IFIG
RESEARCH
REPORT

INSTITUT FÜR INFORMATIK

STRING TRANSFORMATION FOR n-DIMENSIONAL IMAGE COMPRESSION

Martin Kutrib   Jan-Thomas Löwe

IFIG RESEARCH REPORT 0203
MAY 2002

Institut für Informatik
JLU Gießen
Arndtstraße 2
D-35392 Giessen, Germany
Tel: +49-641-99-32141
Fax: +49-641-99-32149
mail@informatik.uni-giessen.de
www.informatik.uni-giessen.de

JUSTUS-LIEBIG-
UNIVERSITÄT
GIESSEN

IFIG Research Report
IFIG Research Report 0203, May 2002

String Transformation for n-dimensional Image Compression

Martin Kutrib[1] Jan-Thomas Löwe[2]

Institute of Informatics, University of Giessen
Arndtstr. 2, D-35392 Giessen, Germany

Abstract. Image compression and manipulation by weighted finite automata exploit similarities in the images in order to obtain notable compression ratios and manipulation tools. The investigations are often based on two-dimensional images. A natural extension is to consider three- or even n-dimensional images which are decomposed in two-dimensional slices, e.g. data produced by tomography. By applying the two-dimensional methods to the slices the volume similarities may be disregarded. Building three-dimensional patterns by merging sequenced images of movie scenes may result in increased similarities. Here we consider transformations of the input strings for weighted finite automata in order to obtain dimension transformations which preserve multidimensional similarities. We focus our investigations on the state complexity and show that a noticeable reduction of the number of states can be achieved.

CR Subject Classification (1998): E.4, F.1, I.4.2, I.5.3

[1]E-mail: kutrib@informatik.uni-giessen.de
[2]E-mail: loewe@informatik.uni-giessen.de

1 Introduction

In computing systems images are usually described by a sequence of color values. With the advance of the information age the need for mass information storage and retrieval grows. Although the capacity of commercial storage devices increases every day, it has not kept pace with the proliferation of image data. Image compression plays an important role in reducing the amount of memory to the essential. But besides the space reduction it is also useful to reduce the time needed to transmit images. Most images contain some amount of redundancy which can be removed by compression algorithms. But this does not lead to high compression ratios in general. Some information are not visible for the human eye and can likewise be removed. Popular compression methods [3] perform a Fourier or a cosines transformation and eliminate high frequency components.

Čulik and Kari [7] used a theoretical approach to image compression. Since points of images in 2-dimensional space are specified by two coordinates, they can be expressed by strings over an arbitrary alphabet [1, 2, 4, 9]. Images or patterns can be interpreted as sets of points. Using an n-letter alphabet patterns are sets of strings i.e. patterns can been seen as formal languages. Choosing a reasonable coding of points it is possible to construct finite state machines to generate (or accept) such patterns. According to the computational result, i.e. either a final or non-final state after successfully reading the input string, the corresponding point will be set black resp. white. For gray-scale images it is useful to consider weighted finite state machines [6] which do not accept input coordinates but compute a real value specifying the gray-scale value.

Other approaches use Levenstein dissimilarity to represent noisy patterns as strings of regular languages and it is also possible to use context-free grammars to represent 2D images [8].

A lot of graphical data consist of three-dimensional information. In some cases it is not sufficient to regard the surface of objects. Take a look at computer tomography. Computer tomography is an imaging process permitting insights into to interior of a spatial body. The resulting images are a sequence of the layers of the object. Although they represent a three-dimensional object they are treated as two-dimensional and will be compressed as is by ordinary compression algorithms. Since these three-dimensional objects obviously contain three-dimensional similarities the redundancy will be destroyed by splitting them into layered images. Even movie sequences can be seen as three-dimensional images when they are put together. Since the distance between two pictures of such a sequence is normally low there exists some kind of three-dimensional similarities. The next step is to extend the well-known finite state machine methods to three or more dimensions. This can be done by choosing other representations of points.

In the present paper we show transformations of n-dimensional data to two dimensional patterns in order to preserve multidimensional similarities. The resulting patterns can be compressed by the ordinary two-dimensional methods. We will show applications where the size of the corresponding automaton noticeable shrinks. Further work has to be done to find a characterization of multidimensional images for which the presented approach works efficiently.

The relationship between patterns and automata are recalled in Section 2. Section 3 introduces the string transformation methods and gives information about the evolution of the number of states of the corresponding automaton. In order to perform a transformation it is required that the strings resp. the underlying alphabet are compatible. Section 4 shows how we are always able to satisfy this condition by enlarging the pattern if the original alphabets are not compatible.

2 Patterns and their automata

We denote the integers by $\mathbb{Z}$, the positive integers $\{1, 2, \ldots\}$ by $\mathbb{N}$, the set $\mathbb{N} \cup \{0\}$ by $\mathbb{N}_0$ and the real numbers by $\mathbb{R}$. The set of strings of length n built from symbols from a set A is denoted by A^n, the set of all such finite strings by A^*. ε is the *empty string*. A^+ is defined to be $A^* \setminus \{\varepsilon\}$. In the sequel we call a finite set *alphabet*. We use $\subseteq$ for inclusion and $\subset$ if the inclusion is strict.

Patterns consist of sets of n coordinates in n-dimensional space. In order to apply formal language and automata theory methods we assign each point a string over an alphabet A as follows. The whole pattern is addressed by ε.

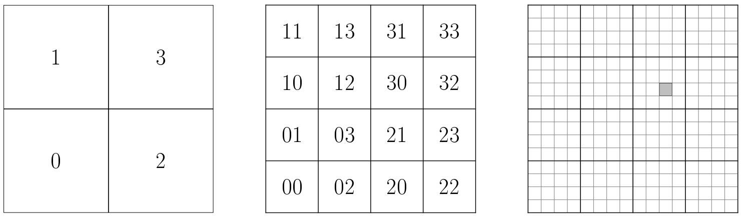

Figure 1: Addressing points by $\{0, 1, 2, 3\}$, the marked point is addressed by 3021

For each letter $a \in A$ we address a (connected) sub-pattern such that each sub-pattern has same size. Every sub-pattern itself will be divided in additional $|A|$ sub-pattern and so on. It is common practice [2, 7, 9] to use 2-dimensional patterns of size $2^m \times 2^m$, to divide them into 4 sub-squares at a time and to use the alphabet $\{0, 1, 2, 3\}$ for coding the coordinates of points. (This is also knwon as *quadtree representation*). Note that it is possible to use any arbitrary alphabet at any size. Although in some cases the visual interpretation may be hard for human eyes, it can be a suitable choice for machines.

Using this method we can formalize patterns as functions.

Definition 1 *A* gray-scale pattern *is a mapping* $p_n : A^n \to \mathbb{R}$, *a* multi-resolution gray-scale pattern *is a mapping* $p : A^* \to \mathbb{R}$. *If* $p(w) \in \{0, 1\}$ *for all* $w \in A^*$, *resp.* $p_n(w) \in \{0, 1\}$ *for all* $w \in A^n$ *the pattern is called* binary.

Binary patterns can be described using finite state machines. We require the machine to accept an input string (which is actually a representation of a point over an alphabet) if and only if the corresponding point is black. Since there exist minimization algorithms we can construct the smallest finite state machine needed to accept the pattern.

The decoding of a given finite state machine is done by generating all input strings resp. coordinates of a length n and checking if the machine accepts or rejects. According to the result the point can be set black or white.

Multi-resolution patterns contain patterns of different resolutions. We usually require the resolution levels to be conform which means that the patterns should be *average preserving*. These patterns can be described by average preserving *weighted finite state machines* [6].

A multi-resolution gray-scale pattern p is called *average preserving* if $p(w) = \frac{1}{|A|} \sum_{a \in A} p(wa)$.

We recall briefly the notion of weighted finite state machines. Pioneering work and theoretical studies have been done, e.g. in [5, 6].

In contrast to the well-known standard finite state machine the weighted finite state machine is equipped with additional *weights* which are assigned both the states and the transitions. During a computation these weights are multiplied for each path and summed for all possible paths according to the current input string. The sum is built using a linear combination according to the initial weights of the states. Thus the machine outputs a value instead of entering a final or non-final state.

Definition 2 *A* weighted finite state machine *is a 5-tuple* $A = (S, \Sigma, f, i, t)$, *where*

1. S *is a finite set of* states,
2. Σ *is a finite* alphabet,
3. $f : S \times \Sigma \times S \to \mathbb{R}$ *is the* weight function,
4. $i : S \to \mathbb{R}$ *is the* initial distribution,
5. $t : S \to \mathbb{R}$ *is the* final distribution.

The distribution function $\delta : S \times \Sigma^* \to \mathbb{R}$ is defined as follows:

1. $\delta(s, \varepsilon) = t(s)$ for all $s \in S$.
2. $\delta(s, aw) = \sum_{q \in S} f(s, a, q) \cdot \delta(q, w)$ for all $s \in S, a \in \Sigma, w \in \Sigma^*$.

The weighted finite state machines defines the function $\phi_A : \Sigma^* \to \mathbb{R}$ specified by $\phi_A(w) = \sum_{s \in S} i(s)\delta(s, w)$.

2.1 Coding

For a given pattern $p : A^* \to \mathbb{R}$ it is possible to construct the corresponding automaton (provided such an automaton exists) as follows: Create a state 0 – this represents the whole pattern. For each sub-pattern which is addressed by $a \in A$ check whether there already exists a state representing such pattern. For gray-scale pattern also check whether there exists a linear combination of existing sub-patterns that can be used to create the current sub-pattern. If `yes` create an edge labeled a to the existing state (for gray-scale add edges that are weighted corresponding to the scalars of the linear combination). If `no` create a new state.

2.2 Decoding

For each input string of a given length check whether the automaton reaches a final state. For gray-scale pattern compute the function ϕ_A for all input strings and assign the value to the corresponding point. In order to completely decode an image one obviously needs additional information about the size and dimension.

3 Transformation

In this section we will discuss transformation of patterns with regard to their underlying alphabet. Practical interests can be transforming patterns from three dimensions to two dimensions and vice versa in order to reduce the amount of states of the corresponding automaton or to allow us to reuse the well-known two-dimensional algorithms. E.g. a representation of points of a three-dimensional pattern of size $2^m \times 2^m \times 2^m$ can be given by strings over the alphabet $\{0, 1, 2, 3, 4, 5, 6, 7\}$. Divisions in sub-quadrants can now be regarded as divisions in sub-cubes.

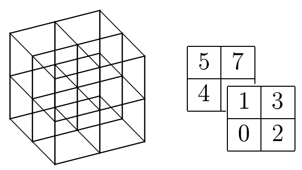

Figure 2: Addressing points in three dimensional space

Given a string of length n over an alphabet which expresses a point we can choose another representation of the same point. A *transformation* is a one to one mapping of strings over one alphabet to another.

Definition 3 *Let A, B be two alphabets. A one to one mapping $\tau : A^k \to B^l$ is called* string transformation. *As usual we extend τ to a mapping $(A^k)^* \to (B^l)^*$ which maps strings to strings as follows:*

$$\begin{aligned} \tau(\varepsilon) &= \varepsilon \\ \tau(uw) &= \tau(u)\tau(w) \text{ for all } u \in A^k, w \in (A^k)^* \end{aligned}$$

Such mapping is only possible if the sizes of the underlying alphabets are compatible, i.e. for two alphabets A, B there must exist integers k, l such that $|A|^k = |B|^l$. Otherwise it is not possible to find a pattern transformation. In Section 4 we will demonstrate how to enlarge the alphabet if such integers do not exist.

A pattern transformation defined in Definition 3 transforms coordinates of a given length k to length l and therefore it is only possible to define the grayscale values for input strings of length $r \cdot l, r \in \mathbb{N}$ directly. Since it should be possible to compute the grayness values of any size we have to define reasonable values, i.e. we choose the average value which is recursively defined.

Definition 4 *Let A, B be two alphabets and $p : A^* \to \mathbb{R}$ be a multi-resolution pattern. If there exist $k, l \in \mathbb{N}$ such that $|A|^k = |B|^l$, then let $\tau : (A^k)^* \to (B^l)^*$ be a string transformation. The pattern $p' : B^* \to \mathbb{R}$ where*

$$p'(v) = \begin{cases} p(\tau^{-1}(v)) & \text{if } v = \tau(w), w \in (A^k)^* \\ \frac{1}{|B|}\sum_{b \in B} p'(vb) & \text{otherwise} \end{cases}$$

is the transformed multi-resolution pattern.

We show that this transformation preserves the average preserving property.

Theorem 5 *Let $p : A^* \to \mathbb{R}$ be an average preserving multi-resolution grayscale pattern. The transformed multi-resolution pattern $p' : B^* \to \mathbb{R}$ is also average preserving.*

Proof. Let k, l be the constants of Definition 4 and τ be the pattern transformation. For all $w \in (B^l)^*$ it holds

$$\begin{aligned} \frac{1}{|B|^l}\sum_{b \in B^l} p'(wb) &= \frac{1}{|B|^l}\sum_{b \in B^l} p(\tau^{-1}(w)\tau^{-1}(b)) = \frac{1}{|A|^k}\sum_{x \in A^k} p(\tau^{-1}(w)x) \\ &= p(\tau^{-1}(w)) = p'(w) \end{aligned}$$

The pattern p is average preserving for all levels in between transformed levels by Definition 4. It remains to show that the average preserving levels fit together.

$$\begin{aligned} \frac{1}{|B|}\sum_{b_1 \in B} p'(wb_1) &= \frac{1}{|B|}\sum_{b_1 \in B}\frac{1}{|B|}\sum_{b_2 \in B} p'(wb_1b_2) = \frac{1}{|B|^2}\sum_{b_{1,2} \in B^2} p'(wb_{1,2}) \\ &= \ldots = \frac{1}{|B|^l}\sum_{b_{1,2,\ldots,l} \in B^l} p'(wb_{1,2,\ldots,l}) = p'(w) \end{aligned}$$

□

3.1 Application

Consider the following three dimensional object. The left figure shows a wedge in its three dimensional representation. For a better visualisation it has been rotated 90 degrees clockwise. If we choose the alphabet $A = \{0,1,2,3,4,5,6,7\}$ we can express it by the set $L = \{w \mid w \in \{1,3,4,6\}^*\{0,2\}A^*\}$. Thus, using finite state machine method it can be expressed by a finite state machine with 3 states.

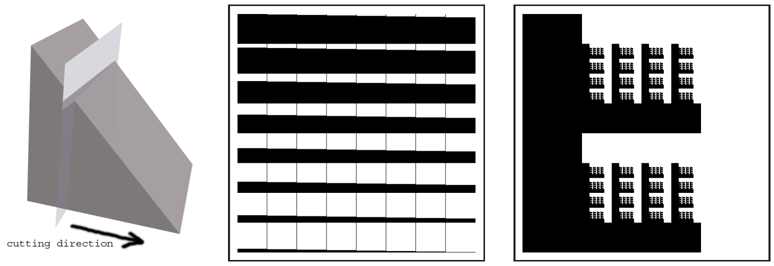

Figure 3: **(a)** Wedge **(b)** Layers **(c)** Transformation

The middle figure is a layer-wise (front to back) representation of the wedge in its original orientation. The sequence of single patterns is put together from left to right such that it fits a squared overall pattern. An additional grid has been inserted for a better understanding. This allows us to compress the two dimensional pattern (without grid) as before. It results in a finite state machine with 85 states(!). Remember that this is a usual procedure in tomography.

The last pattern has been generated using the following transformation from the alphabet A to the alphabet $B = \{0,1,2,3\}$. We use the binary representation for the numbers of each alphabet, i.e.

$$36 = \underbrace{011}_{3}\underbrace{110}_{6} \Leftrightarrow \overbrace{01}^{1}\overbrace{11}^{3}\overbrace{10}^{2} = 132$$

In this case the two dimensional pattern can be expressed by a finite state machine over the alphabet B with only 6 states. The transformation preserves self-similarities and redundancy. If we consider multi-resolution pattern, i.e. wedges at any size the finite state machine for the three dimensional case and for the transformed pattern would keep their sizes. The layered image instead will always increase the number of states of the finite state machine and grow to infinity.

3.2 Increment and decrement of the state complexity

We will examine the evolution of the number of states of the automaton during such a transformation. At first we count the maximal number of states needed to generate a given pattern of fixed size by finite automata.

Let A be an alphabet and $n \in \mathbb{N}$. We denote the smallest integer $v \in \mathbb{N}$ such that

$$|A|^v \geq 2^{(|A|^{n-v})}$$

by $v_{A,n}$.

Note that for any alphabet A with at least two elements there exists such an integer v since the equation holds trivially for $v = n$.

Theorem 6 *Let $p_n : A^n \rightarrow \{0,1\}$ be a binary pattern and F be a finite state machine that accepts p_n. If F is minimized then its number s of states is bounded by*

$$s \leq \sum_{i=0}^{v_{A,n}-1} |A|^i + \sum_{i=v_{A,n}}^{n} (2^{|A|^{n-i}} - 1) + 1$$

Proof. Consider the worst-case for the generation algorithms of a finite state machine. Each state has $|A|$ different successor states, i.e. after reading k input symbols the machine could possibly reach $|A|^k$ different states.

On the other hand there are at most 2^m different binary patterns of size m points. In our case there are at most $2^{(|A|^k)}$ different patterns of size $|A|^k$.

Each state represents a sub-pattern of the original pattern. There are no two states representing the same sub-pattern since the machine is a minimal one. So there exists a branching point where the number of states is higher than the number of distinguishable patterns. This point is reached if

$$|A|^v \geq 2^{(|A|^{n-v})}$$

From this time the number of additional states will shrink ($2^{|A|^k} \rightarrow 2^{|A|^{k-1}}$).

A special case is the empty pattern, i.e. all white points. Since these patterns have to be *rejected* by the machine they are equivalent to a failure state and can be combined to one state. $\square$

Now we are going to explain that the maximal number of states is in fact necessary in the worst case, i.e. the following pattern $p_8 : \{0,1,2,3\}^8 \rightarrow \{0,1\}$ results in a finite state machine with 5478 states. It has been constructed using the 16 distinguishable patterns of size 2×2. These patterns can be used to generate 65536 different patterns of size 4×4. 4096 arbitrary different patterns have been chosen to create the pattern. The number of states can be calculated as

$$\overbrace{1 + 4 + 16 + 64 + 256 + 1024 + 4096}^{\text{branching}} + \underbrace{15 + 1}_{\text{diff. patt.}} + 1$$

The next theorem demonstrates the power of pattern transformation. Given an arbitrary pattern and an arbitrary alphabet it is always possible to find a transformation that results in small finite state machines.

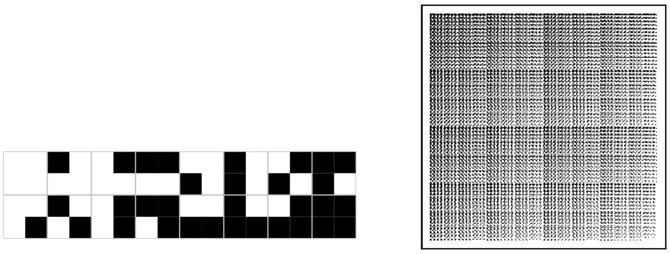

Figure 4: Building blocks and the worst-case pattern

Theorem 7 *Let $p_n : A^n \to \{0,1\}$ be a binary pattern, B be an alphabet which meets the criteria $|A|^n = |B|^l$ for an integer $l \in \mathbb{N}$. Then there exists a string transformation τ and an s-state finite state machine, $s \leq 2l+1$, that accepts the transformed pattern.*

Proof. Consider the set $L = \{w \in A^n \mid p_n(w) = 1\}$ which is the formal language of the pattern p_n. We assign each string of L a number using a one to one mapping $\eta : L \to \{0, \ldots, |L|-1\}$. Any number $\eta(w), w \in L$ can be written as

$$\sum_{i=0}^{l-1} c_i \cdot |B|^i \text{ where } c_i \in \{0, \ldots, |B|-1\}$$

Using a one to one mapping $\zeta : B \to \{0, \ldots, |B|-1\}$ each number can be written as l-tuple of elements of B where $c_i = \zeta(b_i), b_i \in B$. The l-tuples can be used to define strings over the alphabet B and to define L' as the set of all of these strings.

Let $w = w_0 \cdots w_{l-1}$ be the string that represents $|L|$. L' is accepted by the following finite state machine. The states are $\{s_0, \ldots, s_{2l-1}, s_F\}$, starting state s_0, final state s_{2l-1} and transition function $\delta : S \times A \to S$ where

$$\delta(s_i, a) = \begin{cases} s_{i+2} & \text{if } a = w_{\frac{i}{2}} \\ s_{i+1} & \text{if } \zeta(a) < \zeta(w_{\frac{i}{2}}) \end{cases} \quad \text{for even } i$$
$$\delta(s_i, a) = s_{i+2} \text{ for odd } i \leq 2l-3$$

In any other case set $\delta(s_i, a) = s_F$. The basic pattern transformation is defined as $\tau : A^n \to B^l$ where $\tau(w) = \zeta^{-1}(\eta(w))$ □

The following finite state machines accepts all strings w of length 4 that are lower than 1024.

Gray-scale patterns can cause the automaton to branch up to $\frac{n}{2}$ times for a pattern of size $|A|^n$. From this point the existing sub-patterns can be used to build linear combinations, since the number of patterns is larger than the number of points.

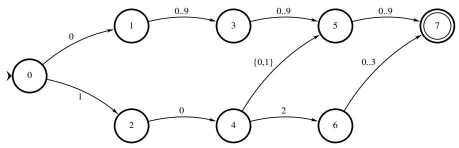

Figure 5: Finite state machine, the failure state s_F is omitted.

4 Enlarging alphabets

In some cases it is not possible to find the constants k, l of Definition 4 to perform a transformation on the alphabet. We will show how to enlarge a given alphabet to preserve the number of states of the corresponding automaton and to keep the average preserving property.

Definition 8 *Let A, B be two alphabets where $A \subset B$. The mapping $\chi : B \to A$ defined by*

$$\chi(a) = \begin{cases} a & \text{for all } a \in A \\ \varepsilon & \text{otherwise} \end{cases}$$

is called characteristic function. *The usual extension to strings $B^* \to A^*$ is defined by*

$$\begin{aligned} \chi(\varepsilon) &= \varepsilon \\ \chi(wu) &= \chi(w)\chi(u) \text{ for all } u \in B, w \in B^* \end{aligned}$$

The embedded pattern is defined such that each additional symbol of the alphabet B is not assigned just the average grayness value but the complete sub-pattern of a lower size. The resulting automaton gets an additional loop weighted 1 at each state. Thus, the number of states will not change when using the automaton generating algorithms.

Definition 9 *Let A, B be two alphabets where $A \subset B$ and $p : A^* \to \mathbb{R}$ be a pattern. The pattern $p' : B^* \to \mathbb{R}$ defined by*

$$p'(w) = p(\chi(w)) \text{ for all } w \in B^*$$

is called embedded pattern of p.

Theorem 10 *Let $p : A^* \to \mathbb{R}$ be an average preserving pattern and $p' : B^* \to \mathbb{R}$ be the embedded pattern of p. Then p' is also average preserving.*

Proof.

$$p'(w) = p(\chi(w)) = \frac{1}{|A|}\sum_{a \in A} p(\chi(w)a) = \frac{1}{|B|}\left(\frac{|B|}{|A|}\sum_{a \in A} p(\chi(w)a)\right)$$

$$
= \frac{1}{|B|} \left(\left(\sum_{a \in A} \underbrace{p(\chi(w)a)}_{=p(\chi(wa)) \text{ for all } a \in A} \right) + (|B| - |A|) \underbrace{\frac{1}{|A|} \sum_{a \in A} p(\chi(w)a)}_{\substack{= p(\chi(w)) = p(\chi(wb)) \\ \text{for all } b \in B \setminus A}} \right)
$$

$$
= \frac{1}{|B|} \left(\sum_{a \in A} p'(wa) + \sum_{b \in B \setminus A} p'(wb) \right) = \frac{1}{|B|} \sum_{b \in B} p'(wb)
$$

□

References

[1] Berstel, J., and Morcrette, M. Compact representation of patterns by finite automata. *Pixim '89, Computer Graphics in Paris, 2nd Annual Conference on Computer Graphics; Selected Papers* (Paris, 1989), pp. 387–401.

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

[3] CCITT SGVIII Joint Photographics Experts Groupt (JPEG), ISO/IEC JTC1/SC2/WG8. *JPEG Technical Specification (Revision 5)*, January 1992. JPEG 8-R5.

[4] Čulik, K., and Dube, S. Affine automata and related techniques for generation of complex images. *Theoretical Computer Science 116* (1993), 373–398.

[5] Čulik, K., and Karhumäki, J. Finite automata computing real functions. *SIAM Journal on Computing 23* (1994), 789–814.

[6] Čulik, K., and Kari, J. Image compression using weighted finite automata. *Computers and Graphics 17* (1993), 305–313.

[7] Čulik, K., and Kari, J. Digital images and formal languages. In *Handbook of Formal Languages*, G. Rozenberg and A. Salomaa, Eds., vol. 3. Springer Verlag, Berlin, 1997, pp. 599–616.

[8] Schlesinger, M. I., and Hlavác, V. *Ten Lectures on Statistical and Structural Pattern Recognition*, vol. 24 of *Computational Imaging and Vision*. Kluwer Academic Publishers, Dordrecht, 2002.

[9] Staiger, L. Quadtrees and the Hausdorff dimension of pictures. *Proceedings of the* 4^{th} *Workshop on Geometrical Problems of Image Processing* (Berlin, 1989), vol. 51 of *Mathematical Research*, Berlin: Akademie-Verlag, pp. 173–178.