Finite State Automata and Image Recognition

Marian Mindek

Department of Computer Science, FEI, VŠB - Technical University of Ostrava,
17. listopadu 15, 708 33, Ostrava-Poruba, Czech Republic
marian.mindek@vsb.cz

Abstract. In this paper we introduce finite automata as a tool for specification and compression of gray-scale image. We describe, what are interests points in pictures and idea if they can hang together with resultant finite automata.

Keywords: finite automata, interest points, image recognition

1 Introduction

Karel Culik II and Vladimir Valenta have proposed fractal-coding technique which is based on automata theory. In their paper [1] describe a inference algorithm for generalized finite automata and a lossy compression system for bi-level images based on this algorithm and vector quantization. In another paper [2] describe a similar algorithm for gray-scale image, which use a weighted finite automata (WFA). We're issue these ideas and describe algorithm for gray-scale pictures based on simple solution for bi-level pictures.

2 Finite automata

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

Here we will consider square images of resolution 2^n x 2^n (typically $6 \leq n \leq 11$). In order to facilitate the application of finite automata to image description we will assign each pixel at 2^n x 2^n resolution a word of length *n* over the alphabet $\Sigma=\{0,1,2,3\}$ as its address. A pixel at 2^n x 2^n resolution corresponds to a sub square 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 shown in Fig. 1 on the left. The four sub square of the square with address *w* are addressed *w0, w1, w2* and *w3*, recursively. Address of all the sub square (pixels) of resolution 4 x 4 are shown in Fig. 1, middle. The sub square (pixel) with address *3203* is shown on the right of Fig. 1.

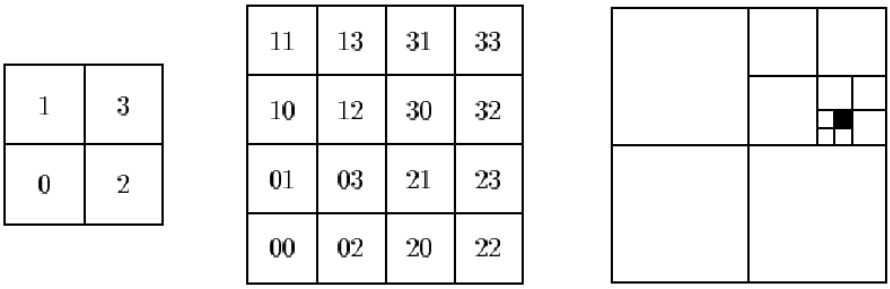

Figure 1. The addresses of the quadrants, of the sub square of resolution 4 x 4, and the sub square specified by the string 3203.

In order to specify a black and white image of resolution 2^m x 2^m, we need to specify a Boolean function $\Sigma^m \rightarrow \{0,1\}$, 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 multi-resolution images, that is images which are simultaneously specified for all possible resolution, usually in some compatible way. (We denote Σ^m the set of all words over Σ of the length *m*, by Σ^* the set of all words over Σ)

In our notation a bi-level multi-resolution image is specified by a language $L \subseteq \Sigma^*$, $\Sigma=\{0,1,2,3\}$, 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. [4].

A finite automaton is displayed by its diagram which is directed graph whose nodes are 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 the final state. The se (language accepted by automaton A) is denoted *L(A)*.

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

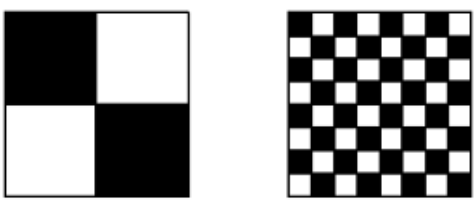

Figure 2. 2 x 2 and 8 x 8 chess-boards.

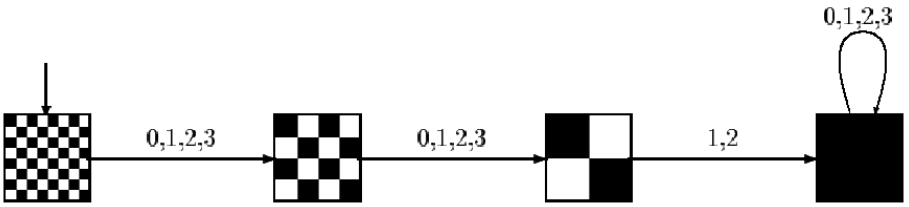

Figure 3. Finite automaton *A* defining the 8 x 8 chess-board.

Notice that here we used the fact that the regular expression $\Sigma^2\{1,2\}\Sigma^*$ is the concatenation of two regular expression Σ^2 and $\{1,2\}\Sigma^*$. It is easy to show that in general if the image is described by the concatenation of two languages $L=L_1L_2$,, then the image L is always obtained by placing copes of the image L_2 into all the squares addressed by the 4 x 4 chess-board $\Sigma\{1,2\}\Sigma^*$ into the squares addressed 0,1,2 and 3, that is as concatenation of Σ and $\Sigma\{1,2\}\Sigma^*$.

Our concatenation decomposition $L=L_1L_2$,, works even when language L_1, is infinite as shown by the following example.

Example. Clearly, $L_1= \{1,2\}^*0$ are addresses of the infinitely many squares illustrated at the left of Fig. 4. If we place the completely black square defined by $L_2=\Sigma^*$ into all these squares we get the image specified by the concatenation $L_1L_2=\{1,2\}^*0\Sigma^*$ which is the triangle shown in the middle of Fig. 4.

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

Example. By placing the triangle $L= L_1L_2$ from the previous example into all the squares with addresses $L_3=\{1,2,3\}^*0$ we get the image $L_3L=\{1,2,3\}^*0\{1,2\}^*0\Sigma^*$ 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 sub square 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 $L_w=\{x\in\Sigma \mid wx\in L\}$. 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.

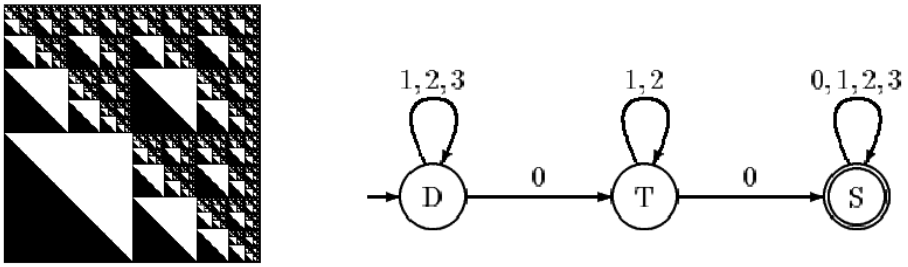

Figure 5. The diminishing triangles defined by {1,2}*0Σ*, and the corresponding automaton.

We have just shown that a necessary condition for black and white multi-resolution image to be represented by a regular set, is that is has only a finite number of different sub images in all the sub squares with addresses from Σ*. 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 by 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 multi-resolution 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 u_x.

1. $i=j=0$.
2. Create state 0 and assign $u_0=I$.
3. Assume $u_i=I_w$. Process state i, that is for k=0,1,2,3 do:
   If $I_{wk}=u_q$ for some atate q, then create an edge labeled k from state i to state q;
   otherwise assign $j=j+1$, $u_j=I_{wk}$,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 algorithm for gray-scale image is based on this procedure, but it will use valuated finite automata (as like WFA) introduced in the section 4 and only replacing black and white color to 256 color (or grayness) image and no creating loop.

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 an

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 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.

3 Interest points

Follows section is based on [4]. The system for image capturing produce mostly the images that are represented discretely by matrices of value. Each element in the matrix expresses either the brightness or the intensities of the color components at the corresponding image point (pixel). In order to capture the image precisely, many pixels are usually used, which yields high volumes of data. If an image is to be analyzed, it is often difficult or even impossible to use all this information directly. Many systems work in such a way that they divide the process of analyzing the image into two steps. In the first step, the important features in the images are found in a rather „mechanical“ way. The features are then used for analyzing the image in the second step. (Let us recall the well-known fact that this scheme need not be accepted without exceptions. The desired result of the first step may depend on the image content which, however, is not known at the time when the first step is carried out.)

The first step, in which the important features are found (usually without deeper understanding the content of the image), has grown into a large and important field in digital image processing that includes finding areas, edges, and corners. Great attention was paid to solve the mentioned problems in the past. Despite this effort, the research in the are does not seem to be closed. If more effective and especially more reliable solutions were available, it could improve the overall performance of the whole systems.

This part of work focuses on the problem of detecting the corners (points of interest, feature points, junction points, dominant points). By the term *corner*, we mean the point at which the direction of the noundary of object changes abruptly. The object is a continuos image area with a constant (or nearly constant) brightness or color. Alternatively, we could say that the corner is an intersection point between two or more edge segments. Let us use Figs. 6-10 to illustrate the term more clearly. Fig. 6 depicts a very simple image containing several objects with the corners indicated in the figure. Fig. 7 shows an example of a typical shape of function of brightness in the neighborhood of a corner. A more complicated brightness function is depicted in Fig. 8. Fig. 9 and 10 show an artificial and real image, respectively, with the corners indicated in them. The simplest possible type of corner depicted in Fig. 7 is called the L-cornel. The image may also contain more complicated corners referee to as T, Y, and X-corners. The corner depicted in Fig. 8, for example, is T-corner. Various types of corners are depicted in Fig. 11. We remark that some authors use the term corner only for the points at which two edges intersect. They then use the term j*unction* or *vertex* for more complicated situations. To be concise, we use the term corner in all that cases.

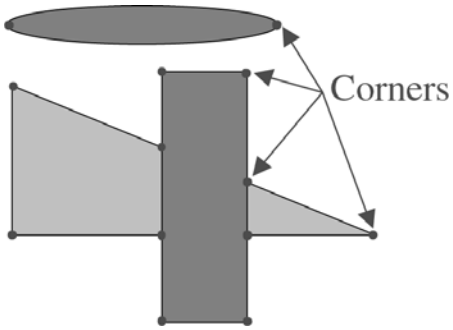

Figure 6. A simple image counting objects (gray areas). The boundaries (solid lines) and corners (small circles) are indicated in the image.

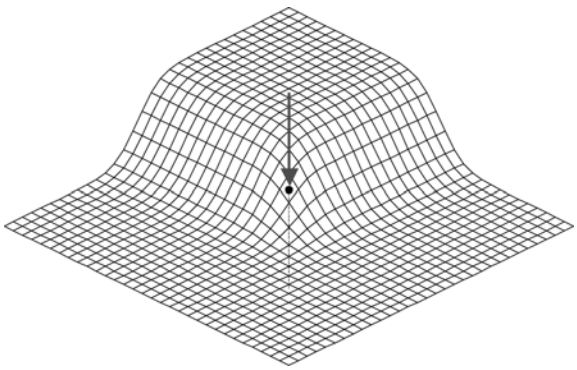

Figure 7. A typical shape of the surface that is defined by the function of brightness in the neighborhood of corner (L-corner). The arrow indicates the theoretical corner point.

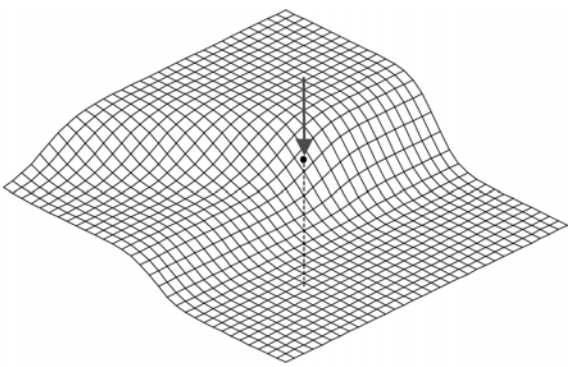

Figure 8. A more complicated surface of brightness (T-corner).

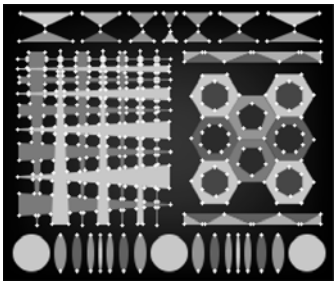

Figure 9. Detected corners (white crosses) in an artificial image.

Figure 10. Detected corners in an image obtained from a CCD camera.

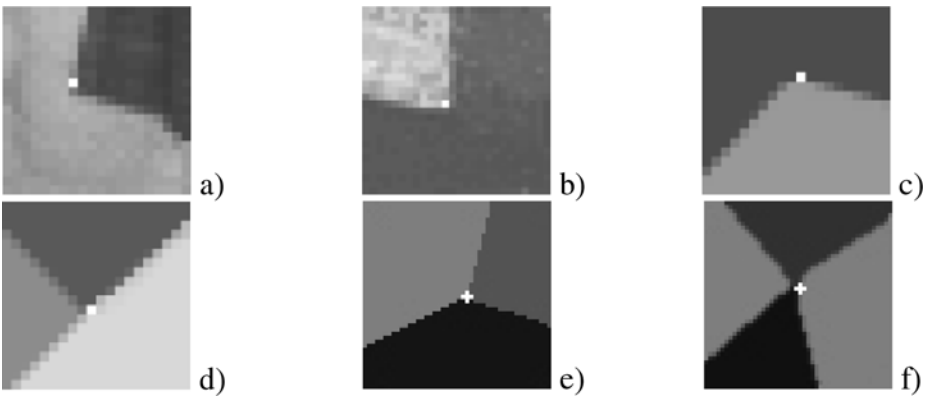

Figure 11. How the corners manifest themselves in real images. L-corners(a,b,c), T-corner (d), Y-corner (e), X-corner (f).

Since the corners convey rich information for many applications in digital image processing and computer vision, the problem of detecting the corners is recognized and well known. Corner detection is often an important step in various image-understanding and scene-reconstructing systems, in which objects are to be detected, tracked, recognized and reconstructed. Some authors claim that the corners play the important role in human perception, which seems probable. It also explain why the use of corner detection may be a logical step in artificial systems too.

If a theoretical analysis is to be carried out, a certain mathematical model of corner is usually required. We will show a widely used model of the L-corner. Let $\psi(\xi)$ be the unit step function defined as follows

$$\psi(\xi) = \left(1 \text{ if } \xi \geq 0, 0 \text{ otherwise} \right) \quad (\mathbf{3.1})$$

Consider an L-corner that is created as an intersection of two non-collinear straight edges. Let φ_1, $\varphi_2 \in <0,2\pi)$ be the directions perpendicular to the edges oriented to the side with higher brightness. We set $n_i=(\cos\varphi_i, \sin\varphi_i)$, i=1,2. Consider the image containing a single convex corner at a point *C*, The brightness function, denoted by *b(X)*, in such an image can be described by the following equations (the term $n_i(X-C)$ expresses the signed distance of *X* from the *i*-th edge)

$$\begin{aligned} b_0(X) &= \psi(n_1 \,.\, (X - C))\, \psi(n_2 \,.\, (X - C)) \,, \\ b(X) &= G(X) * b_0(X) \end{aligned} \quad (\mathbf{3.2})$$

where * means the convolution, * denotes the dot product, and $G(X)$ stands for the two dimensional Gaussian filter.

The corner depicted in Fig. 7 was generated by making use of Eq- (3.1) too. For more complex corners (T, Y, X-corners) the corresponding models can also be introduced [4]. These, however, will not be necessary in this work.

Detecting the corners reliably and effectively in real images that are processed in practice is a difficult problem (Fig. 10). Although many detectors usually work well on simple test images, all the existing algorithm have problem in practical applications if more complicated images are to be processed.

For more information about corners detection, and much more see [5].

4 Image recognition

Our algorithm is based on algorithm shown in section 2. From every node graph lead maximum 4 edges, which they are evaluation numbers of represented image part. At every node is storage information of average grayness in sub square represented thereby state.

Procedure *Construct Automaton for Recognition*
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 u_x.

1. $i=j=0$.
2. Create state 0 and assign $u_0=I$. (Image represented by empty word) and define average grayness of image I.
3. Assume $u_i=I_w$. Process state *i*, that is for k=0,1,2,3 do:
   If $I_{wk}=u_q$ (with small error) or if the image I_{wk} can be expressed as a part or expanded part of the image u_q for some state *q*, then create an edge labeled *k* from state *i* to state *q*;
   otherwise assign $j=j+1$, $u_j=I_{wk}$,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 for Recognition* terminates if there exists an automaton that perfectly (or with small defined error) specifies the given image and produces a deterministic automaton with the minimal number of states. The number of state can be small reduced, or extended by changing error or do tolerance for average grayness of image part. For reconstruct image from automata and compute a interesting point for image recognition we propose follow recursively algorithm.

Procedure *Reconstruct Image for Recognition*
For given automata *A*, we make image I_w the zoomed part of *I* in the square addressed *w*. The image represented by state number *x* is denoted by u_x.

1. Assign the initial state q_o to the image represented by the empty word, that is, to the whole image I, and define $i(q_o)=1$, $t(q_o)=\varnothing(\varepsilon)$, the average grayness of the image I, which we change to computed color, if we wont that.
2. Recursively, for a state *q* assign to square specified by a string *u*, consider four sub square specified by a string *u0, u1, u2, u3*. Denote the image in square by I_{ua}. If the image is everywhere $t(q_o)$ and word has shorter then requested, assign a new input state *q* that representative image specified by a input word *uX* where *X* is denoted part of image. Otherwise, assign a new input state $q(uY)$ where *Y* is a next part of image.
3. Repeat step 3 for each state, and stop if no founded new input state, or input word is a equal to requested.

The procedure *Reconstruct Image for Recognition* was stop for every automata computed by a procedure *Construct Automaton for Recognition*, or other similar algorithm.

With previous procedure can mark the interest point for recognition. There is many method for reflecting point. For example on Fig. 12 (for this and another example on left is original image and on right-hand computed image) is on the left image where lighter color is for state, that construct later (part has longest word) on

the right is lighter color for the state with less sub square (white color is for state, has not sub square).

Figure 12. Reconstructed image on the left-hand with marked later state, on the right-hand with state, has not sub square.

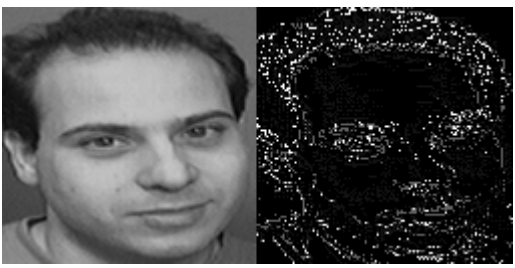

Figure 13. Reconstructed image with marked state.

On Fig. 13 is reconstructed image from automata which have deep 5 and compression is not loss. Number of state is 3317 and 43 latest has not a sub square. Lighter point on right-hand part is marked state without neighbor with same over squere. Some of this point is correspond with interest point (corners) on right part Fig. 10. Finally on Fig. 14 is decompressed image with error 16%, only 16 level of gray and computed automata have 175 state (latest 8 have not sub square). White dot on middle part is state without sub square. On right-hand is marked corners (red dot) computed by method described in [5], many of this point correspond with automata state. For compare on Fig. 15 is all state marked, where darkness color is newer state and white color have latest state.

Figure 14. Reconstructed image with marked state.

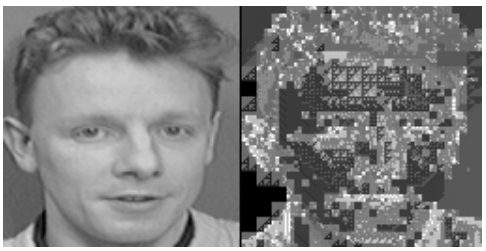

Figure 15. States of computed automata represented by color (on right-hand).

5 Conclusions

In this paper we have proposed an alternative solution for image recognition and finding a interesting points as a corners. This method is based on finite automata compression. The interesting property of this approach is an ability of similarity recognition.

References

1. K. Culik II and V. Valenta. Finite automata based compression of bi-level and Simple Color Images.
2. K. Culik II and J. Kari. Image compression Using Weighted Finite Automata, in *Fractal Image Compression: Theory a Techniques*, Ed. Yuval Fisher, Springer Verlag, pp 243-258 (1994)
3. R. Deriche and G.Giraudon, A computational approach for corner and vertex detection, *International Jurnal of Computer Vision*, 10(2), 101-124 (1993)
4. J.E.Hopcroft and J.D.Ullman. *Introduction to automata theory, languages and computation*. Addison-Wesley (1979).
5. E. Sojka, *A New Algorithm for Direct Corner Detection in Digital Images*, VŠB-Technical University of Ostrava, Faculty of Electrical Engineering and Computer Science, (2002)