Image Compression Method Based on Generalized Finite Automata

Xiaohu Ma, Huanqin Chen
(School of Computer Science and Technology, Soochow University, Suzhou, Jiangsu 215006)
xhma@suda.edu.cn

Abstract

In this paper, we introduce an approach to compress gray image using deterministic Generalized Finite Automata (GFA). By detecting the self-similarity inside an input digitized gray image, a GFA can be constructed to describe the image. The decode algorithm can restore the image from the deterministic Generalized Finite Automata efficiently. This method has a smaller number of states than an equivalent classical finite automaton. Meanwhile it also has an advantage of higher compression without further degradation of quality.

1. Introduction

With the rapid development of multimedia technology in recent years, the digital image transmission plays a very important role in communication field. Digital images generally include a great deal of information, so the compression technology becomes the key of the research. After the research and exploration of recent years, scholars have put forward a lot of encoding methods. Now the main methods of image encoding are fractal theory, neural networks, wavelet transform and so on.

Different from the methods mentioned above, this paper introduces a new algorithm called Generalized Finite Automata (GFA) to encode/decode the images automatically. In essence, the principle of finite automata is that using the self-similarity of the image to reduce the bits which are used to describe the image.

The finite automata method of image-specification has been extended to gray images, represented by Weighted Finite Automata (WFA) which was studied by Karel Culik II and Jarkko Kari [2][3]. This method studied a rule about how to describe images with the automata and put forward an algorithm about how to use nondeterministic finite automata (NFA) to encode/decode the images automatically. When detecting the self-similarity, Karel Culik II and Jarkko Kari only considered the relations of gray proportion transformation; so in order to increase the encoding ability, they adopted the nondeterministic finite automata. In our generalized finite automata we will allow any combination of rotations, flips and complementation of the quadrant image. All of these operations may increase the ability of detecting the self-similarity.

In the next section we introduce the basic concepts of finite automata. In section 3 we explain in detail our notation for gray images and how finite automata are used to specify gray images. In section 4 we introduce generalized finite automata (GFA) and explain how they specify images; and then we describe an encoding algorithm for GFA which for any given gray image finds a GFA that generates a good approximation of the image. In section 5 we apply the GFA algorithm to some commonly gray images. In the last section we summarize the paper and give the advantages of GFA method.

2. Basic concepts of finite automata

A bi-level multiresolution image is specified by assigning the value 0 or 1 to every node of the infinite quadtree (1 for black, 0 for white). 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}. Regular sets of strings are specified by finite automata or regular expressions [6]. Therefore, finite automata can be used to specify multiresolution images. This idea has been recognized independently by several authors in references [4] and [7].

2.1 Several definitions

A finite automata is a mathematical model with a discrete input/output system. It has finite inner structures or states to remember the input information.

And then according to the current input it can make sure the next state or action.

Definition 1: A finite automata is a 5-tuple (S,Σ,δ,s_0,F) in Ref [8] where:

（1） S is a finite set of states;

（2） Σ is a finite alphabet, each of the elements is called input character;

（3） $\delta: S\times\Sigma\to S$ is a transition function. The function δ records the transitions: $\delta(s,a)=s'$ if there is a transition from state s to state s' labeled by a. State s' is called the subsequent state of state s.

（4） $s_0\in S$ is a uniquely initial state;

（5） $F\subseteq S$ is a set of accepting states (may empty).

Definition 2: A finite automaton can be represented diagrammatically by a labeled directed graph as shown in Fig.1, called a transition graph, in which the nodes are the states and the labeled edges represent the transition function. A set of strings which labels a path from the initial state to the finial state can be accepted.

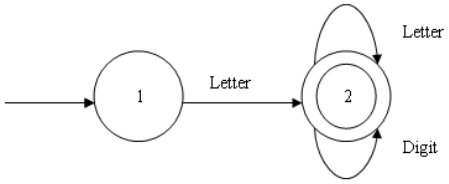


Figure 1. A transition graph

2.2 Image and finite automata

If a 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.

3. Gray image and the procedure of finite automata

3.1 Image partitioning and finite automata

The technique of our method is based on partitioning an image into sub-images, which was introduced in a scheme named quadtree image compression. The quadtree is a hierarchical data structure based on the divide and conquer principle, leading to a recursive partitioning. We examine the region quadtree, which divides the input image recursively into four quadrants of equal size.

Here we will consider square image of resolution $2^n\times 2^n$. 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. We choose the empty string ε as the whole image and e as the address of the whole unit square. Its quadrants are addressed by single digits as shown in Fig 2 on the left. The four subsquares of the square with address w are addressed $w0, w1, w2$ and $w3$, recursively. Addresses of all the pixels of resolution 4×4 are shown in Fig 2, middle. The pixel with address 3203 is shown in the right of Fig 2.

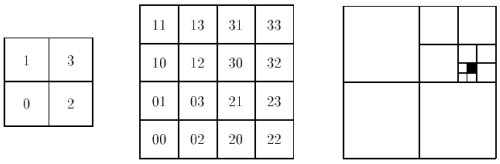


Figure 2. The address of the quadrants, of the subsquares of resolution 4×4 and the subsquare specified by the string 3203

A finite automaton is displayed by its diagram which is 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 $L(A)$.

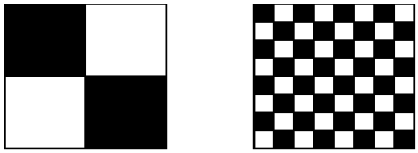


Figure 3. 2×2 chess-board and 8×8 chess-board

The 2×2 chess-board in Fig 3 on the left looks the same for all resolution $2^m\times 2^m$, $m\geq 1$. For depth m, the specification is the finite set $\{1, 2\}\Sigma^{m-1}$ (we denote by Σ^{m-1} the set of all words over Σ of length $m-1$), the multiresolution specification is the regular set $\{1,2\}\Sigma^*$. The 8×8 chess-board in Fig 3 on the right

as a multiresolution image is described by the regular set $\Sigma^2\{1,2\}\Sigma^*$ or by automaton of Fig 4.

3.2 The finite automata algorithm for multiresolution image

Now, given a multiresolution image, we will give a theoretical procedure which finds a finite automaton perfectly specifying it, if such an automaton exists. 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 initial 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 $I_{wk} = \psi_q$ *then* create an edge labeled k from state i to state q;
   else $j = j + 1$;
   $\psi_j = I_{wk}$;
   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;
   else $i = i + 1$;
   go to 3.

The procedure 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 gray images is based on this procedure, and it will use generalized finite automata introduced in the next section.

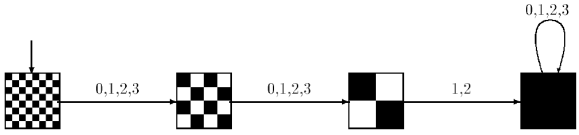


Figure 4. Finite automaton defining the 8×8 chess-board

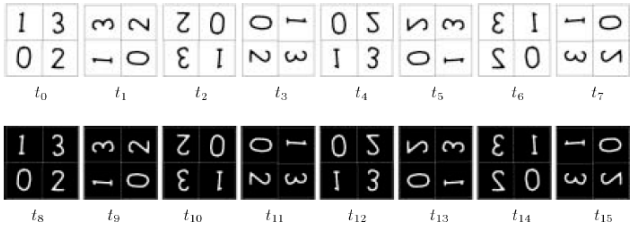


Figure 5. The image transformations

4. Generalized finite automata and compression algorithm

4.1 Generalized finite automata

Each transition of a GFA is labeled by an input symbol, and that it has 16 transformations as shown in Fig 5 [1].

In the diagrams of GFA the transformations are specified by numbers 0-15, i.e. t_i by i. 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 is a transformation. For a GFA, 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 $(p, a - n_i, q_i)$, $i = 1,2,\cdots,r$ 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 $\bigcup_{i=1}^{r} t_{ni}(I_{qi})$.

4.2 The encoding algorithm

Our work combined the GFA with the wavelet transform technique. We constructed a GFA with almost tree-structure such that each state represents one wavelet function from the family of Daubechies's wavelets forming a basis for the wavelet transform. So our method combines the advantages of the "classical" wavelet compression method.

Next we will give the main ideas of the GFA compression algorithm.

Input: A gray image given by a finite labeled quadtree ψ, an $error$ value.

1. The initial state q_0 expresses the subsquare ε (the whole image). For all $q \neq q_0$, the initial distribution will be $\alpha(q_0) = 1$, $\alpha(q) = 0$. 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 $w0$, $w1$, $w2$ and $w3$. Do step 3 with wa for $a = 1$, 2 and 3.
3. Denote the image in the subsquare $wa(a \in \Sigma)$ by ψ'. If ψ'=0, there is no transition from state q with label a. Otherwise, the algorithm searches through all created states and all transformations t_i, $i = 0, \cdots, 15$. If state p and transition t_j is found and $d_k(\psi', t_j(\psi_p)) \leq error$ 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, otherwise, stop.

The decoding algorithm is quite easy. Given a GFA, compute the string s of each pixel at $2^n \times 2^n$ resolution, the length of the string is n. Then according to the string s, the algorithm will transform the states recursively, meanwhile execute the rotations, flips and complementation of each transformation. After all of these operations, we will get the gray values of the pixel and the final image is restored.

5. Experimental results

In order to prove the performance of the GFA algorithm, we use Matlab 6.5 and VC++ as the experimental environment. The experimental gray images (resolution 256×256) are lena, boat, zelda and peppers with the grayscale 256. The experimental results are shown in Fig 6. And all the experimental numbers are listed in Table 1.

The visual quality of the regenerated image can be measured by the following metric: Peak Signal Noise Ratio (PSNR), compression ratio and d_k (the difference between the original image and the regenerated image on a given resolution, i.e. a percentage of pixels where they differ).

The formulas of PSNR and d_k are shown as formula (1) and (3):

$$PSNR = 10 * \log_{10}\left(\frac{255^2}{MSE}\right) \tag{1}$$

Where:

$$MSE = \frac{1}{N^2}\sum_{i=0}^{N^2-1}(f_i - g_i)^2 \tag{2}$$

f_i and g_i are the pixel values of the testing and regenerated images, N^2 is the total number of pixels.

$$d_k(f, g) = \frac{\sum_{w \in \Sigma^k} |f(w) - g(w)|}{2^k \times 2^k} \tag{3}$$

where f and g are two multiresolution images, and k is a positive integer. This formula can meet people's subjective visual feelings well. When the difference between two images is obvious, the value of d_k is much bigger.

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 (VQ) on this resolution. We use a codebook of size 256 created by generalized Lloyd algorithm [5].

6. Conclusions

We propose an image compression scheme using the bitplane modeling and generalized finite automata (GFA) representation aimed at optimally exploring the self-similarity inhabited in gray images. From the experimental results we can see that the regenerated images have no obvious blocking effect; On the other hand, although both DFA and NFA are capable of recognizing precisely the regular set, the GFA with underlying deterministic finite automata are more powerful than "nondeterministic" WFA suggested by Karek Culik II and Jarkko Kari. For example, the "deterministic" GFA can lead to faster recognizers than the "nondeterministic" WFA; second, when encoding the image, GFA doesn't have to solve equations; finally, GFA can make decoding image more quickly.


Gray image of lena

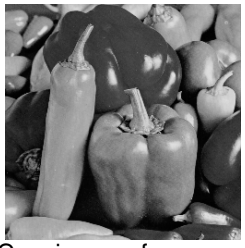

Decoded image of lena


Gray image of boat


Decoded image of boat


Gray image of zelda


Decoded image of zelda


Gray image of peppers

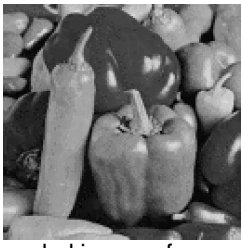

Decoded image of peppers

Figure 6. The comparison of original images and regenerated images

Table 1. Performance of GFA compression

Image Names	States number	Coding bytes	Compression ratio	*d*	PSNR
lena	2388	6605	6.1243：1	9.3891	25.8233
boat	2384	6774	6.0291：1	19.0676	20.2781
zelda	2521	7285	5.7584：1	21.2087	19.9599
peppers	2515	7107	5.8499：1	13.2287	20.8056

7. Acknowledgements

The authors wish to acknowledge the support from the Natural Science Foundation of Jiangsu Province under Grant BK2007050.

8. References

[1] Karel Culik II, Vladimir Valenta. Finite Automata Based Compression of Bi-level and Simple Color Images. Utah:Data Compression Conference,1996.

[2] K.Culik II, J.Karhumaki. Automata Computing Real Functions. SIAM J. on Computing, 1994,23:789-814.

[3] K.Culik II, J.Kari. Image Compression Using Weighted Finite Automata. Computer and Graphics,1993,17(3):305-313.

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

[5] R.J.Clarke. Digital Compression of Still Images and Video. Ltd.:Academic Press,1995.

[6] J.E.Hopcroft, J.D.Ullman. Introduction to automata theory. language and computation, Addison-Wesley,1979.

[7] J.Berstel, A.A.Nait. Quadtrees generated by finite automata. AFCET61/62,1989:167-175.

[8] Chen Huowang, Liu Chunlin, Tan Qingping. Programming Language: Compiler Construction Principles. Beijing: National Defence Industry Press, 2004.