$, so the number of conditioning states is $2^5=32$.

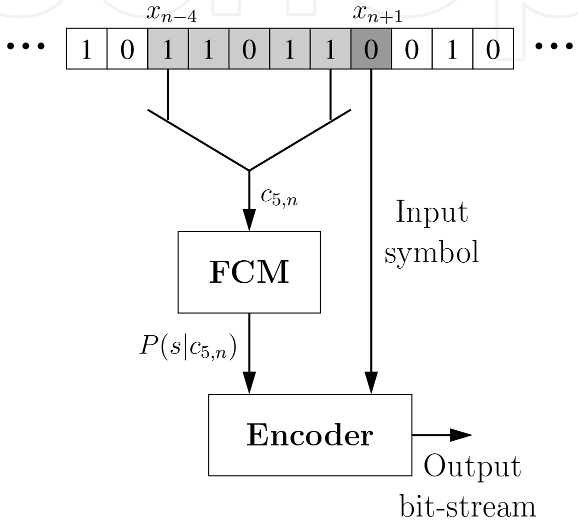


Figure 1. Finite-context model: the probability of the next outcome x_{n+1} is conditioned by the k last outcomes. In this example, $k=5$

The probability estimates $P(x_{n+1} \mid x_{n-k+1\dots n})$ are calculated using symbol counts that are accumulated while processing each pixel of the input image, making them dependent not only on the past k symbols but also on n. The estimator used is

$$P\left(s \mid x_{n-k+1\dots n}\right)=\frac{C\left(s \mid x_{n-k+1\dots n}\right)+\alpha}{C\left(x_{n-k+1\dots n}\right)+|\mathcal{A}|\alpha}, \tag{1}$$

where $C(s \mid x_{n-k+1\dots n})$ represents the number of times that, in the past, symbol s was found having $c_{k,n}=x_{n-k+1\dots n}$ as the conditioning context and where

$$C\left(x_{n-k+1\dots n}\right)=\sum_{a\in\mathcal{A}} C\left(a \mid x_{n-k+1\dots n}\right) \tag{2}$$

is the total number of events that has occurred so far in association with context $c_{k,n}$. Parameter α allows balancing between the maximum likelihood estimator and a uniform distribution (when the total number of events, n, is large, it behaves as a maximum likelihood estimator). For $\alpha=1$, (1) is the well-known Laplace estimator.

Initially, all counters are set to zero, i.e., the symbols are assumed to be equally probable. The counters are updated each time a symbol is encoded. However, it is possible to update the counters according to a specific rule. Since the context templates are causal, the decoder is able to reproduce the same probability estimates without needing additional side information.

Table 1 presents a simple example of how a finite context is typically implemented. In this example, we are dealing with an order-5 finite-context model, which means that the context uses the last five encoded symbols to assign the symbol probabilities. Each row of Table 1 represents a probability model that is used to encode the current symbol, using the last five encoded ones. For example, if the last symbols were "11000," i.e., $c_{5,n}$=11000, then the model sends the following probabilities to the arithmetic encoder (denoted as "Encoder" in Fig. 1): $P0$ and $P1$.

Context, $c_{5,n}$	$C(0\|c_{5,n})$	$C(0\|c_{5,n})$	$C(c_{5,n})$
00000	23	41	64
⋮	⋮	⋮	⋮
00110	14	34	48
⋮	⋮	⋮	⋮
01100	25	12	37
⋮	⋮	⋮	⋮
11000	28	41	69
⋮	⋮	⋮	⋮
11111	8	2	10

Table 1. A simple example illustrating how finite-context models are implemented. The rows of the table correspond to probability models at a given instant n. In this example, the particular model that is chosen for encoding a symbol depends on the last five encoded symbols (order-5 context)

One important aspect that must be considered is the size of the context. For an order- k model and ∣A2 (binary alphabet), the table has 2^k entries and, therefore, its size grows exponentially with k. Using a deeper context, we might achieve higher performance, but this requires also more memory.

2.1.1. Mixtures of finite-context models

In order to attain better compression results, the previous approach can be used in a more robust scheme. The goal here is to use a model mixture with more than one finite-context model. In our case, we decided to use only two different models: the one used by [6] and another one based on *bit modeling by pixel value estimate*. This approach could be extended to use more models, but at a cost of some computation time. In what follows, we will describe how this mixture scheme works.

The per symbol information content average provided by the finite-context model of order-k, after having processed n symbols, is given by

$$H_{k,n} = -\frac{1}{n}\sum_{i=0}^{n-1}\log_2 P\left(x_{i+1} \mid x_{i-k+1\ldots i}\right) \tag{3}$$

bits per symbol (or bits per pixel, bpp, in the case of images). Hence, the $H_{k,n}$ can be viewed as a measure of the performance of the model until that instant. When using several models simultaneously, the probability estimate can be given by a weighted average of the probabilities provided by each model, according to

$$P\left(x_{n+1}\right) = \sum_k P\left(x_{n+1} \mid x_{n-k+1\ldots n}\right) w_{k,n}, \tag{4}$$

where $w_{k,n}$ denotes the weight assigned to model k and

$$\sum_k w_{k,n} = 1. \tag{5}$$

In order to compute the probability estimate for a certain symbol, it is necessary to combine the probability estimates given by (1) using (4). The weight assigned to model k can be computed according to

$$w_{k,n} = P\left(k \mid x_{1\ldots n}\right), \tag{6}$$

i.e., by considering the probability that model k generated the sequence until that point. In that case, we would get

$$w_{k,n} = P\left(k \mid x_{1\ldots n}\right) \propto P\left(x_{1\ldots n} \mid k\right) P\left(k\right), \tag{7}$$

where $P(x_{1\ldots n} \mid k)$ denotes the likelihood of sequence $x_{1\ldots n}$ being generated by model k and $P(k)$ denotes the prior probability of model k. Assuming

$$P\left(k\right) = \frac{1}{K}, \tag{8}$$

where K denotes the number of models, we also obtain

$$w_{k,n} \propto P\left(x_{1\ldots n} \mid k\right). \tag{9}$$

Calculating the logarithm, we get

$$\begin{aligned} \log_2 P\left(x_{1\ldots n} \mid k\right) = \log_2 \prod_{i=1}^{n} P\left(x_i \mid k, x_{1\ldots i-1}\right) = \quad & \text{(a)} \\ = \sum_{i=1}^{n} \log_2 P\left(x_i \mid k, x_{1\ldots i-1}\right), \quad & \text{(b)} \end{aligned} \tag{10}$$

which is related to the code length that would be required by model k for representing the sequence $x_{1\ldots n}$. It is, therefore, the accumulated measure of the performance of model k until instant n. In order to improve the global performance, we decided to use a mechanism that progressively forgets past performances of the models. This mechanism allows each model to progressively forget the past and, consequently, to give more importance to the most recent past. Therefore, we rewrite (11) as

$$\begin{aligned} \sum_{i=1}^{n} \log_2 P\left(x_i \mid k, x_{1\ldots i-1}\right) = \quad & \text{(a)} \\ = \gamma \sum_{i=1}^{n-1} \log_2 P\left(x_i \mid k, x_{1\ldots i-1}\right) + \log_2 P\left(x_n \mid k, x_{1\ldots n-1}\right), \quad & \text{(b)} \end{aligned} \tag{11}$$

where $\gamma \in 0,1$ dictates the forgetting factor to be used. Defining

$$p_{k,n} = \prod_{i=1}^{n} P\left(x_i \mid k, x_{1\ldots i-1}\right) \tag{12}$$

and removing the logarithms, we can rewrite () as

$$p_{k,n} = p_{k,n-1}^{\gamma} P\left(x_n \mid k, x_{1\ldots n-1}\right) \tag{13}$$

and, finally, set the weights to

$$w_{k,n} = \frac{p_{k,n}}{\sum_k p_{k,n}}. \tag{14}$$

Usually, a compression algorithm can be divided into two parts, modeling and coding. The Markov models are responsible for providing a statistical model as reliable as possible to be used later in the coding stage. The coding stage is where the statistical model is used to compress the data. Arithmetic coding is usually the technique that is used. In the following section, we give a brief explanation on how arithmetic coding works.

2.1.2. Arithmetic coding

Arithmetic coding is a compression technique developed by Rissanen [19] in the late 1970s. This method is a good alternative to Huffman coding [20], because it usually generates better compression results. In order to obtain better results, an appropriate probability estimator must be used in the arithmetic encoder (e.g., based on finite-context models, as described above).

This method represents a set of symbols using a single number in the interval [0,1). As the number of symbols of the message grows, the initial interval [0,1) will shrink and the number of bits necessary to represent the interval will increase. When we are processing the pixels of an image in a raster scan order, the probabilities of the intensities of the pixels are conditioned by the context determined by a combination of the already encoded neighboring pixels. The encoder and the decoder estimate this context model dynamically adapting it to the input data, during the encoding/decoding process. According to [16, 17, 18], this arithmetic encoding method generates output bitstreams with average bitrates almost identical to the entropy of the source model.

2.2. Bitplane decomposition

The technique to separate an image into different planes (bitplanes), known as bitplane decomposition, plays an important role in image compression. Usually, each pixel of a grayscale image is represented by 8 bits, or 16 bits as the case of some biomedical images used in the experimental results presented in this chapter. Suppose that the image has $N \times M$ pixels and each one is composed of eight bitplanes, ranging from bitplane 0 for the least significant bitplane (LSBP) to bitplane 7 for the most significant bitplane (MSBP). In fact, plane 0 contains all the lowest-order bits in the bytes comprising the pixels in the image as well as plane 7 holds the most significant bits [21]. Figure 2 illustrates these ideas and Fig. 3 shows the various bitplanes for the image presented on the left. As we can see, the MSBPs (especially 7-4) contain the majority of the visually significant data. On the other hand, the lower planes (namely, planes 0-3) contribute to more subtle details in the image.

The bitplane decomposition technique is very useful on image compression. On one hand, it allows some bi-level compression methods, such as JBIG, to be applied to typical grayscale images. The compression method is applied to each bitplane after the decomposition. On the other hand, it is possible to create sophisticated models that take advantage of this decomposition. For instance, it is possible to use information of the previous bitplanes (usually the MSBPs) to improve the compression performance of the LSBPs.

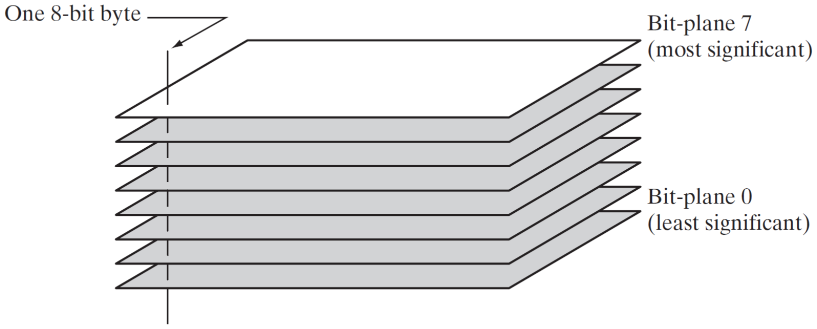


Figure 2. Bitplane representation of an eight-bit image [21]

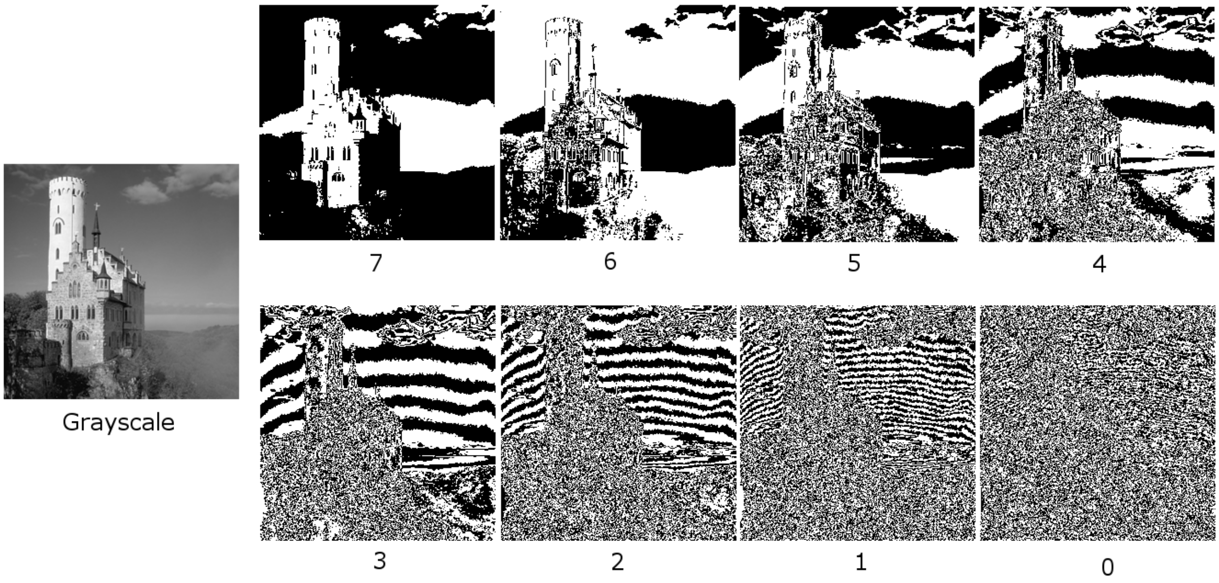


Figure 3. An 8-bit grayscale image and its eight bitplanes. The numbers at the bottom of each image identify the bitplane, where 0 denotes the less-significant plane and 7 the most significant plane (adapted from [22])

In [23], an embedded image-domain adaptive compression (EIDAC) scheme used with success on simple images (with a reduced number of active intensity values) is presented. EIDAC uses a context-adaptive bitplane coder, where each bitplane is encoded using a binary arithmetic coder. Later, in [24], a compression method inspired from EIDAC for microarrays image is proposed. The same authors improved their method using a more robust 3D context modeling [24]. We used this last approach in our experiments and also other alternative versions using some preprocessing techniques, namely, segmentation and bitplane reduction.

2.2.1. Segmentation

In the literature, we can find several algorithms for image segmentation. In our approach, we decided to use a threshold-based method inspired from the work presented in [25]. The segmentation is attained by means of a dynamic thresholding scheme. By applying a threshold value, the pixels of the image can be split in two sets (background and foreground). For each threshold value, it is possible to obtain the number of pixels and the standard deviation of the intensities of these pixels in each set. The desired threshold is calculated according to (17). Using (17), it is always guaranteed that the weighted sum of the standard deviation of both the background and foreground is minimal,

$$\mathcal{T} = \text{argmin}\left\{f\left(T\right)\right\}, \; T = \left\{t \in \mathrm{N} \mid 0 \le t \le 2^{16} - 1\right\}, \tag{15}$$

where

$$f\left(T\right) = \text{stdev}\left(B_T\right) \times \text{size}\left(B_T\right) + \text{stdev}\left(F_T\right) \times \text{size}\left(F_T\right), \tag{16}$$

$B_T\, p \in \text{Image}\, p < T\}$ represents the set of pixels in the background section, $F_T\, p \in \text{Image}\, p \ge T\}$ represents the set of pixels in the foreground section, stdev(x) is the standard deviation of x, and size(y) is the number of pixels of set y. Instead of testing all possible threshold values to find the minimum value of $f(T)$, a recursive search algorithm is used to accelerate the search routine. It is possible to use this recursive search algorithm because $f(T)$ plunges down at a certain threshold value, which is chosen as the final threshold value. After the threshold search is completed, a binary map is created where the foreground pixels will be set to "1" and the background pixels to "0."

2.2.2. Bitplane reduction

Bitplane reduction is an interesting method that can further improve compression efficiency by eliminating redundancy in the pixel precision for simple images. Simple images are images where the number of different intensities that occur is very small, compared to the total number of possible intensities. For example, if we have only 24 different intensities out of 256 for an 8-bit image, we only need 5 bits to represent each pixel intensity. This means that, when we are encoding the image, we only need to encode five bitplanes instead of the original eight bitplanes.

Yoo et al. [26] have shown that it is possible to obtain compression gains using the simplest form of bitplane reduction, known as histogram compaction (HC). Later, [23] presented a more robust bitplane reduction method called scalable bitplane reduction (SBR). This approach finds the reduced bitplane codeword by growing a binary tree. The method splits each node of the binary tree into two nodes using a simple MINMAX metric to measure the distortion.

In order to better understand the SBR algorithm, we present a small example in Fig. 4. Initially, we associate all the active pixel values to the root node. In this small example, the image only has four active pixel values. Starting in the root node, the algorithm splits all the children sub-nodes using a MINMAX criterion. The split process in the root node starts by computing the value 32,767 from $(0 + 65{,}535$. The computed value is then used to split the node. All the intensities that are lower or equal to 32,767 are inserted in the left sub-node. On the other hand, the remaining intensities (>32,767) are associated with the right sub-node. After splitting the root node, the SBR algorithm adds a zero to the left node codeword and a one to the right node codeword. This splitting process is repeated until all sub-nodes have only one intensity associated with them. In this specific example, "0" is a complete codeword for the original pixel value zero, due to the fact that the first left node or partition does not have more sub-nodes. As a result of the variable-length codewords, the average codeword length can be less than 3 bits. This is useful because during the encoding process, it is possible to skip some bitplanes of the pixel codewords with a lower length.

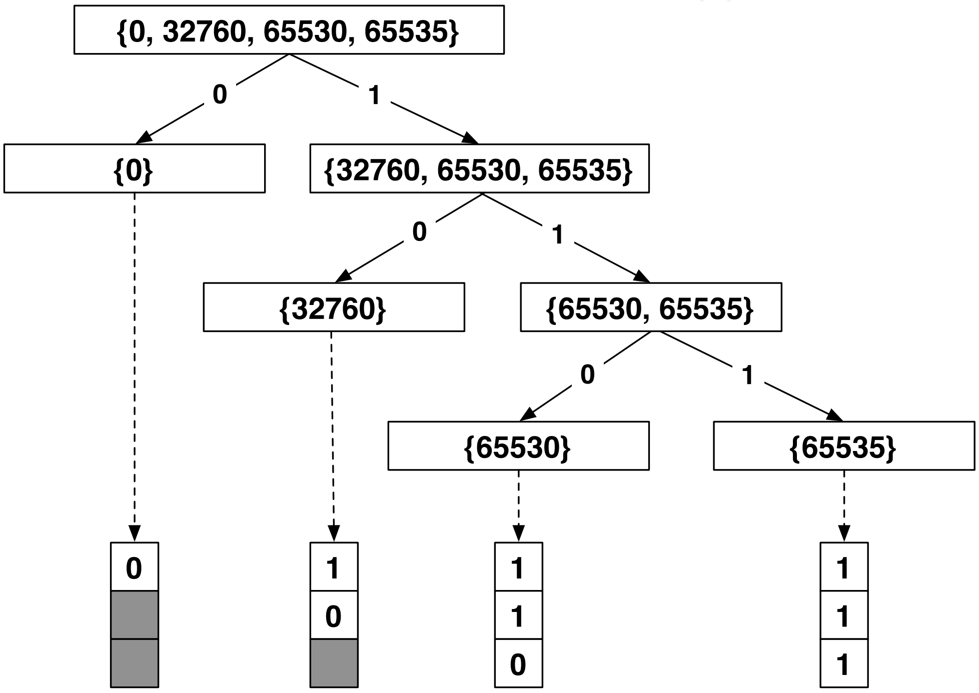


Figure 4. Binary tree to obtain the codewords to represent the active pixel values 0,32,760,65,530,65,535 in the reduced bitplane space after applying the SBR algorithm. Each of the tree nodes is associated with a symbol (an intensity) set to be partitioned, except the end nodes. Each branch defines the bit value of specific symbols at the corresponding bitplane in the reduced bitplane domain. For the intermediate nodes, 32,760,65,530,65,535 and 65,530,65,535, the split process is done using the MINMAX criterion

Taking as example the intensities 0,32,760,65,530,65,535 presented in Fig. 4, we present in Table 2 the codewords for the two histogram reduction methods. As can be seen, the codewords obtained using the HC algorithm are dependent on the number of active intensities. The codeword size can be computed according to

$$S = \lceil \log_2(N) \rceil, \tag{17}$$

where N denotes the number of active intensities. In the example presented in Fig. 4, we have four different intensities, so the codeword size is $\log_2 4=2$. The codeword size is constant for all intensities for the HC method. On the contrary, the resulting codewords obtained using the SBR algorithm have different sizes (see Table 2).

Intensity Value	HC		SBR	
	Codeword	Codeword size	Codeword	Codeword size
0	00	2	0	1
32,760	01	2	10	2
65,530	10	2	110	3
65,535	11	2	111	3

Table 2. A small example showing the differences between the two bitplane reduction methods HC and SBR. The codeword in each column represents the new value that will be assigned to the pixel values of the first column. In HC, the codeword size is constant. In SBR, the codeword size is variable

2.2.3. Simple bitplane coding

In [27], Kikuchi et al. introduced the concept of bit modeling by the pixel value estimates. In their approach, instead of using the true bit values of each bitplane, they used the expectation values of the pixels to build up the contexts. This approach is known as *bit modeling by pixel value estimate*. They extended their work more recently to be applied to various types of images (color, grayscale, color-quantized, bi-level, and halftone) [28] and for HDR (high-dynamic-range) images [29].

Let us consider that a given pixel value at location (i, j) in a given image to be encoded is denoted as $x(i, j)$. Its decoded value is denoted by $y(i, j)$. In order to facilitate the explanation, the location indexes (i, j) are omitted from now on. A typical raster scan order is used to process each pixel of a given image. Similar to Kikuchi's method, as the process of the bitplane coding proceeds to lower bitplanes, the decoded value, y, of the target pixel approaches the true pixel value $x = (x_{16}\, x_{15}\, \ldots\, x_1)$ where x_n denotes the n^{th} bit of x in the case of a 16-bit grayscale image.

Contrarily to Kikuchi's method, our approach uses only one type of context, denoted as *neighborhood context* in Kikuchi's work. The contexts are built by the estimates of partially decoded pixels based on the template depicted in Fig. 5. The pixel location of x is labeled by "X" on the 15-pixel template in Fig. 5, and

$$c_k = \begin{cases} 1, & \text{for } y(k) > y \\ 0, & \text{otherwise} \end{cases}, \tag{18}$$

where $k \in \{1, 2, \ldots, 15$ denotes the spatial location on the template illustrated in Fig. 5. $y(i)$ and y represent the most recent estimates of the neighboring pixels and the target pixel, respectively.

		13	9	12	
	14	5	2	6	8
15	7	1	X	3	
		10	4	11	

Figure 5. Fifteen-pixel template for building up the context. The target pixel to be encoded is labeled by an "X"

In Kikuchi's method, the inter-bit correlation on a bitplane is not used. Instead, for modeling a target bit, the authors used the pixel value estimates of which more significant bits have been already available at the decoder. Their method is referred to as *bit modeling by pixel values*, where the pixel value estimates are used rather than of the unknown true values at the decoder. Contrarily to Kikuchi's method that uses a nine-pixel template, our approach uses a variable-size 15-pixel template (see Fig. 5). In our case, a greedy search routine is performed in each bitplane in order to obtain the context size that attains the best compression performance (lower bits per pixel as possible). According to Kikuchi's method, the decoded pixel values are spatially correlated to each other as significantly high as the true pixel values which will make sense to use a larger template. Furthermore, since the alphabet size is only two, the probability of having context dilution is low. Similar to Kikuchi's method, we are considering some noncausal locations with respect to the scanning order of the pixels (locations $y3y4y10$ and $y11$ in Fig. 5). The usage of noncausal pixels is only possible because the context bits are defined by using the estimates of pixel values, which are available in the decoder.

Let us consider that we are coding an N -bit depth image, where $N \leq 16$. Suppose that the n^{th} bitplane is being encoded at present, where $n \in \{1 \ldots N\}$. For every pixel, the higher bits from the $(n+1)^{\text{th}}$ until N^{th} bitplane are known at the decoder. The other lower n bits are unknown. The value of the unknown part can be distributed over the interval of $[0, 2^n - 1]$. Similar to Kikuchi's method, the values zero and one occur with equal probability in the unknown less significant n bits. Under this assumption, the pixel value estimate of the target pixel is expressed by

$$y^{(n)} = \left\lfloor \frac{y}{2^n} \right\rfloor + 2^{n-1} - 1 \tag{19}$$

at the n^{th} bitplane encoding/decoding, where y is the latest decoded value and . denotes truncation. In Fig. 6, we illustrate an example of a binary representation of the pixel value estimate in the case $n = 8$.

Initially, all the pixel estimate values are set to $y = 2^{N-1} - 1$. The encoding procedure starts at the MSBP and stops at the LSBP. Assuming a target pixel x_n of bitplane n is the one being encoded, under the context of $\{c_k\}$, the pixel estimate y of the target pixel is immediately updated by a simple bit operation as

$$y \leftarrow y + x_n 2^{n-1} - \left\lfloor 2^{n-2} \right\rfloor. \tag{20}$$

The pixel value estimate is used in a coming chance of reference and will be the decoded pixel value, when the decoding is stopped (after all the bitplanes are processed). The most recent estimate of a given pixel is always made up of two parts: its significant bits are those already encoded/decoded true bits and the other less significant bits are 0 (zero) followed by a successive 1s (ones). The value of the less significant bits is equal to the expectation value of the unknown lower bits, if binary symbols of zero and one are assumed to occur in those bits with equal probability.

$y^{(8)} =$	x_{16}	x_{15}	x_{14}	x_{13}	x_{12}	x_{11}	x_{10}	x_9	0	1	1	1	1	1	1	1

Figure 6. Binary representation of a pixel value estimate in the case of *n*=8

2.3. Binary-tree decomposition

Binary trees are also an important data structure that can be used in several algorithms. In the case of image compression, we can associate each leaf node of the binary tree to an image intensity. The binary tree can be viewed as a simple nonlinear generalization of lists; instead of having one way to continue to another element, there are two alternatives that lead to two different elements [30]. Every node (or vertex) in an arbitrary tree has at least two children (see Fig. 7). Each child is designated as left child or right child, according to the position in relation to the tree root.

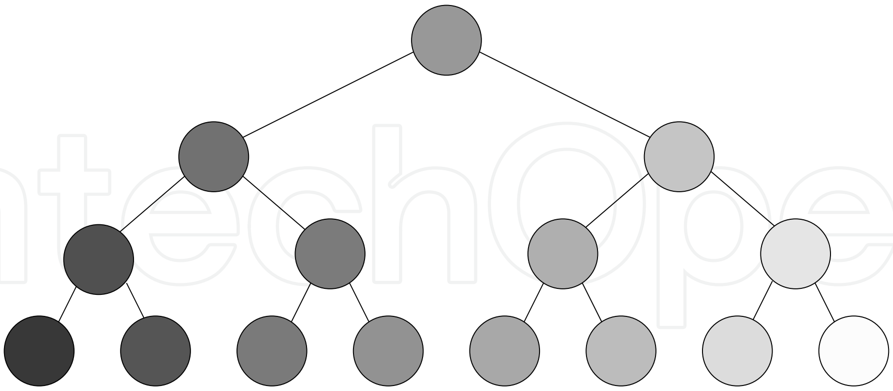


Figure 7. An example of a binary tree with eight intensities or gray levels. Each internal node contains an intensity representative that is computed according to the set of intensities that are associated to the node. The leaf nodes represent the number of different intensities that actually occur; in this case, we have a total of eight intensities

One of the first methods where binary trees were used for image compression was proposed by Chen et al. [31] regarding the compression of color-quantized images. Chen's method uses a binary-tree structure of color indexes instead of a linear list structure. Using this binary-tree structure, Chen's method can progressively recover an image from two colors to all of the colors contained in the original image. Inspired by the work done by [31], a few years later, the authors of [7, 11, 12] developed a lossy-to-lossless method based on binary-tree decomposition and context-based arithmetic coding. In the last approach, the authors studied the performance of their method in several kinds of grayscale images, including medical images. As can been seen, this decomposition approach is very versatile because it can be applied in color-quantized images and also in grayscale images.

This binary-tree decomposition approach was intended to be used in images with a small number of intensities, usually with eight or less bits per pixel, due to a tight relation between the processing time and the number of different intensities of the image. In this work, we intend to study the performance of this approach to compress biomedical images, such as microarray images, RNAi, etc. Another interesting feature of this approach is its capability of progressive decoding, which means that the decoding process can be stopped at any moment according to a specific distortion metric, obtaining an image with some loss. Moreover, it is possible to obtain the original image without any loss if the full decoding process is performed. In the next two sections, we describe in more detail the compression algorithm that is based on a binary-tree decomposition and context-based arithmetic coding.

2.3.1. Hierarchical organization of the intensity levels

This method is based on a hierarchical organization of the intensity levels of the image. This organization of the intensity levels is attained by means of a binary tree. Each node of the binary tree, n, represents a certain subset, S^n, of the intensities of the image. The root node contains all active pixel values of the image $\mathcal{I}I_1,I_2,\ldots,I_N\}$, where N represents the number of different intensities that occur in the image. Therefore, $\mathsf{S}^n \subset \mathcal{I}$ and $\mathsf{S}^1 \equiv \mathcal{I}$. Each node possesses a representative intensity, I^n, given by

$$I^n = \left\lfloor \frac{I_m^n + I_M^n}{2} \right\rfloor, \tag{21}$$

where I_m^n and I_M^n are, respectively, the smallest and largest pixel value in S^n and where x denotes the largest integer less than or equal to x. Computing the value of I^n according to (23) leads to the smallest possible L_{∞} reconstruction error when the intensities associated to node n (those in S^n) are all substituted by I^n. The error is given by

$$\varepsilon_{\infty}^n = I_M^n - I^n. \tag{22}$$

In order to better understand the construction of the binary tree, we present in Fig. 7 a small example for an image with only five active pixel values 32,50,250,33,768,65,530. The construction of this tree begins with the association to the root node of the set of intensities that occur in the original image. After this association, it is necessary to compute I^1 according to (23). In the example depicted in Fig. 8, $I^1 = \lfloor (32 + 65{,}530) / 2 \rfloor = 32{,}781$ and $\varepsilon_\infty^1 = 65{,}530 - 32{,}781$, for the root node. The next step consists of splitting the root node into two sub-nodes and, therefore, splitting S^1 into two subsets. In order to split S^1, we need only to compare the intensity $I \in \mathsf{S}^1$ with I^1. The intensities lower than I^1 are associated with the left node and the other ones with the right one. This procedure is repeated until all nodes are expanded, i.e., until having a tree with N leaves (N is the number of active intensities presented in the original image). The next node to expand is chosen, taking into consideration the smallest possible L_∞ reconstruction error. In case of a tie, one is arbitrarily chosen, although it is necessary that the decoder picks the same one.

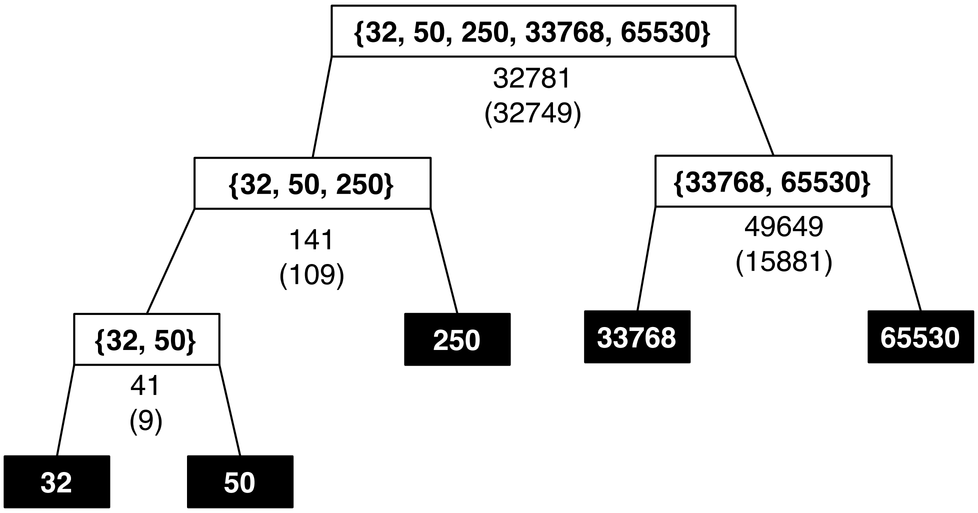


Figure 8. Example of a small binary tree that illustrates the hierarchical organization of the intensity values for an image with five active pixel values 32,50,250,33,768,65,530

In order for the decoder to be able to build the same tree, it is necessary to send the information of the active pixel values to the decoder using a 65,536-bit indicator. In order to encode this indicator, the encoder uses the following strategy. First, the maximum intensity value I_N is sent. After that, a string of I_n bits is transmitted, such that if the n^{th} bit of the string is one, it means that the intensity $n-1$ is present in the image and zero otherwise. The previous 65,536-bit indicator is enough for the decoder to construct exactly the same binary tree.

2.3.2. Encoding pixel locations

After each node is expanded, two new nodes are created, each one with a representative intensity (I_l^n for the left node and I_r^n for the right node). This step can be seen as a region of arbitrary shape, containing zeros (relative to the left node) and ones (regarding the right node), that needs to be communicated to the decoder. The position where the pixels in the image are associated with the parent node that was expanded is known by the decoder. However, it is necessary to communicate to the decoder the zeros and ones that correspond to the pixels that after the expand procedure will be associated to the left and right nodes, respectively. Since the decoder has access to the pixels associated to the parent node that was expanded, it is enough to encode a binary mask, where zero indicates that the pixel needs to change its intensity to I_l^n and one indicates a change to I_r^n. This binary mask is encoded using arithmetic coding based on variable-size finite-context models [16, 17, 18].

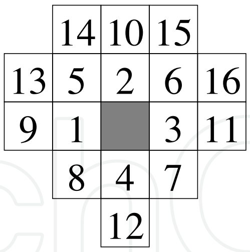
The performance of the compression method is directly dependent on the encoding of these binary masks. The encoding efficiency of these binary masks can be controlled by a carefully chosen context modeling that will then drive the binary arithmetic encoder. The context is constructed based on the template depicted in Fig. 9. The number of context pixels can go up to 16 at most, and they are numbered according to their distance to the encoding pixel (represented in gray in Fig. 9). A particular context is represented using a sequence of bits,

$$b_1 b_2 \ldots b_k \tag{23}$$

where

$$b_i = \begin{cases} 0, & \text{if} \quad \left| I(i) - I_l^n \right| \leq \left| I(i) - I_r^n \right| \\ 1, & \text{otherwise} \end{cases},$$

and where $I(i)$ denotes the intensity of the pixel in the current reconstructed image corresponding to position i of the context template.



	14	10	15	
13	5	2	6	16
9	1	■	3	11
	8	4	7	
		12		

Figure 9. Context template used in this work. The use of noncausal pixels is possible, because context information can be obtained from the previous version of the reconstructed image

The value k defines the model order used. In this case, the k value varies as the encoding proceeds. This variation is necessary in order to improve the compression performance. Furthermore, it is expected to have larger mask regions initially in the first nodes that are expanded and smaller regions when $n \approx N$. This variation is also useful to avoid the problem of context dilution. In this research work, we present two modes of context creation. One is denoted as "greedy" where the context size is first chosen using a k value according to [32]. After that, the method tests incrementally several context sizes bigger and smaller than k and stops when it reaches one context that produces worse results than the previous best. In the end, the algorithm has two context sizes. One attained when applying an increment to k and the other one when applying a decrement to k. The best context size is then chosen to encode the binary mask. The other mode is slower because it tests all possible context sizes. This second mode, denoted as "best," always attains the best context size that minimizes the bitrate. For both cases, the context size needs to be sent to the decoder, for each node that is expanded.

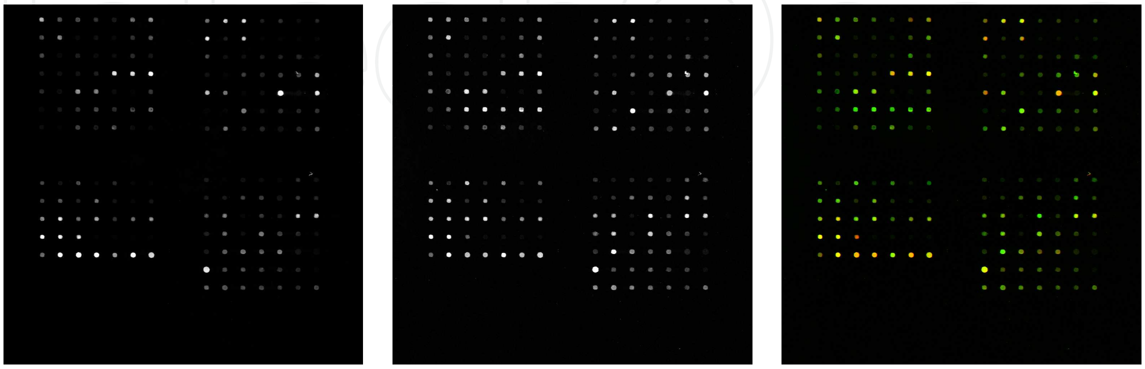
There is also an alternative way to encode the binary mask. If the number of bits required to encode the mask and the context size is bigger than the total number of pixels associated with the node to be expanded, the encoder sent the binary mask as a binary string, without compression. In order for the decoder to differentiate between these two modes, a binary stream is needed to be encoded for each node that is expanded.

3. Evaluation of the compression methods

In this section, we present compression results obtained by the compression methods presented earlier in Sect. 4. We decided to present only the results of the best version of each method in order to avoid extending this section. We start with the description of the data sets used in this work. Then, we provide the obtained compression results. At the end of this section, we present a study of the rate distortion, comparing the two compression approaches described in Sect. 4 and two image coding standards, JBIG and JPEG2000.

3.1. Data sets

To evaluate the compression methods presented in this chapter, we used three types of images: microarray, medical, and RNAi. The output data obtained in a microarray experiment is a pair of 16-bits-per-pixel grayscale images, one from the so-called red channel and the other from the green channel (see Fig. 10). We use a total of 298 microarray images of nine different data sets as described in Table 3.



(a) “Deff661Cy5” (b) “Deff661Cy3” (c) Color version

Figure 10. Example of a microarray experiment from the *ISREC* data set. (a) and (b) correspond to the pair of microarray images. (c) a color version created from (a) as the red channel and (b) as the green channel

Regarding the medical images, we used images of four modalities: computed radiography (CR), computed tomography (CT), magnetic resonance (MR), and ultrasound (US). In Fig. 11, we can find five examples of medical images used in this work. The data set used in [33] was also considered as a medical reference data set. All the medical image data sets are depicted in Table 4, a total of 370 images.

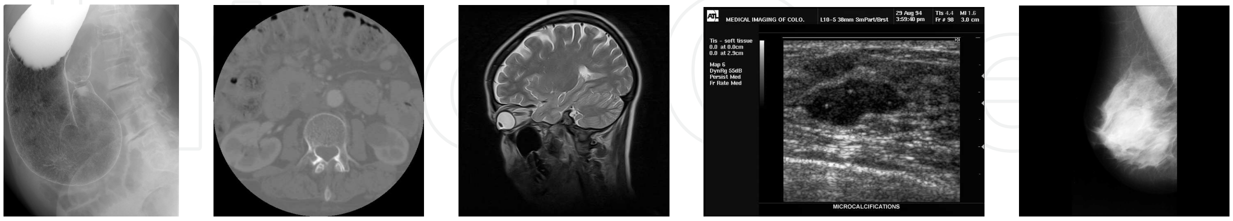

(a) "cr_4503" (b) "ct_17" (c) "mr_2321" (d) "us_3393" (e) "mdb001"

Figure 11. Example of medical images from different modalities

Finally, we also used a recent and popular image type that can be extracted from RNA interference (RNAi) experiments. Those experiments involve marking of cells with various fluorescent dyes to capture three components of interest, namely, DNA, actin, and PH3 channels [4]. In Table 5, we can find the RNAi image data sets used in this work, a total of 34,560 images. The images were retrieved from the RNAi experiments without any kind of transformation (no normalization was applied). For us, it does not make sense to perform a normalization operation in order to reduce the image depth from 10 to 8 bits per pixel. The normalization is a lossy process, and hence, applying a lossless compression method to images that suffered some loss is not appropriate. In Fig. 12 we present some examples of RNAi images.

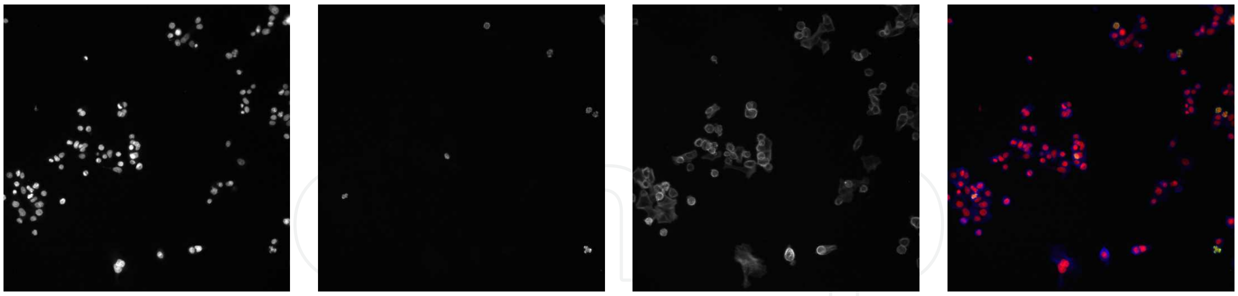

(a) DNA (b) PH3 (c) Actin (d) Merged

Figure 12. Examples of RNAi image, (a) DNA channel, (b) action channel, (c) PH3 channel, and (d) all the previous channels merged in an RBG image

In Tables 3-5, we can see several important measures: image size, depth, entropy, intensity usage, and a sparsity measure called Gini index or GI [34]. The GI is a normalized measure that assumes values between zero and one. Values close to zero means that image has a lower sparsity histogram. On the other hand, values close to one represent an image that has a very sparse histogram.

Data sets	Year	Images	Approximate size (cols × rows)	Depth	Average entropy (bpp)	Average intensity usage (percentage)	Gini Index [0-1]
ApoA1 [35]	2001	32	> 1,044×1,041	16	11.038	39.507%	0.494
Arizona [36]	2011	6	=13,800×4,400	16	9.306	82.821%	0.774
IBB [37]	2013	44	= 2,019×6,235	16	8.503	54.072%	0.806
ISREC [38]	2001	14	= 1,000×1,000	16	10.435	33.345%	0.710
Omnibus-LM [39]	2006	25	=12,200×4,320	16	5.713	50.130%	0.726
Omnibus-HM [40]	2006	25	=12,200×4,320	16	7.906	98.076%	0.892
Stanford [41]	2001	40	> 1,900×2,000	16	8.306	27.515%	0.615
Yeast [42]	1998	109	= 1,024×1,024	16	6.614	5.391%	0.518
YuLou [43]	2004	3	> 1,800×1,900	16	9.422	36.906%	0.556
Overall		298		16	7.415	67.003%	0.782

Table 3. Microarray image data sets used in this work. The number of images represents the total number of images that each data set contains (each image corresponds to one channel)

Data sets	Year	Images	Approximate size (cols × rows)	Depth	Average entropy (bpp)	Average intensity usage (percentage)	Gini Index [0-1]
CR	2003	12	> 612× 746	10-16	9.905	11.487%	0.240
CT	2003	12	> 340× 340	12-16	8.032	2.879%	0.389
MR	2003	12	> 256× 256	16	6.837	1.636%	0.624
US	2003	12	> 584× 476	8	4.561	0.358%	0.714
MIAS [33]	1995	322	=1,024×1,024	8	4.544	0.357%	0.681
Overall		370		8-16	5.169	1.596%	0.631

Table 4. Medical image data sets used. The computed radiography (CR), computed tomography (CT), magnetic resonance (MR), and ultrasound (US) data sets can be found in [44]. The last data set, mini-MIAS, contains 322 mammography images

Taking into consideration Table 3, we can see that the image sizes of the microarray data sets are very large (ranging from 1,000×1,000 to 13,800×4,400) and the bit depth is 16 for all the data sets. The average entropy goes from 5.7 bpp to 11 bpp, and the percentage of active intensities varies from 5.4% to 98.1%. The GI measures are all higher than 0.49 for all data sets reaching values close to 0.89 (very sparse) for the *Omnibus-HM* data set.

Data sets		**Year**	**Images**	**Approximate size (cols × rows)**	**Depth**	**Average entropy (bpp)**	**Average intensity usage (percentage)**	**Gini Index [0-1]**
RNAi D1	| (DNA)	2006	2,304	=512×512	12	7.408	4.140%	0.274
	| (PH3)		2,304			5.631	1.019%	0.056
	| (Actin)		2,304			7.583	2.287%	0.192
RNAi D2	| (DNA)	2006	2,304	=512×512	12	6.617	2.925%	0.172
	| (PH3)		2,304			5.572	0.661%	0.050
	| (Actin)		2,304			6.565	1.334%	0.099
RNAi D3	| (DNA)	2006	2,304	=512×512	12	7.210	4.195%	0.257
	| (PH3)		2,304			5.659	0.981%	0.054
	| (Actin)		2,304			7.653	3.215%	0.242
RNAi D4	| (DNA)	2006	2,304	=512×512	12	7.530	4.249%	0.286
	| (PH3)		2,304			5.662	1.080%	0.057
	| (Actin)		2,304			7.907	3.070%	0.244
RNAi D5	| (DNA)	2006	2,304	=512×512	12	7.281	4.046%	0.256
	| (PH3)		2,304			5.622	0.904%	0.052
	| (Actin)		2,304			7.649	2.772%	0.221
RNAi all	| (DNA)	2006	11,520	=512×512	12	7.209	3.911%	0.249
	| (PH3)		11,520			5.629	0.929%	0.054
	| (Actin)		11,520			7.471	2.536%	0.199
Overall		2006	34,560	=512×512	12	6.770	2.458%	0.167

Table 5. Five RNAi image data sets extracted from the first five plates that can be found in [45]. Contrary to other authors, the images were extracted from the RNAi files without any loss. The exact image intensity values were used to create the images. No normalization was performed

Regarding the medical image data sets (see Table 4), we can see that the image sizes are lower when compared to the microarray images (except for the *mini-MIAS* data set). The bit depth varies from 8 to 16 bits and the average entropy from 4.5 bpp to 9.9 bpp. The average intensity usage is very low for these medical images. For the *CR* data set, only 11.5% of the total available intensities are actually used. For the other data sets, this measure is even lower (3). For the GI measure, we can see that the *MR*, *US*, and *mini-MIAS* data sets have a GI higher than 0.6. On the other hand, for the *CR* and *CT* data sets, the GI values are lower (0.24 and 0.39, respectively).

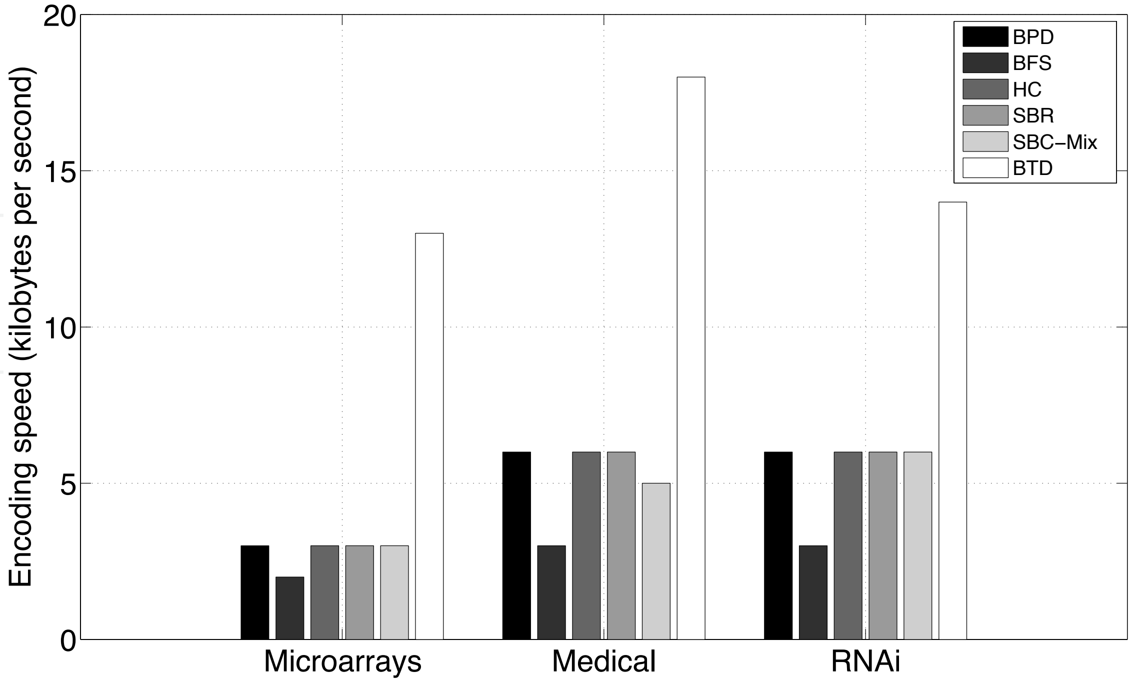


Figure 13. Encoding speed in kilobytes per second for the methods evaluated in this work. BPD (bitplane decomposition), BFS (background-foreground separation), HC (histogram compaction), SBR (scalable bitplane reduction), SBC-Mix (simple bitplane coding-mixture), and BTD (binary-tree decomposition)

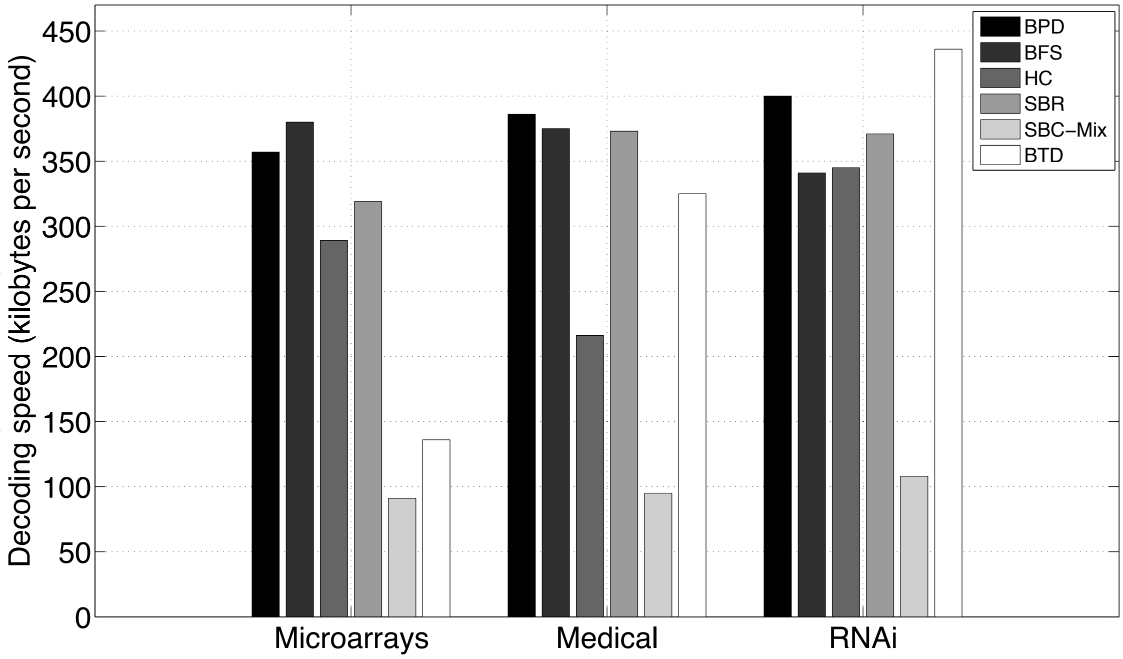


Figure 14. Decoding speed in kilobytes per second for the methods evaluated in this work. BPD (bitplane decomposition), BFS (background-foreground separation), HC (histogram compaction), SBR (scalable bitplane reduction), SBC-Mix (simple bitplane coding-mixture), and BTD (binary-tree decomposition)

Finally, we provide 5 RNAi image data sets in Table 5. We split each data set into three channels (DNA, PH3, and actin). All the images have size 512×512 and depth of 12 bits. Globally, the average entropy is about 6.77 bpp and the average intensity usage is around 2.5%. Regarding the GI, the values are very low (<0.17), which means that these images are not very sparse.

3.2. Experimental results

In this subsection, we present the results obtained using several general purpose compression tools (e.g., gzip, bzip2, PPMd, and LZMA), image coding standards (e.g., PNG, JBIG, JPEG-LS, and JPEG2000), and the methods described in Sect. 4. In Tables 6-8, we can find the compression results for all data set described earlier in Sect. 5.1, using several general compression tools and image coding standards. Default parameters were used in all compression tools. Hence, we did not try to adjust some parameters in order to improve the compression results. Only the lossless mode was imposed.

JBIG results were obtained using version 2.0 of the JBIG-Kit package[1]. The results for the JPEG-LS standard were obtained using version 2.2 of the SPMG JPEG-LS codec[2]. JPEG2000 lossless compression was obtained using version 5.1 of JJ2000 codec with default parameters for lossless compression[3]. For additional reference, we also provided compression results using the gzip, bzip2, ppmd, and lzma general-purpose compression tools.

	Compression methods							
Data sets	**Gzip**	**Bzip2**	**PPMd**	**LZMA**	**PNG**	**JBIG**	**JPEG-LS**	**JPEG2000**
ApoA1	12.711	11.068	10.984	11.374	12.568	10.851	10.608	11.063
Arizona	11.263	9.040	8.980	9.402	11.017	8.896	8.676	9.107
IBB	10.453	9.081	8.495	8.985	10.090	9.344	9.904	10.516
ISREC	12.464	10.922	10.730	11.126	12.476	10.925	11.145	11.366
Omnibus (LM)	7.124	5.346	4.977	5.527	6.781	5.130	4.936	5.340
Omnibus (HM)	9.558	7.523	7.219	7.787	9.160	7.198	6.952	7.587
Stanford	9.972	7.961	7.809	8.273	9.776	7.906	7.684	8.060
Yeast	7.672	6.075	5.794	6.389	8.303	6.888	8.580	9.079
YuLou	11.434	9.394	9.285	9.708	11.428	9.298	8.974	9.515
Average	9.044	7.189	6.859	7.388	8.729	7.051	6.996	7.511

Table 6. Lossless compression results, in bits per pixel (bpp), using gzip, bzip2, PPMd, LZMA, PNG, JBIG, JPEG-LS, and JPEG2000 for the microarray image data sets. Default compression parameters have been used for all algorithms. The best results are highlighted in bold

[1] http://www.cl.cam.ac.uk/ mgk25/jbigkit

[2] The original website of this codec, http://spmg.ece.ubc.ca, is currently unavailable. However, it can be obtained from http://sweet.ua.pt/luismatos/codecs/jpeg_ls_v2.2.tar.gz

[3] The original website of this codec, http://jj2000.epfl.ch, is currently unavailable. Nevertheless, this codec can be obtained from http://sweet.ua.pt/luismatos/codecs/jj2000_5.1-src.zip

	Compression methods							
Data sets	**Gzip**	**Bzip2**	**PPMd**	**LZMA**	**PNG**	**JBIG**	**JPEG-LS**	**JPEG2000**
CR	9.284	6.140	6.230	7.197	7.991	6.152	5.784	5.845
CT	8.790	5.604	5.847	6.996	9.411	8.822	8.089	8.334
MR	8.342	5.313	5.514	6.663	9.795	9.940	9.321	9.389
US	3.496	2.742	2.642	2.820	3.061	2.908	2.750	3.138
mini-MIAS	2.637	1.696	1.590	2.022	1.672	1.562	1.416	1.443
Average	3.452	2.242	2.163	2.660	2.484	2.183	2.005	2.040

Table 7. Lossless compression results, in bits per pixel (bpp), using gzip, bzip2, PPMd, LZMA, PNG, JBIG, JPEG-LS, and JPEG2000 for the medical image data sets. Default compression parameters have been used for all algorithms. The best results are highlighted in bold

		Compression methods							
Data sets		**Gzip**	**Bzip2**	**PPMd**	**LZMA**	**PNG**	**JBIG**	**JPEG-LS**	**JPEG2000**
RNAi D1	| (DNA)	9.120	6.846	6.832	7.314	9.857	6.802	6.311	6.533
	| (PH3)	7.461	5.933	5.804	6.008	11.552	5.994	5.806	5.830
	| (Actin)	9.274	6.807	6.839	7.353	11.801	6.790	6.304	6.452
RNAi D2	| (DNA)	8.358	6.469	6.389	6.714	9.421	6.469	6.094	6.233
	| (PH3)	7.396	5.884	5.760	5.963	12.420	5.964	5.793	5.804
	| (Actin)	8.309	6.378	6.315	6.638	12.449	6.422	6.032	6.122
RNAi D3	| (DNA)	8.948	6.735	6.637	7.153	9.665	6.704	6.238	6.436
	| (PH3)	7.490	5.952	5.819	6.030	11.756	6.042	5.812	5.842
	| (Actin)	9.362	6.874	7.387	7.431	10.582	6.838	6.353	6.509
RNAi D4	| (DNA)	6.243	6.925	6.915	7.415	10.014	6.878	6.372	6.612
	| (PH3)	7.491	5.950	5.824	6.028	11.548	6.028	5.813	5.845
	| (Actin)	9.586	7.014	7.070	7.629	10.955	6.977	6.460	6.630
RNAi D5	| (DNA)	9.007	6.765	6.745	7.198	9.775	6.729	6.259	6.459
	| (PH3)	7.450	5.928	5.798	6.001	11.745	5.994	5.804	5.825
	| (Actin)	9.346	6.857	6.893	7.414	11.185	6.828	6.342	6.493
RNAi all	| (DNA)	8.935	6.748	6.703	7.159	9.746	6.716	6.255	6.455
	| (PH3)	7.458	5.929	5.801	6.006	11.804	6.004	5.806	5.829
	| (Actin)	9.176	6.786	6.900	7.293	11.394	6.771	6.298	6.441
Average		8.523	6.488	6.468	6.819	10.982	6.497	6.120	6.242

Table 8. Lossless compression results, in bits per pixel (bpp), using gzip, bzip2, PPMd, LZMA, PNG, JBIG, JPEG-LS, and JPEG2000 for the RNAi image data sets. Default compression parameters have been used for all algorithms. The best results are highlighted in bold

According to the results depicted in Tables 6-8, it seems that, globally, JPEG-LS is the best image coding standard. Among the general compression methods used, PPMd is the one that attained the best compression results.

Tables 9-11 provide lossless compression results using the methods described in Sect. 4. BPD (bitplane decomposition) corresponds to the results using method [6]; BFS (background-foreground separation) corresponds to using the previous approach with segmentation. The columns HC (histogram compaction) and SBR (scalable bitplane reduction) correspond to the approaches that use bitplane reduction in method [6]. The SBC-Mix (simple bitplane coding-mixture) corresponds to the approach described in Sect. 4.2.3 using a mixture of finite-context models. The last column, denoted as BTD (binary-tree decomposition), corresponds to the results obtained using the method described in Sect. 4.3.

If we look closely to Table 9, we can see that the method based on binary-tree decomposition is the one that attained the best compression results among all the others, for the case of microarray images. The best method is about 11% better than JPEG-LS. However, the method SBC-Mix is the best for three data sets: *ApoA1*, *ISREC*, and *Yulou*.

Regarding the experiments performed on medical images and taking into consideration the results presented in Table 10, we can conclude once again that the method based on binary-tree decomposition outperforms all the others and is about 9% better than JPEG-LS.

	Compression methods						
Data sets	**BPD**	**BFS**	**HC**	**SBR**	**SBC**	**SBC-Mix**	**BTD**
ApoA1	10.194	10.234	10.231	10.232	10.205	10.142	10.194
Arizona	8.242	8.245	8.244	8.243	10.308	8.219	8.186
IBB	7.974	7.982	7.978	7.978	8.537	7.966	7.943
ISREC	10.159	10.193	10.195	10.199	10.260	10.148	10.198
Omnibus (LM)	4.567	4.570	4.561	4.565	4.645	4.545	4.539
Omnibus (HM)	6.471	6.479	6.473	6.472	6.581	6.443	6.400
Stanford	7.379	7.349	7.350	7.349	7.403	7.305	7.303
Yeast	5.453	5.395	5.527	5.466	5.492	5.326	5.318
YuLou	8.619	8.641	8.626	8.626	8.669	8.591	8.592
Average	6.284	6.288	6.286	6.285	6.437	6.257	6.235

Table 9. Average compression results, in bits per pixel using the methods described in Sect. 4, BPD (bitplane decomposition), BFS (background-foreground separation), HC (histogram compaction), SBR (scalable bitplane reduction), SBC-Mix (simple bitplane coding-mixture), and BTD (binary-tree decomposition), for the microarray image data sets

	Compression methods						
Data sets	**BPD**	**BFS**	**HC**	**SBR**	**SBC**	**SBC-Mix**	**BTD**
CR	5.338	5.207	5.291	5.214	5.764	5.137	5.136
CT	4.836	4.840	4.857	4.875	8.302	4.762	4.809
MR	4.654	4.702	4.835	4.833	9.382	4.594	4.783
US	2.472	2.564	2.478	2.471	2.722	2.439	2.462
mini-MIAS	1.390	1.378	1.391	1.391	1.430	1.358	1.352
Average	1.877	1.853	1.874	1.865	2.016	1.826	1.822

Table 10. Average compression results, in bits per pixel using the methods described in Sect. 4, BPD (bitplane decomposition), BFS (background-foreground separation), HC (histogram compaction), SBR (scalable bitplane reduction), SBC-Mix (simple bitplane coding-mixture), and BTD (binary-tree decomposition), for the medical image data sets

		Compression methods						
	Data sets	**BPD**	**BFS**	**HC**	**SBR**	**SBC**	**SBC-Mix**	**BTD**
RNAi D1	| (DNA)	6.250	6.293	6.250	6.255	6.225	**6.179**	6.186
	| (PH3)	5.559	5.570	5.570	5.569	5.571	**5.545**	5.555
	| (Actin)	6.255	6.311	6.244	6.258	6.220	**6.185**	6.205
RNAi D2	| (DNA)	5.959	5.986	5.968	5.968	5.960	**5.918**	5.930
	| (PH3)	5.537	5.547	5.539	5.540	5.545	**5.518**	5.525
	| (Actin)	5.919	5.948	5.904	5.928	5.904	**5.876**	5.892
RNAi D3	| (DNA)	6.175	6.213	6.169	6.164	6.145	**6.106**	6.111
	| (PH3)	5.580	5.598	5.584	5.585	5.590	**5.564**	5.572
	| (Actin)	6.313	6.366	6.299	6.306	6.277	**6.236**	6.247
RNAi D4	| (DNA)	6.309	6.355	6.312	6.317	6.289	**6.239**	6.244
	| (PH3)	5.577	5.593	5.582	5.584	5.587	**5.561**	5.570
	| (Actin)	6.423	6.484	6.412	6.419	6.395	**6.348**	6.363
RNAi D5	| (DNA)	6.195	6.233	6.194	6.198	6.166	**6.126**	6.133
	| (PH3)	5.558	5.568	5.564	5.567	5.569	**5.543**	5.552
	| (Actin)	6.299	6.355	6.286	6.298	6.263	**6.224**	6.240
RNAi all	| (DNA)	6.178	6.216	6.179	6.182	6.157	**6.113**	6.121
	| (PH3)	5.562	5.575	5.568	5.569	5.573	**5.546**	5.555
	| (Actin)	6.242	6.293	6.229	6.242	6.212	**6.174**	6.189
Average		5.994	6.028	5.992	5.998	5.980	**5.945**	5.955

Table 11. Average compression results, in bits per pixel using the methods described in Sect. 4, BPD (bitplane decomposition), BFS (background-foreground separation), HC (histogram compaction), SBR (scalable bitplane reduction), SBC-Mix (simple bitplane coding-mixture), and BTD (binary-tree decomposition), for the RNAi image data sets

Finally, in Table 11, we present the results for the RNAi images, and in this case, method SBC-Mix is the one that attained the best compression results, almost 3% better than JPEG-LS. The difference between the SBC-Mix and BTD is very low (close to 0.01 bpp). In our opinion, it seems that the methods described in Sect. 4 attain better compression results when compared to JPEG-LS for images with higher GI values (sparse images). This is the reason why the binary-tree decomposition approach attains between 9% to 11% better results than JPEG-LS for the microarray and medical images. On the other hand, the RNAi images have a lower GI (around 0.17), therefore the lower performance of our methods when compared to JPEG-LS.

In Figs. 15 and 16, we can see the encoding and decoding speed in kilobytes per second for the compression methods described in Sect. 4. In the encoding phase, it seems that the method based on binary-tree decomposition is the fastest one among all the others for all data sets. In the decoding phase, the fastest method is different from data set to data set. For the microarray and medical image data sets, the methods based on bitplane decomposition are faster than the method based on binary-tree decomposition. On the other hand, for the RNAi images, the method based on binary-tree decomposition is faster than all the others. The SBC-Mix method is the slowest one in all data sets. The lower decoding speed for the SBC-Mix is due to the mixture procedure. For each symbol that is processed, the model weights need to be updated, which is a very time-consuming task.

3.3. Rate distortion study

The two decomposition approaches described in Sect. 4 have progressive decoding capabilities (also known as lossy to lossless). Due to that, it is important to understand what is the error induced in the image during the decoding phase when only part of the data is decoded. This characteristic allows also to obtain the original image without any loss.

In Figs. 15 and 16, we present the rate distortion curves of two images, "array1" from the *YouLou* data set and "cr_17218" from the medical *CR* data set, according to two metrics: L_2-norm (root-mean-square error or RMSE) and L_∞-norm (maximum absolute error or MAE). We used JBIG, JPEG2000, the approach based on bitplane decomposition [6], and the other based on binary-tree decomposition [8].

Regarding Fig. 15, we can notice that the method based on binary-tree decomposition outperforms all the other methods in terms of L_2-norm and L_∞-norm. JBIG and method [6] have a similar performance in terms of rate distortion. The JPEG2000 standard attains globally worse results in terms of L_2. Furthermore, we can notice a sudden degradation of the rate distortion, for higher bitrates L_∞. We believe that this phenomenon is probably related to the default parameters used by the encoder, which might not be well suited for microarray images.

In Fig. 16, we can see in the first plot, related to the L_2-norm, that method [6] attains the worse distortion results. JBIG is slightly better but worse than JPEG2000 and the method [8]. We can also see that JPEG2000 outperforms all the other methods in terms of L_2-norm for lower bitrates. For bitrates higher than 2 bpp, the best method is the one based on binary-tree decomposition. For the L_∞-norm, JPEG2000 and method [6] are the ones that attained the

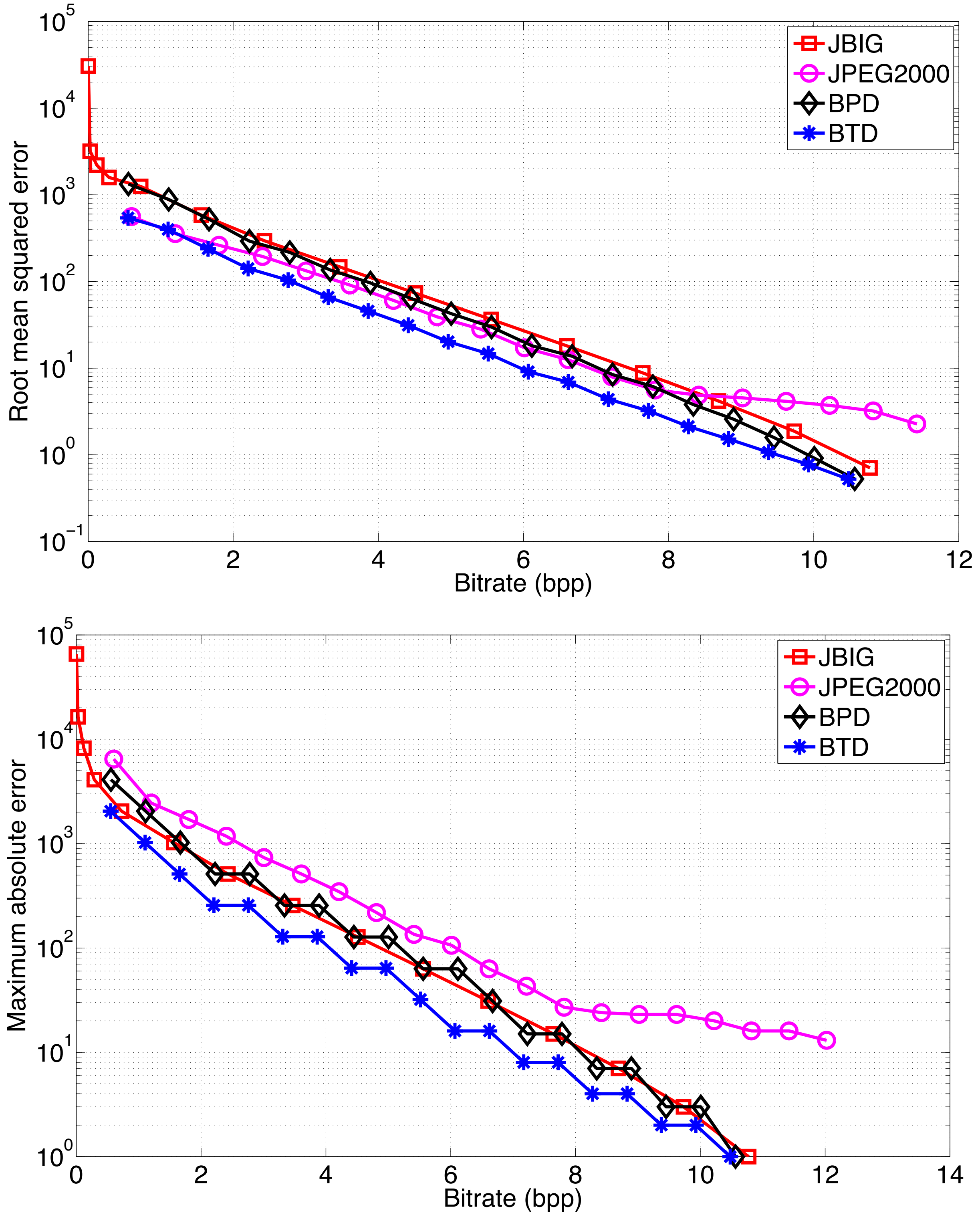


Figure 15. Rate distortion curves showing the performance of the bitplane decomposition (BPD), the binary-tree decomposition (BTD), and the JPEG2000, and JBIG standards, in lossy-to-lossless mode for the microarray image "array1" from the *YuLou* data set. Results are given both for the L_2 (root-mean-square error or RMSE) and L_∞ (maximum absolute error or MAE) norms

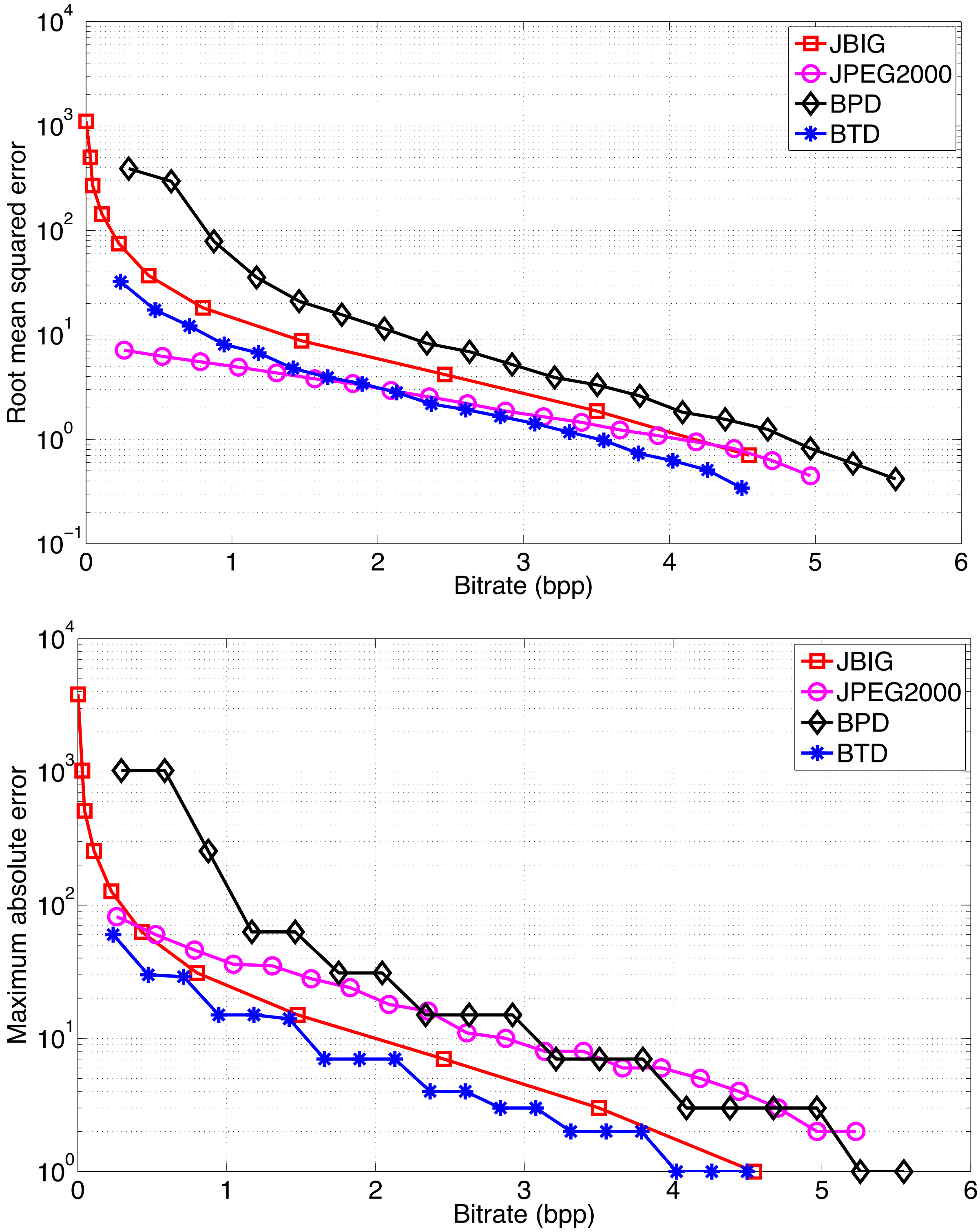


Figure 16. Rate distortion curves showing the performance of the bitplane decomposition (BPD), the binary-tree decomposition (BTD), and the JPEG2000 and JBIG standards, in lossy-to-lossless mode for the medical image "cr_17218" from the *CR* data set. Results are given both for the L_{∞} (root-mean-square error or RMSE) and $(>8\,\mathrm{bpp})$ (maximum absolute error or MAE) norms

worse results. On the other hand, method [8] outperforms all the others, regardless of the bitrate. We did not include results for other images to avoid extending this chapter, but the results obtained with the medical images are very similar with the results obtained using RNAi images. On the other hand, we can conclude that the rate distortion results for the microarray images are not similar to the other data set types (medical and RNAi).

4. Conclusions

The use of biomedical imaging has increased in the last years. These biomedical images include microarray, medical, and RNAi images. In this chapter, we have studied two image decomposition approaches to be applied to this type of images in order to compress them efficiently.

We presented a set of comprehensive results regarding the lossless compression of biomedical images using general coding methods (e.g., gzip), image coding standards (e.g., JPEG2000), and the two image decomposition approaches that rely on finite-context models and arithmetic coding.

From the obtained experimental results, we conclude that JPEG-LS gives the best compression results, among the image coding standards, but lacks lossy-to-lossless capability, which may be a decisive functionality if remote transmission over possibly slow links is a requirement.

We developed compression algorithms based on two decomposition approaches: bitplane decomposition and binary-tree decomposition. In the bitplane decomposition, we also used segmentation, bitplane Reduction, and an approach based on bit modeling by the pixel value estimates. All the developed methods allow lossy-to-lossless compression and are based on finite-context models and arithmetic coding. According to the obtained results, the approach based on binary-tree decomposition seems to be the one that attains the best compression results. The only exceptions are the RNAi images, for which the best method is the one based on a mixture of finite-context models.

In terms of coding speed, the compression algorithm based on a binary-tree decomposition seems to be the fastest one among all the others in the encoding phase. In the decoding stage, the approaches based on bitplane decomposition seem to be faster for the microarray and medical image data sets. For the RNAi images, the method based on binary-tree decomposition seems to be the fastest one.

A rate distortion study was also performed and, according to the obtained results, it seems that the method based on binary-tree decomposition attains the best results for the majority of the cases. The results obtained by the developed compression algorithms have been compared with general-purpose compression methods and also with image coding standards. According to the results, we can conclude that these compression methods have better compression performance in all the image test sets used.

Acknowledgements

This work was developed at IEETA (Institute of Electronics and Telematics Engineering of Aveiro).It was partially supported by FEDER, through the Operational Program Competitiveness Factors (COMPETE), and by National Funds through FCT (Foundation for Science and Technology) in the context of a PhD Grant (FCT reference SFRH/BD/85855/2012) and projects PEst-OE/EEI/UI0127/2014 and Incentivo/EEI/UI0127/2014.

Author details

Luís M. O. Matos*, António J. R. Neves and Armando J. Pinho

*Address all correspondence to: luismatos@ua.pt

IEETA/DETI, University of Aveiro, Aveiro, IEETA/DETI, Portugal

References

[1] A. Nait-Ali and C. Cavaro-Menard. *Compression of Biomedical Images and Signals*. Wiley-ISTE, 1st edition, 2008.

[2] P. Hegde, R. Qi, K. Abernathy, C. Gay, S. Dharap, R. Gaspard, J. Earle-Hughes, E. Snesrud, N. Lee, and John Q. A concise guide to cDNA microarray analysis. *Biotechniques*, 29(3):548–562, September 2000.

[3] S. K. Moore. Making chips to probe genes. *IEEE Spectrum*, 38(3):54–60, March 2001.

[4] C. Falschlehner, S. Steinbrink, G. Erdmann, and M. Boutros. High-throughput RNAi screening to dissect cellular pathways: a how-to guide. *Biotechnology Journal*, 5(4): 368–376, 2010.

[5] D. S. Taubman and M. W. Marcellin. *JPEG2000: Image Compression Fundamentals, Standards and Practice*. Kluwer Academic Publishers, 2002.

[6] A. J. R. Neves and A. J. Pinho. Lossless compression of microarray images using image-dependent finite-context models. *IEEE Transactions on Medical Imaging*, 28(2):194–201, February 2009.

[7] A. J. Pinho and A. J. R. Neves. Progressive lossless compression of medical images. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP-2009*, Taipei, Taiwan, April 2009.

[8] L. M. O. Matos, A. J. R. Neves, and A. J. Pinho. Compression of microarrays images using a binary-tree decomposition. In *Proceedings of the 22th European Signal Processing Conference, EUSIPCO-2014*, pages 531–535, Lisbon, Portugal, September 2014.

[9] MENT, 2014. Last accessed on February, 2015. http://bioinformatics.ua.pt/software/ment.

[10] A. J. R. Neves and A. J. Pinho. Compression of microarray images. In S. Miron, editor, *Signal Processing*, pages 429–448. INTECH, March 2010.

[11] A. J. Pinho and A. J. R. Neves. L-infinity progressive image compression. In *Proceedings of the Picture Coding Symposium, PCS-07*, Lisbon, Portugal, November 2007.

[12] A. J. Pinho and A. J. R. Neves. Lossy-to-lossless compression of images based on binary tree decomposition. In *Proceedings of the IEEE International Conference on Image Processing, ICIP-2006*, pages 2257–2260, Atlanta, Georgia, USA, October 2006.

[13] A. J. Pinho, D. Pratas, and P. J. S. G. Ferreira. Bacteria DNA sequence compression using a mixture of finite-context models. In *Proceedings of the IEEE Workshop on Statistical Signal Processing*, Nice, France, June 2011.

[14] A. J. Pinho, A. J. R. Neves, D. A. Martins, C. A. C. Bastos, and P. J. S. G. Ferreira. Finite-context models for DNA coding. In S. Miron, editor, *Signal Processing*, pages 117–130. INTECH, March 2010.

[15] A. J. Pinho, A. J. R. Neves, V. Afreixo, Carlos A. C. Bastos, and P. J. S. G. Ferreira. A three-state model for DNA protein-coding regions. *IEEE Transactions on Biomedical Engineering*, 53(11):2148–2155, November 2006.

[16] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text Compression*. Prentice Hall, 1990.

[17] K. Sayood. *Introduction to Data Compression*. Morgan Kaufmann, 3rd edition, 2006.

[18] D. Salomon. *Data Compression - The Complete Reference*. Springer, 4th edition, 2007.

[19] J. Rissanen and G. G. Langdon, Jr. Arithmetic coding. *IBM Journal of Research and Development*, 23(2):149–162, March 1979.

[20] David Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, September 1952.

[21] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Prentice Hall, 2nd edition, 2002.

[22] P. Buonora and F. Liberati. A format for digital preservation of images: a study on JPEG 2000 file robustness. *D-Lib Magazine*, 14(7/8), August 2008.

[23] Y. Yoo, Y. G. Kwon, and A. Ortega. Embedded image-domain compression using context models. In *Proceedings of the IEEE International Conference on Image Processing, ICIP-99*, volume I, pages 477–481, Kobe, Japan, October 1999.

[24] A. J. R. Neves and A. J. Pinho. Lossless compression of microarray images. In *Proceedings of the IEEE International Conference on Image Processing, ICIP-2006*, pages 2505–2508, Atlanta, Georgia, USA, October 2006.

[25] A. Neekabadi, S. Samavi, S. A. Razavi, N. Karimi, and S. Shirani. Lossless microarray image compression using region based predictors. In *Proceedings of the IEEE International Conference on Image Processing, ICIP-2007*, volume 2, pages 349–352, San Antonio, Texas, USA, September 2007.

[26] Y. Yoo, Y. G. Kwon, and A. Ortega. Embedded image-domain adaptive compression of simple images. In *Proceedings of the 32nd Asilomar Conference on Signals, Systems, and Computers*, volume 2, pages 1256–1260, Pacific Grove, California, USA, November 1998.

[27] H. Kikuchi, K. Funahashi, and S. Muramatsu. Simple bit-plane coding for lossless image compression and extended functionalities. In *Proceedings of the Picture Coding Symposium, PCS-09*, pages 1–4, Chicago, Illinois, USA, May 2009.

[28] H. Kikuchi, R. Abe, and S Muramatsu. Simple bitplane coding and its application to multi-functional image compression. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E95.A(5):938–951, 2012.

[29] H. Kikuchi, T. Deguchi, and M. Okuda. Lossless compression of LogLuv32 HDR images by simple bitplane coding. In *Picture Coding Symposium, PCS-2013*, pages 265–268, San Jose, California, USA, December 2013.

[30] S. Baase and A. V. Gelder. *Computer Algorithms: Introduction to Design and Analysis*. Addison Wesley, 3rd edition, November 1999.

[31] X. Chen, S. Kwong, and J.-F. Feng. A new compression scheme for color-quantized images. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(10):904–908, October 2002.

[32] A. J. Pinho and A. J. R. Neves. A context adaptation model for the compression of images with a reduced number of colors. In *Proceedings of the IEEE International Conference on Image Processing, ICIP-2005*, volume 2, pages 738–741, Genova, Italy, September 2005.

[33] mini-MIAS, 1995. Last accessed on February, 2015. http://peipa.essex.ac.uk/pix/mias.

[34] D. Zonoobi, A. A. Kassim, and Y. V. Venkatesh. Gini index as sparsity measure for signal reconstruction from compressive samples. *IEEE Journal of Selected Topics in Signal Processing*, 5(5):927–932, September 2011.

[35] ApoA1, 2001. Last accessed on February, 2015. http://www.stat.berkeley.edu/ terry/zarray/Html/apodata.html or http://sweet.ua.pt/luismatos/microarrays/apoa1.html.

[36] Arizona, 2011. Last accessed on February, 2015. http://deic.uab.es/ mhernandez/media/imagesets/arizona.tar.bz2 or http://sweet.ua.pt/luismatos/microarrays/arizona.html.

[37] IBB, 2013. Last accessed on February, 2015. http://deic.uab.es/ mhernandez/media/imagesets/ibb.tar.bz2 or http://sweet.ua.pt/luismatos/microarrays/ibb.html.

[38] ISREC, 2001. Last accessed on February, 2015. http://ccg.vital-it.ch/DEA/module8/P5_chip_image/images or http://sweet.ua.pt/luismatos/microarrays/isrec.html.

[39] Omnibus-LM, 2006. Last accessed on February, 2015. http://deic.uab.es/ mhernandez/media/imagesets/omnibus.txt or http://sweet.ua.pt/luismatos/microarrays/omnibus-lm.html.

[40] Omnibus-HM, 2006. Last accessed on February, 2015. http://deic.uab.es/ mhernandez/media/imagesets/omnibus.txt or http://sweet.ua.pt/luismatos/microarrays/omnibus-hm.html.

[41] Stanford, 2001. Last accessed on February, 2015. ftp://smd-ftp.stanford.edu/pub/smd/transfers/Jenny or http://sweet.ua.pt/luismatos/microarrays/stanford.html.

[42] Yeast, 1998. Last accessed on February, 2015. http://genome-www.stanford.edu/cell-cycle/data/rawdata/individual.html or http://sweet.ua.pt/luismatos/microarrays/yeast.html.

[43] YuLou, 2004. Last accessed on February, 2015. http://www.cs.ucr.edu/yuluo/MicroZip or http://sweet.ua.pt/luismatos/microarrays/yulou.html.

[44] MEDNAT, 2004. Last accessed on February, 2015. http://sun.aei.polsl.pl/ rstaros/mednat.

[45] BBBC017v1, 2006. Last accessed on February, 2015. http://www.broadinstitute.org/bbbc/BBBC017.