Image Storage, Indexing and Recognition with Finite State Automata

Marian Mindek, Michal Burda, *VSB**

Abstract: **In this paper, we introduce the weighted finite state automata (WFA) as a tool for image specification and loss or loss-free compression. We describe how to compute WFA from input images and how the resultant automaton can be used to store images (or to create image database) and to obtain additional interesting information usable for image indexing or recognition. Next, we describe an automata composition technique. We also present a possible way of storing automata in persistent storage. Finally, we depict some tests.**

The benefit of our approach is that some beneficial information for indexing and recognition without knowledge of scene does not need to be computed from compressed images using other algorithms since the resultant WFA already contains it.

Index Terms: **indexing, compression, automata, pre-processing**

I. Introduction

In this paper we introduce a weighted finite state automaton as a tool for image loss or loss-free compression, where a resultant automaton contains some useful information. In [4] and [6] one can found an automata-based technique for simple bi-level images coding. This approach is later generalized for gray-scale images and for simple color images [5]. Finally, [32] proposes wavelet compression approach. This article is based on ideas presented in papers enumerated above. We describe here an approach to image compression and storage based on finite automata. Such way an image database could be created. Such database could be used to obtain additional information from images without any effort since finite state automata approach generates such information implicitly. For simplicity, we do not describe compression or other useful transformations of an automaton (e.g. wavelet or other).

II. Finite Automata Foundations

In this section, we assume the reader to have a fundamental knowledge about automata theory (see e.g. [1]). Basic information about finite automata used as a tool for specifying images could be found in [4], [5] or [22]. To understand the following text we firstly allege the necessary background.

A digitized image of the finite resolution m x n consists of m x n pixels each of which takes a Boolean value for bi-level image, or real value for a gray-scale or color image, where we have 255 tint of source color - R, G, B (24bit color) [5].

Here we will consider square images of resolution 2^n x 2^n. 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 $\mathbf{\Sigma = \{0, 1, 2, 3\}}$ for basic approach and $\Sigma=\{0,1\}$ for offset (read vector) approach, as its address.

Σ's symbols represent the address of a sub-square (The number of four is not a necessary condition, but we use it now for simplicity). A pixel at 2^n x 2^n resolution corresponds to a sub-square of size 2^{n-1} of the unit square. We choose ε as an address of the whole unit square. The four sub-squares in Fig. 1 - 2D 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 is shown in Fig. 1, middle. For simplicity we can address only used sub-square (pixel), with this way we can save more space and machine time in computing with images. This is useful e.g. for large sparse matrixes, BW images or trends.

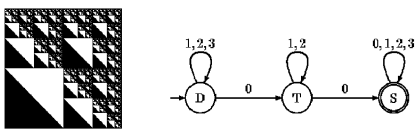
In order to specify a black and white image of resolution $2m$ x $2m$, we need to specify a language $L \subseteq \Sigma^m$.

Frequently, it is useful to consider multi-resolution images, sounds or el. signals, which are parts that is simultaneously specified for all possible resolution (discriminability), 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. [7].

The automaton accepts a word in the input alphabet if there exist labeled path from the initial state to the final state. The set (language accepted by automaton A) is denoted $L(A)$.



Example. By placing the triangle $L=L_1L_2$ from the Fig. 2 into all the squares with addresses $L_3=\{1,2,3\}^*0$ we get the image $L_3=\{1,2,3\}^*0\{1,2\}^*0\Sigma^*$ - diminishing triangles depicted hereinbefore.

* Manuscript received April 6, 2006. M. Mindek., M. Burda., This work was supported by the Department of Computer Science, FEI, VŠB - Technical University of Ostrava, 17. listopadu 15, 708 33, Ostrava-Poruba, CZ, {Marian.Mindek, Michal.Burda}@vsb.cz

Zooming [6] is easily implemented for images represented by regular sets and is very important for image compression shown in next section.

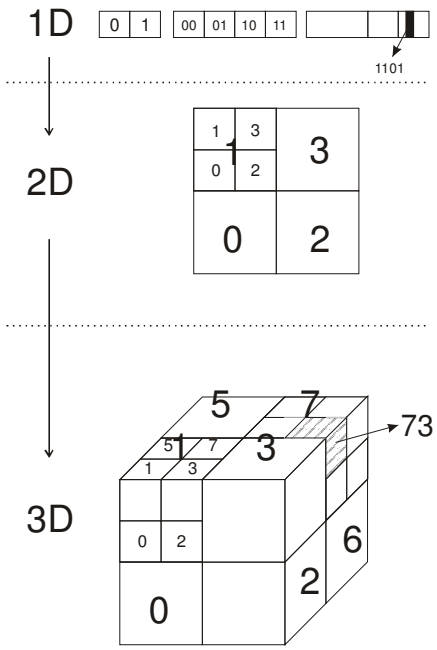


Figure 1. Addressing sub-parts in one and *n*-dimensional space.

Figure 2. The squares specified by {1,2}*0, a triangle defined by {1,2}*0Σ*, and the corresponding automaton.

1) Construction of Finite 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 it must have 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 (or images with many similar parts). Self-similarity is a typical property of fractals. Any image can by approximated by a regular expression however; an approximation with a smaller error might require a larger automaton.

Our algorithm for image compression (approaches for *n*-dimensional) is based on basic procedure for black-and-white images proposed in [4], but it will use evaluated finite automata (like WFA) introduced in [6] and only replacing black and white color to real values. Also be revamped to no possibility to create loops and for adding some option for setup compression and facilitation storage for likely representation.

Now we demonstrate in brief a generalized method for image compression applicable on construction of resultant image storage, or image database with included information. There lead four edges from each node at most (for offset approach lead two edges at most, n-dimensional more edges) and these are labeled with numbers representing image part. Every state can store information of average value of sub-part represented thereby state or other needed information.

The procedure *Construct Automaton* terminates if exists an automaton that perfectly (or with small-defined error) specifies the given image and produces a deterministic automaton with the minimal (interpret as optimal for our problem solution) number of states. The count of states can be reduced a bit or extended by changing error or do tolerance for average values of image part. This principle is naturally useful only for images, where we can obtain image reconstructed with small error (only if we make tolerance, it is loss-compression.)

Changing the part (or only one image element) in source image can change the count of states in resultant automata. We can use certain principle to optimize this algorithm for non-recompress all image. Details are described furthermore in the text.

Procedure *Construct Automaton*

For given image M, we denote M_w the zoomed part of M in the part addressed w, where $w \in \{0,1,2,3...X\}$. For simplicity we use $w \in \{0,1\}$, see figure 1. The image represented by state numbered x is denoted by u_x.

Procedure Construct Automaton
  i = j = 0, create state 0 and assign $u_0 = M$
    (image represented by empty word and define average value of M) an assume $u_i = M_w$
loop for k ∈ {0,1} **do**
 if $M_{wk} = u_q$
  (or with small error, only for loss-compression)
   or if the image M_{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
    else j = j + 1 and $u_j = M_{wk}$
    create an edge labeled k from state i to the
    new state j
  end if ; end for
  if i = = j **than** Stop (all states have been processed)
  else i = i + 1
 end if
end loop
end procedure

It is clear, that procedure for vector or 3D approach is very

similar.The procedure *Reconstruct Image* stop for every automata computed by a procedure *Construct Automaton,* or other similar algorithm, for more information see [23].

Example: Reconstruction with defined error (loss-decompression). Original computed word is $w = \{3203\}$. The length of word is $4 \Rightarrow$ counts of pixels are 256. If we process all words we obtain all image elements, if we process only first 2 symbols from word, we obtain only image with 2^2 pixels. The average value of element (if value is from interval 0-255) with address $w = \{32\}$ is stored in elements, then we obtain image, where sub-square with address $w = \{32\}$ have every elements e.g. equal 16. The procedure *Reconstruct Image* can reconstruct all image elements (same dimension as original) with defined error if we use word with first 2 symbol and append empty word $w=\{32\varepsilon\varepsilon\}$.

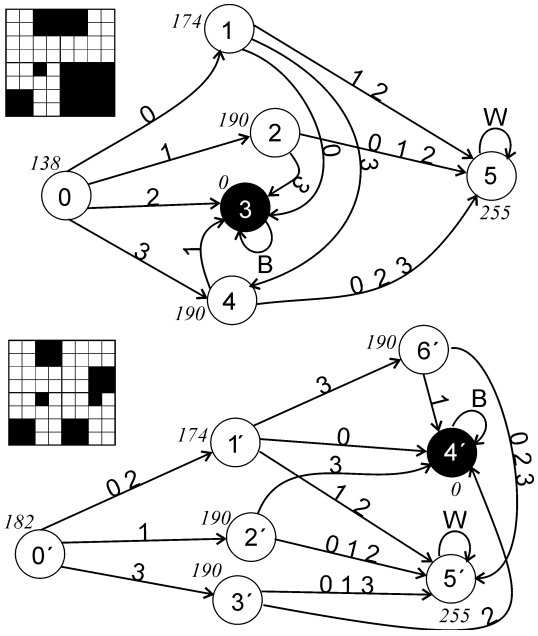


Figure 3. Two sample images with common parts and the corresponding automata.

2) Resultant automata - coalescence

There exist many methods for assembling images represented by finite automata. One of better ones is assembling resultant automata in direction from leaf node to the root. Suppose a couple of images and their representation by means of final state automata computed with procedure 1, depicted in figure 3. Moreover, the states of automata contain also the average grayness of corresponding picture parts. These values are in interval 0 – 255, where 0 represent black and 255 white colors.

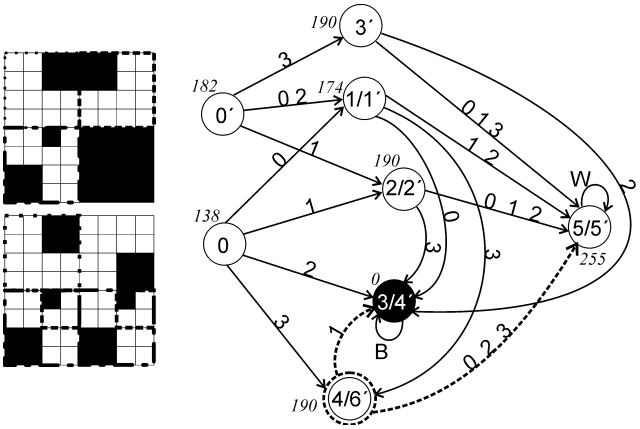


Figure 4. Composite automata from figure 3 respecting the common parts.

Example: It is clear that images in figure 3 have some common parts, highlighted in following figure 4. These common parts could be joined into the same state in composed resultant automata (see right side of figure 4). The corresponding automata with images in figure 2 have some common state and edges labeled with same symbol. There exists some similar or the equal word *w* depicting the way from the root to the corresponding part represented by a state.

This principle can be used for loss-free compression saving images. Additional information can be obtained from structure of resultant automaton, for example the information about the similarity of the stored images, common lines, and alternatively equal parts in images. We can easily get the group of equal images parts.

Resultant automaton is in most cases smaller than original automata and contains interesting information from both automata. In some cases, the resultant automaton could be larger, but if we will store more and more images, the resultant automaton will increase less and less. It is clear, when we are storing images of the same domain (e.g. medical super-sound radiograph, X-ray, satellite images, air photo, etc.) we could obtain an interesting knowledge about stored images hand in hand with saving the memory space and network capacity.

III. INTERESTING INFORMATION

If we store some similar images in one automaton, we can obtain interesting information about changes in resultant automaton. Concretely, if our resultant automaton has more states then the new states represent the differences between input images. With procedure *Construct Automaton,* we can control the type of acceptable (or unacceptable) changes, e.g. trivial differences in value, small noise, etc. With this, we can separate interesting changes from those that are rather to be ignored.

For example, in medicine, we can obtain many similar images and only medical specialist can mark an interesting parts. We can help him or her to reduce the number of non-interesting

changes and of course notice him to some small differences, which can be important, but hard to find.

With procedure *Composition Automaton,* we can increase the ability to make the difference between interesting and non-interesting parts of the resultant automaton. We can even do it by involving the tolerance in value (average value), or similarity of words, etc.

If we store our source matrix in more than one automaton, we can focus on our interesting part of the matrix and there compute the profundity automaton. On other part of matrix, we can compute automaton with less number of states. For this purpose, we can use the pattern matrix shown in table 1, where the values in cells are the counts of profundity of automaton, which represents that part of matrix. This principle can be used only for loss-compression (e.g. images, signals, etc.). The part with less count of states stores much fewer information than the part with more states.

3	3	3	3	3	3
3	5	5	5	5	3
3	5	10	10	5	3
3	5	10	10	5	3
3	5	5	5	5	3
3	3	3	3	3	3

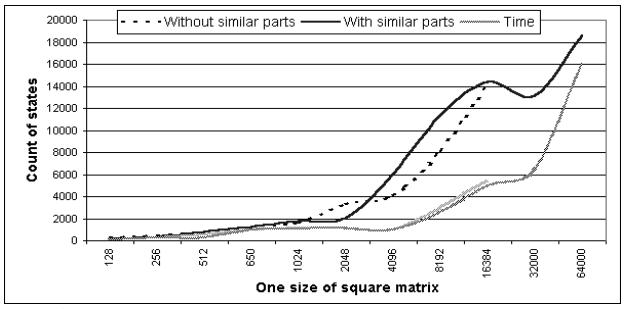
Table 1: Pattern of approach with 36 automata.

It is clear that with this principle we can save much more space and also preserve high information value of our data. We can transfer only interesting part of matrix or any nearest part and save machine time or network capacity. It is sufficient to choose a state from resultant automata, which represents our interesting part of the matrix, and operate with this as with the root. This principle is used with the principle automata composition. Pattern may be arbitrary. Also is possible connect final state of automaton with other one, ore with some other object (e.g. vector image, notations, etc.)

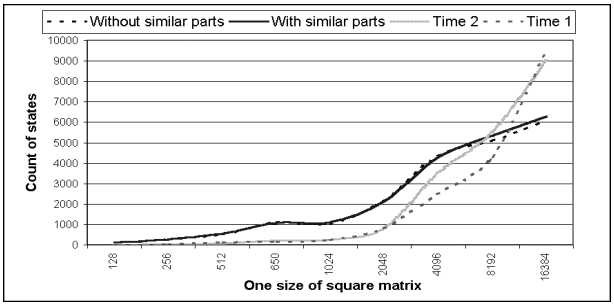
Now we have a background for using finite state automata as a database with included information.

IV. TESTS - EXHIBITS

Every test was carried out on standard PC with Intel Celeron 1,3GHz processor and 512MB RAM. We used two different algorithms for computing resultant automata loss-free compression, but results are very similar, such that we describe only one of the results. Tested data was generated randomly. Some of the test matrixes correspond with the worst test data (for our procedure) for comparison. On graph 1 can be found some disadvantage of our approach. If we want have high compression ratio hand to hand with good informative ability of our automata and high resolution source images, we must compute with more similar parts. But this leads to markedly increase of machine time and counts of states. In this case we can use some compromise usage e.g. comparing only similar sized parts. If we want safe machine time, we can decrease number of transformation or by contrast increase for produce fewer states of resultant automata (more tests in [24]).



Graph 1. Graph of results for offset approach, where count of states is X states plus Y states.



Graph 2. Graph of results for matrix approach.

Graph 2 depicts tests with same source data, but with matrix oriented approach. Is evident, that this approach is a little bit faster for smaller source matrices and produce fewer states. But for very high resolution source image is like as vector approach. The vector approach is much better for bi-level images or large sparse source matrices.

Figure 5. Reconstructed image on the left-hand with marked later state, on the right-hand with state, has not sub-square.

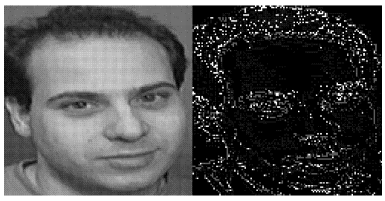


Figure 6. Reconstructed image with marked state on the right.

Figure 7. Reconstructed image with marked state, on the right is marked corners.

On Fig. 5 are depicted samples of resultant automaton with alternative representation. On the left side is source image and corresponding automaton with highlighted states computed later and on the right side is corresponding automaton with highlighted states without descendant i.e. this states represents parts of image without sub-parts.

On Fig. 6 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 neighbors with same over-square. Some of this point is correspond with interest point (corners). Finally on Fig. 7 is decompressed image with error 16%, only 16 level of gray and computed automata have 175 states (latest 8 have not sub-square). White dot on middle part is state without sub-square. On right-hand is marked corners (white dot) computed by classical method [31], many of this point s correspond with automaton states. For compare on Fig. 8 are all states marked, where darkness color is newer states and white color have latest states.

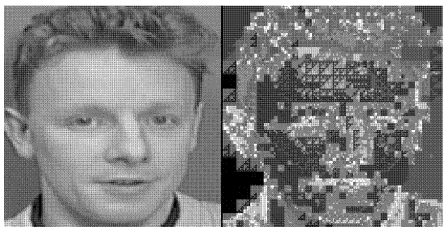


Figure 8. States of computed automata represented by shade.

V. CONCLUSION

In this paper, we have presented a Weighted Finite State Automata (WFA) for image storage and compression. The WFA allows us to capture a large class of images and to obtain interesting information that is implicitly carried by the WFA without any special algorithmic effort. We have shown that data stored in WFA save more space, network capacity, make it possible to access and manipulate images without decompressing process and allow call attention to interesting parts. Storing data in automaton allows us to send via network only a part of data. We do not need to have available the whole image if we are interested in a small part of it only. Additional benefit is included. Composition also allows us to use WFA as a database of images with the ability to simpler search of interesting parts of our data.

In our future work, we focus on manipulating with data in database (at various structures), describe language or a set of tools that is able to query stored data.

REFERENCES

[1] Alur, R. and Dill, D. L. A Theory of Timed Automata. In Theoretical Computer Science, 126(2):183–235, 1994.

[2] A. Lipson, The use of FPGAs in *DNA Pattern Matching. Final Research Report*, October 29, 2001

[3] Burda, M., Mindek, M., Šarmanová, J.: Characteristics of cosymmetric association rules. In *Dateso 2005.* Ed. Karel Richta, Václav Snášel, Jaroslav Pokorný, Ostrava:, 2005, vol. 2005, 20-31, ISBN 80-01-03204-3

[4] K. Culik II and J. Kari. Image compression using weighted finite automata. In *Computers & Craphics*, 17:305–313, 1993.

[5] K. Culik II and V. Valenta. Finite automata based compression of bi-level and simple color images. In *Computers & Craphics*, 21:61–68, 1997.

[6] K. Culik II and J. Kari. Image compression Using Weighted Finite Automata, in Fractal Image Compression. In *Theory a Techniques*, Springer Verlag, pp 243-258, 1994.

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

[8] Martin Kutrib, Jan Thomas Lowe, String Transformation for *n*-dimensional Image Compression, in W.I. Grosky and F. Plasil (Eds.): *SOFSEM 2002, LNCS 2540, pp. 208–217, 2002*. Springer-Verlag Berlin Heidelberg 2002

[9] M. Crochemore, W. Rytter. Text Algorithms. Oxford University Press, 1994.

[10] M. Crochemore, C. Iliopoulos, Ch. Makris, W. Rytter, A. Tsakalidis, K. Tsichlas. Approximate String Matching with Gaps, Nordic Journal of Computing 9, 2002, pp. 54-65.

[11] M. Crochemore, T. Lecroq, Pattern Matching and Text Compression Algorithms, The Computer Science and Engineering Handbook, A.B. Tucker, Jr, ed., CRC Press, Boca Raton, 2003, Chapter 8.

[12] M. Crochemore, Reducing space for index implementation, Theoretical Computer Science, 292-1 2003.

[13] M. Crochemore, W. Rytter, Jewels of Stringology, World Scientific, Hong Kong, 2002

[14] D. Perrin. Finite Automata. Handbook of theoretical Computer Science. Elsevier Science Published 1990.

[15] Galil, Z., Park, K.: Truly alphabet-independent two-dimensional pattern matching. In: *Proceedings of the 33rd IEEE Annual Symposium on Foundations of Computer Science*. (1992) 247–256

[16] J. Holub. Reduced Nondeterministic Finite Automaton for Approximate String Matching. Proceedings of the Prague Stringology Club Workshop 1996.

[17] H. R. Lewis, C. H. Papadimitriou. Elements of the Theory of Computation. Prentice Hall 1981.

[18] B. Melichar, J. Holub. 6D Classification of pattern Matching Problems. Proceedings of the Prague Stringology Club Workshop 1997.

[19] B. Mikolajczek ed. Algebraic and Structual Automata Theory. Norton Holland 1991

[20] M. Mindek, V. Snášel, Algebra of Approximate Pattern Matching Problem, in *Znalosti 2006*

[21] Mindek, M., Hynar, M.: Image database using fnite state automata. In *ISIM '05*. Ed. Jaroslav Zendulka, 2005, 287-294, 80-86840-09-3

[22] Mindek, M.: Finite State Automata and Image Recognition: *DATESO 2004*, pp 132-143 (2004), ISBN: 80-248-0457-3

[23] Mindek, M., Burda, M.: Finite State Automata - a concept for data representation, *SYRoDIS St. - Petersburg, Russia*:, VVM. com. Ltd., 2005. 5-9651-0087-6

[24] Mindek, M., Hynar, M.: Finite State Automata as Data Storage. *Dateso 2005*. Ed. Karel Richta, Václav Snášel, Jaroslav Pokorný, VŠB-TU Ostrava, p. 9-19, 80-01-03204-3

[25] M. Mohri. Marching Patterns of an Automaton. Combinatorial Pattern Matching, 6th Annual Symposium, CPM 95, Espo, Finland, Springer Verlag 1995

[26] P. Mužátko. Approximate Regular Expression Matching. Proceedings of the Prague Stringology Club Workshop 1996.

[27] G. Navarro. A Guided Tour to Approximate String Matching. ACM Computing Surveys, 33(1) 2001

[28] G. Navarro, R. Baeza-Yates. Improving an Algorithm for Approximate String Matching. Algorithmica, 30(4) 2001

[29] Tomas Polca and Boivoj Melicha, in M. Domaratzki et al. (Eds.): *CIAA 2004, LNCS 3317,* pp. 327–328, 2005. Springer-Verlag Berlin Heidelberg 2005.

[30] S.V.Ramasubramanian and Kamala Krithivasan, Finite Automata and Digital Images, IJPRAI, Vol. 14, No. 4, pp. 501-524, 2000.

[31] 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)

[32] Leonardo E. Urbáno Schánchez, Wavelet Image Compression Using Universal Coefficients Matrix Detail Estimation, in CONIELECOMP 2004, 076952074-X/04 - IEEE 2004