Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
Theoretical Computer Science 313 (2004) 133 – 144

Theoretical Computer Science

www.elsevier.com/locate/tcs

Enhancements of partitioning techniques for image compression using weighted finite automata

F. Katritzke[a,*], W. Merzenich[a], M. Thomas[b]

[a]Department of Electronic Engineering and Computer Science, University of Siegen, Siegen, Germany
[b]Mathematical Department, University of Siegen, Siegen, Germany

Abstract

Weighted finite automata are efficient structures for the storage of digital images. The choice of the image partitioning technique is important to achieve good compression results. In this paper we examine two promising techniques by measuring the compression performance at well-known test images.
© 2003 Elsevier B.V. All rights reserved.

Keywords: Image compression; Weighted finite automata; Partitioning techniques

1. Introduction

The efficient storage and transmission of digital images is gaining increasing interest, for example in multimedia and WWW applications. Pictures require a huge amount of storage capacity, even in compressed form (as GIF or JPEG). For this reason, many researchers study the area of digital image compression extensively.

Weighted finite automata (WFAs) are structures which have been utilized successfully for the efficient storage of digital images. One of the most important aspects for data compression with WFAs is the choice of the image partitioning technique. In this paper we examine the suitability of some techniques for our purposes. The most widely used partitioning techniques for WFAs are quadtrees and bintrees. We examine the application of HV partitioning which is a common technique for image partitioning in image coding with iterated function systems (IFS) and suggest a modification (called light HV) that leads to good results when it is applied to WFAs.

* Corresponding author. Tel.: +49-271-740-2311; fax: +49-271-740-2532.
E-mail address: frankka@informatik.uni-siegen.de (F. Katritzke).

0304-3975/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2003.10.011

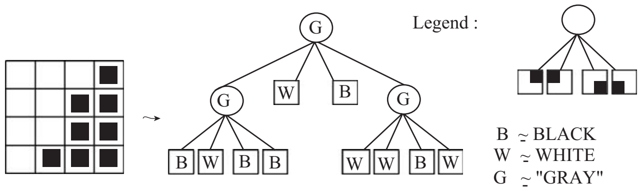

Fig. 1. Region quadtree. The ordering of the quadrants and the node labels are indicated in the legend on the right.

The reader who is already familiar with weighted finite automata may skip to Section 4.

2. Basic techniques for image partitioning

The technique of WFA coding is based on partitioning an image into sub-images, which was introduced in a scheme named quadtree image compression. It can be refined using bintrees and HV bintrees. We represent images with a real-valued $2^k \times 2^k$ matrix ($k \in \mathbb{N}$) and address the elements in the matrix as pixels.

2.1. Quadtrees

The original image partitioning in the WFA technique uses quadtrees [1]. The *quadtree* is a hierarchical data structure based on the divide and conquer principle, leading to a recursive partitioning. We examine the *region quadtree*, which divides the input image recursively into four quadrants of equal size. The quadtree is therefore a tree of degree four, with each inner node having four children. Its leaves have to fulfill a certain simplicity criterion, e.g., all pixels in the pertaining quadtree have equal gray values.

The quadtree generation algorithm is illustrated in Fig. 1. The nodes are traversed in the order represented by the tree on the right. The leaves store the gray values “B” (black) and “W” (white). The inner nodes are labeled with “G” (gray) to indicate that they have to be partitioned further.

A quadtree is suited for image compression because large areas can often be stored in a single node. The quadtree can also be used for lossy image coding if the recursive division is stopped if a special “homogeneity property” is fulfilled by the addressed segment. The associated node of the quadtree could store a gray value which minimizes a given distortion function, like the mean value for a quadratic error measure. Note that 4 bits indicating the presence or absence of a further subtree are sufficient to store an inner node of the quadtree. An entropy encoder can significantly reduce the requirement to store 4 bits per node.

2.2. Bintrees

If we divide each part of an image in only two parts instead of four, we obtain *bintrees*. Horizontal and vertical subdivisions are applied at alternate levels of the tree.

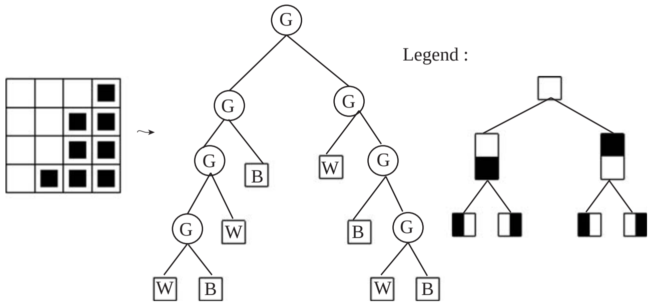

Fig. 2. Region bintree. The ordering of the segments and the node labels are indicated in the legend on the right.

It is obvious that bintrees having an even number of levels yield the same image partition as quadtrees. Using bintrees, it is possible to stop the division process employed in its generation at an odd number niveau, yielding larger parts of the image and thus facilitating a more efficient coding. Similar to quadtrees, a bintree is stored with 2 bits per node. See Figs. 2 and 6 for an illustration of bintrees.

3. Weighted finite automata

In this section, we introduce the main topic of interest in this paper: the weighted finite automaton (WFA). Image coding with this structure was introduced by Culik and Kari [1]. In order to simplify the approach, we introduce this kind of structure with quadtrees. As we see later, WFAs may be defined in a similar manner using bintrees and even HV bintrees (see Section 4.1). Since a WFA may be stored with less bits than the original image, such automata are of interest for image compression.

3.1. Definition of a WFA

In order to apply methods of formal languages to images, we associate words over an alphabet Σ with two-dimensional image segments. As usual, let Σ^n denote all words (strings) of length n and $\Sigma^* := \bigcup_{n \geqslant 0} \Sigma^n$.

We explain the concept of a WFA for the quadtree case only and thus associate real numbers with the image segments defined by the nodes of a quadtree. These numbers will express the average gray value of the addressed image segment. In order to describe the quadtree nodes we label the four children of a node with elements of an alphabet $\Sigma = \{1, 2, 3, 4\}$ (and $\Sigma = \{1, 2\}$ in the non-quadtree cases considered in this paper). The string defined by concatenating the labels assigned to the quadrants on the path from the root to the desired node is associated with the image segment defined by that node. For an illustration of this addressing concept see Fig. 3.

A function $\Psi : \Sigma^* \to \mathbb{R}$ is called a *multi-resolution image.* For $b_1 \dots b_k \in \Sigma^*$, we interpret the real number $\Psi(b_1 \dots b_k)$ as the gray value of the image segment addressed

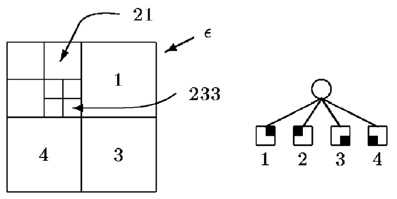

Fig. 3. Interpretation of a quadtree as a trie.

by $b_1 \dots b_k$. Let $V := \{\Psi : \Sigma^* \to \mathbb{R}\}$ be the set of all multi-resolution images. Furthermore, we define the set of conservative (or average preserving) functions

$$\mathscr{D} := \left\{ \Psi \in V\colon\ \Psi(\sigma) = \frac{1}{|\Sigma|} \sum_{b\in\Sigma} \Psi(\sigma b),\ \sigma \in \Sigma^* \right\}, \tag{1}$$

where σb is the usual concatenation of strings [1]. A conservative function with $|\Sigma| = 4$ can be interpreted as a quadtree, where the brightness of each node is the average brightness of its children.

V is a vector space over $\mathbb{R}$ if we define the operations

$$(\Psi_1 + \Psi_2)(\sigma) = \Psi_1(\sigma) + \Psi_2(\sigma) \quad (\Psi_1, \Psi_2 \in V, \sigma \in \Sigma^*), \tag{2}$$

$$(c\Psi)(\sigma) = c\Psi(\sigma) \quad (\Psi \in V, c \in \mathbb{R}, \sigma \in \Sigma^*). \tag{3}$$

It is obvious that the set of conservative functions $\mathscr{D}$ is a subspace of the vector space V.

Definition 1. A WFA $\mathscr{A} = (S, \Sigma, W, I, F)$ is given by
1. a finite set of *states* $S = \{1, \dots, n\}$,
2. a finite *alphabet* $\Sigma = \{a_1, \dots, a_m\}$,
3. a *weight function* $W : S \times \Sigma \times S \to \mathbb{R}$,
4. an *initial distribution* $I : S \to \mathbb{R}$,
5. a *final distribution* $F : S \to \mathbb{R}$.

We either choose $\Sigma = \{1, 2, 3, 4\}$ (quadtree partitioning) or $\Sigma = \{1, 2\}$. We write $(W_b)_{i,j} = W(i, b, j)$ with $b \in \Sigma$, $i, j \in S$ and say that (i, b, j) is an *edge* of the WFA iff $(W_b)_{i,j} \neq 0$. To perform calculations with the WFA $\mathscr{A}$, we consider W_b as a real valued $n \times n$-matrix, I and F as n-dimensional (column) vectors.

For each state $i \in S$ a multi-resolution image Φ_i (*state image*) is defined by

$$\Phi_i : \begin{cases} \quad \Sigma^* \to \mathbb{R}, \\ b_1 \dots b_k \mapsto e_i^{\mathrm{t}}(W_{b_1} \dots W_{b_k} F), \end{cases} \tag{4}$$

where e_i is the ith canonical unit vector and x^{t} is the transposition of the vector x. Moreover, we write $\Psi(b\cdot)$ to denote the multi-resolution image

$$\Psi(b\cdot) : \begin{cases} \quad \Sigma^* \to \mathbb{R}, \\ b_1 \dots b_k \mapsto \Psi(b b_1 \dots b_k) \end{cases} \tag{5}$$

obtained by zooming into the quadrant specified by $b \in \Sigma$. The multi-resolution image $\Psi_{\mathscr{A}}$ given by the WFA $\mathscr{A} = (S, \Sigma, W, I, F)$ is the linear combination of the state images Φ_i

$$\Psi_{\mathscr{A}} := \sum_{j=1}^{n} I_j \Phi_j. \tag{6}$$

By inserting (4), we obtain

$$\Psi_{\mathscr{A}}(b_1 \dots b_k) = I^{\mathrm{t}}(W_{b_1} \dots W_{b_k} F) \tag{7}$$

for $k \in \mathbb{N}$ and $b_1 \dots b_k \in \Sigma^*$.

Note that (7) yields a (slow) WFA decoding algorithm, since we can calculate the associated brightness of each pixel in the target image with this formula. Faster WFA decoding algorithms are given in [1,8].

3.2. The first WFA Coding Algorithm introduced by Culik and Kari

To construct a WFA for a given image (described by a conservative function $\Psi \in \mathscr{D}$), we must calculate the vectors I, F and the matrices W_b ($b \in \Sigma$). The vector I depends on the encoding algorithm and is specified later. From the definition of the state images (4) we infer

$$\Phi_i(\varepsilon) = e_i^{\mathrm{t}} F = F_i, \tag{8}$$

showing that F_i is the average brightness of the state image Φ_i. Furthermore, for $b \in \Sigma$ we consider the quadrant $\Phi_i(b\,\cdot)$ of the state image Φ_i and obtain

$$\Phi_i(b\cdot)(b_1 \dots b_k) = \Phi_i(bb_1 \dots b_k) \tag{9}$$

$$= e_i^{\mathrm{t}} W_b W_{b_1} \dots W_{b_k} F \tag{10}$$

$$= (e_i^{\mathrm{t}} W_b)(W_{b_1} \dots W_{b_k} F) \tag{11}$$

$$= (e_i^{\mathrm{t}} W_b) \begin{pmatrix} e_1^{\mathrm{t}} W_{b_1} \dots W_{b_k} F \\ \vdots \\ e_n^{\mathrm{t}} W_{b_1} \dots W_{b_k} F \end{pmatrix} \tag{12}$$

$$= ((W_b)_{i,1}, \dots, (W_b)_{i,n}) \begin{pmatrix} \Phi_1(b_1 \dots b_k) \\ \vdots \\ \Phi_n(b_1 \dots b_k) \end{pmatrix} \tag{13}$$

$$= \sum_{j=1}^{n} (W_b)_{i,j} \Phi_j(b_1 \dots b_k) \tag{14}$$

for all $b_1 \dots b_k \in \Sigma^*$, and thus

$$\Phi_i(b\cdot) = \sum_{j=1}^{n} (W_b)_{i,j} \Phi_j. \tag{15}$$

One can see that the quadrants of the state images are linear combinations of other state images, which yields the straightforward WFA construction algorithm to approximate a conservative function Ψ shown in the following listing. One should note some details.

- We start with a state representing the whole input image Ψ. Then we try to express the quadrants of a given state image as a linear combination of other state images. If that is not possible, a new state is generated for the corresponding quadrant.
- The matrices $W_1,\dots,W_4$ must grow from an $n\times n$ to an $(n+1)\times(n+1)$ matrix when a new state is added, which is achieved by adding zeroes.
- The algorithm terminates because a state representing a single pixel can be expressed as a linear combination of other states.

It can be shown that the WFA generated by this method possesses the minimal number of states of all WFAs that generate Ψ [1]. For more details concerning this method see [2] or [3].

```
*Input*: Image as a conservative function $\Psi$
*Output*: Number of states $n, W_1,\dots,W_4, I$
$\Phi_1 := \Psi$
$n := 1$          {number of states}
$U := \{1\}$      {set of unprocessed states}
while $U \neq \emptyset$
    {
    choose a state $i \in U$
    for $b \in \Sigma$
        if $\exists (r_1,\dots,r_n) \in \mathbb{R}^n : \Phi_i(b\cdot) = r_1\Phi_1 + \cdots + r_n\Phi_n$
            $((W_b)_{i,1},\dots,(W_b)_{i,n}) := (r_1,\dots,r_n)$
        else
            {
            $\mathrm{n} := \mathrm{n}+1$      {create a new state}
            $U := U \cup \{n\}$
            $\Phi_n := \Phi_i(b\cdot)$
            $((W_b)_{i,1},\dots,(W_b)_{i,n}) := (0,\dots,0,1)$
            }
    $U := U \setminus \{i\}$
    }
$I := (1,0,\dots,0)^{\mathrm{t}}$
```

Listing 1. The first WFA coding algorithm.

We illustrate this coding process in Fig. 4, where the image on the top is encoded. Assume that the quadrants are processed[1] in the order 2, 3, 1, 4. The generated automaton has four states and Fig. 4 shows the state images of these states. The upper

[1] Note that the order in which the quadrants are processed is arbitrary. We have chosen this processing order because of visualizing reasons.

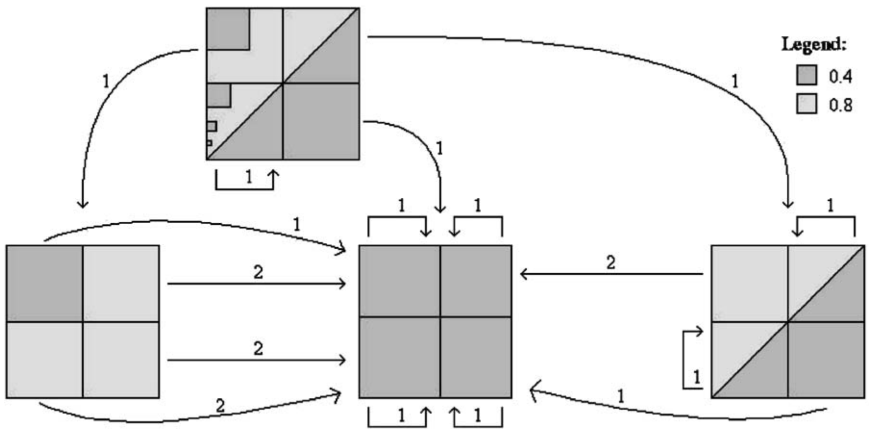

Fig. 4. Construction of a WFA. The edges are labeled with the corresponding weights and the labels of the edges are defined by the quadrants of the starting points.

state image shows $\Phi_1 = \Psi$ (the original image), the three lower state images are (from left to right) Φ_2, Φ_3 and Φ_4.

The created WFA $\mathscr{A} = (S, \Sigma, W, I, F)$ has the following components:

- the states $S = \{1,2,3,4\}$,
- $\Sigma = \{1,2,3,4\}$ since in this example we apply quadtrees for image partitioning,
- $I^t = (1,0,0,0)$ since the original image is the state image of the first state,
- $F^t = (0.5\bar{6}, 0.7, 0.4, 0.6)$ (average brightnesses of the state images),
- and the weight matrices

$$W_1 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \qquad W_2 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 \end{pmatrix},$$

$$W_3 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \qquad W_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

A lossy compression can be achieved if we only approximate the quadrants of the state images by linear combinations of other state images. For details on the lossy compression algorithm, the reader is referred to [7].

4. Advanced techniques for image partitioning

In Section 2 we have introduced some basic techniques for image partitioning. We now examine how the WFA compressor can be improved by using enhanced partitioning methods.

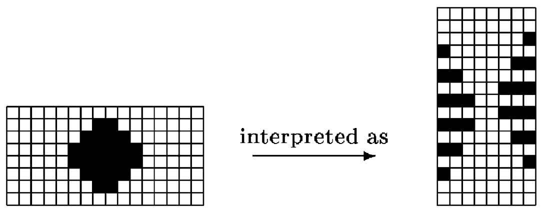

Fig. 5. Interpretation of a vector as an image of wrong resolution.

4.1. Light HV partitioning

We point out an important property of bintrees: if the first bintree partitioning is made horizontally, the image parts belonging to the leaves of the bintree are always either squares or twice as wide as high. To give the bintree more flexibility, we introduced an option to partition the square image segments either horizontally or vertically. We call this method for partitioning *light HV partitioning* since it is a special case of the HV partitioning method described in the next section. Note that this method requires the efficient storage of additional data and the improvement in image quality must be high enough to exceed the extra cost.

To explain some difficulties with the light HV partitioning when applying it to real pictures instead of multi-resolution images, we consider an image as an $n \times m$ matrix with $n, m \in \mathbb{N}$. Using the raster scan order, such an image can be interpreted as a nm vector.

It is obvious that different shapes of the image segments occur from horizontal and vertical splits. As an example, consider an image segment of size 16×16. When the image segment is partitioned vertically, we have to approximate two image segments of size 8×16. The straightforward utilization of all raster scan vectors of image segments of the sizes 8×16 and 16×8 leads to scattered images as shown in Fig. 5 (right side), which are not well suited for WFA coding. The high frequencies arise because neighboring relations (and thus correlations) are destroyed.

The following solutions exist to solve the problem of different image shapes:

1. Scaling images to the required size: As the scaling from a 16×8 to a 8×16 image wastes details in the picture, it is undesirable.

2. Scaling images to the next square size: Because the size of pictures is doubled, memory consumption and calculation time of the required scalar products are also doubled.

3. Transposition of the image matrix: This technique avoids all drawbacks mentioned above and leads to good results.

We use the heuristic of Fisher to determine the partitioning line; it is described in the next section. Fig. 6 illustrates the differences between the bintree and light HV partitioning using the well-known Lenna image.

One can see that differences occur at the left bar and the right hair strand because of the strong vertical edges.

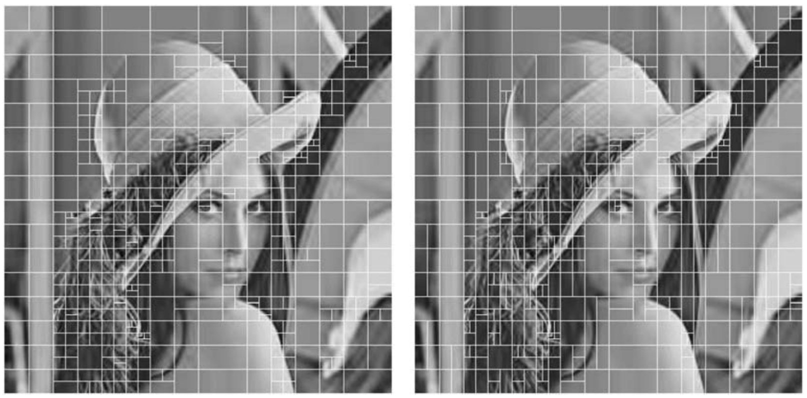
Fig. 6. Bintree (left) and light HV partitioning (right) of the image Lenna.

4.2. HV partitioning

A more flexible approach for partitioning an image was introduced by Fisher and Menlove [5] with good experimental results using an IFS encoder. In this partitioning method, the codec is able to move the partitioning line from the middle towards outer regions of the image segment.

This approach was also tested in our implementation of the WFA codec. However, we observed that the gain in image quality did not compensate the extra cost of storing the information added to the bintree structure. For this reason, the light HV partitioning mentioned above was introduced, where extra storage cost is small. Another reason why this scheme did not yield good results in our implementation might be the problem of scaling. As mentioned in the previous paragraph, image details are lost by scaling down. On the other hand, scaling up is not desirable.

We implemented the heuristic used in [5] to determine the partitioning line. For an $m \times n$ image segment with gray values $r_{i,j}$ ($i \in \{1,\dots,n\}$, $j \in \{1,\dots,m\}$), Fisher computes the *biased differences*

$$h_j = \frac{\min(j-1, m-j)}{m-1} \left| \sum_{i=1}^{n} (r_{i,j} - r_{i,j+1}) \right| \tag{16}$$

and

$$v_i = \frac{\min(i-1, n-i)}{n-1} \left| \sum_{j=1}^{m} (r_{i,j} - r_{i+1,j}) \right|, \tag{17}$$

chooses the biggest value of the set $\{h_1,\dots,h_{m-1},v_1,\dots,v_{n-1}\}$ and splits the image segment at the corresponding index (horizontally if the biggest value is one of the h_i and vertically otherwise). The resulting partitioning cuts the image segments at strong horizontal and vertical edges while avoiding narrow image segments.

Fig. 7. Grayscale images of the Waterloo image site: Barb, Boat, Goldhill, Lenna, Mandrill and Peppers.

A better partitioning could be obtained by a backtracking algorithm that checks the compression results of each choice. We did not implement that approach because the running time seems to be too high for a practical solution.

However, other refinements of HV partitioning might work well with WFA encoding. This point is a research topic for the future of WFA encoding.

5. Results

We applied the WFA codec using the standard bintree partitioning and the light HV partitioning to some commonly used images of the Waterloo Fractal Image Coding Project [9] (see Fig. 7). These are grayscale images using 8 bits per pixel.

The diagrams in Fig. 8 show the image quality measured in PSNR(dB) (peak signal to noise ratio, see [4]) plotted against the number of bits per pixel.

As one can see, the image quality can be enhanced by approximately up to 0.25 dB PSNR by using the light HV partitioning scheme. However, there is no increase in quality for some pictures (e.g., Mandrill). We also added samples of the image coding system JPEG [6], which is outperformed by the WFA codec for all of the above test images.

There are no data for the quadtree and HV partitioning scheme. The bintree partitioning scheme is a generalization of the quadtree partitioning scheme and is thus at least as powerful, while the HV partitioning scheme was dropped in an early stage of our implementation because of inferior results. The same range was used for all diagrams to visualize the compressibility of the different test images.

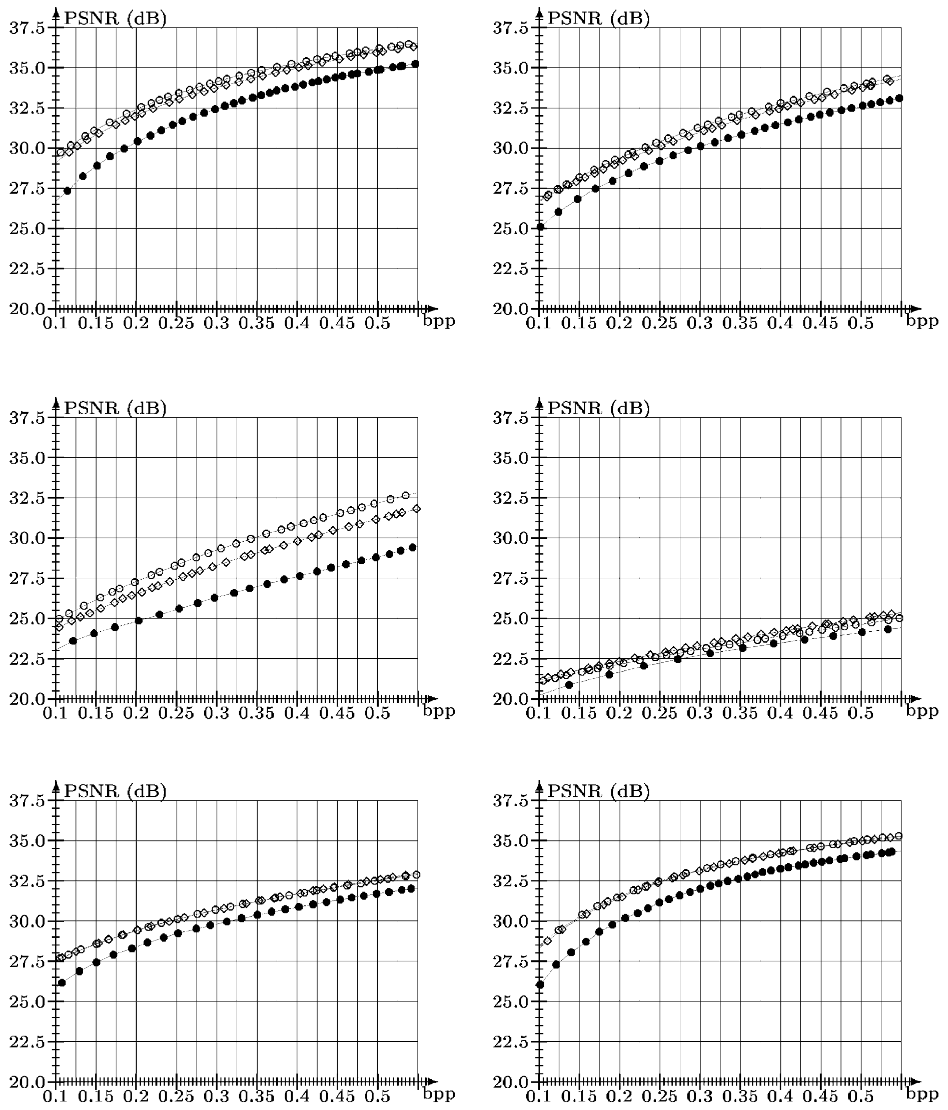

Fig. 8. Rate–distortion diagram of the images Lenna (upper left), Boat (upper right), Barb (middle left), Mandrill (middle right), Goldhill (lower left) and Peppers (lower right), comparing the different techniques: ○ ≃ WFA with light HV, ◇ ≃ WFA with bintree, ● ≃ IJPEG (Trans.).

Finally, in Fig. 9 we present two decoded Lenna images achieving approximately the same quality, measured in PSNR. However, the required bit rate drops from 0.144 bits per pixel to 0.139 when switching from the bintree partioning to the light HV partitioning.

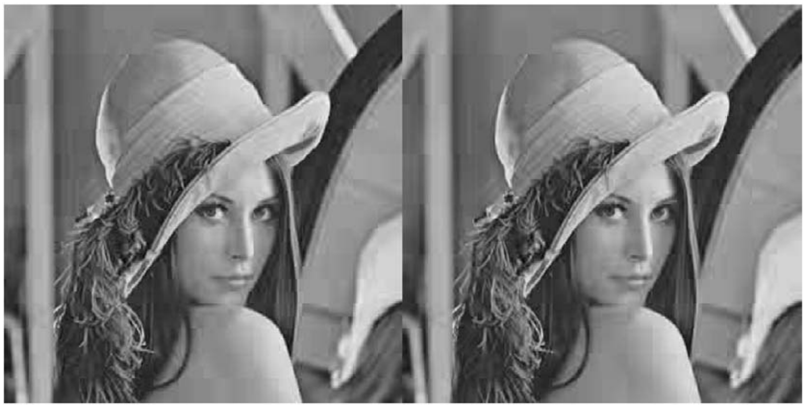

Fig. 9. Decoded images of Lenna using bintree partitioning (left: 30.65 dB PSNR, 0.144 bpp) and light HV partitioning (right: 30.83 dB PSNR, 0.139 bpp).

References

[1] K. Culik II, J. Kari, Image compression using weighted finite automata, Comput. Graphics 17 (1993) 305–313.

[2] K. Culik II, J. Kari, Inference algorithms for WFA and image compression, in: Y. Fisher (Ed.), Fractal Image Compression, Springer, Berlin, Heidelberg, New York, 1995, pp. 243–258.

[3] K. Culik II, V. Valenta, Finite automata based compression of bi-level and simple color images, Comput. Graphics 21 (1997) 61–68.

[4] Y. Fisher (Ed.), Fractal Image Compression, Springer, Berlin, Heidelberg, New York, 1995.

[5] Y. Fisher, Fractal Encoding with HV Partitions, in: Y. Fisher (Ed.), Fractal Image Compression, Springer, Berlin, Heidelberg, New York, 1995, pp. 119–136.

[6] The Independent JPEG Group, JPEG software release 6b, 1998. ftp://ftp.uu.net/graphics/jpeg/jpegsrc.v6b.tar.gz.

[7] F. Katritzke, Refinements of data compression using weighted finite automata, Ph.D. Thesis, University of Siegen, 2001.

[8] A. Rao, V.D. Pandit, R.U. Udupa, Efficient decoding algorithms for weighed finite automata, Department of Computer Science, S.J. College of Engineering, Mysore, 1996.

[9] University of Waterloo, Ontario, Canada, Waterloo BragZone: comparison of image compression systems, http://www.uwaterloo.ca/bragzone.base.html.