Journal of Universal Computer Science, vol. 3, no. 10 (1997), 1100-1113
submitted: 11/3/97, accepted: 2/7/97, appeared: 28/10/97 © Springer Pub. Co.

Compression of Silhouette-like Images based on WFA *

Karel Culik II, Vladimir Valenta

Department of Computer Science

University of South Carolina

Columbia, S.C. 29208, U.S.A.

Jarkko Kari

Department of Computer Science

University of Iowa

Iowa City, IA 52242-1419

Abstract

We describe a new approach to lossy compression of silhouette-like images. By a silhouette-like image we mean a bi-level image consisting of black and white regions divided by a small number of closed curves. We use a boundary detection algorithm to express the closed curves by chain codes, and we express the chains as one function of one variable. We compress this function using WFA over two letter alphabet. Finally, we use arithmetic coding to store the automaton.

*This work was supported by the National Science Foundation under Grant No. CCR-9417384. Preliminary version was presented in DCC 1997.

1 Introduction

The objective of this paper is to design a lossy fractal compression method for silhouette-like bi-level images that would have an excellent quality to compression rate ratio. This is our second and better performing approach to this problem. First we discuss the background.

Weighted Finite Automata (WFA) have been introduced in [1, 2] as a way to specify real functions on $[0,1]^n$, in particular, for $n = 2$ grayscale functions (grayscale images). In [2] a theoretical inference algorithm for WFA was given and finally in [3] a recursive inference algorithm for WFA which led to well performing data-image compression software, see also [12]. In [8, 9] we have described the first approach to compression of bi-level images based on finite automata. Since in the case of a bi-level image we need to define the set of black pixels instead of the grayness function, we have not used WFA but rather Generalized Finite Automata (GFA) where the generalization means that we used rotations, flippings and negations when expressing subsquares of each state in the terms of the other states. However, no weights were used. It turned out that for GFA it was not advantageous to allow nondeterminism, i.e. to express some subsquares as unions of the other states. That led to a nonrecursive, fast encoding algorithm which performed well, however, comparatively not as impressively as the recursive WFA algorithm for grayscale and color images. Further considerable improvement was obtained by using vector quantization for the small (8×8) subsquares.

Here, we are returning to WFA. We will reduce the problem of the encoding of a silhouette-like bi-level image to the encoding of two one-variable functions describing the boundary (-ies) of the black and white regions of the given image. Our method assumes that the given bi-level image consists of black and white regions separated by k closed curves $c_1, \ldots, c_k$, for relatively small k. Using the algorithm from [11] we find k *NESW* (north-east-south-west) chain codes for the regions. We convert each chain code into two functions x_i, y_i expressing c_i in parameterized form for $i = 1, \ldots, k$. Now we combine the $2k$ functions (tables) $x_1, \ldots, x_k$, $y_1, \ldots, y_k$ into one function (longer table) z by concatenating the domains. Finally, we compress the function (table) z by the one-dimensional version of the recursive WFA algorithm.

We measure the quality of the regenerated images by the percentage of wrong pixels. Subjectively, they look even better, in particular, the smooth features, notice that the regenerated images have some features smoother than the originals. We show the performance of our method on some examples.

Our new method extends to cartoon-like color images in the same way as it is described in [9] for our GFA method. An image with less than 2^m color values is expressed by m bitplanes and then each bitplane is encoded as a bi-level image. A further advantage is that the automata encoding different bitplanes can share states.

2 Weighted finite automata and their inference

In [1] the weighted finite automata (WFA) were introduced as devices computing real functions on $[0,1]^n$ for any n. In [2, 3, 4] we used the case $n = 2$ interpreted as the grayscale function specifying a grayscale image. Here we will use two functions of one variable to specify the color boundary. In order to compress tables for these functions we will need to infer one-dimensional ($n = 1$) WFA. The inference algorithms from [2, 3, 4] work for any $n \geq 1$, we will use the recursive algorithm from [3]. We remind the definition of WFA for the specific case $n = 1$, in this case we use alphabet $\Sigma = \{0, 1\}$.

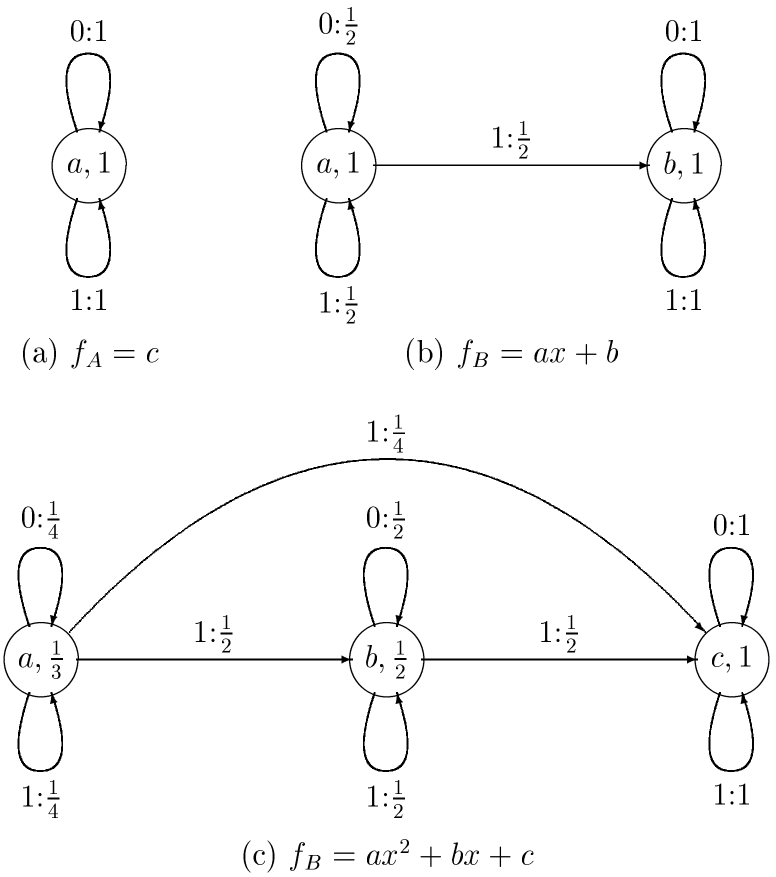

Figure 1: WFA defining constant, linear or quadratic function.

Binary words in $\Sigma^\star$ are interpreted as addresses of subintervals of the interval $[0,1]$. The whole interval is addressed by ε, then recursively the left half of the interval with address w is addressed by $w0$, the right half by $w1$. For example, the address of the interval $[\frac{1}{8}, \frac{3}{16}]$ is 0010. The words over Σ of length n are denoted by Σ^n, all the words over Σ by $\Sigma^\star$. Hence, a function $g : \Sigma^n \rightarrow I\!R$ can be interpreted as a table of length 2^n of a real function (R is the set of reals). A function $f : \Sigma^\star \to R$ is a *multiresolution* specification of a function, i.e. specifies a table of length 2^n for all $n \geq 1$. We use a WFA to specify a multiresolution function $f : \Sigma^\star \to R$ and possibly a function $\hat{f} : [0,1] \to R$ as the limit of tables of size 2^n for $n \to \infty$.

An m-state WFA A over $\Sigma = \{0,1\}$ is defined by

(1) a row vector $I \in R^{1\times m}$, the *initial distribution*,

(2) a column vector $F \in R^{m\times 1}$, the *final distribution*,

(3) two weight matrices $W_0, W_1 \in R^{m\times m}$, the weight matrices for input 0 and 1,
respectively.

WFA A defines the multiresolution function $f_A : \Sigma^\star \to R$ specified for all $a_1 a_2 \ldots a_k \in \Sigma^\star$ by $f_A(a_1 a_2 \ldots a_k) = I \cdot W_{a_1} \cdot W_{a_2} \ldots W_{a_k} F$. A diagram of (small) WFA is similar to that used for finite automata. States are represented by nodes (circles), the components of the initial and final distribution are shown inside the circles. If, for $a = 0, 1$, $(W_a)_{i,j} \neq 0$, then there is an directed edge from node i to node j labeled by $a : (W_a)_{i,j}$.

Example. The WFA in Fig. 1(a) specifies the constant function $f_1(x) = c$, for initial distribution (c). The WFA in Fig. 1(b) specifies the linear function $f_2(x) = ax + b$ for initial distribution (a, b). The WFA in Fig. 1(c) specifies the quadratic function $ax^2 + bx + c$ for the initial distribution (a, b, c). Hence, it can specify any polynomial of degree up to 2. In [1] we give a fixed WFA with $n + 1$ states with initial distribution $(a_0, a_1, \ldots, a_n)$ that generates the polynomial $a_0 x^n + a_1 x^{n-1} + \ldots + a_n$, for any $a_0, a_1, \ldots, a_n$, thus any polynomial of degree smaller or equal to n.

In [3] there is discussed an efficient algorithm for inferring a WFA generating the greyness function of an image in pixel form. Here we need a simple version of this algorithm. It infers a WFA generating a function of one variable specified by a table. We define size(A) to be the actual storage space in bytes required to store the WFA A. The quality of the regenerated function will be measured by the square of the L^2-metric: Let f and g be two multiresolution functions, and let k be a positive integer. Define

$$d_k(f,g) = \sum_{w \in \{0,1\}^k} (f(w) - g(w))^2.$$

Given a function f at resolution 2^k, the algorithm tries to find a WFA A that would minimize the value of

$$d_k(f_A, f) + G \cdot \text{size}(A), \tag{1}$$

where G is a positive real number. Parameter G controls the compression ratio and the quality of the regenerated function. With large values of G a small automaton with poor approximation of f is produced. When G is made smaller, the approximation improves while the size of the automaton increases.

The main structure of the recursive inference algorithm is given in Table 1. The algorithm uses global variables n (the number of states in the automaton at the moment), and $\psi_1, \psi_2, \ldots, \psi_n$ (the functions corresponding to the states). A recursive call $make\_wfa(i, k, max)$ processes state number i: Each quadrant of ψ_i is expressed either as a linear combination of existing states or by introducing a new state. The new state is processed by a recursive call to *make_wfa*. The function ψ_i is studied at level k, and is considered an function of size 2^k. The algorithm tries to minimize the value of

$$cost = d_k(\psi_i, \psi_i') + G \cdot s, \tag{2}$$

where ψ_i' denotes the approximation of the function ψ_i that is obtained, and s is the increase in the size of the automaton caused by the new states and edges that are added to the automaton in order to compute ψ_i'. If $cost > max$ function *make_wfa* returns value ∞ (indicating that max could not be improved), otherwise it returns $cost$.

During the recursive call all four quadrants of ψ_i are approximated in two different ways: by linear combinations of existing states (step 1 in Table 1), and by adding a new state corresponding to the function in the present quadrant and recursively using *make_wfa* to approximate it (steps 2 and 3). Whichever alternative yields a better result is chosen (steps 4 and 5).

Finally, we replace the present quadrant in ψ_i with the approximation that was obtained, and we then update the other function $\psi_1, \psi_2, \ldots, \psi_n$ that depend on ψ_i (step 6). If this is not done, in any future approximation the "ideal" ψ_i is used instead of the "real", slightly different approximation of ψ_i. This would cause further errors. Note that before we know the "real" ψ_i' we have no alternative but to use the "ideal" ψ_i in the linear combinations in step 1. This causes errors that are difficult to deal with, but at least we should use the "real" value as soon as we know it. One possible way to avoid the uncontrolled errors is not to use a state in linear combinations until we know the real function it represents. This is the approach used in our present implementation. It has the drawback of not allowing loops in the WFA.

Initially, before calling the recursive function *make_wfa* for the first time, we must set $n \leftarrow 1$ and $\psi_1 \leftarrow f$, where f is the image that needs to be approximated at level k. The approximation is done by calling $make\_wfa(1, k, \infty)$.

The algorithm produces a WFA with fixed initial distribution $I_1 = 1$, $I_i = 0$ for all $i \geq 2$. Therefore, the initial distribution does not need to be stored. See [3] for details related to storing the automaton using arithmetic coding.

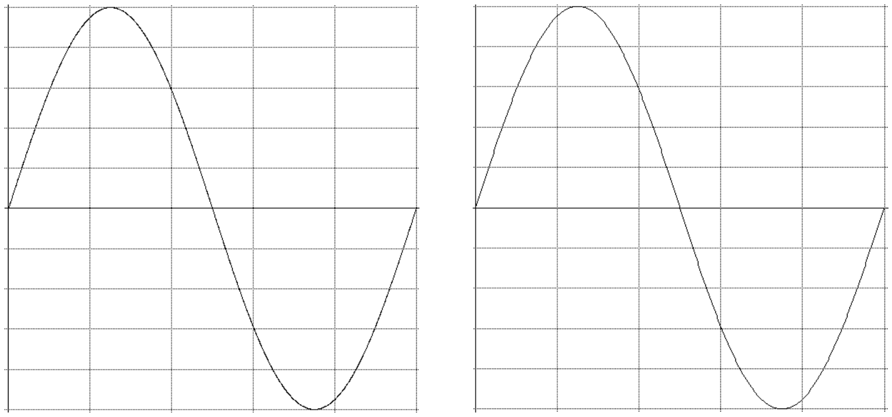

Figure 2: Function $\sin(x)$ and its approximation by a six state WFA

A fixed set of functions can be used in all linear combinations starting from step 1 of the algorithm. Such functions are said to form an *initial basis*. Before calling *make_wfa* for the first time, we set $n \leftarrow N+1$, $\psi_1 \leftarrow f$, where f is the image to be compressed, and $\psi_2, \psi_3, \ldots, \psi_{N+1}$ are set to be the N fixed images in the initial basis. The functions in the basis can be chosen arbitrarily — they do not need to be definable by a WFA. The choice of the initial basis can of course depend on the application, that is, on the type of images one wants to compress. The initial basis resembles the code book in vector quantization [14] which can be viewed as a very restricted version of our method. Here we have used the initial basis containing three functions: constant $x \to 1$, linear $x \to x$ and quadratic $x \to x^2$. All quadratic functions $x \to ax^2 + bx + c$ can be expressed as their linear combinations, so the basis functions alone provide good piecewise approximations of smooth functions.

Fig. 2 shows the graph of function $\sin(x)$ on the left and its approximation by a WFA with six states inferred by our algorithm.

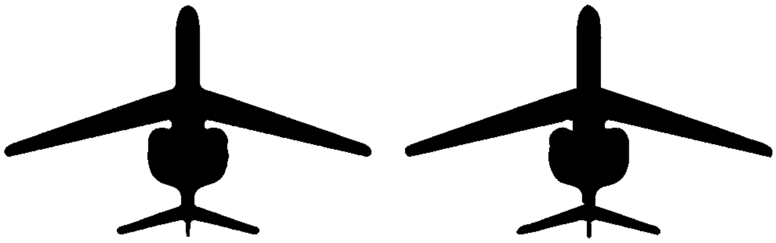

Figure 3: original 512×512, 396 bytes, 0.01208 bpp, 0.11% wrong pixels

3 Compression algorithm and results

Given a bi-level image as input we start by using the algorithm from [11] to obtain $NESW$ chain code for each segment i, $i = 1, \ldots, k$ where k is a number of segments. We convert each chain code into two parametric functions x_i, y_i for $i = 1, \ldots, k$ in the following way. Denote $a_i(j)$ to be a j-th letter in chain code i. We set $x_i(0) = y_i(0) = 0$ for all i. We continue generating x_i, y_i functions as follows:

$$a_i(j) = \begin{cases} E & : \quad x_i(j) = x_i(j-1) + 1,\ y_i(j) = y_i(j-1) \\ W & : \quad x_i(j) = x_i(j-1) - 1,\ y_i(j) = y_i(j-1) \\ N & : \quad y_i(j) = y_i(j-1) + 1,\ x_i(j) = x_i(j-1) \\ S & : \quad y_i(j) = y_i(j-1) - 1,\ x_i(j) = x_i(j-1) \end{cases}$$

We merge the $2k$ functions (tables) into one function (longer table) z by concatenating the domains. Next, the function z is compressed using the inference algorithm described in the previous chapter. The inference algorithm produces a WFA A such that the function f_A it defines is a close approximation of function z (under MSE metric). The WFA is then compactly stored using arithmetic coding, as explained in detail in [13].

The decoder works in the following way: first we compute the function f_A defined by WFA A using the fast decoding algorithm

Figure 4: original 512×496, 496 bytes, 0.01514 bpp, 0.25% wrong pixels

from [2]. Then we decompose function f_A into k parts $\bar{x}_i$, $\bar{y}_i$ for $i = 1, \ldots, k$. Since we conducted a lossy compression, functions $\bar{x}_i$ and $\bar{y}_i$ only approximate functions x_i and y_i, respectively. Finally, let l_i denote a length of the table specifying function x_i (the length of y_i is the same). We reconstruct the segment i for each $i = 1, \ldots, k$ as follows: we connect by a straight line each pair of points $[(\bar{x}_i(j),\ \bar{y}_i(j)),\ (\bar{x}_i(j+1),\ \bar{y}_i(j+1))]$, for $j = 0, \ldots, l_i$, and also the pair $[(\bar{x}_i(l_i),\ \bar{y}_i(l_i)),\ (\bar{x}_i(0),\ \bar{y}_i(0))]$.

The results of three of our experiments are shown in Figures 3, 4 and 5. The original images are on the left, the reconstructed images on the right. Note that the lossy compression actually smoothes some jagged lines, e.g. the wheel and the barrel of the cannon. This is because, within the allowed error, the automaton describing the smooth curve is simple and hence results in small cost.

Conclusion. The new method outperforms the GFA method in case if high quality of regenerated images is desired. We could expect further improvement of this method if the inference algorithm stops subdividing the tables specifying the functions at the size 16 and uses vector quantization at this point.

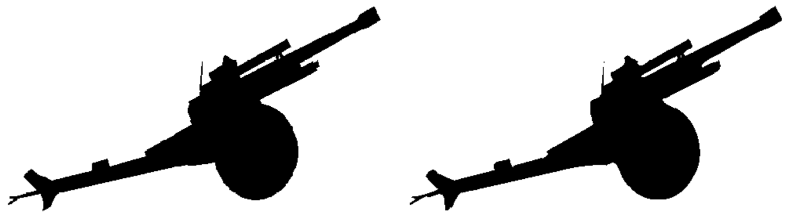

Figure 5: original 512×512, 511 bytes, 0.01559 bpp, 0.19% wrong pixels

Acknowledgment. The authors are grateful to Robert R. Estes for discussion of his results and for allowing the authors to use his chain code routines.

References

[1] K. Culik II and J. Karhumäki, Automata Computing Real Functions, *SIAM J. on Computing* **23**, 789-814 (1994).

[2] K. Culik II and J. Kari, Image Compression Using Weighted Finite Automata, *Computer and Graphics* **17**, 305-313 (1993).

[3] K. Culik II and J. Kari, Image-Data Compression Using Edge-Optimizing Algorithm for WFA Inference, *Journal of Information Processing and Management* **30**, 829-838 (1994).

[4] K. Culik II and J. Kari, Efficient Inference Algorithm for Weighted Finite Automata, in: *Fractal Image Compression*, ed. Y.Fisher, Springer-Verlag (1994).

[5] K. Culik II and J. Kari, J. Finite state methods for image manipulation, Proceedings of ICALP 95, *Lecture Notes in Computer Science* **944**, Springer-Verlag, 51-62 (1995).

[6] K. Culik II and J. Kari, Finite State Methods for Compression and Manipulation of Images, *Proceedings of Data Compression Conference 1995*, Snowbird, Utah, 142-151 (1995).

[7] K. Culik II and J. Kari, Finite State Transformation of Images, *Computer and Graphics*, 20, 125-135 (1996)

[8] K. Culik II and V. Valenta, Finite Automata Based Compression of Bi-level Images, in: *Proceedings of the Data Compression Conference*, Snowbird, Utah, 280-289 (1996).

[9] K. Culik II and V. Valenta, Finite Automata Based Compression of Bi-level and Simple Color Images, *Computer and Graphics*, 21, 61-68 (1997).

[10] K. Culik II and V. Valenta, Generalized finite automata and transducers, submitted to *Journals of Automata, Languages and Combinatorics* .

[11] Robert R. Estes, Jr. and V. Ralph Algazi, Efficient error free chain coding of binary documents, in: *Proceedings of the Data Compression Conference*, Snowbird, Utah, April, 122-132 (1995).

[12] U. Hafner, Refining Image Compression with Weighted Finite Automata, in: *Proceedings of DCC 96,* 359-368, Snowbird, Utah (1966).

[13] J. Kari and P. Fränti, Arithmetic coding of weighted finite automata, *R.A.I.R.O. Theoretical Informatics and Applications*, 28, 3-4, 343-360 (1994).

[14] M. Rabbani and P. W. Jones, *Digital Image Compression Techniques*, Tutorial Texts in Optical Engineering, TT7, SPIE, Optical Enginering Press, Bellingham, Washington (1991).

make_wfa(i,k,max) :
If $max < 0$ then return(∞);
If k=0 return($d_0(f,0)$);

$cost \leftarrow 0$;
do the steps 1–5 with $\psi = (\psi_i)_a$ for $a = 0$ and 1:

1. Find $r_1, r_2, \ldots r_n$ such that the value of

$$cost1 \leftarrow d_{k-1}(\psi, r_1\psi_1 + \ldots + r_n\psi_n) + G \cdot s$$

   is small, where s denotes the increase in the size of the automaton caused by adding edges from state i to states j with non-zero weights r_j and label a, and d_{k-1} denotes the distance between two multiresolution functions at level $k-1$.

2. $n_0 \leftarrow n$, $n \leftarrow n+1$, $\psi_n \leftarrow \psi$ and add an edge from state i to the new state n with label a and weight 1. Let s denote the increase in the size of the automaton caused by the new state and edge.

3. $cost2 \leftarrow s +$ *make_wfa(n,k-1,min{max−cost,cost1}−s)*;

4. If $cost2 \leq cost1$ then $cost \leftarrow cost + cost2$;

5. If $cost1 < cost2$ then $cost \leftarrow cost + cost1$, remove all outgoing transitions from states $n_0+1, \ldots n$ (added during the recursive call), as well as the transition from state i added on step 2. Set $n \leftarrow n_0$ and add the transitions from state i with label a to states $j = 1, 2, \ldots n$ with weights r_j whenever $r_j \neq 0$.

6. Update functions $\psi_1, \psi_2, \ldots, \psi_n$ to reflect the error done in quadrant a of ψ_i.

If $cost \leq max$ return($cost$) else return(∞);

Table 1: Outline of the recursive inference algorithm for WFA.