Chapter 11: Digital Image Compression

Jürgen Albert[1] and Jarkko Kari[2]

[1] Informatik II, Universität Würzburg, 97074 Würzburg, Germany
albert@informatik.uni-wuerzburg.de
[2] Department of Mathematics, University of Turku, 20014 Turku, Finland
jkari@utu.fi

1 Introduction

Regular languages have a self-similar structure, given explicitly in their finite automata descriptions. This fractal nature can be observed visually by using a suitable interpretation of words of the language as addresses of black image pixels.

Under a similar addressing scheme, weighted languages and automata define grayscale images: the weight of a word gives the intensity of the corresponding image pixel. It turns out that weighted finite automata (WFA) are already powerful enough to describe both fractal-like and smooth images. Simple inference algorithms exist for constructing WFA representations of given images, and standard automata minimization techniques can be used to minimize the number of states. This naturally leads to the idea of using

M. Droste, W. Kuich, H. Vogler (eds.), *Handbook of Weighted Automata*, Monographs in Theoretical Computer Science. An EATCS Series,
DOI 10.1007/978-3-642-01492-5_11,

such automata as compressed representations of images. This concept was introduced in the early 1990s [7, 9–11], and it soon led to further improved and generalized algorithms; see, e.g., [3, 4, 16, 21, 22, 25, 26]. See also [24] for a tutorial and further examples. In this article, the basic concepts and algorithms are presented.

We start in Sect. 2 by discussing our addressing scheme of pixels using words over a four-letter alphabet. Infinite languages define infinitely sharp images, so we next briefly discuss the concept of multi-resolution images. A multi-resolution image is simply a formal power series, that is, a function that assigns colors to words. In case the color set is a semiring, weighted finite automata can be used to describe power series. This is the topic of Sect. 3. Most of the time, the semiring used is $\mathbb{R}$; the set of real numbers under normal addition and multiplication. In Sect. 4, we discuss how one can efficiently draw the image specified by a given WFA. Next, we turn our attention to the converse problem: Sect. 5 concentrates on the problem of inferring a WFA that represents a given input image. We provide an efficient algorithm for this task. The algorithm is guaranteed to produce the minimum state WFA. In Sect. 6, we outline the ideas used to transform the theoretical inference algorithm into a practical image compression technique. The details of these ideas are skipped and the interested reader is referred to the more technical descriptions [10, 23]. We compare our algorithm with the image compression standard JPEG [31] that is based on the discrete cosine transform. Using several test images, we show the difference in performance, and we demonstrate that the WFA technique performs especially well on images with sharp edges. Then in Sect. 7, we briefly discuss one nice aspect of the WFA representation of images: several natural image transformations can be implemented directly in the compressed form, without the need to decode the image first. We show several examples of image operations that can be defined using weighted finite transducers (WFT). Finally, we stress the fractal nature of WFA-concepts in Sect. 8 by generalizing from WFA to higher-dimensional parametric weighted finite automata, which can, e.g., simulate any iterated function system easily.

2 Image Types

A *finite resolution* image is a $w \times h$ rectangular array of pixels, each with its own color. In the context of this work, squares of dimensions that are powers of two come up naturally, so that

$$w = h = 2^k \quad \text{for some } k \in \mathbb{Z}_+.$$

Definition 2.1. *The image is then just a function*

$$\{1, 2, \ldots, w\} \times \{1, 2, \ldots, h\} \to S$$

that assigns to each pixel a color from a set S of possible colors. The color set depends on the type of the image. It can be:

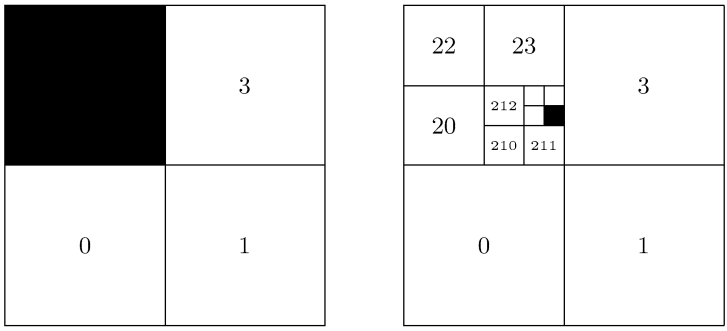


Fig. 1. Pixels with addresses 2 and 2131 respectively

- $S = \mathbb{B} = \{0, 1\}$ *on bi-level or binary images, where* 0 *and* 1 *represent white and black, respectively.*
- $S = \mathbb{R}$*, the set of reals, on grayscale images. The color value represents the intensity of the pixel. Sometimes only non-negative real numbers* $S = \mathbb{R}_+$ *are allowed, sometimes the values are restricted to an interval, e.g.,* $S = [0, 1]$*. In digital images the intensity values are quantized in a finite number of intensity levels. However, in this article, we always use* $S = \mathbb{R}$*.*
- $S = \mathbb{R}^3$*, vectors of three real numbers, on color images. Three values represent the intensities of three color components. Also,* $\mathbb{R}^3_+$ *or* $[0, 1]^3$ *may be used.*

Rather than addressing the pixels by their x- and y-coordinates, let us address the pixels using words over the four-letter alphabet $\Sigma = \{0, 1, 2, 3\}$. Pixels of a $2^k \times 2^k$ image are addressed by words of length k as follows.

Definition 2.2. *The image is first divided into four quadrants. The first letter of the address of a pixel is determined by the quadrant that contains the pixel.*

The rest of the address is defined recursively as the word of length $k - 1$*, that is, the address of the pixel in the quadrant when the quadrant is viewed as a* $2^{k-1} \times 2^{k-1}$ *image.*

The only pixel of a 1×1 resolution "image" has the empty word ε as address.

Example 2.3. Figure 1 shows the locations of the pixels addressed by 2 at resolution 2×2 and 2131 at resolution 16×16, respectively.

Our addressing scheme thus defines a *quadtree* structure on the pixels of the images. Quadtrees (also octtrees, etc.) can be embedded into *bintrees* in a canonical way when encoding the alphabet symbols $0, 1, 2, 3$ by their binary number-representations $00, 01, 10, 11$. For addresses of odd length, this opens a possibility to address also images and sub-images of sizes $2^{k+1} \times 2^k$. Bintrees have been used successfully for fine-tuning the encoding algorithm presented

in the section on practical image compression (see, e.g., [15]). For simplicity of the arguments, we will stick to the common quadtree representations for the following.

A $2^k \times 2^k$ resolution image is now a function

$$r_k : \Sigma^k \to S$$

that assigns a color to each pixel.

Definition 2.4. *A multi-resolution image is a function that assigns a color to each pixel in resolutions $2^k \times 2^k$ for all $k \geq 0$. In other words, it is a formal power series $r \in S\langle\langle \Sigma^* \rangle\rangle$ where $\Sigma = \{0, 1, 2, 3\}$ and S is the color set. Coefficient (r, w) is the color of pixel w.*

A multi-resolution image can be viewed as a sequence $r_0, r_1, r_2, \ldots$ of images where r_k, the restriction of r to words of length k, is a $2^k \times 2^k$ image.

Note that without any additional constraint on a multi-resolutions image r, the images r_k may be completely unrelated. However, one would like them to represent approximations of the same image at different resolutions. This constraint is captured into the concept of average preservation. Let us consider grayscale images, i.e., the case $S = \mathbb{R}$. A simple way to interpolate a $2^k \times 2^k$ resolution grayscale image into a $2^{k-1} \times 2^{k-1}$ resolution image is to average the intensity values on 2×2 blocks of pixels.

Definition 2.5. *Let r be a multi-resolution grayscale image. If*

$$(r, w) = \frac{(r, w0) + (r, w1) + (r, w2) + (r, w3)}{4}$$

for every $w \in \Sigma^*$, then r_{k-1} is the interpolation of the next sharper image r_k, for every $k = 1, 2, \ldots$. In this case, we say that r is average preserving, or ap for short.

In the quadtree form, this property simply states that each node is the average of its four children. If r is average preserving, the images in the sequence $r_0, r_1, r_2, \ldots$ form sharper and sharper approximations of some infinitely sharp grayscale image.

In image processing algorithms, we typically rely on efficient algorithms of linear algebra. These are available because $\mathbb{R}\langle\langle \Sigma^* \rangle\rangle$ is a linear space over $\mathbb{R}$ under the usual pointwise sum and scalar product. Note that the set of average preserving multi-resolution images is a linear subspace.

In an ideal situation, multi-resolution images are used. In practical applications in digital image processing, one only has finite resolution $2^k \times 2^k$ images available. They also form a linear space that is isomorphic to $\mathbb{R}^{4^k}$. Here, we can find an expression of a given finite resolution image r_k as a linear combination of finite resolution images $\psi_1, \ldots, \psi_n$, if such an expression

$$r_k = c_1 \cdot \psi_1 + c_2 \cdot \psi_2 + \cdots + c_n \cdot \psi_n$$

with $c_1, c_2, \ldots, c_n \in \mathbb{R}$ exists.

3 Weighted Finite Automata and Multi-resolution Images

Suppose the color set S is a semiring. This is the case in all our examples where $S = \mathbb{B}$ (bi-level image), $S = \mathbb{R}$ (grayscale image) or $S = \mathbb{R}^3$ (color image). Then multi-resolution images can be described using weighted automata.

Weighted finite automata (WFA) are finite automata whose transitions are weighted by elements of the coloring semiring S, and whose states have two associated weights called the initial and the final distribution values. In our applications, the input alphabet is $\Sigma = \{0, 1, 2, 3\}$, and the semiring S is $\mathbb{B}$, $\mathbb{R}$, or $\mathbb{R}^3$.

Definition 3.1. *A WFA is specified by:*

- *The finite state set Q*
- *Four transition matrices $A_0, A_1, A_2, A_3 \in S^{Q\times Q}$*
- *A final distribution vector $F \in S^{Q\times 1}$*
- *An initial distribution vector $I \in S^{1\times Q}$*

The WFA defines a multi-resolution image r as follows:

$$(r, a_1 a_2 \dots a_k) = I A_{a_1} A_{a_2} \dots A_{a_k} F$$

for all $a_1 a_2 \dots a_k \in \Sigma^*$. Let us use the following shorthand notation: For every $w = a_1 a_2 \dots a_k \in \Sigma^*$, let

$$A_w = A_{a_1} A_{a_2} \dots A_{a_k}$$

be the product of the matrices corresponding to the letters of the word w. Then $(r, w) = I A_w F$.

Usually a WFA is drawn as a labeled, weighted directed graph. The vertex set is the state set Q, and there is an edge from vertex $i \in Q$ to vertex $j \in Q$ with label $a \in \Sigma$ and weight $s \in S$ iff $(A_a)_{ij} = s$. Edges with weight 0 are usually not drawn. The initial and final distribution values are marked inside the vertices.

Example 3.2. Let $S = \mathbb{B}$, $|Q| = 2$ and

$$I = \begin{pmatrix} 1 & 0 \end{pmatrix} \qquad A_0 = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \qquad A_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$
$$F = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad A_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad A_3 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}.$$

Weighted finite automata over $\mathbb{B}$ are classical finite automata. The directed graph representation of the sample automaton is

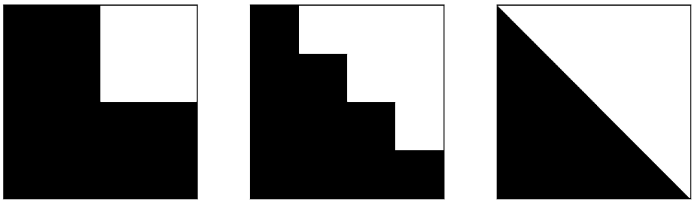
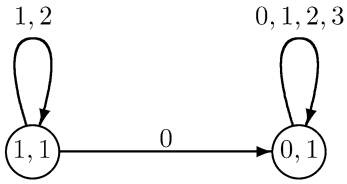
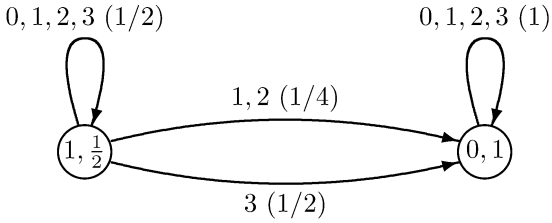


Fig. 2. Finite resolution images defined by the finite automaton of Example 3.2



Here, all edges are with weight 1. A regular expression for the accepted language is $(1 + 2)^* + (1 + 2)^*0\Sigma^*$. Figure 2 shows the corresponding finite resolution images at resolutions 2×2, 4×4, and 256×256.

Example 3.3. Let $S = \mathbb{R}$, $|Q| = 2$ and



The weight of each edge is given in parentheses after the label.

The finite resolution images defined by this WFA at resolutions 2×2, 4×4, and 256×256 are shown in Fig. 3. Here (and in all our grayscale examples), value 0 is drawn black, value 1 in white, and intermediate values give different shades of gray.

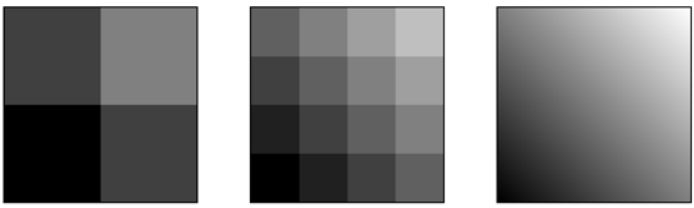


Fig. 3. Finite resolution images defined by the weighted finite automaton of Example 3.3

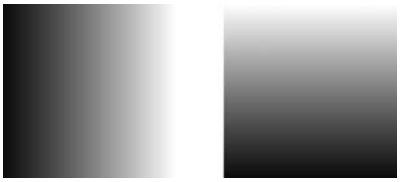


Fig. 4. High resolution images generated by the two state WFAs of Example 3.4

Let $S = \mathbb{R}$. If

$$(A_0 + A_1 + A_2 + A_3) \cdot F = 4F \tag{1}$$

holds then

$$(r, w0) + (r, w1) + (r, w2) + (r, w3) = 4(r, w)$$

for all $w \in \Sigma^\star$. In other words, the multi-resolution image specified by the automaton is then average preserving. Therefore, a WFA is called *average preserving* if (1) is satisfied. Note that condition (1) states that number 4 is an eigenvalue of the matrix $A_0 + A_1 + A_2 + A_3$, and F is a corresponding eigenvector.

Given an n-state WFA A and an m-state WFA B that compute multi-resolution functions r_1 and r_2, respectively, it is easy to construct:

- An $(n + m)$-state WFA that computes the sum $r_1 + r_2$
- An n-state WFA that computes the point-wise scalar multiple sr_1 of r_1 by $s \in \mathbb{R}$
- An nm-state WFA that computes the Hadamard product $r_1 \odot r_2$

Example 3.4. The 2-state WFAs

generate the linear functions of Fig. 4. From these and the constant function 1, one can build a WFA for any polynomial.

4 Drawing WFA Images

In this section, we consider the following *decoding problem*: Given a WFA, draw the corresponding image at some specified finite resolution $2^k \times 2^k$. Decoding at resolution $2^k \times 2^k$ involves forming the matrix products IA_wF for all $w \in \Sigma^k$. Note that the number of multiplications (and additions) required by the trivial algorithm to compute the product of:

- Two $n \times n$ matrices is n^3
- An $n \times n$ matrix and an n-vector is n^2
- Two n-vectors is n

Observe that it is naturally better to multiply vectors than matrices.

Decoding Algorithm #1

1. Form the products IA_w for all non-empty words $w \in \Sigma^{\leq k}$ in the order of increasing length of w. Because $IA_{ua} = IA_uA_a$, we need one product of a vector and a matrix for each word $w = ua$.
2. In the end, for every $w \in \Sigma^k$, multiply vectors IA_w and F.

Let us analyze the complexity of the algorithm. Let $N = 4^k$ be the number of pixels, and let n be the number of states. The number of non-empty words of length $\leq k$ is

$$4 + 16 + \cdots + 4^k \leq N\left(1 + \frac{1}{4} + \frac{1}{16} + \cdots\right) = \frac{4}{3}N.$$

Step 1 of the algorithm requires then at most $\frac{4}{3}Nn^2$ multiplications. Step 2 requires Nn multiplications so the total number of multiplications is at most $Nn(1 + \frac{4n}{3})$. Let us consider then a more efficient alternative.

Decoding Algorithm #2

1. Use the first step of the decoding algorithm #1 to compute the products IA_u for words u of length $k/2$ and products A_vF for words v of length $k/2$. If k is odd, then we round the lengths so that u has length $\lfloor k/2 \rfloor$ and v has length $\lceil k/2 \rceil$.
2. Form all possible products $(IA_u)(A_vF)$ for all $u, v \in \Sigma^{k/2}$.

Step 1 requires at most $2 \times \frac{4}{3}4^{k/2}n^2 = \frac{8}{3}\sqrt{N}n^2$ multiplications. Step 2 requires Nn multiplications. Now the total is $Nn(1 + \frac{8n}{3\sqrt{N}})$ multiplications. This is considerably better than algorithm #1, especially when N, the number of pixels, is very large.

5 An Encoding Algorithm

In this chapter, we assume that $S = \mathbb{R}$. For $r \in S\langle\langle\Sigma^*\rangle\rangle$ and $w \in \Sigma^*$ we denote by $w^{-1}r$ the left quotient of r by w. It is the power series defined as follows:

$$\left(w^{-1}r, u\right) = (r, wu)$$

for all $u \in \Sigma^*$. Intuitively speaking, $w^{-1}r$ is the image obtained from image r by zooming into the sub-square whose address is w.

In a WFA A, the transitions A_0, A_1, A_2, and A_3 and the final distribution F define a multi-resolution image ψ_i for every state i: Image ψ_i is the multi-resolution image computed by the WFA that is obtained from A by changing the initial distribution so that the initial distribution value of state i is 1, and all other states have initial distribution 0. In other words, for every $w \in \Sigma^*$, we have

$$(\psi_i, w) = (A_w F)_i$$

where we use the notation that $(A_w F)_i$ is the ith component of the vector $A_w F$. Multi-resolution ψ_i is called the image of state i. It is average preserving if the WFA is.

The WFA gives a mutually recursive definition of the ψ_i multi-resolutions:

- $(\psi_i, \varepsilon) = F_i$, that is, the final distribution values are the average intensities of the state images.
- For every $a \in \Sigma, w \in \Sigma^\star$

$$\begin{aligned}\left(a^{-1}\psi_i, w\right) = (\psi_i, aw) = (A_{aw}F)_i = [A_a(A_w F)]_i = \sum_{j=1}^{n} (A_a)_{ij}(A_w F)_j \\ = s_1(\psi_1, w) + s_2(\psi_2, w) + \cdots + s_n(\psi_n, w)\end{aligned}$$

  where $s_j = (A_a)_{ij}$ is the weight of the transition from state i into state j with label a. Since the coefficients $s_j = (A_a)_{ij}$ are independent of w, we have

$$a^{-1}\psi_i = s_1\psi_1 + s_2\psi_2 + \cdots + s_n\psi_n$$

  in the vector space of multi-resolution images. In other words, the ith row of the transition matrix A_a tells how the quadrant a of the state image ψ_i is expressed as a linear combination of state images $\psi_1, \ldots, \psi_n$. See Fig. 5 for an illustration.
- The initial distribution I tells how the multi-resolution image r computed by the WFA is composed of state images $\psi_1, \psi_2, \ldots, \psi_n$:

$$r = I_1 \cdot \psi_1 + I_2 \cdot \psi_2 + \cdots + I_n \cdot \psi_n.$$

The final distribution values and the transition matrices define state images $\psi_1, \ldots, \psi_n$ uniquely.

The following algorithm infers a minimum state WFA for a given multi-resolution function r:

Encoding Algorithm for Grayscale Images

Input: multi-resolution image r
Variables: n : number of states so far
i : first non-processed state
ψ_j : Image of state j

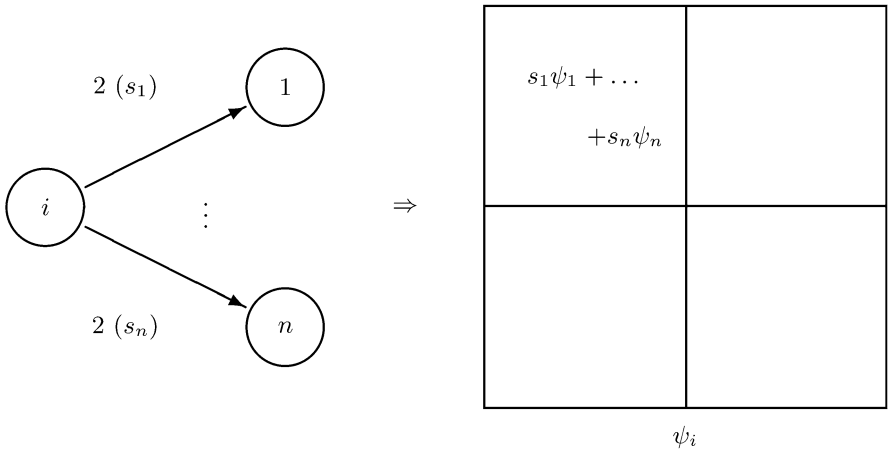
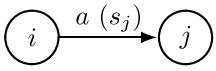
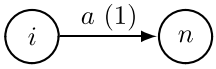


Fig. 5. The transitions from state i with label a specify how the quadrant a of the state image ψ_i is expressed as a linear combination of the state images $\psi_1, \ldots, \psi_n$

1. $n \leftarrow 1, i \leftarrow 1, \psi_1 \leftarrow r$
2. For quadrants $a = 0, 1, 2, 3$ do
   (a) If $\exists s_1, \ldots, s_n \in \mathbb{R}$ such that $a^{-1}\psi_i = s_1\psi_1 + \cdots + s_n\psi_n$, then add the transitions

   for all $j = 1, 2, \ldots, n$
   (b) Else create a new state: Set $n \leftarrow n + 1$, $\psi_n \leftarrow a^{-1}\psi_i$, and add the transition

3. $i \leftarrow i + 1$. If $i \leq n$, then goto 2
4. Initial distributi.on: $I_1 = 1, I_i = 0$ for $i = 2, 3, \ldots, n$
   Final distribution: $F_i = (\psi_i, \epsilon)$ for $i = 1, 2, \ldots, n$

Theorem 5.1 ([11]). *Let $r \in \mathbb{R}\langle\langle \Sigma^* \rangle\rangle$.*

- *Multi-resolution image r can be generated by a WFA if and only if the multi-resolution images*

$$u^{-1}r, \quad \textit{for all } u \in \Sigma^\star,$$

  generate a finite dimensional vector space. The dimension of the vector space is the same as the smallest possible number of states in any WFA that generates r.
- *If r can be generated by a WFA, then the algorithm above produces a WFA with the minimum number of states.*

- *If r is average preserving then the algorithm produces an average preserving WFA.*

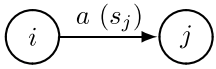
The setup of the encoding algorithm above is theoretical: the input r is an infinite object but the input of a concrete algorithm should have a finite presentation. In practice, the input to the algorithm is a finite resolution image. When encoding a $2^k \times 2^k$ size image, the subtrees $w^{-1}r$ are only known up to depth $k - |w|$. When forming linear combinations, the values deeper in the tree are "do not care" nodes, that is, it is enough to express the known part of each subtree as a linear combination of other trees and the do not care nodes may get arbitrary values. Fortunately, as the process is done in the breadth-first order, the sub-images are processed in the decreasing order of size. This means that all previously created states have trees assigned to them that are known at least to the depth of the current subtree. Hence, the don't care values of prior images only can affect the don't care values of the present image, and the linear expressions are precise at the known part of the tree.

6 Practical Image Compression Using WFA

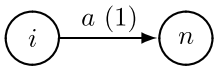
In the previous section we saw how we can find a minimum state WFA for a given grayscale image. If the WFA is small in size, then the WFA can be used as a compressed representation of the image. However, the encoding algorithm of the previous section as such is ill suited for image compression purposes. Even though the resulting WFA is minimal with respect to the number of states, it may have a very large number of edges. The algorithm also represents the image exactly, and the last details of the image often require a very large increase in the size of the WFA. This is called *lossless* image compression. More often one is interested in *lossy* compression where small errors are allowed in the regenerated image, if this admits sufficient decrease in the size of the compressed image file. In this section, we outline how the encoding algorithm can be modified into a practical image compression method. Note that the techniques are heuristic in nature and no guarantees of optimality exist. We can only use compression experiments and comparisons with other algorithms to show their usefulness.

Recall step 2 of the encoding algorithm from the previous section:

(a) If $\exists s_1, \ldots, s_n \in \mathbb{R}$ such that $a^{-1}\psi_i = s_1\psi_1 + \cdots + s_n\psi_n$, then for all $j = 1, 2, \ldots, n$ add the transitions



(b) Else create a new state: Set $n \leftarrow n+1$, $\psi_n \leftarrow a^{-1}\psi_i$, and add the transition



It turns out that better results are obtained in practice if we try both alternatives (a) and (b), and choose the one that gives the smaller automaton. It may namely happen that the linear combination in (a) contains so many coefficients that it is better to create a new state in (b) and process its quadrants.

In order to make a fair comparison between (a) and (b), we need to process the new state created in (b) completely before we can determine which choice to make. Therefore, we change the order of processing the states from the breadth-first order into the depth-first order: instead of processing all four quadrants of ψ_i before advancing to the next state image, all new states created by a quadrant are processed before advancing to the next quadrant.

Let us first measure the size of the automaton by the quantity

$$|E| + P \cdot |V|$$

where $P \in \mathbb{R}_+$ is a given constant, $|E|$ is the number of transitions, and $|V|$ is the number of states of the automaton. Constant P is a Lagrange multiplier that formulates the relative cost of states vs. edges in the automaton. If we want to minimize the number of edges, we set $P = 0$. The goal of the next inference algorithm is to find for a given multi-resolution image ψ a small WFA in the sense that the value $|E| + P \cdot |V|$ is small.

Because we now process the quadtree in the depth-first order, it is natural to make the algorithm recursive. The new encoding algorithm consists of a recursive routine *make_wfa(ψ_i,max)* that adds new states and edges to the WFA constructed so far, with the goal of representing the state image ψ_i in such a way that the value $\Delta_E + P \cdot \Delta_V$ is small, where Δ_E and Δ_V are the numbers of new transitions and states added by the recursive call. If $\Delta_E + P \cdot \Delta_S \leq max$ then the routine returns the value $\Delta_E + P \cdot \Delta_V$, otherwise it returns ∞ to indicate that no improvement over the target value *max* was obtained.

Encoding Algorithm #2 [10]

Input: Multi-resolution image r and a positive real number P
Global variables used: n : number of states
ψ_j : image of state j

1. $n \leftarrow 1, \psi_1 \leftarrow r$
2. *make_wfa(ψ_1, ∞)*

make_wfa(ψ_i,max) :

1. If $max < 0$ then return(∞)
2. Set $cost \leftarrow 0$

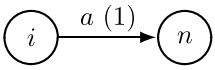
3. For quadrants $a = 0, 1, 2, 3$ do
   (a) If $\exists s_1, \ldots, s_n \in \mathbb{R}$ such that $a^{-1}\psi_i = s_1\psi_1 + \cdots + s_n\psi_n$ then

$$cost1 \leftarrow \text{number of non-zero coefficients } s_j$$

   else

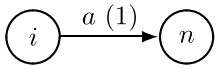
$$cost1 \leftarrow \infty$$

   (b) Set $n_0 \leftarrow n$, $n \leftarrow n + 1$, $\psi_n \leftarrow a^{-1}\psi_i$ and add the transition

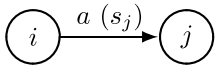


   (c) Set $cost2 \leftarrow P + make\_wfa(\psi_n, min\{max - cost - P, cost1 - P\})$
   (d) If $cost2 \leq cost1$ then
   - $cost \leftarrow cost + cost2$

   else
   - $cost \leftarrow cost + cost1$
   - remove all transitions from states $n_0 + 1, \ldots, n$, and set $n \leftarrow n_0$
   - remove the transition



   - add transitions



   for $s_j \neq 0$
4. If $cost \leq max$ then return($cost$) else return(∞)

A few words to explain the algorithm: The main step is line 3 where we try to find a WFA representation for each of the four quadrants of ψ_i. For each quadrant, we try two alternatives: (a) to express the quadrant as a linear combination of existing states, and (b) to create a new state whose state image is the quadrant, and to recursively process the new state. In variable *cost1,* we store the cost of alternative (a), i.e., the number of transitions created in the automaton, and in *cost2* we store the cost of alternative (b), i.e., the sum of P (for the new state created) and the cost returned from the recursive call to process the new state. In step 3(d), the algorithm chooses the better of the two alternatives.

The algorithm above is still lossless. The WFA represents the input image precisely, without any loss of detail. Much better compression is obtained if we allow small errors in the regenerated image whenever that helps to compress the image more. There is a trade-off between the amount of image degradation and the size of the compressed image. Let us measure the amount of degradation by the square difference metric $d(\cdot, \cdot)$. For two images ψ and ϕ at resolution $2^k \times 2^k$, this metric defines the image difference value as

$$d(\psi, \phi) = \sum_{w \in \Sigma^k} \big((\psi, w) - (\phi, w)\big)^2.$$

This is a reasonable measure for the reconstruction error. It is also convenient for our linear combinations approach since it is the square of the normal Euclidean distance.

We also introduce a new Lagrange multiplier G that controls the trade-off between the image size and the reconstruction error. Parameter G is given as an input by the user, and the algorithm will produce a WFA that generates an image ψ' such that the value of

$$d(\psi, \psi') + G \cdot S$$

is small, where ψ is the image to be compressed and S is the size of the WFA constructed by the algorithm. We may continue using

$$S = |E| + P \cdot |V|,$$

but in practical image compression it is better to define S as the actual number of bits required to store the WFA in a file. The WFA is stored using suitable entropy coder, e.g., an arithmetic coder. See [23] for details on how the different items such that states, transitions, and weights are stored using arithmetic coding.

The Lagrange multiplier G is the parameter that the user can change to adjust the file size and the image quality:

Small $G \Rightarrow$ big automaton, small error
Large $G \Rightarrow$ small automaton, big error

The following modifications to the algorithm were also made:

- Edges back to states that have not yet been completely processed are problematic in lossy coding, as the actual image of those states is not yet precisely known. Therefore, we opted to only allow edges to states already completely processed. Note that this prevents the creation of any loops in the WFA.
- In order to have at least one processed state to begin with, we introduce an initial base: Before calling *make_wfa* the first time, we set $n \leftarrow N$ with some fixed images $\psi_1, \ldots, \psi_N$. In our tests below, we used $N = 6$ base images that were linearly independent quadratic polynomials, i.e., functions 1, x, y, x^2, y^2, and xy.

With these modifications, we have a practical encoding algorithm that compares favorably with other compression techniques. Let us compare WFA compression with the JPEG image compression standard using the test image of Fig. 6. This color image contains large smooth areas, and is therefore well suited for JPEG compression. A color image consists of three color layers, each

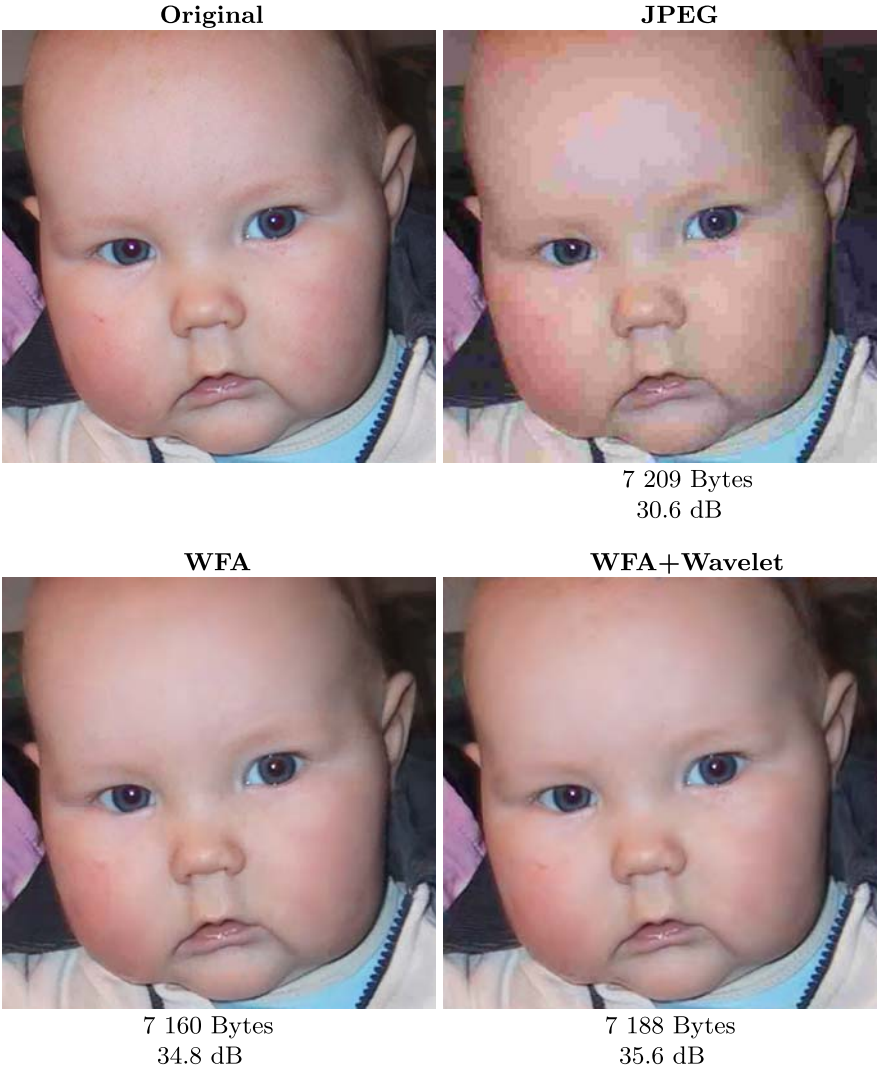


Fig. 6. A comparison of JPEG and the WFA compression at a low bitrate

of which is compressed as a grayscale image. However, only one automaton is built, that is, different color components can refer to the sub-images of other color layers.

In the compression experiment, the reconstruction errors are reported as the peak signal-to-noise ratio (PSNR). The units of this measure are decibels (dB). The PSNR value is directly obtained from the square difference $d(\cdot,\cdot)$ as follows:

$$\mathrm{PSNR}(\cdot,\cdot) = 10\log_{10}\left(\frac{A^2}{\sigma}\right)$$

Fig. 7. The rate-distortion comparison of JPEG, WFA compression, and WFA compression of the wavelet coefficients

where A is the maximum intensity value ($A = 255$ in our 8 bits-per-pixel image) and

$$\sigma = \frac{d(\cdot,\cdot)}{\#\text{ of pixels}}$$

is the average square difference between the pixel values of the two images. Notice that larger PSNR values mean better quality.

Our visual comparison in Fig. 6 is between compressed images at the very low bitrate of 7.2 KBytes. The WFA compressed image is more than 4 dB better than the JPEG compressed image at the same bitrate. As we increase the bitrate, the JPEG algorithm catches up with WFA.

Very good compression performance is obtained if the WFA encoding algorithm is applied to an image composed of wavelet coefficients instead of the original image. The sub-bands obtained from the wavelet transformation are arranged into a so-called Mallat pyramid, and this is compressed as an image using the WFA encoding algorithm. The WFA algorithm is able to take advantage of the self-similarity of different sub-bands in the wavelet transformation. The subdivision into quadrants used by our algorithm matches with the organization of the sub-bands in the Mallat form. A result is provided in Fig. 6 for visual comparison. In our tests, the Daubechies W6 wavelets are used.

Figure 7 summarizes numerically the rate-distortion performances of JPEG, WFA, and WFA with wavelets for the test image of Fig. 6. All three algorithms were used at various bitrates, and the bitrate versus image quality values were plotted. Note how JPEG surpasses WFA at 16 KByte compression for this type of image characteristics.

WFA (without wavelets) compress very well images with sharp edges. As a second experiment, let us compare WFA compression and JPEG on the test image shown in Fig. 8. The difference between JPEG and WFA is clear even at high quality setting, as seen in Fig. 8.

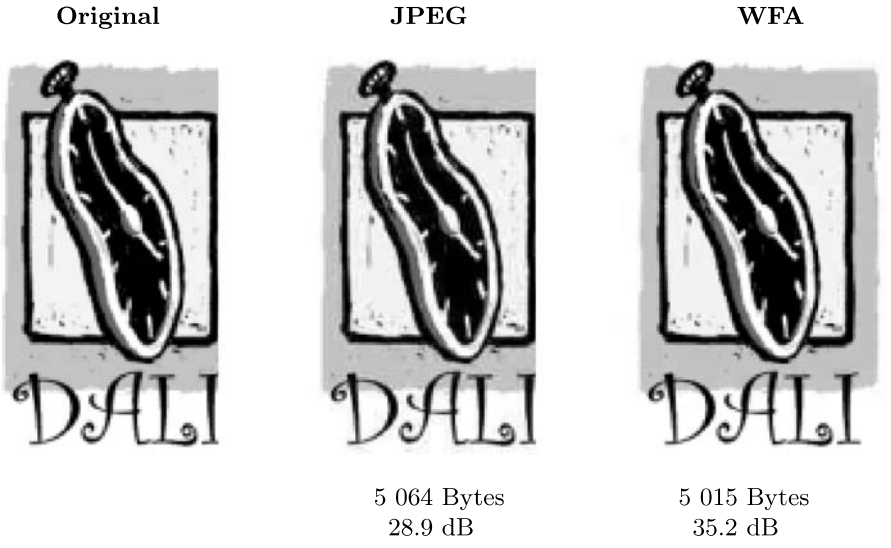


Fig. 8. The second test image that contains sharp edges, and the compression results using JPEG and WFA compression at 5 KBytes

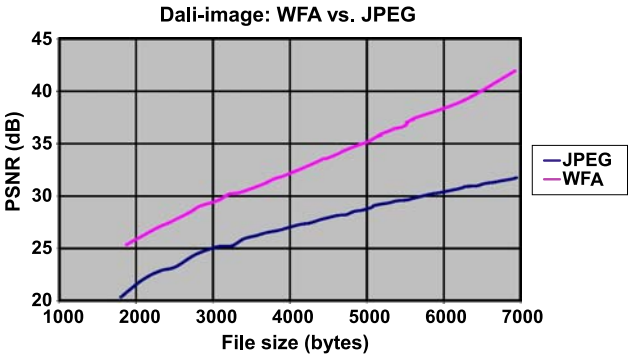


Fig. 9. The rate-distortion comparison of JPEG and WFA compression on the second test image

Numerical comparisons in Fig. 9 indicate that WFA compression remains superior through all bitrates.

7 Weighted Finite Transducers (WFT)

A nice feature of WFA image representations is the property that one can perform interesting and useful image operations directly in the WFA form. Bi-level images and regular languages can be transformed using finite state transducers. Analogously, grayscale images and WFA are transformed using

finite state transducers with real weights. More details of the examples and results in this section can be found in [8]. We assume throughout this chapter that the semiring used is $S = \mathbb{R}$.

A *Weighted Finite Transducer (WFT)* is obtained by introducing edge weights and initial and final distribution values to an ordinary finite state transducer. The transitions are labeled by pairs a/b where $a, b \in \Sigma \cup \{\varepsilon\}$.

Definition 7.1. *More precisely, a WFT is specified by:*

- *The finite state set* Q
- *Transition matrices* $A_{a,b} \in \mathbb{R}^{Q\times Q}$ *for all* $a, b \in \Sigma \cup \{\varepsilon\}$
- *Final distribution vector* $F \in \mathbb{R}^{Q\times 1}$
- *Initial distribution vector* $I \in \mathbb{R}^{1\times Q}$

The WFT is called ε*-free if the weight matrices* $A_{a,\varepsilon}$*,* $A_{\varepsilon,b}$*, and* $A_{\varepsilon,\varepsilon}$ *are zero matrices for all* $a, b \in \Sigma$*. The WFT defines a function*

$$\rho : \Sigma^* \times \Sigma^* \to \mathbb{R}$$

called a weighted relation as follows: For every $u, v \in \Sigma^*$*, we have*

$$\rho(u, v) = IA_{u,v}F$$

where

$$A_{u,v} = \sum_{\substack{a_1 \dots a_m = u \\ b_1 \dots b_m = v}} A_{a_1,b_1} \dots A_{a_m,b_m}$$

if the sum converges. The sum is over all decompositions of u *and* v *into symbols* $a_i, b_i \in \Sigma \cup \{\varepsilon\}$*. Note that the sum is finite (and hence converges) if the WFT does not contain any cycles that read* ε/ε*.*

If the WFT is ε*-free, then*

$$\rho(a_1 \dots a_k, b_1 \dots b_k) = IA_{a_1,b_1} \dots A_{a_k,b_k}F$$

where all $a_i, b_i \in \Sigma$*, and* $\rho(u, v) = 0$ *when* $|u| \neq |v|$*.*

Next, we define the action of a weighted relation ρ on a multi-resolution image f. The result is a new multi-resolution function $g = \rho(f)$, defined by

$$g(w) = \sum_{u\in\Sigma^*} f(u)\rho(u, w), \quad \text{for all } w \in \Sigma^*,$$

provided the sum converges. The sum is finite if the WFT is ε-free, or more generally, if the weight matrices $A_{a,\varepsilon}$ are zero, for all $a \in \Sigma \cup \{\varepsilon\}$. In this case, the sum is over all words u whose length is not greater than the length of w.

It is easy to see that the operator

$$\rho : \mathbb{R}\langle\langle \Sigma^* \rangle\rangle \to \mathbb{R}\langle\langle \Sigma^* \rangle\rangle$$

(a) (b) (c)

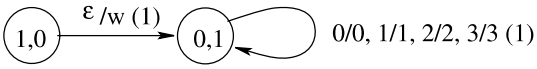
Fig. 10. The output of transforming the test image using the WFT of Examples 7.2, 7.3, and 7.4

is linear, that is, for arbitrary multi-resolution functions f and g, and arbitrary real numbers x and y we have

$$\rho(xf + yg) = x\rho(f) + y\rho(g).$$

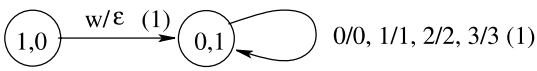
Many interesting and natural linear image transformations can be implemented as a WFT. In the following, we see several examples.

Example 7.2. Let $w \in \Sigma^*$ be a fixed word. The WFT



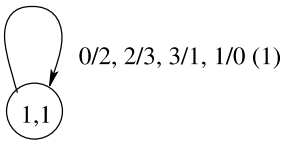
computes the weighted relation $\rho(u, wu) = 1$ for every $u \in \Sigma^*$, and $\rho(u, v) = 0$ if $v \neq wu$. The effect is to shrink the input image and place it at the sub-square addressed by w. For example, Fig. 10(a) shows the result of our test image for $w = 21$.

Example 7.3. Consider then the WFT



It computes the weighted relation $\rho(wu, u) = 1$ for every $u \in \Sigma^*$, and $\rho(v, u) = 0$ if $v \neq wu$. Now the effect is to zoom the sub-square of the input image whose address is w. For example, with $w = 30$, our test image is mapped into the image shown in Fig. 10(b).

Example 7.4. The WFT



rotates the image 90°, as shown in Fig. 10(c).

Fig. 11. The output of transforming the test image using an 11 state WFT

Also, fractal-like transformations can be defined. For example, 11 states are enough to implement a transformation that produces Fig. 11.

Next, we define the action of WFTs on WFAs.

Definition 7.5. *An application of an ε-free n-state WFT M to an m-state WFA A is the mn state WFA $M(A)$ whose states are the pairs (p, q) of states of A and M, the initial and final distribution values are obtained by multiplying the corresponding distributions of A and M, and the weight of the transition*

$$(p, q) \xrightarrow{a} (s, t)$$

is

$$\sum_{x \in \Sigma} (A_x)_{p,s} (M_{x,a})_{q,t}$$

where A_x and $M_{x,a}$ are the weight matrices of A and M, respectively.

This is a straightforward generalization of the usual applications of a finite letter-to-letter transducer on a finite automaton. It is easy to see that $M(A)$ generates the multi-resolution function $\rho(f)$ where ρ is the weighted relation of M and f is the multi-resolution determined by A.

Applying WFT M directly to a WFA A has the advantages that:

- It is fast if A and M are small. There is no need to decode the image into the usual pixel form before applying the operation.
- The result is correct at every resolution.

A concept of average preservation can also defined for WFT; see [8] for details and for more examples.

8 Parametric Weighted Finite Automata (PWFA)

We will generalize now the way input words can define pixel positions in WFA. Instead of using a fixed binary (or k-ary) representation of addresses, we allow

that the pixel positions are also computed by some WFA [1]. We call these automata *parametric weighted finite automata*, because the input string acts as a parameter binding the functions for different dimensions together. Instead of computing single values for the input strings, in parametric weighted finite automata, we get points of higher dimensional spaces. This requires only one change in the definition for WFA, namely our initial distribution vector I becomes an initial distribution matrix of size $d \times Q$ for some $d > 0$. For these parametric WFA, we will only consider weights and vectors over $S = \mathbb{R}$.

Definition 8.1. *A Parametric Weighted Finite Automaton (PWFA) is thus specified by:*

- *The finite state set* Q
- *The input alphabet* $\Sigma = \{0, 1, \ldots, m-1\}$
- *The weight matrices for transitions* $A = (A_0, A_1, \ldots, A_{m-1}),\ A_i \in \mathbb{R}^{Q \times Q}$
- *The final distribution* $F \in \mathbb{R}^{Q \times 1}$
- *The initial distribution matrix* $I \in \mathbb{R}^{d \times Q}$

In the transition diagrams for PWFAs now inside every node d, initial distribution values and one final distribution value are inserted. And computing (r, w) with an initial distribution matrix I of size $d \times Q$ looks exactly as in the case of WFA: $(r, w) = IA_wF$.

Definition 8.2. *For the given PWFA* P *over the alphabet* Σ*, let* $R_n(P)$ *denote the set of points computed by* P *on inputs of length* n*, and* $R_{\geq n}(P)$ *the set of points computed on inputs of length at least* n*:*

$$R_n(P) = \big\{(r, w) \;\big|\; w \in \Sigma^n\big\}, \qquad R_{\geq n}(P) = \bigcup_{i=n}^{\infty} R_i(P).$$

Now a topologically closed set $R(P)$ *can be associated with a PWFA* P*:*

$$R(P) = \bigcap_{n=0}^{\infty} \overline{R_{\geq n}(P)}$$

where $\overline{R_{\geq n}(P)}$ *is the topological closure of* $R_{\geq n}(P)$*.*

In other words, a point $\overline{x} \in \mathbb{R}^d$ is in $R(P)$ if either (i) there exist infinitely many words w such that $(r, w) = \overline{x}$, or (ii) there exist points $(r, w) \neq \overline{x}$ arbitrarily close to $\overline{x}$.

Multi-dimensional sets $R(P)$ given by a PWFA P can be interpreted as relations or images in many different ways. If $d = 2$, it is natural to interpret pairs (x, y) as points of the Euclidean plane, so $R(P)$ becomes a bi-level image. In case $d = 3$, we might have a set of points (x, y, z) of a 3D object or a description of pixel locations x, y and intensities z of a 2D image, or a description of a moving 2D bi-level object where the third dimension is interpreted as the time coordinate. Case $d = 4$ could be a description of a 3D

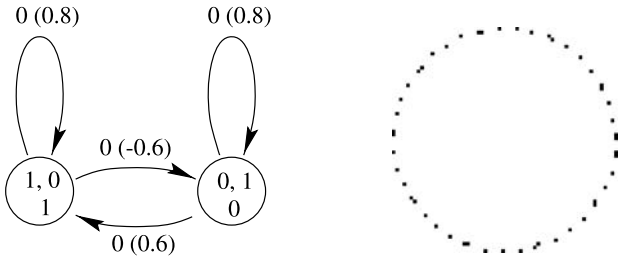


Fig. 12. (**left**) Rotation by $\cos^{-1}(0.8)$. (**right**) First 50 points of the circle

grayscale object, or a 2D grayscale video, etc. In all cases, PWFA-decoding computes d-dimensional points, followed by their interpretation. This new degree of freedom by separating the generation of real values from their interpretation yields a high descriptional power. For example, PWFA are used as representations of multi-dimensional wavelets [27], shapes of figures [30], spline-curves, and textured 3D patches [2].

PWFA over Unary Alphabet

WFA over a unary alphabet do not define useful functions, but from PWFA over a single input symbol 0 one can derive already interesting structures.

Example 8.3. Consider the following one-symbol, two-states PWFA C in Fig. 12.

The corresponding weight matrix A_0 defines a rotation of the plane $\mathbb{R}^2$ by the angle $\alpha = \cos^{-1}(0.8)$. The ratio of α to π is irrational. To see this, we can use the following nice, short proof from [20]. We have $\cos\alpha = \frac{4}{5}$. Using repeatedly the formula $\cos 2\alpha = 2\cos^2\alpha - 1$, we easily obtain

$$\cos 2^k\alpha = \frac{a_k}{5^{2^k}}$$

for all k, where a_k is an integer not divisible by 5. Hence, all $\cos 2^k\alpha$ are distinct, so the set $\{\cos i\alpha \mid i \in \mathbb{Z}\}$ is infinite. This means that α/π is irrational.

The orbit of a point under the iterated rotation by an irrational angle defines a dense subset of a circle, so

$$R(C) = \{(x, y) \mid x^2 + y^2 = 1\} = \{(\cos(t), \sin(t)) \mid t \in \mathbb{R}\}.$$

The unary alphabet PWFAs have been characterized by decidability results and closure properties [28, 19]. Furthermore, it is not hard to prove that two symbols in Σ actually suffice. Please note, that on the other hand the number of states gives rise to an infinite hierarchy, as can be concluded from the facts about polynomials of degree m.

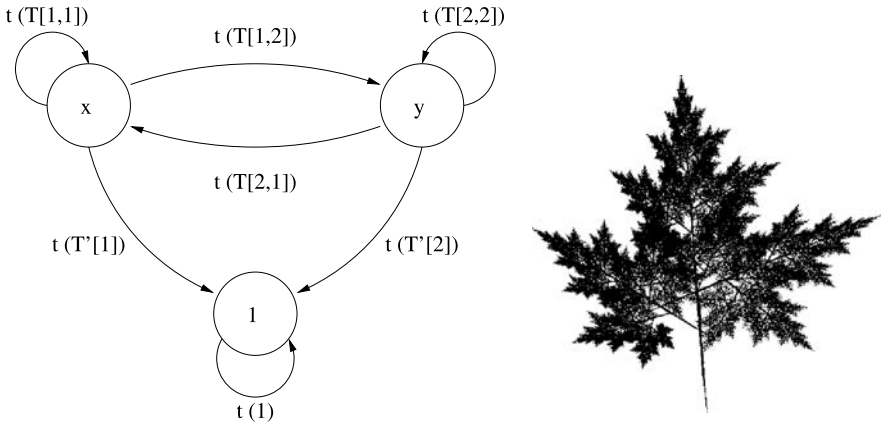


Fig. 13. Simulating the IFS for the maple leaf

PWFA and Iterated Function Systems

Consider any Iterated Function System (IFS) with k contractive affine maps of $\mathbb{R}^2$ [5]. A PWFA simulating the IFS just needs k symbols. Each symbol t corresponds to one affine transform represented by some 2×2 matrix T for scaling and/or rotating the plane and a 2×1 translation vector T'.

Two states are needed to represent the x- and y-coordinates and one state for the constant 1. The weights for transitions between states are assigned for each symbol t in a straightforward manner from the matrix- and vector-entries of T and T'.

Example 8.4. The fractal maple leaf is generated this way by a 3-state, 4-symbols PWFA (see Fig. 13):

PWFA and Polynomial Curves

If each of the d functions computed by a PWFA is a polynomial, we can produce a very compact automaton for the corresponding polynomial curve in $\mathbb{R}^d$.

Example 8.5. Consider the 4-state, 2-symbols, 2-dimensional PWFA P as given in Fig. 14. It has initial distributions $(1, -1, 0, 0)$ and $(0, 1, -1, 0)$, if the states are numbered from left to right.

In the bintree representation of a WFA, the four states as such—from left to right—would compute the functions t^3, t^2, t, and 1, respectively, over the interval $[0, 1)$. Let us interpret now the two dimensions of the PWFA P as the x- and y-coordinates of points. Then the given PWFA computes the points of the curve segment shown in the middle of Fig. 14. The second image is also

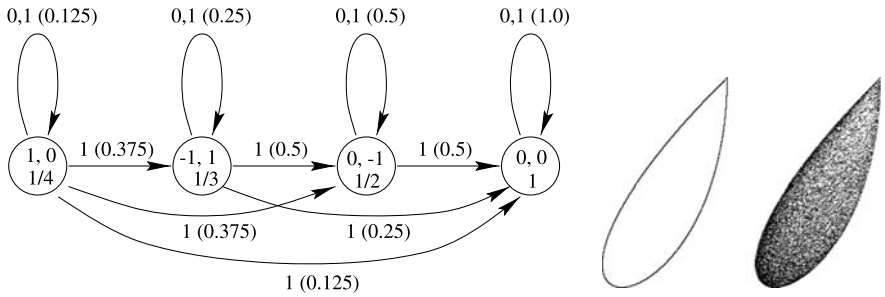


Fig. 14. PWFA for $\{(t^3 - t^2, t^2 - t) \mid 0 \leq t \leq 1\}$, curve and filling the interior

generated by some PWFA, which essentially holds two copies of the PWFA P and three extra symbols plus two helper states. Then for any two points on the curve of P, the line between those points is gradually filled with black pixels in some random fashion. For the displayed picture, the computation was stopped intentionally to leave some pixels of the interior unchanged.

We had mentioned already that any polynomial $p(x)$ of degree m can be computed by a standard one-dimensional WFA with $m + 1$ states, as shown already in [6]. Therefore, any d-dimensional curve

$$\big\{\big(p_1(t), p_2(t), \ldots, p_d(t)\big) \;\big|\; 0 \leq t \leq 1\big\}$$

with parametric representation using the polynomials $p_1(t), p_2(t), \ldots, p_d(t)$ is computable by a PWFA. Furthermore, if the highest degree of the polynomials $p_1(t), p_2(t), \ldots, p_d(t)$ is m, the PWFA will only need $m + 1$ states again. It is worthwhile noting that these polynomial curves include, e.g., the square root (t^2, t), which can only be approximated by WFA, but is not computable with arbitrary precision [14, 13]. For many practical purposes, these polynomial curves are used to approximate figures in the plane like shapes of, e.g., font letters. This has been studied for WFA and chain-code languages in [12] and in [29, 30] for several types of splines and also for textured 3D Bezier-spline surfaces [2].

9 Conclusions and Open Problems

For weighted finite automata and their extensions to, e.g., weighted finite transducers or parametric WFA several attractive possibilities of image generation and image compression have been demonstrated. The WFA inference algorithm works very well, especially for the combination with wavelet-transforms as preprocessing. This seems to indicate that there is still quite a bit of potential to improve the WFA compression rates in hybrid WFA image compression heuristics.

WFT and PWFA have been explored up to now mostly with respect to inclusion properties and decidability questions and a small number of interesting "hand-made" examples have been provided. For practical applications, an important question is whether the WFA inference algorithm can be extended to WFT or PWFA, e.g., to PWFA-representation of 3D spline-patches.

For some of the published examples of PWFA, it seemed essential that irrational weights can be employed. In a strict sense, it is arguable here, whether the attribute "finite" is indeed justified for those PWFA, since we do not generate the irrational number by some kind of finite state device. There are results on language families and decidability questions for integer weighted finite automata by Halava and Harju [17, 18], but PWFA with rational weights are still to be studied in detail.

References

1. J. Albert and J. Kari. Parametric weighted finite automata and iterated function systems. In M. Dekking et al., editors, *Proc. Fractals: Theory and Applications in Engineering, Delft*, pages 248–255, 1999.
2. J. Albert and G. Tischler. On generalizations of weighted finite automata and graphics applications. In S. Bozapalidis and G. Rahonis, editors, *Proc. CAI 2007*, volume 4728 of *Lecture Notes in Computer Science*, pages 1–22. Springer, Berlin, 2007.
3. S. Bader, S. Hölldobler, and A. Scalzitti. Semiring artificial neural networks and weighted automata—And an application to digital image encoding. In *Advances in Artificial Intelligence*, volume 3238 of *Lecture Notes in Computer Science*, pages 281–294. Springer, Berlin, 2004.
4. P. Bao and X.L. Wu. L-infinity-constrained near-lossless image compression using weighted finite automata encoding. *Computers & Graphics*, 22:217–223, 1998.
5. M. Barnsley. *Fractals Everywhere*, 2nd edition. Academic Press, San Diego, 1993.
6. K. Culik II and J. Karhumäki. Finite automata computing real functions. *SIAM Journal on Computing*, 23(4):789–814, 1994.
7. K. Culik II and J. Kari. Digital images and formal languages. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages, volume 3*, pages 599–616. Springer, Berlin, 1997.
8. K. Culik II and J. Kari. Finite state transformations of images. *Computers & Graphics*, 20:125–135, 1996.
9. K. Culik II and J. Kari. Finite state methods for compression and manipulation of images. In J.A. Storer and M. Cohen, editors, *Proceedings of the Data Compression Conference*, pages 142–151. IEEE Computer Society Press, Los Alamitos, 1995.
10. K. Culik II and J. Kari. Image-data compression using edge-optimizing algorithm for WFA inference. *Information Processing & Management*, 30:829–838, 1994.

11. K. Culik II and J. Kari. Image compression using weighted finite automata. *Computers & Graphics*, 17:305–313, 1993.
12. K. Culik II, V. Valenta, and J. Kari. Compression of silhouette-like images based on WFA. *Journal of Universal Computer Science*, 3(10):1100–1113, 1997.
13. D. Derencourt, J. Karhumäki, M. Latteux, and A. Terlutte. On computational power of weighted finite automata. In *Proceedings of MFCS'92*, volume 629 of *Lecture Notes in Computer Science*, pages 236–245. Springer, Berlin, 1992.
14. M. Droste, J. Kari, and P. Steinby. Observations on the smoothness properties of real functions computed by weighted finite automata. *Fundamenta Informaticae*, 73(1,2):99–106, 2006.
15. U. Hafner. Refining image compression with weighted finite automata. In J.A. Storer and M. Cohn, editors, *Proc. Data Compression Conference*, pages 359–368, 1996.
16. U. Hafner, J. Albert, S. Frank, and M. Unger. Weighted finite automata for video compression. *IEEE Journal on Selected Areas in Communication*, 16:108–119, 1998.
17. V. Halava and T. Harju. Undecidability in integer weighted finite automata. *Fundamenta Informaticae*, 38(1,2):189–200, 1999.
18. V. Halava and T. Harju. Languages accepted by integer weighted finite automata. In J. Karhumäki, H. Maurer, G. Paun, and G. Rozenberg, editors, *Jewels Are Forever*, pages 123–134. Springer, Berlin, 1999.
19. V. Halava, T. Harju, M. Hirvensalo, and J. Karhumäki. Skolems problem—On the border between decidability and undecidability. Technical Report 683, Turku Centre for Computer Science, 2005
20. J. Jahnel. When is the (co)sine of a rational angle equal to a rational number? Unpublished, available online at www.uni-math.gwdg.de/jahnel/Preprints/cos.pdf, 2005.
21. Z.H. Jiang, O. de Vel, and B. Litow. Unification and extension of weighted finite automata applicable to image compression. *Theoretical Computer Science*, 302:275–294, 2003.
22. Z.H. Jiang, B. Litow, and O. de Vel. Similarity enrichment in image compression through weighted finite automata. In *Computing and Combinatorics*, volume 1858 of *Lecture Notes in Computer Science*, pages 447–456. Springer, Berlin, 2000.
23. J. Kari and P. Fränti. Arithmetic coding of weighted finite automata. *Informatique Théorique et Applications, RAIRO*, 28:343–360, 1994.
24. J. Kari. Image processing using finite automata. In Z. Esik, C. Martin-Vide, and V. Mitrana, editors, *Recent Advances in Formal Languages and Applications*, volume 25 of *Studies in Computational Intelligence*, pages 171–208. Springer, Berlin, 2006.
25. F. Katritzke, W. Merzenich, and M. Thomas. Enhancements of partitioning techniques for image compression using weighted finite automata. *Theoretical Computer Science*, 313:133–144, 2004.

26. S.V. Ramasubramanian and K. Krithivasan. Finite automata digital images. *International Journal of Pattern Recognition and Artificial Intelligence*, 14:501–524, 2000.
27. G. Tischler, J. Albert, and J. Kari. Parametric weighted finite automata and multidimensional dyadic wavelets. In *Proc. Fractals in Engineering, Tours, France*. Published on CD-ROM, 2005.
28. G. Tischler. Theory and applications of parametric weighted finite automata. Dissertation, Würzburg, 2008.
29. G. Tischler. Properties and applications of parametric weighted finite automata. *Journal of Automata, Languages and Combinatorics*, 10(2/3):347–365, 2005.
30. G. Tischler. Parametric weighted finite automata for figure drawing. In *Implementation and Application of Automata*, volume 3317 of *Lecture Notes in Computer Science*, pages 259–268. Springer, Berlin, 2004.
31. G.K. Wallace. The JPEG still picture compression standard. *Communications of the ACM*, 34(4):30–44, 1991.