Finite State Automata as a Data Storage

Marian Mindek and Martin Hynar

Department of Computer Science, VŠB – Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava–Poruba, Czech Republic
{marian.mindek, martin.hynar}@vsb.cz

Abstract. In this paper, we summarize ideas to use finite automata as a tool for specification and compression of data aggregates (e.g. images, electrical signals, waves, large (sparse) matrixes, etc.). We describe different ways of data access. Then we describe an approach how make a resultant automata with included interesting information, how to focus on interesting information in our data, and how to link together resultant automata.

Keywords: finite automata, compression, large sparse matrix, searching, pattern

1 Introduction

Finite automata is an useful tool for matrix representation of commonly used information resources (e.g. images, texts, sound waves, electrical signals etc.), for their compression and for obtaining interesting information about given data [1,2,4,5,7,8,9].

In our opinion, such technique could be used also to represent large matrixes, which are usually hard to manipulate. A traditional approach (compression using common algorithms) solves only part of the problem. It consumes less space but on the other hand, there is no way to make changes to original matrix. Moreover, there is no way to use another additional information and if it is required it has to be computed using other means (e.g. nearest neighbors of some 1-position, interest points, carrier, base, etc.). With our approach, we can focus at this issue and improve predication capabilities about data. The resultant automaton (or automata) contains this interesting information. We can use it for comparing per pattern or search similar information (e.g. part of faces, medical pictures, buildings tracing, part of large sparse matrixes, similar noise, similar trends, etc.).

If we want to have certain benefit from such advantages and if we want to have some mean to store matrixes in database with included interesting information, we can use the approach of storing resultant automata in some well known structure such as table, matrix or XML.

In the following examples we describe for simplicity our approach on the images, if will not remark alternatively.

K. Richta, V. Snášel, J. Pokorný (Eds.): Dateso 2005, pp. 9–19, ISBN 80-01-03204-3.

2 Data specification

2.1 Finite State Automata (FSA)

Background about automata theory in this chapter is the most necessary. We describe only simple procedure for storing matrixes to automata too. For more about automata theory please read [6] and for more about automata as a tool for specifying image, please read [3,4,5,7].

In order to facilitate the application of FSA to matrix description we will assign each pixel at 2^n x 2^n resolution (2^n for vector) a word of length n over the alphabet $\Sigma=\{0,1,2,3\}$ for basic approach and $\Sigma=\{0,1\}$ for offset (read vector) approach, as its address. Offset (vector) approach is useful for matrix approach too.

Example. The large sparse matrix can by represented as set of coordinates [*x*, *y*] where *x* is a row and *y* is a column. If we separate *x* part and *y* part we obtain two vectors. These vectors can have value as set of positions in matrix, or difference between previous and followed position.

A part at 2^n x 2^n (2^n) resolution corresponds to a sub-part of size 2^{-n} of the unit part. We choose ε as the address of the whole unit part. Single digits as shown in figure 1a; on the left address its quadrants. The four sub-squares of the square with address *w* are addressed *w0*, *w1*, *w2* and *w3*, recursively. Addresses of all the sub-squares (pixels) of resolution 4 x 4 are shown in figure 1, middle. The sub-square (pixel) with address *3203* is shown on the right of figure 1. Clearly for offset (vector) approach is sub-part with address *w* denoted only as *w1* and *w2*, recursively. For comparison see figure 1b, black part of vector has address 1101.

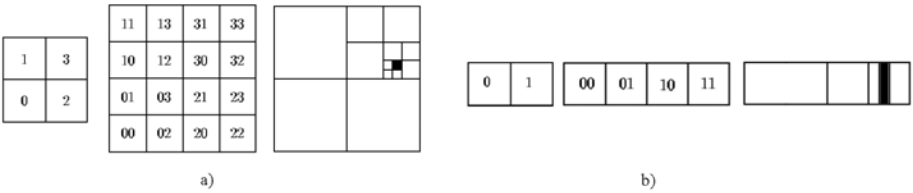


Fig. 1. The addresses of the quadrants, of the sub-square of resolution 4 x 4, and the sub-square specified by the string 3203.

For simplicity, in the following we describe theory only for matrixes. (Offset) vectors approach is very similar.

In order to specify a binary matrix of resolution 2^m x 2^m, we need to specify a language $L \subseteq \Sigma^m$. Frequently, it is useful to consider multi-resolution images, sounds or el. signals simultaneously specified for all possible resolutions (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 binary matrix is specified by a language $L \subseteq \Sigma^*$, $\Sigma=\{0,1,2,3\}$, i.e. the set of addresses of all the evaluated squares.

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

Example. The 2 x 2 chessboards in figure 2 (a) look identically for all resolutions. The multi-resolution specification is the regular set {1,2}Σ*. The 8 x 8 chessboard in figure 2 (b) is described by the regular set Σ^2{1,2}Σ* or by FSA *A* figure 2(c).

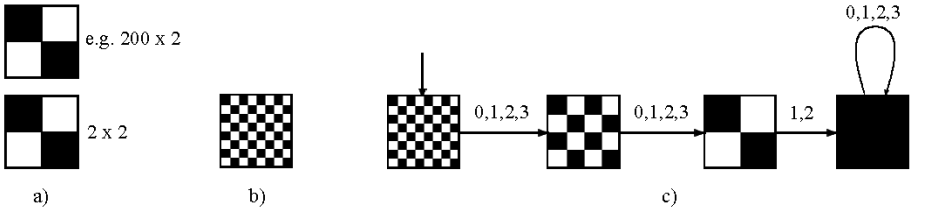


Fig. 2. 2 x 2, 8 x 8 chessboards and corresponding automaton.

Note that here we used the fact that the regular expression Σ^2{1,2}Σ* is the concatenation of two regular expression Σ^2 and {1,2}Σ*.

Example. By placing the triangle $L = L_1L_2$ where L_1= {1,2}*0 and L_2=Σ* into all squares with addresses L_3={1,2,3}*0 we get the image L_3L={1,2,3}*0{1,2}*0Σ* shown at the left of figure 5.

Zooming [3] is easily implemented for matrixes represented by regular sets (automaton) and is very important for loss compression.

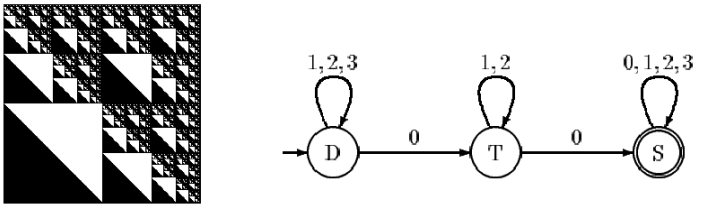


Fig. 3. The diminishing triangles defined by {1,2,3}*0{1,2}*0Σ*, and the corresponding automaton.

We have just shown that a necessary condition for binary matrixes to be represented by a regular set (FSA) is that it must have only a finite number of different sub-matrixes in all the sub-squares with addresses from Σ*. We will show that this condition is also sufficient. Therefore, matrixes that can be perfectly (i.e. with infinite precision for loss compression) described by regular expressions are matrixes of regular or fractal character. Self-similarity is a typical property of fractals. Any matrix can by approximated by a regular expression however; an approximation with a smaller error might require a larger automaton. Multi-resolution (fractal) principle is at most useful for images, sounds, and descriptions of function or electrical signals.

2.2 Basic procedure

Our algorithm for matrix compression (both approaches) is based on basic procedure for black-and-white images proposed in [4], but it will use evaluated finite automata (like WFA) introduced in [3] and only replacing black and white color to real values, without possibility to create loops and adding some option for setup compression and facilitation storage for likely representation.

Example. For the image *diminishing triangles* in figure 3, the procedure constructs the automaton shown at the right-hand side of figure 3. First, the initial state *D* is created and processed. For sub-square with address 0 a new state *T* is created, for addresses 1,2 and 3 create the new states with deep *n* (where deep is length of route from the root and *n* is length of part of word *w* with the same symbol; loop of edge from previous algorithm). Then state *T** is processed for sub-square with address 0 and new state *S* is created, for 1 and 2 a connection to a last of new states. There is no edge labeled 3 coming out of *T* since the quadrant 3 for *T* (triangle) is empty (in binary matrix there is 0 everywhere). Finally, the state *S* (square) is processed by creating edge back to *S* for all four inputs. In this way it represents end of automaton or loop to state itself for multi-resolution approach.

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

The procedure *Construct Automaton for compression* terminates if exists an automaton that perfectly (or with small-defined error) specifies the given matrix 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 matrix part. This principle is naturally useful only for matrixes, where we can obtain matrix reconstructed with small error (only if we make tolerance, it is loss-compression.)

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

Procedure *Construct Automaton for Compression*
For given matrix M (in arbitrary representation e.g. full matrix, difference vector, [x, y] representation, etc.), 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 matrix represented by state numbered x is denoted by u_x.

Procedure *Construct Automaton for Compression*
$i = j = 0$
create state *0* and assign $u_0 = M$ (matrix represented by empty word and define average value of M)
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 matrix M_wk can be expressed as a part or expanded part of
                        the matrix u_q for some state q
                then    create an edge labelled k from state i to state q
                else    j = j + 1
                        u_j = M_wk
                        create an edge labelled 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 (offset) approach is very similar. We do not describe it, but we give some confrontation later.

Procedure for reconstruction matrixes from automaton is very simple, for more information see [9].

2.3 Tests

Every test was carried out on standard PC with Intel Celeron 1,3GHz and 384MB RAM. We used two different algorithms for computing resultant automata without loss 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.

In table 1 there are depicted matrixes and corresponding counts of evaluated elements. The last column contains counts of similar parts of matrixes. Sizes of these parts are between 1/16 - 1/128, for larger matrixes. Maximum range of similar parts is 6 for matrix with 32000x32000 elements. This setup is only for testing, real data are generally more similar but we show tests for worse cases of data. In next comparison, we test matrixes without similar parts. The resultant automata were perfectly (without loss) representing the source matrix.

In table 2 there are depicted results of our tests. In first column there are source matrixes, in second resulting time for procedure with using vector approach and then follow two columns with counts of state of corresponding resultant automata. In last two columns there are times for procedure using full matrix (*Procedure Construct Automaton for Compression* described before) and counts of states of resultant automata.

Table 1. The tested matrixes.

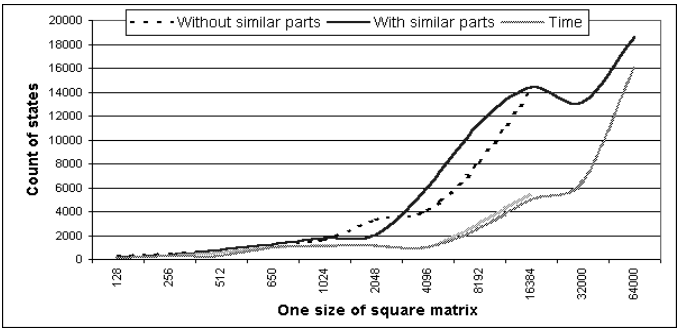
Matrix	Count of elements	Similar parts
128 x 128	67	2
	57	0
256 x 256	128	2
	124	0
512 x 512	266	2
	244	0
650 x 650	515	2
	526	0
1024 x 1024	533	3
	500	0
2048 x 2048	1035	2
	1011	0
4096 x 4096	2058	2
	2022	0
8192 x 8192	4000	3
	4059	0
16000 x 16000	*1*	*0*
16384 x 16384	8193	4
	8000	0
32000 x 32000	9000	6
64000 x 64000	18124	0

Table 2. The results of tests. From the left: source matrixes, time for offset approach in seconds, count of states of resultant automata for X and Y parts, time for classical procedure and counts of states of resultant automata.

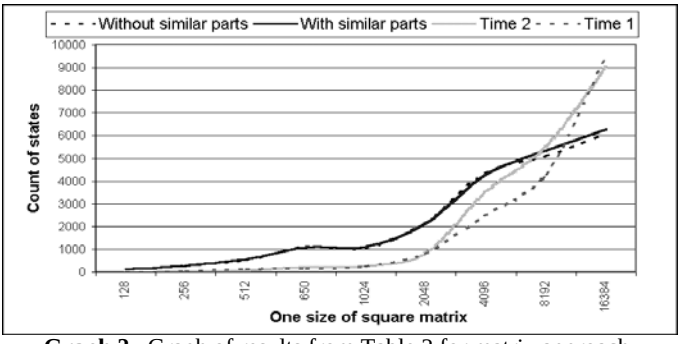
Matrix	time (XY)	X states	Y states	time	states
128 x 128	0,1	72	132	0,1	134
	0,1	111	119	0,2	118
256 x 256	0,3	129	254	0,8	271
	0,3	163	249	0,9	280
512 x 512	0,3	310	516	1,2	552
	0,5	289	491	2,3	516
650 x 650	1	427	813	4,5	1080
	1	419	835	3	1126
1024 x 1024	1,1	740	1027	4,9	1120
	1,2	593	1003	5	1044
2048 x 2048	1,2	1008	1028	16	2161
	1,2	1279	2027	16	2145
4096 x 4096	1	2019	4049	70	4289
	1	2016	2052	50	4353
8192 x 8192	2,6	4010	7350	110	5350
	3	4304	3890	86	5112
16000 x 16000	*0,1*	*13*	*13*	*240*	*14*
16384 x 16384	5	6218	8192	180	6308
	5,5	5905	8100	190	6100
32000 x 32000	6,5	4929	8190	NA	NA
64000 x 64000	16	8111	10450	NA	NA

Highlighted row is the worst case for our solution for matrix approach (Source matrix contains only one element at unlikely position.) It is clear that offset approach is faster for larger matrixes but produces more states. Matrix approach is better for

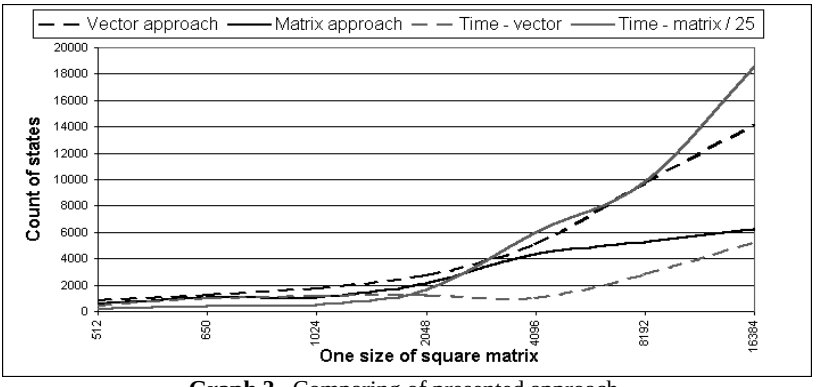
compression of small matrixes. If we want to have more compressed large matrix then the matrix approach is useful too but at the expense of machine time. For lucidity see following graph, where time is only on informative scale.



Graph 1. Graph of results from Table 2 for offset approach, where count of states is X states plus Y states.



Graph 2. Graph of results from Table 2 for matrix approach.



Graph 3. Comparing of presented approach.

2.4 Changes in source

In this section we describe in brief, how to solve the changes in the matrix. If we compress the matrix with traditional algorithm (e.g. zip, LZW, Huffman, etc.) and some element is changed, we must re-compress all matrixes every time. But if we represent matrix as a FSA, we can change/re-compress only the corresponding part of resultant automaton, the one with changed element.

There exist at least three basic solutions for selection of corresponding part of Finite State Automata or corresponding sub-square of source matrix:

1) Re-calculating the biggest corresponding sub-square:
This approach leads to a big quantum of data manipulation (up to one quarter), but with this approach we can reach the high compression ratio. The method is useful for all types of source matrixes.

2) Re-calculating the least corresponding sub-square:
This approach re-calculates the least quantum of data (approximately tens elements), but with this approach we can gain only very small compression ratio and changes often lead to the growth of the automata. Compression become here the disutility, but if we want only obtain the interesting information from our resultant automata, this method is useful too. This approach is useful for all types of source matrixes.

3) Re-calculating the optimal corresponding sub-square:
Retrieval of such sub-square may be difficult, but in most cases, it shows that it has no sense to work with sub-square greater than three or four least corresponding sub-squares. Naturally, it greatly depends on the character of the matrixes. If we know that the matrixes contain many equal blocks, we can state the amount of levels which we should still take in consideration. This choice naturally has not influence on algorithm, but only on machine time and resultant size of compressed matrixes.

3 Resultant aggregate / database

3.1 Resultant automata

Composition is useful for storing resultant FSA in one structure with value-added information. In this section, we describe only necessary generalized procedure, for more information read [9]. This procedure can be used for both approaches; *object oriented* and *prevailing* approach. This approach can be simply upgraded to loss-composition and makes possible to save more space and setup some additional options. We focus on this in future work.

Procedure *Composition Automaton for Storage*
For given automaton A and automaton B (resultant automaton from previously composition) compute new resultant automaton $B' \in A \cup B$ and combine similar parts of both ones.

Procedure Composition Automaton (Automaton *A*, Stored automaton *B*)
Assign state q_x from *A* to the corresponding state in stored automaton *B*.
if such case does not exists, assign a new state and take q_{x+1} from *A*.
end if
for all state of automaton *A* **do**
if not exists edge from state q_i labeled with same word *w* as edge from correspond state in stored automaton **then**
create a new edge labeled *w* to a new state *i*
otherwise
take next edge
end if
if all edges from actual state is processed, take next state
end if
end for
end procedure

This principle can be used for no-loss or loss compression for saving matrixes. Additional information can be obtained from structure of resultant automaton, for example the information about the similarity of the stored matrixes or its parts, common lines, etc. We can also easily get the group of equal matrix parts.

3.2 Focus on a interesting information

If we store the source matrixes in more than one automaton, we can focus on the interesting part of the matrix and compute the automaton with various lengths. This principle was introduced in [10].

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 3, where the values in cells are the counts of profundity of automaton, which represents that part of matrix. This matrix can be used for some image, see figure 4. 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.

It is clear that with this principle we can save much more space preserving high information value of 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 the interesting part of the matrix, and operate with this as with the root. This principle is used in the automata composition. Procedure for focusing on interesting information is very simple. Pattern may be arbitrary.

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

Procedure *Focus on interesting information*
For given matrix *M* and pattern matrix *P* compute resultant automaton. This procedure use procedure *Construct Automaton for Compression (CAfC)*, there in before.

Procedure Focus on interesting information (Matrix *M*, pattern matrix *P*)
    $i = j = 0$
    create state *0* and assign $u_0 = M$
    assume $u_i = M_w$
    loop
        for $x \in S$ (part of pattern matrix)
            assume $|w| = P(u_i)$
            $i = CAfC(M, \text{new } w)$
            j++;
        end for
        if $i = = j$ **than**
            Stop (all parts from pattern are processed)
        else $S = S + 1$
        end if
    end loop
end procedure

Table 3. Example pattern for procedure *Focus on interesting information*

2	2	2	2	2	2	2	2	2	2	...	2
2	2	3	3	3	3	3	3	3	2	...	2
2	2	3	4	4	4	4	4	3	2	...	2
2	2	3	4	5	5	5	4	3	2	...	2
2	2	3	4	5	6	4	4	3	2	...	2
2	2	3	4	5	5	5	4	3	2	...	2
2	2	3	4	4	4	4	4	3	2	...	2
2	2	3	3	3	3	3	3	3	2	...	2
2	2	2	2	2	2	2	2	2	2	...	2

Fig. 4. Example image with used focus on interesting information.

4 Conclusions

In this paper was summarized an idea to use finite state automata as a tool for specification and compression of data aggregates. We compared two basic approaches of computing the resultant automata, namely: the matrix approach and the vector (or offset) approach. The first one is better applicable for representation and compression of smaller matrixes and for manipulation after decompression. The vector approach is pretty faster for larger matrixes, but on the other hand it produces more states. If we want to have large matrix to be more compressed (independently on the machine time) then the matrix approach is useful too. Both methods can be loss or loss-free, and both types have high predicate ability about stored matrixes and save space and network capacity.

The linked resultant automaton is able to create matrix database with included value and to make manipulation with matrixes easier. We also described simple and generalized procedure *focus on interesting information* in the source data and some illustrative results were shown. This procedure can make a better compression ratio together with maintaining the high information level of data.

5 References

1. Alur, R. and Dill, D. L. A Theory of Timed Automata. In Theoretical Computer Science, 126(2):183–235, 1994.
2. Daniela Berardi, Fabio De Rosa, Luca De Santis and Massimo Mecella. Finite State Automata as Conceptual Model for E-Services. In Integrated Design and Process Technology, IDPT- 2003, June 2003.
3. K. Culik II and J. Kari. Image compression using weighted finite automata. In Computers & Craphics, 17:305–313, 1993.
4. K. Culik II and V. Valenta. Finite automata based compression of bi-level and simple color images. In Computers & Craphics, 21:61–68, 1997.
5. K. Culik II and J. Kari. Image compression Using Weighted Finite Automata, in Fractal Image Compression. In Theory a Techniques, Ed. Yuval Fisher, Springer Verlag, pp 243-258, 1994.
6. J.E.Hopcroft and J.D.Ullman. Introduction to automata theory, languages and computation. In Addison-Wesley, 1979.
7. Marian Mindek. Finite State Automata and Images. In Wofex 2004, PhD Workshop, Ed. V. Snášel, ISBN: 80-248-0596-0, 2004
8. Marian Mindek. Finite State Automata and Image Recognition. In Dateso 2004, Ed. V. Snášel, J. Pokorný, K. Richta, pp 132-143, ISBN: 80-248-0457-3, 2004
9. Marian Mindek. Finite State Automata and Image Storage. In Znalosti 2005, Eds. Lubomír Poplínksý, Michal Krátký, ISBN: 80-248-0755-6
10. Marian Mindek. Konečné automaty jako obrázky s multi-rozlišením. In posters Znalosti 2005
11. W3C (2004) XML Protocol. XML Protocol Web Page http://www.w3.org/XML (January 2005)