Finite Automata Based Compression of Bi-level and Simple Color Images *

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

Abstract

We introduce generalized finite automata as a tool for specification of bi-level images. We describe an inference algorithm for generalized finite automata and a lossy compression system for bi-level and simple color images based on this algorithm and vector quantization.

1 Introduction

A bi-level multiresolution image is specified by assigning the value 0 or 1 to every node of the infinite quadtree. If the outgoing edges of each node of the quadtree are labeled $0, 1, 2, 3$, we get a uniquely labeled path to every node; its label is called the address of the node. The address of a node at depth k is a string of length k over the alphabet $\{0, 1, 2, 3\}$. Hence, a bi-level multiresolution image can be specified as a subset of strings over the alphabet $\{0, 1, 2, 3\}$, namely the collection of the addresses of the nodes assigned value 1 (black). Regular sets of strings are specified by finite automata or regular expressions [13]. Therefore, finite automata can be used to specify (regular) multiresolution images. This idea has been recognized independently by several authors [1, 2, 4, 14], but has not led to a successful image-data compression method.

The finite automata method of image-specification has been extended to gray-scale images, represented by Weighted Finite Automata (WFA) [6, 7]. A theoretical WFA inference algorithm was proposed in [7]. This algorithm finds a WFA with the minimal number of states

*This work was supported by the National Science Foundation under Grant No. CCR-9417384. Preliminary version of a part of this paper was presented at the Data Compression Conference, Snowbird, Utah, 1996.

for an image which can be perfectly (without error) represented by a WFA. However, for realistic images (photographs), which must be approximated, this algorithm infers automata with many edges (transitions) and it is not useful for image compression. In [8] a recursive algorithm for WFA-inference was proposed; based on this algorithm a (lossy) image-data compression method has been developed which gives an excellent relation between the image quality and the compression rate. It also has additional advantages: it works well for the widest variety of images, and it allows zooming and many other image transformations to be done easily on compressed images. That is, we can decode a part of an image, its filtered version, etc. without decoding the original image, see [10, 12, 11].

The objective of our work is to design a lossy compression method for bi-level images based on finite automata which, at least for the silhouette-like images, would have an excellent quality/size ratio. The first idea is to modify the successful algorithm from [8]. That, however, would not work since the important property of WFA is that the WFA with underlying nondeterministic automata are more powerful than "deterministic" WFA. For example, the linearly sloping grayness function can be implemented by a two state "nondeterministic" WFA, but not by any "deterministic" one. The recursive algorithm from [8] efficiently finds the best way to express every "subsquare image" as a linear combination of the existing states (images) and thus takes advantage of the nondeterminism of WFA.

For bi-level images the situation is quite different. It is well known that nondeterministic finite automata define the same (regular) sets as deterministic [13], even if the nondeterministic automaton might be much smaller than the equivalent deterministic automaton. Because our experiments have shown that nondeterminism does not cause a substantial increase in the quality/size ratio for bi-level images, and it is expensive in the terms of the running time, our algorithm proposed here is completely different from the algorithm for gray-scale images in [8], in particular, it is not recursive. The resulting automaton is deterministic, but it is a generalized finite automaton. We introduce new features into the automata that allow a briefer description of an image and thus higher compression without further degradation of quality.

If a (classical) deterministic finite automaton A represents an image I, then each state of A must correspond to a subsquare of I, with the initial state corresponding to the whole I. Moreover, if there is a transition from state i to state j labeled by 0 (1,2,3), then the image corresponding to state j is the south-west (NW, SE, NE) quadrant of the image corresponding to state i.

In our generalized finite automata we will allow any combination of rotations, flips and complementation of the quadrant image. Typically, a generalized finite automaton will have a smaller number of states than an (almost) equivalent classical finite automaton. The price to pay is the need to remember the transformation done on the subsquares. Since we have 16 different transformations available, we will need 4 more bits for each edge of the generalized automaton.

In the next section we explain in detail our notation for bi-level images and how finite

automata are used to specify bi-level images. In section 3 we introduce generalized finite automata (GFA) and again explain how they specify images. We show examples of images that can be perfectly specified by GFA. In section 4 we describe an encoding algorithm for GFA which for any given bi-level image finds a GFA that generates a good approximation of the image. The quality of the regenerated image and the compression ratio is determined by a given parameter. If no self-similarity is found on the higher levels our algorithm uses vector quantization for subimages of size 8×8. We show examples which demonstrate quality/size performance of our compression program.

In the last section we explain the extension of our algorithm which allows us to apply our method to restricted class of color images. Assuming that such an image has at most 2^n different colors, i.e. its digital representation uses n bits per pixel. We will apply our algorithm to each bitplane with the modification that the resulting automata share states. We show some examples. Color quantization as a preprocessing tool is used to simplify images with too many colors.

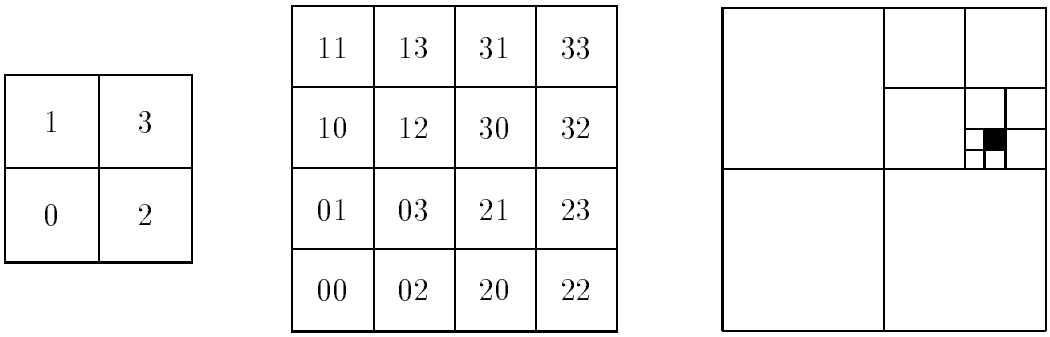


Figure 1: The addresses of the quadrants, of the subsquares of resolution 4×4, and the subsquare specified by the string 3203.

2 Black and white images and finite automata

A digitized image of the finite resolution $m \times n$ consists of $m \times n$ pixels each of which takes a Boolean value (1 for black, 0 for white) for bi-level image image, or a real value (practically digitized to an integer between 0 and 256) for a gray-scale image.

Here we will consider square images of resolution $2^n \times 2^n$ (typically $7 \leq n \leq 11$). In order to facilitate the application of finite automata to image description we will assign each pixel at $2^n \times 2^n$ resolution a word of length n over the alphabet $\Sigma = \{0, 1, 2, 3\}$ as its address. A pixel at $2^n \times 2^n$ resolution corresponds to a subsquare of size 2^{-n} of the unit square. We choose ϵ as the address of the whole unit square. Its quadrants are addressed by single digits as

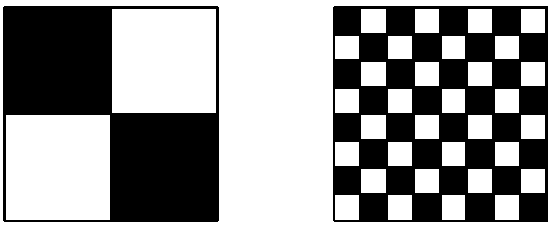


Figure 2: 2×2 and 8×8 chess-boards.

shown in Fig. 1 on the left. The four subsquares of the square with address w are addressed $w0, w1, w2$ and $w3$, recursively. Addresses of all the subsquares (pixels) of resolution 4×4 are shown in Fig. 1, middle. The subsquare (pixel) with address 3203 is shown on the right of Fig. 1.

In order to specify a black and white image of resolution $2^m \times 2^m$, we need to specify a Boolean function $\Sigma^m \to \{0,1\}$, or alternately we can specify just the set of pixels which are black, i.e. a language $L \subseteq \Sigma^m$. Frequently, it is useful to consider multiresolution images, that is images which are simultaneously specified for all possible resolutions, usually in some compatible way. (We denote by Σ^m the set of all words over Σ of length m, by $\Sigma^\star$ the set of all words over Σ.)

In our notation a bi-level multiresolution image is specified by a language $L \subseteq \Sigma^\star$, $\Sigma = \{0,1,2,3\}$, i.e. the set of addresses of all the black squares, at any resolution. Now, we are ready to give some examples. We assume that the reader is familiar with the elementary facts about finite automata and regular sets, see e.g. [13].

A finite automaton is displayed by its diagram which is a directed graph whose nodes are the states, with the initial node indicated by an incoming arrow and the final nodes by double circles. An edge labeled a from state i to state j indicates that input a causes the transition from state i to state j. A word in the input alphabet is accepted by the automaton if it labels a path from the initial state to a final state. The set (language accepted by automaton A) is denoted by $L(A)$.

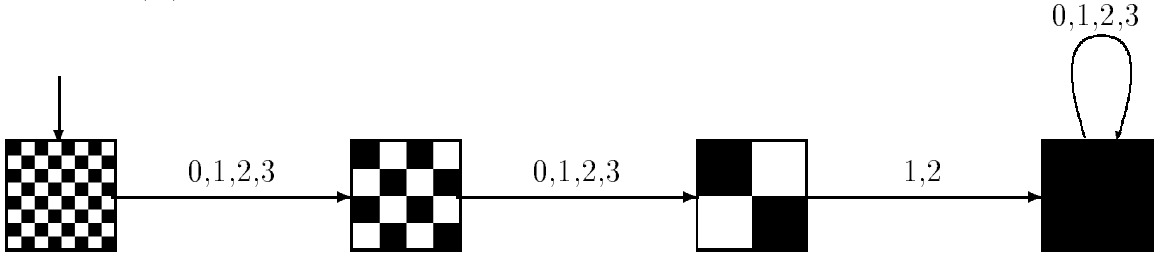
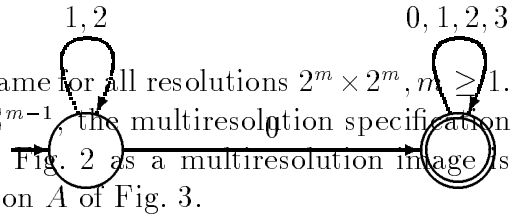


Figure 3: Finite automaton A defining the 8×8 chess-board.

Example. The 2×2 chess-board in Fig. 2 looks the same for all resolutions $2^m \times 2^m, m \geq 1$. For depth m, the specification is the finite set $\{1,2\}\Sigma^{m-1}$, the multiresolution specification is the regular set $\{1,2\}\Sigma^\star$. The 8×8 chess-board in Fig. 2 as a multiresolution image is described by the regular set $\Sigma^2\{1,2\}\Sigma^\star$ or by automaton A of Fig. 3.



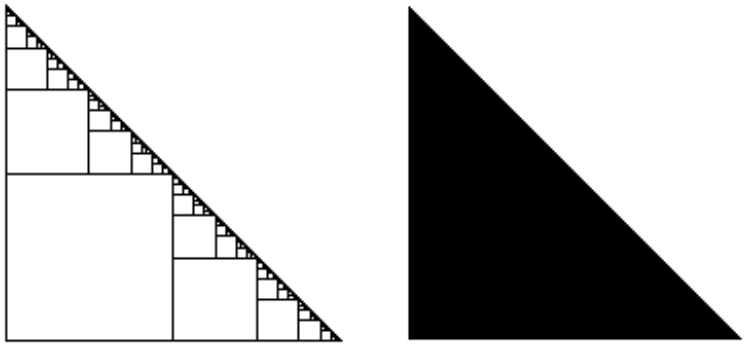


Figure 4: The squares specified by $\{1,2\}^\star 0$, a triangle defined by $\{1,2\}^\star 0\,\Sigma^\star$, and the corresponding automaton.

Notice that here we used the fact that the regular expression $\Sigma^2\{1,2\}\Sigma^\star$ is the concatenation of two regular expressions Σ^2 and $\{1,2\}\Sigma^\star$. It is easy to show that in general if the image is described by the concatenation of two languages $L = L_1 L_2$, then the image L is always obtained by placing copies of the image L_2 into all the squares addressed by the strings of L_1. Note that we can also obtain the 8×8 chess-board by placing 4 copies of the 4×4 chess-board $\Sigma\{1,2\}\Sigma^\star$ into the squares addressed $0, 1, 2$ and 3, that is as concatenation of Σ and $\Sigma\{1,2\}\Sigma^\star$.

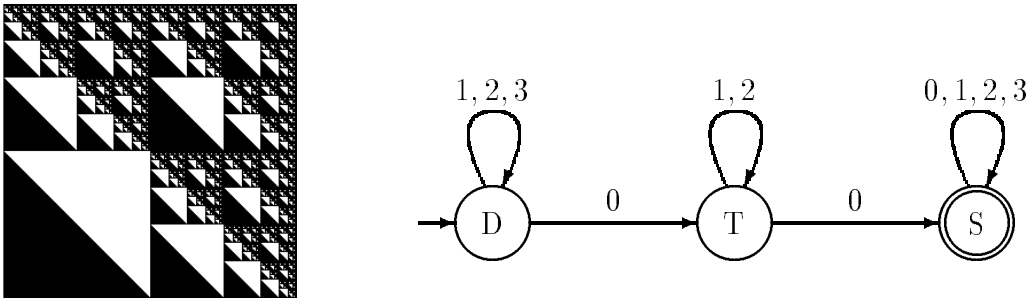


Figure 5: The diminishing triangles defined by $\{1,2\}^\star 0\,\Sigma^\star$, and the corresponding automaton.

Our concatenation decomposition $L = L_1 L_2$ works even when language L_1 is infinite as shown by the following example.

Example. Clearly, $L_1 = \{1,2\}^\star 0$ are addresses of the infinitely many squares illustrated at the left of Fig. 4. If we place the completely black square defined by $L_2 = \Sigma^\star$ into all

these squares we get the image specified by the concatenation $L_1L_2 = \{1,2\}^\star 0\Sigma^\star$ which is the triangle shown in the middle of Fig. 4.

Example. By placing the triangle $L = L_1L_2$ from the previous example into all the squares with addresses $L_3 = \{1,2,3\}^\star 0$ we get the image $L_3L = \{1,2,3\}^\star 0\{1,2\}^\star 0\Sigma^\star$ shown at the left of Fig. 5.

Zooming is easily implemented for images represented by regular sets. Let an image be represented by language L. Zooming to subsquare with address w, i.e. expanding the image in square w to the whole unit square, is done as follows. We take the left quotient of L with respect to w, that is $L_w = \{x \in \Sigma^\star \mid wx \in L\}$. This is especially easy when L is specified by a deterministic finite automaton (DFA) A. The DFA A_w accepting L_w is obtained by simply replacing the initial state of A by the state reached by input string w.

We have just shown that a necessary condition for a black and white multiresolution image to be represented by a regular set, is that it has only a finite number of different subimages in all the subsquares with addresses from $\Sigma^\star$. We will show that this condition is also sufficient. Therefore, images that can be perfectly (i.e. with infinite precision) described by regular expressions (finite automata) are images of regular or fractal character. Self-similarity is a typical property of fractals. Any image can be approximated by a regular expression (finite automaton), however, an approximation with a smaller error might require a larger automaton.

Now, we will give a theoretical procedure which, given a multiresolution image, finds a finite automaton perfectly specifying it, if such an automaton exists.

Procedure "Construct Automaton":
For given image I, we denote I_w the zoomed part of I in the square addressed w. The image represented by state number x is denoted by ψ_x.

1. $i = j = 0$.
2. Create state 0 and assign $\psi_0 = I$.
3. Assume $\psi_i = I_w$. Process state i, that is for $k = 0,1,2,3$ do:
   If $I_{wk} = \psi_q$ for some state q, then create an edge labeled k from state i to state q;
   otherwise assign $j = j + 1$, $\psi_j = I_{wk}$, and create an edge labeled k from state i to the new state j,
4. If $i = j$, that is all states have been processed, stop;
   otherwise $i = i + 1$, go to 3.

The procedure "Construct Automaton" terminates if there exists an automaton that perfectly specifies the given image and produces a deterministic automaton with the minimal number of states. Our lossy compression algorithm for bi-level images is based on this procedure, but it will use generalized finite automata introduced in the next section and only approximate the given image.

For the image "diminishing triangles" in Fig. 5, the procedure constructs the automaton shown at the right-hand side of Fig. 5. First the initial state D is created and processed. For 0 a new state T is created, for $1, 2$ and 3 a loop to itself. Then state T is processed for 0 a new state S is created, for 1 and 2 a loop back to T. There is no edge labeled 3 coming out of T since the quadrant 3 for T (triangle) is empty. Finally the state S (square) is processed by creating loops back to S for all 4 inputs.

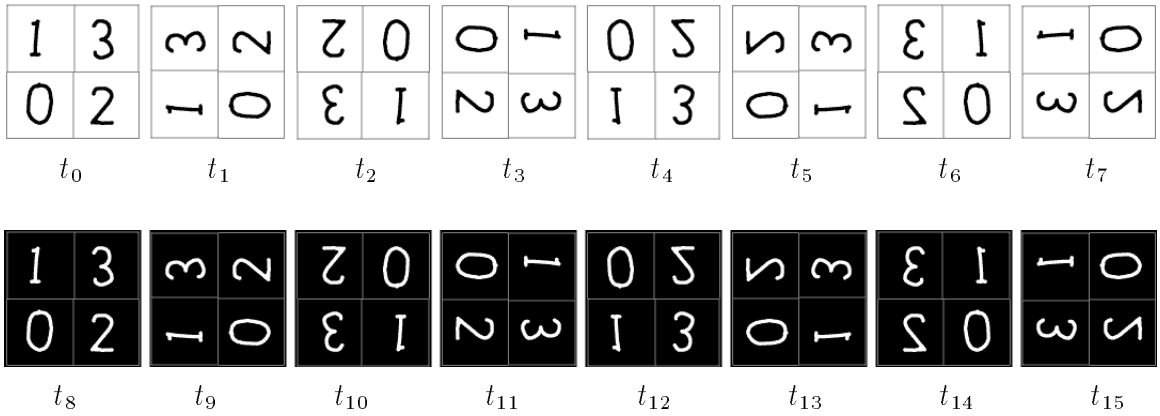


Figure 6: The image transformations

3 Generalized finite automata

Now, we introduce generalized finite automata (GFA) that specify bi-level images. Each transition of a GFA is labeled by an input symbol (like for classical FA) and optionally also by one of the 15 transformations that can be obtained by composition of rotation by 90°, flipping and complementation. In the diagrams of GFA the transformations are specified by numbers 0 – 15, i.e. t_i by i, and their meaning is indicated in Fig. 6. Identity transformation (0) can be omitted. A diagram of GFA is like a diagram of a FA with labels of the form $a{-}n$, where a is an input and n a transformation. For GFA A, with the set of states Q and initial state q_0, we say that state q represents the image I_q, for each $q \in Q$, if the following holds. For each $p \in Q$ and $a \in \{0, 1, 2, 3\}$, let $(p, a{-}n_i, q_i)$, $i = 1, 2, \ldots, r$ be all edges leading from state p and labeled by $a - n_i$. Then the image obtained from I_p by zooming into quadrant a is $\bigcup_{i=1}^{r} t_{n_i}(I_{q_i})$, where the union of images is defined as Boolean OR on pixel values. The

image $I(A)$ defined by GFA A is I_{q_0}, the image represented by its initial state. All the states of a GFA are always final so we do not have to indicate them. In the following diagrams we show at most one edge (in each direction) and we will use multiple labels between any two nodes.

Example. The GFA B shown in Fig. 7 specifies the image shown in Fig. 9(a), and the GFA C shown in Fig. 8 specifies the image shown in Fig. 9(b). Note that the rightmost state of GFA B alone generates the triangle of Fig. 4.

All the examples shown up to now have given infinitely precise description of the images, i.e. each of the images can be regenerated at any resolution without error. For these regular or fractal-like images our algorithm achieves extremely high lossless compression. In the next section we describe how our algorithm infers an approximation of any given finite resolution image.

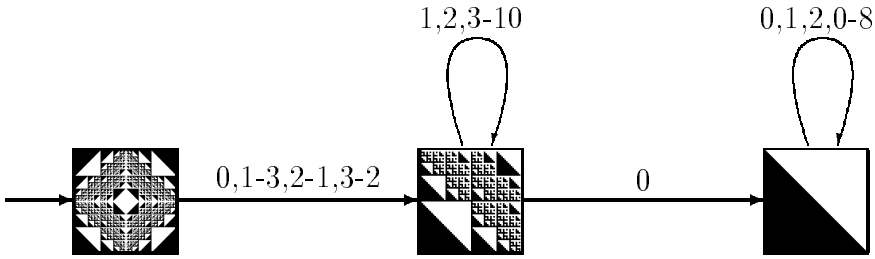


Figure 7: GFA B

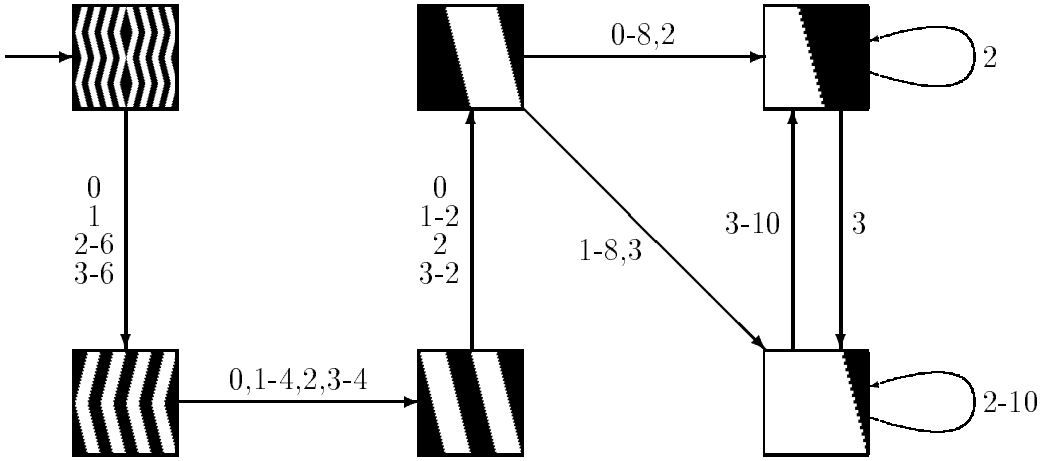
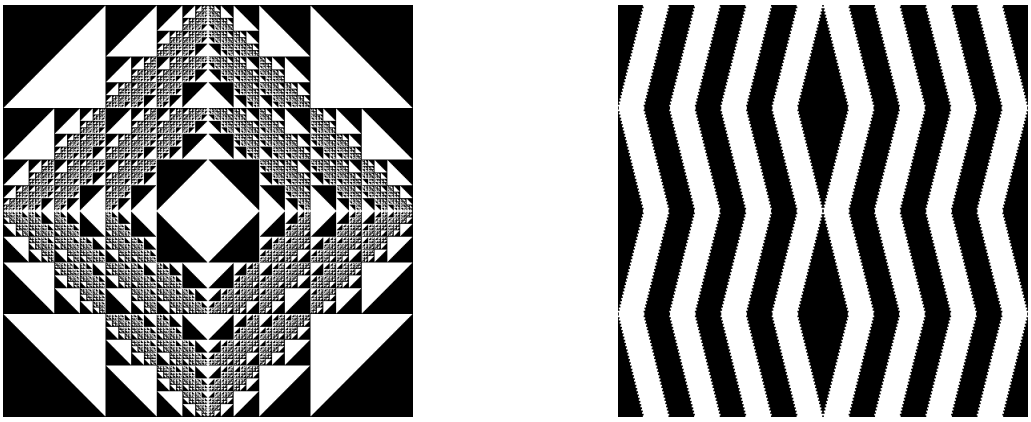


Figure 8: GFA C



(a) (b)

Figure 9: The images generated by GFA B and C

4 Compression algorithm and results

In this section we describe an algorithm that finds, for a given bi-level image, a GFA that generates a good approximation of the image. The quality of the regenerated image will be measured by the following metric: Let f and g be two multiresolution images, and let k be a positive integer. Define

$$d_k(f,g) = \frac{\sum_{w\in\Sigma^k} |f(w) - g(w)|}{2^k \times 2^k}$$

In other words, the difference between two images on a given resolution is a percentage of pixels where they differ. We interpret 0 as white and 1 as black so, for example, $\psi = 0$ means that image ψ is completely white.

Algorithm

Input: A bi-level image given by a finite labeled quadtree ψ, *error* value.

1. Assign initial state q_0 to the subsquare ε (the whole image). State q_0 will be the only one displayed, i.e. the initial distribution will be $\alpha(q_0) = 1, \alpha(q) = 0$ for all $q \neq q_0$. The final distribution β is $\beta(q) = 1$ for all q.
2. Recursively, process each state as follows: for next unprocessed state q, assigned to the square w, divide w into four subsquares $w0, w1, w2, w3$. Do step 3 with wa for $a = 0, 1, 2$ and 3:
3. Denote the image in the subsquare $wa, a \in \Sigma$, by ψ'. If $\psi' = 0$ there is no transition from q with label a. Otherwise, the algorithm searches through all created states and all transformations t_i, $i = 0 \ldots 15$. If state p and transition t_j is found such that

Figure 10: Original 512×512, 365 bytes, 0.01114 bpp, 0.23% wrong pixels

$d_k(\psi', t_j(\psi_p)) \leq$ *error* where ψ_p is a subsquare image assigned to the state p, we create a new edge $(q, a-j, p)$. If there is no such state and transformation, we assign a new (unprocessed) state r to ψ' and create a new edge $(q, a-0, r)$.

4. Go to step 2 if there is an unprocessed state, stop otherwise.

Even though the general concept of the inference algorithm is quite simple, there are many details that need to be solved in order to make the implementation efficient.

The first difficulty is in step 3 — to choosing a state p and transformation t_j to express ψ. We use two different approaches, *first fit* and *best fit*. When using the first approach, we search through all created states and use all our defined transformations on each state. Then we compute the distance between the images. This process is terminated if a "good" approximation is found. In the latter approach we always conduct the full search and we choose the best approximation. *Best fit* gives generally better results in quality, but it takes longer. However, even when using *best fit* the encoding of a typical image, including arithmetic coding of the automaton, takes only about 2 – 10 sec on PC 486 (depending on the given image and the *error* value).

Our experiments show that it is not efficient to express images of size 8×8 pixels or smaller by GFA. We get significantly better results (in terms of the time needed for encoding as well as the quality/size ratio) if we use a vector quantization on this resolution. We use a code-book of size 256 created by generalized Lloyd algorithm [3]. However, if a self-similarity is found at higher resolution, VQ is not used for that part of the image. Thus for the images shown in Fig. 9 no VQ is used at all.

The final step of our compression algorithm is storing the automaton in a file. The file is divided into three parts. The first part contains the information about edges created during step 4 in the algorithm. In the second part, we store the indices of states where the transitions go. Finally, the codewords used in the vector quantization are stored in the third part of the file. Arithmetic coding is used to encode all three parts of the file.

For decoding we use the fast algorithm described in [7] with an appropriate modification in order to account for the 16 transformations (see Fig. 6) that might be applied to the

Figure 11: Original 512×512, 512 bytes, 0.01562 bpp, 0.37% wrong pixels

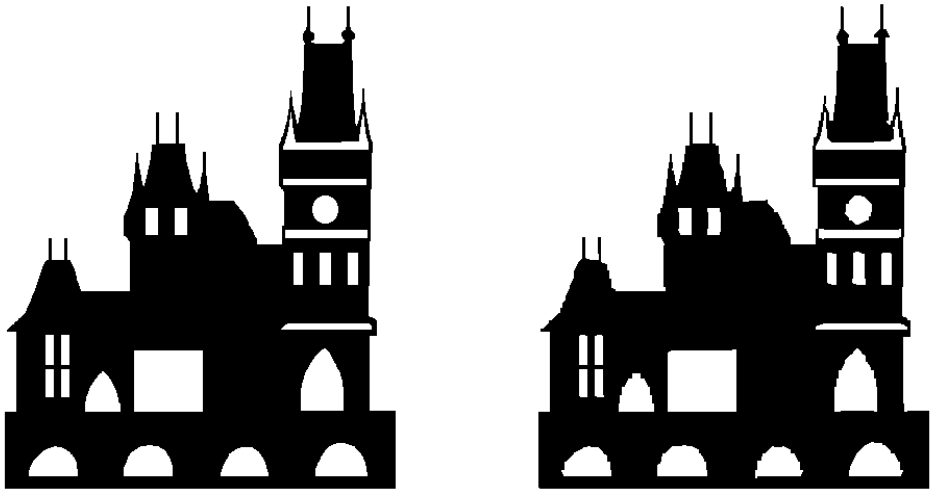


Figure 12: Original 512×512, 813 bytes, 0.02481 bpp, 0.32% wrong pixels

subimages.

The results of three of our experiments are shown in Figures 10, 11 and 12.

5 Compression of color images

Now we are ready to describe an extended algorithm that will allow us to find for a given color image a GFA that generates its approximation. We will restrict our attention to the images which consist of several areas with clear boundaries. Each area is filled with only one color that is distinct from all colors of the neighboring areas. However, most images consist of a large amount of different color shades that are spread all over the picture and where a clear border line between the different color shades cannot be found.

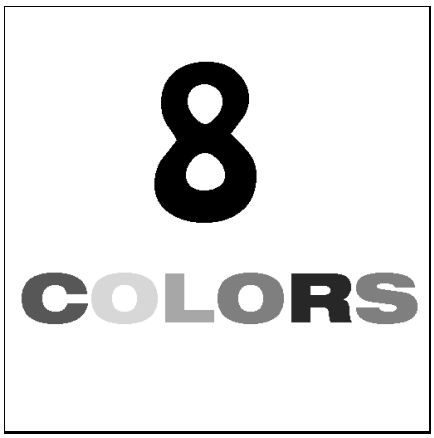


Figure 13: Original 512×512

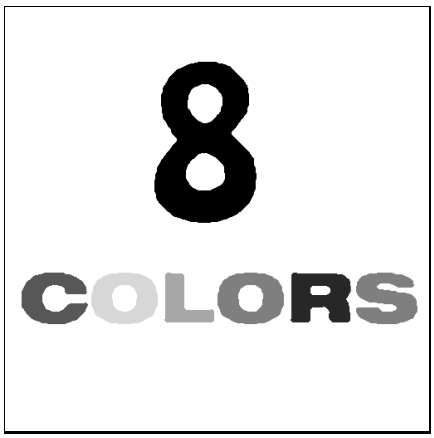


Figure 14: 638 bytes, 0.01947 bpp, 0.26% wrong pixels

In order to apply our algorithm to such color images some preprocessing is needed first, we will use color quantization. However, our approach is intended mainly to be used in cases where no or minimal color quantization is needed.

The most straightforward way how to extend algorithm from the previous section to color images would be to represent n-bit image by n bitplanes. Then we could use the algorithm described in the previous section and find a separate GFA A_i for each bitplane i. Although this technique would work, it would not take the advantage of the fact that some images in one bitplane can be expressed by other images in a different bitplane using one of our 16 transformations (usually by identity t_0 or negation t_8).

Consider the eight color image in Fig. 13 (the colors are here represented by different shades of gray) and its three bitplanes in Fig. 15. Clearly, the subimage with address 23 in the first bitplane (letter "S") is identical to the subimage with the same address in the third bitplane. Once this subimage has been expressed by a GFA in the first bitplane, we do not need to repeat the same process for the subimage in the third bitplane. Clearly, it is desirable that GFA A_3 share states with GFA A_1 because we reduce the time needed for building GFA A_3 and also gain better compression.

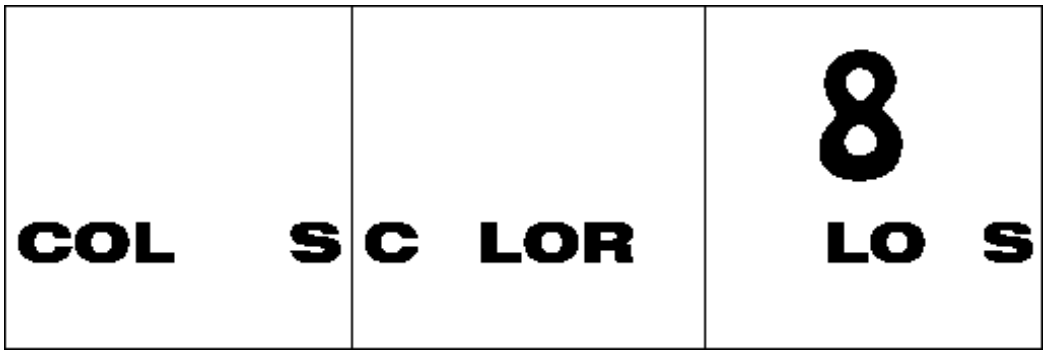


Figure 15: Three bitplanes

The algorithm for color images works the following way: First, color quantization is applied to the image. This step is very helpful since we get less colors and thus less bitplanes with clearer boundaries. This speeds up the algorithm and enables us to obtain a higher compression. Also, when using color quantization, we are able to reduce the amount of space that is needed for saving and restoring the color palette (list of colors used in the original image) and thus achieve a higher compression. We use the most recent color quantization technique [15] where colors are stored in leaves of an octree. The tree is then reduced by recursively merging the colors at the leaf level into their parent nodes until the desired number of colors is obtained. After the color quantization we decompose a given image into bitplanes. Then, using the algorithm described in the previous chapter, we find a GFA for the first bitplane. The same procedure with only a minor modification is used for all other bitplanes : the algorithm in step 3 searches through all created states of all bitplanes that have already been processed. This process can be viewed in two different ways. If n is the number of bitplanes we either construct n GFA's (one GFA for each bitplane) and resulting GFA's share states or we design only one GFA for all bitplanes together and this resulting GFA has n initial states.

The reconstructed image is shown in Fig. 14.

References

[1] J. Berstel and M. Morcrette, Compact representation of patterns by finite automata, *Proceedings Pixim'89*, Paris, 387-402 (1989).

[2] J. Berstel and A. A. Nait, Quadtrees generated by finite automata, *AFCET* 61/62, 167-175 (1989).

[3] R.J. Clarke, *Digital Compression of Still Images and Video*, Academic Press, Ltd. (1995).

[4] K. Culik II and S. Dube, Affine automata and related techniques for generation of complex images, *Proceedings of MFCS 1990, Lecture Notes In Computer Science* **452**, 224-231, Springer, Berlin (1990).

[5] K. Culik II and S. Dube, Rational and Affine Expressions for Image Description, *Discrete Applied Mathematics* **41**, 85-120 (1993).

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

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

[8] 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).

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

[10] 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).

[11] 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).

[12] K. Culik II and J. Kari, Finite State Transformation of Images, *Computer and Graphics*, to appear.

[13] J.E. Hopcroft and J.D. Ullman, *Introduction to automata theory, languages and computation*, Addison-Wesley (1979).

[14] L. Staiger, Quadtrees and the Hausdorff dimensions of pictures, Workshop on Geometrical Problems of Image Processing, Georgenthal, 173-178 (1989).

[15] Dean Clark, Color Quantization using Octrees Dr. Dobb's Journal, January 1996