Journal of Universal Computer Science, vol. 5, no. 4 (1999), 227-242
submitted: 1/12/98, accepted: 20/4/99, appeared: 28/4/99 © Springer Pub. Co.

Generalized Weighted Finite Automata Based Image Compression

Karel Culik II
Department of Computer Science
University of South Carolina
Columbia, S.C. 29208, U.S.A.

Peter C. von Rosenberg
Department of Computer Science
University of South Carolina
Columbia, S.C. 29208, U.S.A.

Abstract: The Culik-Kari recursive inference algorithm for WFA is based on an efficient way of expressing subsquares of the given image as linear combinations of available states. Here we improve it in two ways. First, we allow the use of rotations, flippings and negations of the states in the linear combination. Second, in order to get the best possible representation of simple fractal images we allow the creation of edges pointing to ancestors of states under construction which, for technical reasons, was not done in the original recursive algorithm.

Key Words: Image-data compression, finite automata, WFA, fractal-image compression.

1 Introduction

Finite automata can be used to represent fractal images in the following way. One state in the automaton is used to describe the original image. The other states describe other images that are used. All images are of size $2^n \times 2^n$ for some integer n. Each state's image is described by dividing the image into four quadrants and then expressing each quadrant as a linear combination of states. For each quadrant, edges are drawn to these states. Each edge is labeled with a quadrant number and a "weight", its coefficient in the linear combination. We refer to such an automaton as a "Weighted Finite Automaton" (WFA).

Mathematical properties of WFA have been studied in [4]. The first inference program to generate a WFA from an image is described in [5]. The recursive inference algorithm designed by first author and J. Kari is described in [6]. This algorithm yields an automaton with near optimal size as the amount of space needed to store is concerned. Because of this algorithm an efficient image-compression software has been implemented. A summary of [5] and [6] can be found in [7]. Another algorithm for inferring an automaton from an image was described in [14].

The recursive inference algorithm compresses an image by constructing a WFA which describes an approximation of the image. We implement an extension of the recursive algorithm that allows transformations (rotations, flippings, and negations) to be used in the linear combinations. That is, each quadrant is now expressed as a linear combination of transformations of states. This requires the edges to be labeled with a transformation as well. We refer to the automaton obtained by this process as a "Generalized Weighted Finite Automaton" (GWFA). This approach creates an automaton with fewer states, the drawback being that more information must be stored for each edge.

If we disregard the negative weights, WFA and GWFA are a special case of PMRFS [3] where only four affine transformations, the mappings of the unit square into the four quadrants, are used for WFA, and the compositions of these four mappings with 90^o rotations, and flippings are used for GWFA. Another special case of MRFS are IFS of Barnsley where arbitrary affine transformations are allowed but only one variable (state) is used. There is no encoding algorithm known for PMRFS, the edge optimizing algorithm for WFA clearly outperforms the IFS based algorithms [2, 10], both in compression ratio and speed. Clearly, the possibility to express a subsquare as a linear combination of other subsquares in WFA is more important than the possibility to use general affine transformation in IFS based systems. In this work we try a small compromise. GWFA unlike PMRFS or IFS use only a special kind of affine transformations but have a somewhat wider selection than WFA.

2 Grayscale images and WFA

A grayscale image of finite resolution $m \times n$ can be expressed by assigning a grayness value to each of the $m \times n$ pixels. Typically the values are digitized using integers between 0 and $2^k - 1$ for some positive integer k. Often $k = 8$ is used, giving values from 0 (black) to 255 (white) which is the method used in our implementation.

This section gives definitions and terminology from [6]. First, we will describe the method they used to represent multiresolution images as functions on words. An image consists of four quadrants labeled 0, 1, 2, and 3 as shown in Fig. 1. These quadrants can also be divided into quadrants. This process can be continued thereby yielding smaller and smaller **subsquares**.

Definition 1. Consider a subsquare of a $2^n \times 2^n$ resolution image.

1. Its **address** is the string of quadrant labels listed in the order in which they were chosen to arrive at the subsquare.
2. Its **depth** is the integer d such that the subsquare is of size $2^d \times 2^d$. The entire image has depth n.

In computer graphics quadtrees whose leaves are pixels of an $2^n \times 2^n$ digitelized image. With the obvious labeling of the edges, the address of a pixel shows the path in the quadtree from the root to the pixel. Some examples of subsquares and their addresses are shown in Fig. 1. Let $\Sigma = \{0, 1, 2, 3\}$ denote the 4-letter alphabet of quadrant labels. Given an image of resolution $2^n \times 2^n$, the pixels are the subsquares with addresses of length n. Letting Σ^n denote the set of strings of length n in the alphabet Σ and using this addressing scheme, we can represent the image as a function $g : \Sigma^n \to \mathbb{R}$. A multiresolution image refers to one in which the image is described at an infinite number of resolutions. By letting Σ^* denote the set of all strings in the alphabet Σ we can represent a multiresolution image as a function $f : \Sigma^* \to \mathbb{R}$. We will let ε denote the empty string. The subsquare with address ε is the entire image. The set of functions $g : \Sigma^* \to \mathbb{R}$

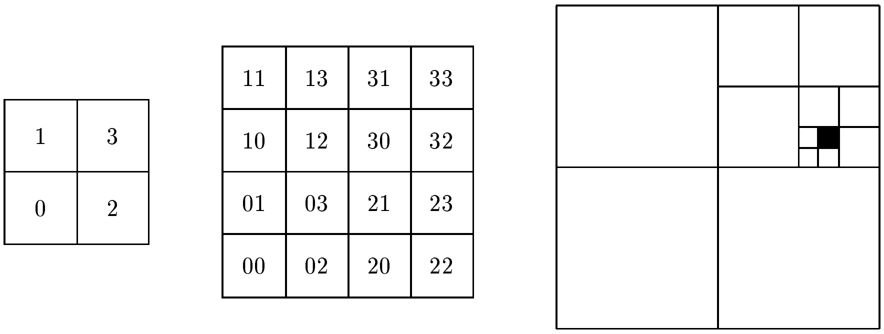
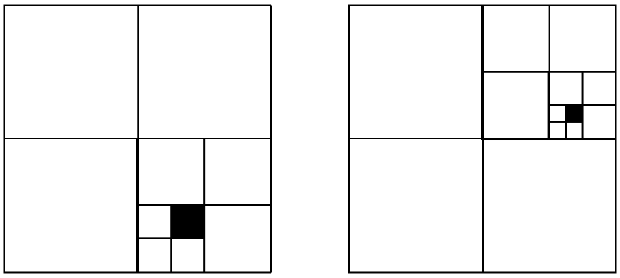


Figure 1: The addresses of the quadrants, of the subsquares of resolution 4×4, and the subsquare with address 3203.



ψ_j subsquare $\quad$ ψ_i subsquare

$w = 203$ $\quad$ $aw = 3203$

Figure 2: Example with $a = 3$ and $w = 203$.

can be added and multiplied by scalars and so forms a vector space. For any functions $f_1, f_2 : \Sigma^* \to \mathbb{R}$ and any $c \in \mathbb{R}$ we define

$$\begin{aligned} (f_1 + f_2)(w) &= f_1(w) + f_2(w) \qquad && \text{for any } w \in \Sigma^* \\ (cf_1)(w) &= cf_1(w) && \text{for any } w \in \Sigma^*. \end{aligned}$$

Weighted Finite Automata (WFA) introduced in [4, 5] give an efficient tool to describe these functions on Σ^*

Definition 2. A **weighted finite automaton** or **WFA** $A = (Q, \Sigma, \mathcal{W}, I, F)$ is specified by

1. $Q = \{q_1, q_2, \ldots, q_n\}$ is a finite set of states,
2. Σ is a finite alphabet (here we use the alphabet $\Sigma = \{0, 1, 2, 3\}$),
3. $\mathcal{W} = \{W_a : a \in \Sigma\}$, where $W_a : Q \times Q \to \mathbb{R}$ are the weights at edges labeled by a,
4. $I : Q \to \mathbb{R}$ is the initial distribution,
5. $F : Q \to \mathbb{R}$ is the final distribution.

In the following I will be used to denote the row vector $(I(q_1), \dots, I(q_n))$ and F the column vector $(F(q_1), \dots, F(q_n))$. We say that $(p, a, q) \in Q \times \Sigma \times Q$ is an edge (transition) of A if $W_a(p, q) \neq 0$. This edge has label a and weight $W_a(p, q)$. For $|Q| = n$ we will usually view W_a as an $n \times n$ matrix of real numbers and I and F as real vectors of size n. The WFA A defines a final multiresolution image $f_A : \Sigma^* \to \mathbb{R}$ by

$$f_A(a_1 a_2 \dots a_k) = I \cdot W_{a_1} \cdot W_{a_2} \cdot \ \dots \ \cdot W_{a_k} \cdot F.$$

Each state q_i of the WFA defines a multiresolution image $\psi_i : \Sigma^* \to \mathbb{R}$, the **state image**. Another way to define f_A is to use these images. We will define the ψ_i's by defining a function $\psi : \Sigma^* \to \mathbb{R}^n$ such that $\psi_i(w)$ is the i^{th} entry of the vector $\psi(w)$. Let $\psi(\varepsilon) = F$ and

$$\psi(aw) = W_a \cdot \psi(w)$$

where $a \in \Sigma$ and $w \in \Sigma^*$. This says that each quadrant of each state image is a linear combination of state images since

$$\psi_i(aw) = \sum_{j=1}^{n} (W_a)_{ij} \cdot \psi_j(w)$$

where $(W_a)_{ij}$ is the $(i, j)^{\text{th}}$ entry of the matrix W_a. Note that aw is the address of a subsquare in quadrant a of of image ψ_i and w is the address of the corresponding subsquare in the images ψ_j. See Fig. 2 for an example. Note that we can also define ψ by

$$\psi(a_1 a_2 \dots a_k) = W_{a_1} \cdot W_{a_2} \cdot \ \dots \ \cdot W_{a_k} \cdot F.$$

Finally f_A is defined by

$$f_A(w) = I \cdot \psi(w) = \sum_{i=1}^{n} I_i \cdot \psi_i(w)$$

whereI_i is the i^{th} entry of I. And so the final image is a linear combination of the state images as specified by I. Note that since $\psi(\varepsilon) = F$, F specifies the average grayness for each state.

We display WFA using diagrams that are similar to those used for finite automata. States are represented by circles containing their initial and final distributions. If the $(i, j)^{\text{th}}$ entry of W_a is $(W_a)_{ij} \neq 0$, then there is an edge from state i to state j labeled with "$a : (W_a)_{i,j}$". (Multiple edges with the same weights are often combined in the diagram.) The following example is from [6].

Example 1. A WFA can be specified as a diagram with states $\{1, \dots, n\}$. There is an edge from state i to state j with label $a \in \Sigma$ and weight $r \neq 0$ if and only if $(W_a)_{ij} = r$. The initial and final distribution values are shown inside the nodes, as illustrated in Fig. 3, where the left circle is state 1, the right circle is state 2, $I = (1, 0)$, $F = (\frac{1}{2}, 1)$,

$$W_0 = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 1 \end{pmatrix}, \quad W_1 = \begin{pmatrix} \frac{1}{2} & \frac{1}{4} \\ 0 & 1 \end{pmatrix}, \quad W_2 = \begin{pmatrix} \frac{1}{2} & \frac{1}{4} \\ 0 & 1 \end{pmatrix}, \text{ and } \quad W_3 = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{pmatrix}.$$

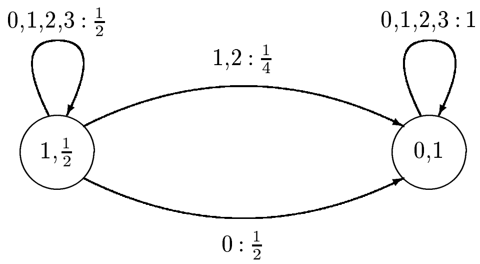


Figure 3: A diagram for WFA A defining the linear grayness function f_A.

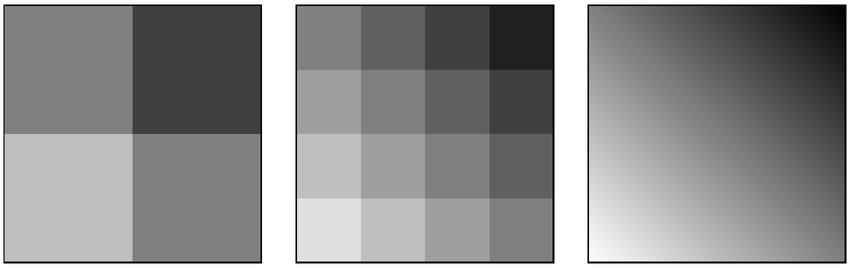


Figure 4: The image f_A in resolutions 2×2, 4×4, and 128×128.

The multiresolution image f_A can be read as follows. Considering subsquare 03 and using the above formula we get $f_A(03) = IW_0W_3F = \frac{5}{8}$. To read this off of the WFA diagram we consider all two edge paths that exist with the first edge labeled 0 and the second edge labeled 3. Each of these contributes a weight w obtained by multiplying the initial distribution of the starting state, the edge weights, and the final distribution of the final state. In this case there are three such paths: take the state 1 loop twice ($w = 1 \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{8}$), take the state 1 loop and then take the bottom edge to state 2 ($w = 1 \cdot \frac{1}{2} \cdot 1 \cdot 1 = \frac{1}{2}$), or take the state 2 loop twice ($w = 0 \cdot 1 \cdot 1 \cdot 1 = 0$). Adding these weights gives $f_A = \frac{1}{8} + \frac{1}{2} + 0 = \frac{5}{8}$. The image f_A for resolutions 2×2, 4×4, and 128×128 is shown in Fig. 4.

Definition 3. A function $f : \Sigma^* \to \mathbb{R}$ is an **average preserving function** or **ap-function** if

$$f(w) = \frac{1}{|\Sigma|} \sum_{a \in \Sigma} f(wa)$$

for each $w \in \Sigma^*$.

For $\Sigma = \{0, 1, 2, 3\}$ this says that

$$f(w) = \frac{1}{4}(f(w0) + f(w1) + f(w2) + f(w3))$$

or that f evaluated on a subsquare equals the average of f evaluated on its 4 quadrants. This is usually required of multiresolution images since it makes the resolution levels compatible. Note that the set of ap-functions forms a linear sub-space of the vector space mentioned above since any linear combination of ap-functions is an ap-function.

Definition 4. A WFA A is **average preserving** if

$$\sum_{a\in\Sigma} W_a \cdot F = p \cdot F,$$

or equivalently

$$\sum_{a\in\Sigma} \psi(a) = p \cdot F$$

where $p = |\Sigma|$ is the cardinality of the alphabet Σ. In other words, a WFA A is average preserving if its final distribution is an eigenvector of $\sum_{a\in\Sigma} W_a$ corresponding to its eigenvalue $|\Sigma|$.

The below result comes from [5] by combining Theorems 1 and 3 from that paper.

Theorem 5. *Let $f : \Sigma^* \to \mathbb{R}$ be a multiresolution image computable by a WFA. Then the following hold:*

1. *if f is computed by an average preserving WFA then f is average preserving,*
2. *if f is average preserving then f can be computed by an average preserving WFA.*

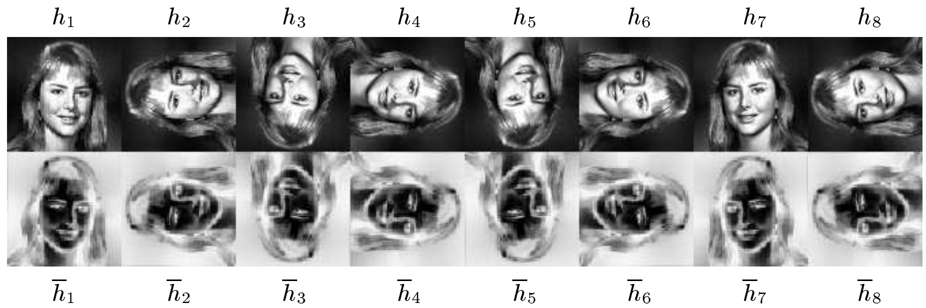
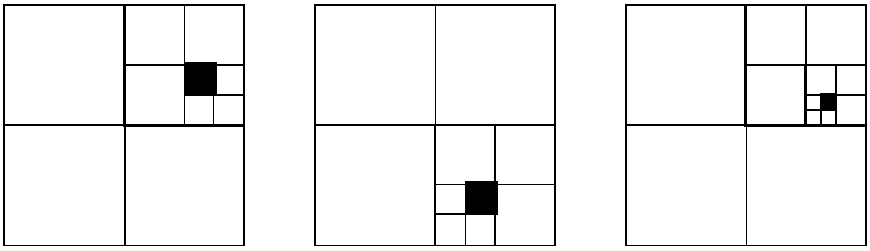


Figure 5: The image transformations

3 Grayscale images and GWFA

In [8] and [9] the first author and Valenta defined generalized finite automata (GFA) by adding transformations to finite automata. GWFA were used to represent bi-level images. By combining the definitions for WFA and GFA we obtain the following.



ψ_j subsquare $h_k^{-1}(w) = 321$ | h_k maps it to $w = 203$ | ψ_i subsquare $aw = 3203$

Figure 6: Example with $a = 3$, $w = 203$, and $k = 4$.

Definition 6. A **generalized weighted finite automaton** (**GWFA**) is a five-tuple $A = (Q, \Sigma, \mathcal{W}, I, F)$ where

1. $Q = \{q_1, q_2, \ldots q_n\}$ is a finite set of states,
2. Σ is a finite alphabet (here we use the alphabet $\Sigma = \{0, 1, 2, 3\}$),
3. $h_1, \ldots, h_r : \Sigma^* \to \Sigma^*$ are letter-to-letter permutation morphisms (transformations) where $r \geq 1$,
4. $\mathcal{W} = \{W_{ak}, \overline{W}_{ak} : a \in \Sigma, i \in \{1, 2, \ldots, r\}\}$ where
   $W_{ak} : Q \times Q \to \mathbb{R}$, the weights at edges labeled by a and h_i, and
   $\overline{W}_{ak} : Q \times Q \to \mathbb{R}$, the weights at edges labeled by a and $\overline{h}_i$,
5. $I : Q \to \mathbb{R}$ is the initial distribution,
6. $F : Q \to \mathbb{R}$ is the final distribution.

Letting $H = \{h_1, \overline{h}_1, \ldots, h_r, \overline{h}_r\}$, we say that $(p, a, h, q) \in Q \times \Sigma \times H \times Q$ is an edge (transition) of A if $W_{ak}(p, q) \neq 0$ when $h = h_k$ or $\overline{W}_{ak}(p, q) \neq 0$ when $h = \overline{h}_k$. This edge has label a, transformation h, and weight $W_{ak}(p, q)$ or $\overline{W}_{ak}(p, q)$. For $|Q| = n$ we will usually view W_{ak} and $\overline{W}_{ak}$ as $n \times n$ matrices of real numbers and I and F as real vectors of size n. The permutations used are as follows:

h_1 is the identity morphism,
$h_2(0) = 2$, $h_2(1) = 0$, $h_2(2) = 3$, $h_2(3) = 1$,
$h_3(0) = 3$, $h_3(1) = 2$, $h_3(2) = 1$, $h_3(3) = 0$,
$h_4(0) = 1$, $h_4(1) = 3$, $h_4(2) = 0$, $h_4(3) = 2$,
$h_5(0) = 1$, $h_5(1) = 0$, $h_5(2) = 3$, $h_5(3) = 2$,
$h_6(0) = 0$, $h_6(1) = 2$, $h_6(2) = 1$, $h_6(3) = 3$,
$h_7(0) = 2$, $h_7(1) = 3$, $h_7(2) = 0$, $h_7(3) = 1$,
$h_8(0) = 3$, $h_8(1) = 1$, $h_8(2) = 2$, $h_8(3) = 0$.

The interpretation of these morphisms as transformations on images is shown in Fig. 5 with their negations in the second row.

Each state q_i of the GWFA has a corresponding multiresolution image $\psi_i : \Sigma^* \to \mathbb{R}$, the **state image**. The GWFA A determines a final multiresolution image $f_A : \Sigma^* \to \mathbb{R}$ which can be defined in terms of these state images. We will define the ψ_i's by defining a function $\psi : \Sigma^* \to \mathbb{R}^n$ such that $\psi_i(w)$ is the i^{th} entry of the vector $\psi(w)$. Let $\psi(\varepsilon) = F$ and

$$\psi(aw) = \sum_{k=1}^{r} W_{ak} \cdot \psi(h_k^{-1}(w)) + \sum_{k=1}^{r} \overline{W}_{ak} \cdot [X - \psi(h_k^{-1}(w))]$$

where $a \in \Sigma$, $w \in \Sigma^*$ and X is the vector containing all 1's. This says that each quadrant of each state image is a linear combination of transformations of state images. Consider the i^{th} entry $\psi_i(aw)$ of $\psi(aw)$. Note that aw is the address of a subsquare in quadrant a of of image ψ_i. Given some h_k and some ψ_j, $h_k^{-1}(w)$ is the address of the subsquare in the image ψ_j which h_k maps onto address w (which corresponds to aw). See Fig. 6 for an example. If a negated transformation $\overline{h}_k$ is involved, then the second summation is used. The result is negated by subtracting from X and $\overline{W}_{ak}$ is used. Finally f_A is defined by

$$f_A(w) = I \cdot \psi(w) = \sum_{i=1}^{n} I_i \cdot \psi_i(w)$$

where I_i is the i^{th} entry of I. And so the final image is a linear combination of the state images as specified by I. Again, since $\psi(\varepsilon) = F$, F specifies the average grayness for each state.

An example of a GWFA is shown in Fig. 7. The image of each state is shown in the diagram. The corresponding WFA is shown in Fig. 8. In this example the GWFA requires only half as many states as the WFA and also uses fewer edges. In general the GWFA will use fewer states than the WFA but will also require a transformation to be stored for each edge.

The concept of average preserving can be applied to a GWFA as follows.

Definition 7. A GWFA A is **average preserving** if

$$\sum_{a\in\Sigma} \left[\sum_{k=1}^{r} W_{ak} \cdot F + \sum_{k=1}^{r} \overline{W}_{ak}[X - F] \right] = p \cdot F,$$

or equivalently

$$\sum_{a\in\Sigma} \psi(a) = p \cdot F$$

where $p = |\Sigma|$ is the cardinality of the alphabet Σ.

Clearly every WFA is also a GWFA. For all $a \in \Sigma$ let $W_{a1} = W_a$, $W_{ak} = 0$ for all $k \neq 1$ and $\overline{W}_{ak} = 0$ for all k. Also, if the WFA is average preserving then this GWFA is also average preserving since

$$\sum_{a\in\Sigma} \left[\sum_{k=1}^{r} W_{ak} \cdot F + \sum_{k=1}^{r} \overline{W}_{ak}[X - F] \right] = \sum_{a\in\Sigma} W_a \cdot F = p \cdot F.$$

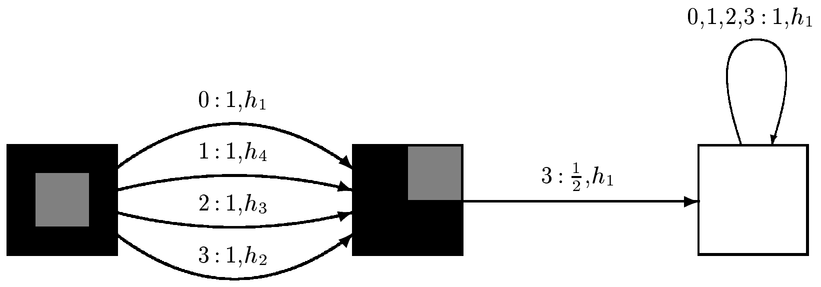


Figure 7: GWFA for a framed square.

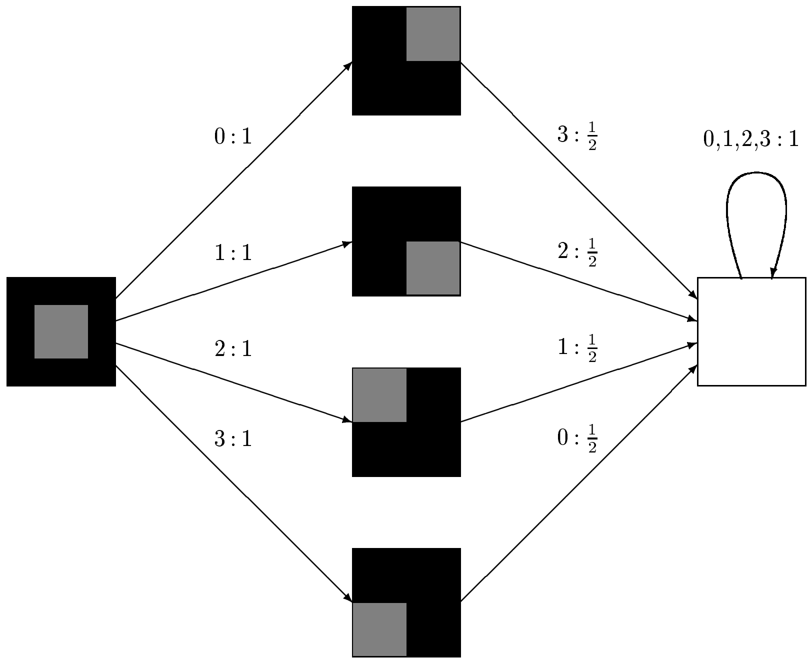


Figure 8: WFA for a framed square.

4 WFA encoding algorithm

In this section we reviw the WFA encoding algorithm from [6], in the next section we modify it for GWFA. We consider the set of depth d images to be an ℓ^2 space. Given depth d images $\phi, \psi : \Sigma^d \to \mathbb{R}$ we define the inner product by

$$\langle \phi, \psi \rangle_d = \sum_{w \in \Sigma^d} \phi(w) \cdot \psi(w)$$

and the norm by

$$\|\phi\|_d^2 = \langle \phi, \phi \rangle_d.$$

Sometimes in the program recursion is used to compute these values, using the fact that

$$\langle \phi, \psi \rangle_d = \sum_{i=0}^{3} \langle \phi_i, \psi_i \rangle_{d-1}$$

and

$$\|\phi\|_d^2 = \sum_{i=0}^{3} \|\phi_i\|_{d-1}^2$$

where ϕ_0, ϕ_1, ϕ_2, and ϕ_3 are the four quadrants of ϕ.

Algorithm 4.1 is the recursive algorithm we used to construct a WFA which approximates a given image.

Algorithm 4.1
build(ϕ, d, c)
1. *Find the edges* $c_0, c_1, \ldots, c_{n-1}$ *such that cost1 is small*
   $\phi' = c_0\psi_0 + \cdots + c_{n-1}\psi_{n-1}$
   cost1 $= \|\phi - \phi'\|_d^2 + G \cdot$ *mem*(*edges for* $c_i \neq 0$)
2. *Execute the following:* (ϕ_i *is quadrant* i *of* ϕ)
   $n_0 = n$
   cost2 $= G \cdot$ *mem*(*new state and one edge*)
   for $i = 0$ *to* 3
      cost2 $=$ *cost2* $+$ *build*$(\phi_i, d-1, cq[i])$
3. *If cost2* $<$ *cost1 then*
   $c_i = 0$ *for* $i = 0, 1, \ldots, n_0 - 1$
   $c_n = 1$
   add_state(n, cq) [*generate* ψ_n]
   $n = n + 1$
   return(*cost2*)
4. *If cost1* $\leq$ *cost2 then*
   remove states $n_0, \ldots, n-1$ (*added in step 2*)
   set $n = n_0$
   return(*cost1*).

Initially the WFA contains only the basis states, the black (or white) square and perhaps some other simple images. At any given point the state images are denoted by $\psi_0, \psi_1, \ldots, \psi_{n-1}$ where n is the number of states. The parameters passed to *build* are the image ϕ and its depth d. The final parameter c is an array which is used to return the coefficients $c_0, c_1, \ldots, c_{n-1}$ of the edges used to describe the image. (If $c_i \neq 0$ then there is an edge to state ψ_i with weight c_i.) Also the *cost* is returned. In general,

$$\textit{cost} = \textit{error} + G \cdot \textit{memory}.$$

Here G is a constant, *memory* is the number of bits needed to store the WFA additions, and

$$error = \|\phi - \phi'\|_d^2$$

where ϕ' is the WFA approximation of the image ϕ. The value of G is an input to the program. The smaller the value of G is, the smaller the final error (regenerated image of higher quality) and the larger the resulting WFA. (A small value of G implies a small penalty for memory used.)

When an image ϕ is sent to *build* two costs are calculated. The value of *cost1* is the cost if the image is expressed in terms of existing states. The value of *cost2* comes from creating a new state for the image and processing each of its four quadrants with calls to *build*. If *cost2* is less than *cost1* then a new state is added (using information from the four calls to *build*), the coefficients are set to indicate a single edge to the new state, and *cost2* is returned. Otherwise, the states added by the four calls to *build* are removed from the WFA and *cost1* is returned.

The first step is the difficult one: expressing the image in terms of existing states. Algorithm 4.2 shows how this is done. For more details see [6].

Algorithm 4.2

find_edges(ϕ, d, c)

1. *Initialization: price* $= 0$, *error* $= 0$, *and* $\eta_i = \psi_i$ *for all* i *with* $0 \leq i < n$.
2. *Loop until there is a break at step b.*
   a. *Find next edge – the one with the largest* $\Delta cost$ *where*
   $\Delta cost = \left\langle \phi, \frac{\eta_i}{\|\eta_i\|} \right\rangle_d^2 - G \cdot ($ *memory for edge* $)$.
   b. *If this* $\Delta cost$ *is negative or very small then break out of the loop*
   c. *Update error* $=$ *error* $- \left\langle \phi, \frac{\eta_i}{\|\eta_i\|} \right\rangle_d$
   and price $=$ *price* $+ ($ *memory for edge* $)$.
   d. *Store edge information: into*$[n] = i$ *and weight*$[n] = \frac{\langle \phi, \eta_i \rangle}{\|\eta_i\|_d^2}$
   e. *Make unused* η_j *orthogonal to* η_i, *that is, for all unused* η_j
   $\eta_j = \eta_j - \left\langle \eta_j, \frac{\eta_i}{\|\eta_i\|} \right\rangle_d \cdot \frac{\eta_i}{\|\eta_i\|_d}$.
3. *Compute the actual edge weights and store in array c.* (*The array weight contains weights for "orthogonalized states" rather than actual states.*)
4. *Return(error* $+ G \cdot$ *price).*

Initially the η_i values are set equal to the states ψ_i. The loop finds the edges. At the end of each loop the unused η_j values are adjusted so that they are orthogonal to the chosen state η_i. The weights from the loop correspond to the "orthogonalized states". These are used to compute the actual edge weights.

5 New encoding algorithm for GWFA

Algorithm 5.1 is the recursive algorithm we used to construct a GWFA which approximates a given image.

Algorithm 5.1
build(ϕ, d, c, t)
1. *Find the edges:* $c_0, c_1, \ldots, c_{n-1}$ *and* $t_0, t_1, \ldots, t_{n-1}$ *such that cost1 is small*
   $\phi' = c_0 t_0(\psi_0) + \cdots + c_{n-1} t_{n-1}(\psi_{n-1})$
   $cost1 = \|\phi - \phi'\|_d^2 + G \cdot mem(edges\ for\ c_i \neq 0)$
2. *Execute the following:* (ϕ_i *is quadrant* i *of* ϕ)
   $n_0 = n$
   $cost2 = G \cdot mem(new\ state\ and\ one\ edge)$
   $for\ i = 0\ to\ 3$
   $cost2 = cost2 + build(\phi_i, d-1, cq[i], s[i])$
3. *If* $cost2 < cost1$ *then*
   $c_i = 0$ *and* $t_i = id$ *for* $i = 0, 1, \ldots, n_0 - 1$
   $c_n = 1$ *and* $t_n = id$
   $add\_state(n, cq, s)$ [*generate* ψ_n]
   $n = n + 1$
   $return(cost2)$
4. *If* $cost1 \leq cost2$ *then*
   remove states $n_0, \ldots, n-1$ (*added during the recursive calls*)
   set $n = n_0$
   $return(cost1)$.

The difference between this algorithm and algorithm 4.1 is that, for each edge, a transformation must be returned (in array t) as well as a coefficient. We only allowed one transformation to be used for each state. Algorithm 4.2 to find the edges is the same except that instead of just searching through each state in step 2a we search through all transformations of each state.

6 Encoding the Automaton

The GWFA is encoded in four parts. For a WFA the method is the same except that the final transformation part is omitted. Each part is expressed as a list of symbols which is then encoded using adaptive arithmetic encoding. The four parts are as follows.

1. The **tree** of edges which point to newly created states. These edges all have weight 1 and transformation h_1 (the identity transformation). The tree T for a given state is listed as $T = X_0 X_1 X_2 X_3$ where

$$X_i = \begin{cases} 1\,T_i & \text{if quadrant i is given by a new state with tree } T_i \\ 0 & \text{otherwise.} \end{cases}$$

   The tree stored here is of course the tree for the root state (the final image).
2. The **matrix** which lists all edges not indicated in the tree. The rows of the matrix are all of the (state,label) pairs which are marked with a 0 in the tree, listed in order. The columns are all of the states listed in order. A 1 in the matrix indicates that there exists an edge from the row state to the column state labeled with the row quadrant number. Each row must contain zeros for all states with numbers greater than or equal to the row state (since

Original

bpp=8.00

WFA

bpp=0.53 psnr=32.57 dB

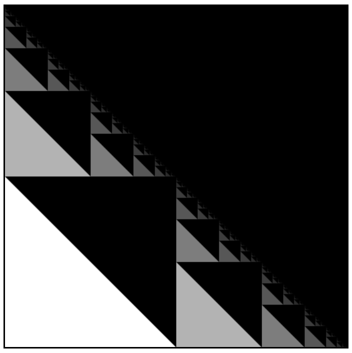
GWFA (4 trans)

bpp=0.46 psnr=32.57 dB

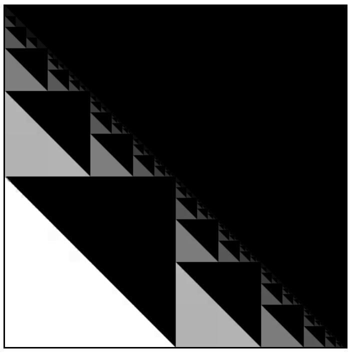
Figure 9: Framed image of resolution 256×256 using 6-state basis.

these states were not created until after the row state was created). These zeros are removed from the matrix and the remaining numbers are encoded column by column.

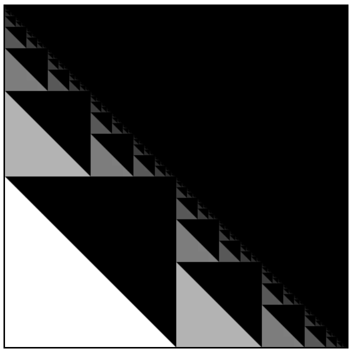
3. The **transformations** of the edges are listed in the order in which they appear in the matrix.
4. The **weights** of the edges are listed in the order in which they appear in the matrix. The number line is divided into intervals and the number of the interval containing the weight is encoded. The intervals $(-\infty, -0.5)$ and $[1, \infty)$ are always used as the first and last intervals. The portion of the number line from -0.5 to 1 is divided into intervals whose length depends upon the desired precision. For precision 0 these intervals are $[-0.5, 0)$ and $[0, 1)$. For a precision of k bits with $1 \leq k \leq 3$ these intervals have length 2^{-k}. For precisions higher than 3 these intervals are of length $2^{-3} = \frac{1}{8}$. In this last case, and also when $(-\infty, -0.5)$ or $[1, \infty)$ is the interval chosen, extra bits are appended to the end of the file to provide the desired accuracy.



Original
bpp=8.00



WFA
bpp=0.0022 psnr=45.01 dB



WFA – fractal mode
bpp=0.0014 psnr=57.43 dB

Figure 10: Fractal image of resolution 512×512 using 1-state basis.

For more details see [12].

The WFA algorithm in [6] does not create loops or edges to ancestor states since such edges would complicate the error computation. These edges are not important for real life images but are needed to obtain small WFA's or GWFA's for simple fractal images. Therefore we added a new feature to the implementation of the GFWA algorithm to allow it to produce edges from states corresponding to subsquares of size $2^{n-1} \times 2^{n-1}$ and $2^{n-2} \times 2^{n-2}$ to the state corresponding to the original image of size $2^n \times 2^n$. The advantages resulting from these edges outweigh any error considerations. This is accomplished by temporarily inserting the original image as a state in *find_edges* (Algorithm 4.2) when these subsquares are being processed. Normally, edges are only allowed to point to states which have already been generated by the recursive algorithm. Another problem arising from this fractal approach occurs during decompression. The final distribution of the original image is needed during the computation. The final distribution could have been stored during the encoding but this information would have been redundant. Instead, this problem was resolved by using a limiting process as shown in Algorithm 6.1 where ε is a small positive value. An example using a fractal image is shown in Fig. 10.

Algorithm 6.1

1. Set fd = maximum possible final distribution.
2. Compute the image using fd in place of its final distribution.
3. Set fdnew = final distribution of the resulting image.
4. If $|\, fdnew - fd \,| > \varepsilon$ *then*
  a. Set fd = fdnew,
  b. Go to step 2.

7 Results

In testing the program we used only the first 4 or the first 8 transformations. The negations are not as beneficial since they can easily be obtained by subtracting the image to be negated from the white square. The improvement in compression resulting from the transformations is best demonstrated when the image clearly contains transformed portions. Such an example, a picture with a gray frame, is shown in Fig. 9 where 4 transformations are used with the GWFA. For most real life images the GFWA based compression resulted only in marginal or no improvement in compression (for the same quality) compared to WFA. For comparison of WFA performance to other methods, e.g. wavelets, see [6].

The GWFA image compression retains the advantages of the WFA approach. It works well for all types of images, including e.g. cartoon-like images with many edges, and it works especially well for color images. For a color image the GWFA compression algorithm builds just one GWFA for all three color components taking advantage of their similarities. All three color components share the states and each has just its own initial distribution.

Acknowledgement. Research was supported by the National Science Foundation under Grant No. CCR-9417384.

References

[1] Barnsley, M.: "Fractals Everywhere"; Academic Press, San Diego, CA (1988).
[2] Barnsley, M., Hurd, L.: "Fractal Image Compression"; AK Peters, Ltd., Wellesley, Massachusetts (1993).
[3] Culik II, K., Dube, S.: "Affine Automata and Related Techniques for Generation of Complex Images"; Theoretical Computer Science 116, 373-398 (1993).
[4] Culik II, K., Karhumäki, J.: "Automata computing real function"; SIAM J. on Computing 23, 4, 789-814 (1994).
[5] Culik II, K., Kari, J.: "Image compression using weighted finite automata"; Comput. Graphics 17, 305-313 (1993).
[6] Culik II, K., Kari, J.: "Image-data compression using edge-optimizing algorithm for WFA inference"; Journal of Information Processing and Management 30, 829-838 (1994).
[7] Culik II, K., Kari, J.: "Efficient inference algorithm for weighted finite automata"; in: Fractal Image Encoding and Compression (Ed. Y. Fisher), Springer-Verlag (1995).

[8] Culik II, K., Valenta, V.: "Finite Automata Based Compression of Bi-level and Simple Color Images"; Computers & Graphics 21, 61-68 (1997).

[9] Culik II, K., Valenta, V.: "Generalized Finite Automata and Transducers"; Journal on Automata, Languages, and Combinatorics 2, 3-17 (1997).

[10] Jacquin, J.: "Image coding based on a fractal theory of iterated contractive image transformations"; IEEE Transactions on Image Processing 1(1), 19-30 (1992).

[11] Mauldin, R.D., Williams, S.C.: "Hausdorff Dimension in graph directed constructions"; Trans. Am. Math. Soc. 309, 811-829 (1988).

[12] Rosenberg, von P.C.: "Generalized Weighted Finite Automata Based Image Compression"; Master's Thesis, Dept. of Computer Science, University of South Carolina (1998).

[13] Royden, H.L.: "Real Analysis"; Macmillan Publishing Company, New York, NY (1988).

[14] Hafner, U.: "Refining image compression with weighted finite automata"; Proceedings Data Compression Conference, (Ed. James A. Storer and Martin Cohn), IEEE Computer Society Press, 359-368 (1996).