Image Recognition Using Finite Automata

Tomáš Skopal, Václav Snášel, Michal Krátký

Department of Computer Science
VŠB-Technical University Ostrava
17. listopadu 15, 708 33 Ostrava
Czech Republic

e-mail: {tomas.skopal, vaclav.snasel, michal.kratky}@vsb.cz

Abstract. In this paper we introduce an idea of image recognition using conventional (single-dimensional) finite automata. This approach could be an elegant alternative to complicated solutions based on two-dimensional languages and two-dimensional automata. In consequence, this method could be generally extended to the context of higher-dimensional languages beyond the scope of image recognition.

1 Introduction

Image recognition recently became an object of interest for theory of automata. The picture, a rectangular raster, can be considered as a sentence of a two-dimensional language where the pixels of picture are characters of a finite alphabet.

It is obvious that sentences of two or more-dimensional language cannot be recognized by "conventional" automata. Conventional automaton takes the characters from the input one by one as they appear in a single-dimension sentence. On the other side, two-dimensional sentence processing (e.g. picture) is not so unambiguous, there exist four directions in which the sentence can be processed in each step – *left, right, upwards, downwards*. There were several two-dimensional automata designed, e.g. *4-way* finite-state automata [BH67].

Our solution tries to exploit the existing well-established area of "conventional" automata together with the transformation of the two-dimensional language into a single-dimensional one. The transformation of a picture (or two-dimensional sentence) consists of *space linearization*. This means that the pixels of a picture are linearly ordered and the resultant ordering along with the original picture define the appropriate single-dimensional sentence. The linear order is performed using a space filling curve. In this paper we propose certain curves which were proved to be the good space-describing curves in many applications (especially in data storage and retrieval). However, the quality of the curves may differ in our case and therefore we refer to [SKS02] where we discuss some general properties of space filling curves.

Once we have chosen the curve for language description we must construct an automaton that recognizes a given picture in its "flat shape". However, none of the space filling curves describe the space (and picture) perfectly, some distortion of the picture recognition must be taken into account. This seeming drawback can turn over

to an advantage if we realize that the measure of recognition distortion may represent *similarity* of the recognized picture to the prospective pattern.

Automaton construction for recognition of the linearized picture is based on the Levenshtein DFA where the Levenshtein metric (edit distance) serves as the measure for the allowed picture distortion.

2 Two-dimensional Languages

Informally, a two-dimensional string is called a picture and is defined as a rectangular array of symbols taken from finite alphabet Σ. A two-dimensional language (e.g. picture language) is then a set of pictures.

A generalization of formal languages to two dimensions is possible in different ways, and several formal models to recognize or generate two-dimensional objects have been proposed in the literature (see [KM1, KM2, LMN98]). These approaches were initially motivated by problems arising in the framework of pattern recognition and image processing.

Definition [RS97] A two-dimensional string (e.g. picture) over Σ is a two-dimensional rectangular array of elements from Σ. The set of all two-dimensional strings over Σ is denoted as Σ^{**}. A two-dimensional language over Σ is a subset of Σ^{**}.

Given a picture $p \in \Sigma^{**}$, $l_1(p)$ denotes the number of rows and $l_2(p)$ denotes the number of columns of p.

The pair $(l_1(p), l_2(p))$ is called the size of the picture p. The set of all pictures over Σ of size (m, n), with $m, n > 0$ will be indicated as $\Sigma^{m \times n}$. Furthermore, if $1 \leq i \leq l_1(p)$ and $1 \leq j \leq l_2(p)$, then $p(i, j)$ (or equivalently $p_{i,j}$) denotes the symbol in picture p on coordinates (i, j).

Two-dimensional languages, or picture languages, are an interesting generalization of the standard languages of computer science. Rather than one-dimensional strings, we consider two-dimensional arrays of symbols over a finite alphabet. These arrays can then be accepted or rejected by various types of automata. The introduction of two-dimensional automata brought a new sort of automata on the stage, with its own huge theoretical background.

3 Another Approach

Our approach is to reuse the existing traditional (single-dimensional) automata (languages respectively) and simplify the automaton construction problem. The most important thing is to transform the two-dimensional language (pictures) into one-dimensional strings. This can be done using space filling curves. The consecutive automaton construction depends on the properties of space filling curve we have chosen.

3.1 Space Filling Curves

We want to transform the *two-dimensional string* over Σ into the *one-dimensional string* over Σ. The two-dimensional string over Σ is a two-dimensional rectangle array of elements of Σ. We can look at the array as a two dimensional space $\Omega = D_1 \times D_2$, where the cardinality of domain D_1 ($|D_1|$) is equal to the rows count of the array and $|D_2|$ is equal to the columns count. The tuple (point) with coordinates $(column, row)$ within the space will have a value in Σ.

Many space filling curves have been developed, for example *C-curve*, *Z-curve* or *Hilbert curve* ([Ma99]). For deeper acquaintance with the topic of general space filling curves we refer to the comprehensive monography by Hans Sagan [Sa94].

The usage of the curves isn't in two-dimensional space only, but the curves fill any vector space with arbitrary dimension. It is possible to use the curves for transformation of the n-dimensional string over Σ into the one-dimensional string over Σ. We can see C-curve, Hilbert curve, and Z-curve filling the two dimensional space 8×8 in Figure 1.

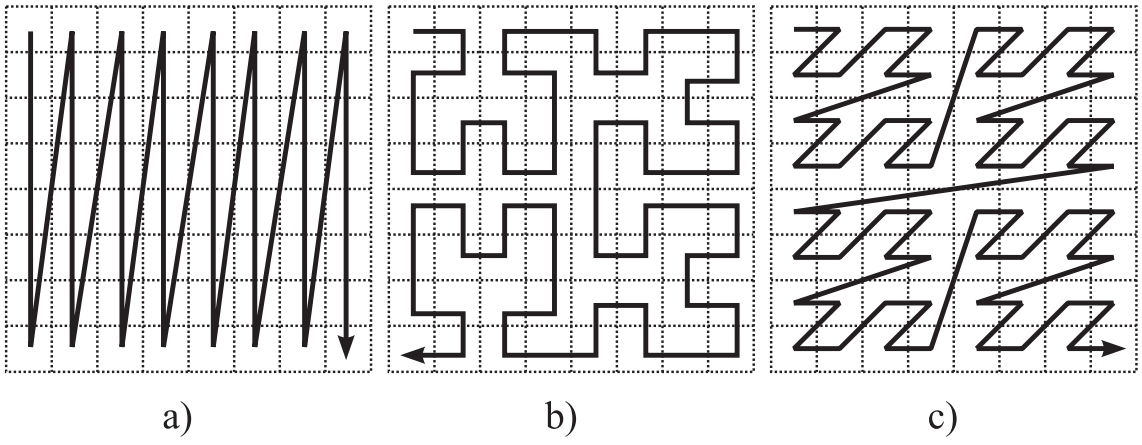

Figure 1: The space filling curves. a) C-curve, b) Hilbert curve, and c) Z-curve.

We can consider several curves, but it is convenient to choose the curve that is highly *self-similar* ([Ma99], [SKS02]) – informally, it means that points that are geometrically close, would have to lie close on the curve. For example, the Z-curve is used for indexing of multidimensional data with UB-trees ([Ba97]). In the following section we will describe the Z-curve as an example of space filling curve.

3.2 Z-address

Definition 1 **(Z-address)**
Let Ω be an n-dimensional space. For tuple $\mathcal{O} \in \Omega$ with n attributes and binary representation attribute value $A_i = A_{i,s-1}A_{i,s-2}\ldots A_{i,0}$, where $1 \leq i \leq n$. Then

$$Z(\mathcal{O}) = \sum_{j=0}^{s-1} \sum_{i=1}^{n} A_{i,j} 2^{jn+i-1}$$

is the *Z-address* function for space Ω.

The attributes of tuple define the coordinates of point representing tuple in the space Ω. If we are calculating the Z-address for all points of n-dimensional space Ω and order the points according their Z-address value, we get the Z-curve filling the entire space Ω (see Figure 2a). For calculation of tuple Z-address exists algorithm with linear complexity - so called *bit interleaving algorithm* (see bellow).

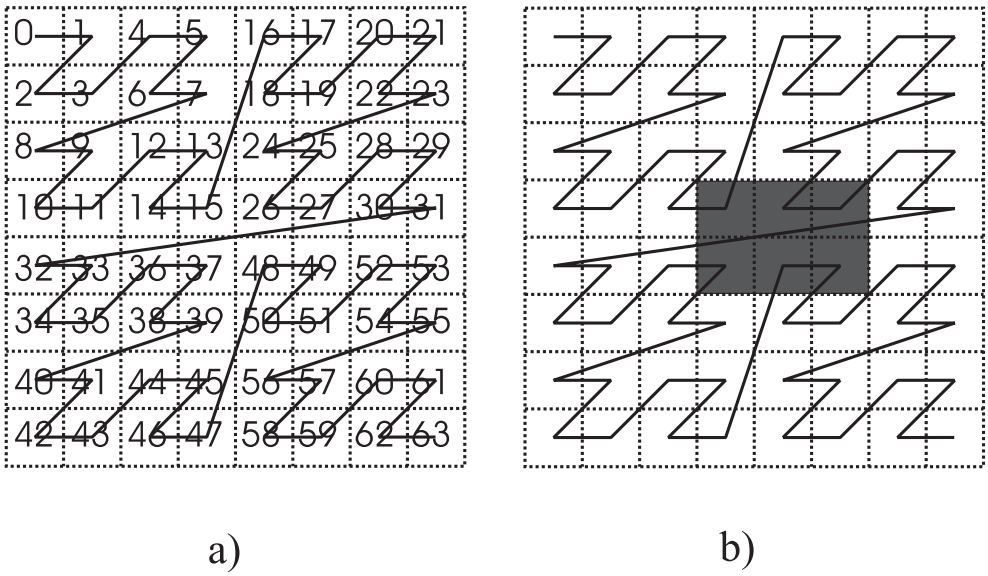

Figure 2: a) The two-dimensional space (image) 8×8 filled by Z-curve. b) Picture in image interleaved by the Z-curve.

Z-address calculation example

We see the calculation of Z-address according bit interleaving algorithm for point (6,13) in two-dimensional space in Figure 3. Numbers 6 and 13 have the binary form 0110 and 1101 respectively. We obtain the coordinate values as four places bit strings. Maximal values for four places binary number is 16. The domains D_i of both attribute are sets {0,1, ..., 14,15}, point lies in two-dimensional space 16×16. The result point Z-address is then 10110110 (182 decimal).

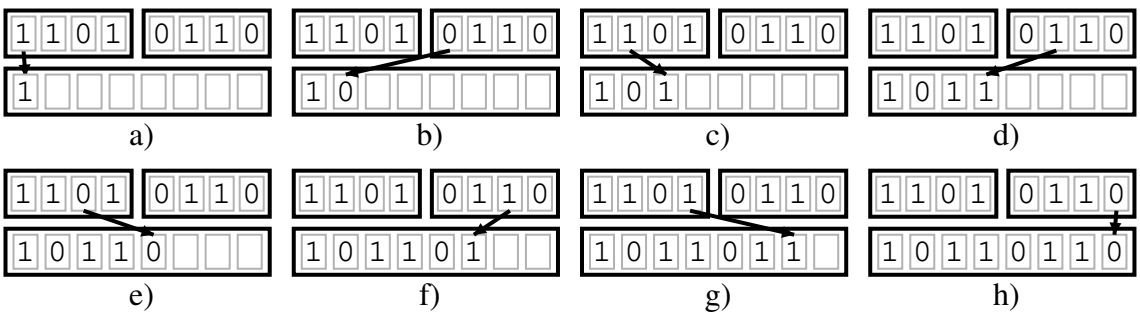

Figure 3: The Z-address calculation according to bit interleaving algorithm for point $(6, 13)$ in two-dimensional space 16×16.

It is possible to go through the entire space passing upon Z-curve. We interleave a picture (the two-dimensional string) over Σ by Z-curve and we recognize the picture by the "classical" one-dimensional finite automata (e.g. see Figure 2b). The automata construction of the picture recognition is outlined in the next section.

3.3 Automaton Construction

The automaton will recognize a square picture of size $x \times y$ within an image of a greater size (see Figure 2b).

Construction
The automaton type is well-known NDFA for matching patterns with k differences – in other words, it is an automaton for approximate string matching using Levenshtein metric. The construction takes as a parameter the pattern sentence (picture to be recognized) and a Levenshtein distance threshold which defines the maximal tolerance value of the above mentioned picture distortion. For detailed information on construction of the Levenshtein automata see [Ho96].

The Levenshtein distance threshold is computed as the minimal distance of the pattern picture to an input picture when the *correct* input is still recognizable. More clearly, the correct input picture may appear on any position in the image and the automaton must recognize the picture on this position. However, the threshold value may cause that they can be recognized also incorrect pictures. This imprecise behaviour could serve as a similarity recognition because the recognized picture is always within the Levenshtein distance threshold which guarantees only a limited number of differences between the pattern picture and the input picture. Pictures that are close (in terms of Levenshtein distance) could be considered as similar to each other.

In following we will focus on measuring of the pattern picture and input picture using Levenshtein metric.

3.3.1 What is the Levenshtein Distance?

Levenshtein distance (LD) is a measure of the similarity between two strings, which we will refer to as the source string (s) and the target string (t). The distance is the number of deletions, insertions, or substitutions required to transform s into t. For example,

If s is `"test"` and t is `"test"`, then LD(s,t) = 0, because no transformations are needed. The strings are already identical.

If s is `"test"` and t is `"tent"`, then LD(s,t) = 1, because one substitution (change "s" to "n") is sufficient to transform s into t. The greater the Levenshtein distance, the more different the strings are.

Levenshtein distance is named after the Russian scientist Vladimir Levenshtein, who devised the algorithm in 1965 [Le66]. If you can't spell or pronounce Levenshtein, the metric is also sometimes called *edit distance*.

The Levenshtein distance algorithm (based on dynamic programming) has been used in:

- Spell checking
- Speech recognition

- DNA analysis
- Plagiarism detection

The Algorithm – step description

1. Set n to be the length of s.
   Set m to be the length of t.
   If $n = 0$, return m and exit.
   If $m = 0$, return n and exit.
   Construct a matrix containing $0 \ldots m$ rows and $0 \ldots n$ columns.
2. Initialize the first row to $0 \ldots n$.
   Initialize the first column to $0 \ldots m$.
3. Examine each character of s (i from 1 to n).
4. Examine each character of t (j from 1 to m).
5. If $s[i]$ equals $t[j]$, the cost is 0.
   If $s[i]$ doesn't equal $t[j]$, the cost is 1.
6. Set cell d[i,j] of the matrix equal to the minimum of: a. The cell immediately above plus 1: $d[i-1, j] + 1$.
   b. The cell immediately to the left plus 1: $d[i, j-1] + 1$.
   c. The cell diagonally above and to the left plus the cost: $d[i-1, j-1] + cost$.
7. After the iteration steps (3, 4, 5, 6) are complete, the distance is found in cell $d[n, m]$.

3.4 Examples

As we have said earlier, the Levenshtein threshold value is computed as a maximum distance of the pattern picture and the *correct* input picture on any position in the image being recognized. In Figure 4 are depicted three examples of pictures (sized 3×3) in images (sized 8×8) and its distances to pattern pictures.

Note that the pixel values are characters from a finite alphabet. The numbers next to the pixels are the character identifiers. The gaps denoting those pixels of image that are not pixels of the picture are represented with appropriate characters but in our examples, for simplicity and clarity, the gap is represented with a special character that is not contained in the alphabet Σ. This special character ensures the worst matching case, thus the real distance computations will be always smaller or equal.

3.5 Extension to Multidimensional Languages

Because the space filling curve remains single-dimensional even for multidimensional spaces, we can extend the scope of two-dimensional languages to the multidimensional languages without the need of changing the automaton construction. Then, multidimensional sentences can be constructed simply by extending the language with additional coordinates.

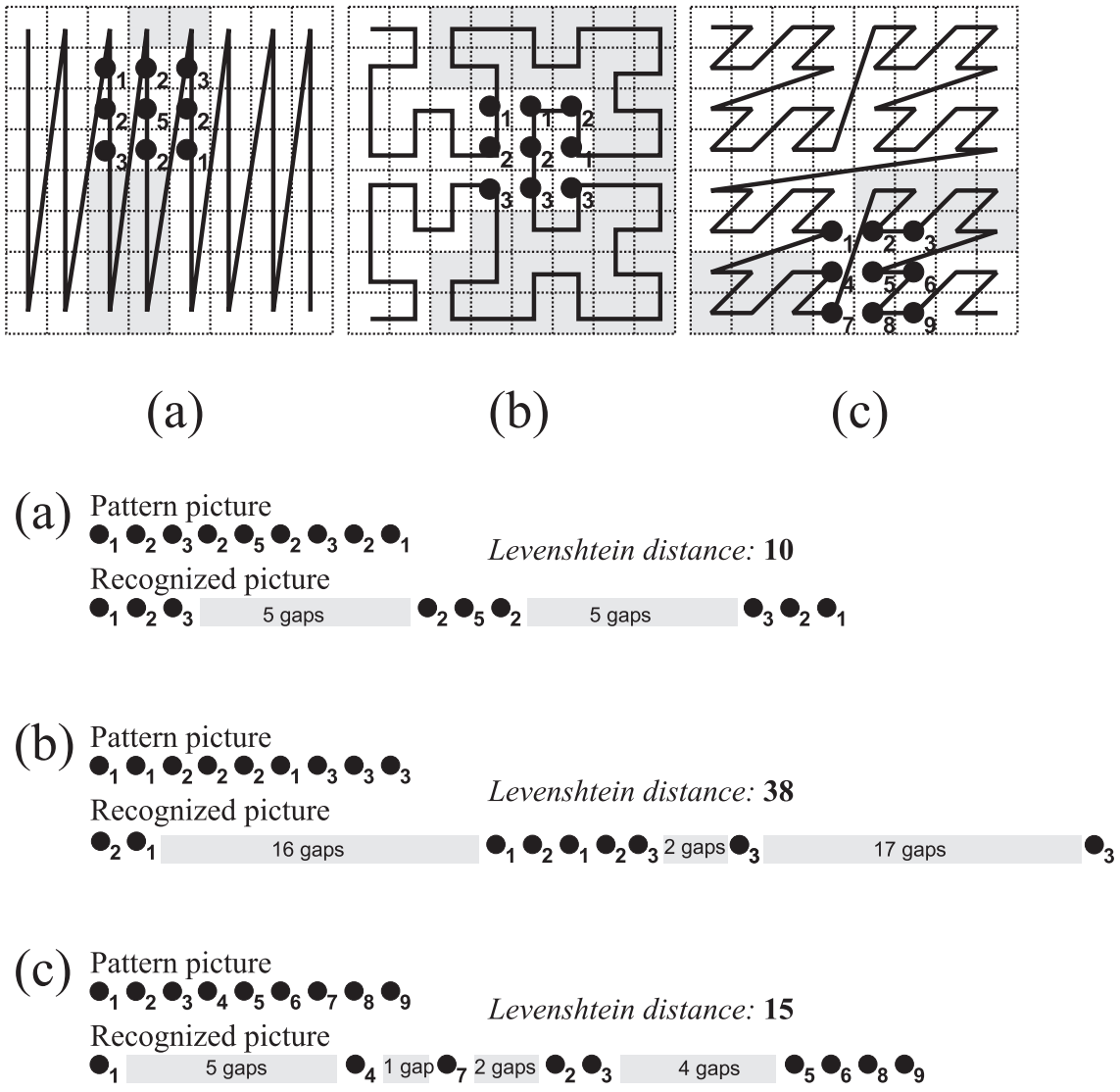

Figure 4: Measuring the Levenshtein distance on pictures.

In general, we can say that the imprecision caused by the Levenshtein distance threshold will increase with increasing dimension. This fact arises from the behaviour of the space filling curves in high-dimensional vector spaces. The other factor is the relation of sentence size to space size. The longer sentences and smaller sentence/space size ratio, the lower imprecision.

4 Conclusions

In this paper we have proposed an alternative solution of image recognition and even multidimensional language recognition. This method is based on space filling curves and Levenshtein automaton construction. The interesting property of this approach is an ability of similarity recognition.

References

[Ba97] Bayer R. The Universal B-Tree for multidimensional indexing: General Concepts. In: *Proc. Of World-Wide Computing and its Applications 97 (WWCA 97)*. Tsukuba, Japan, 1997.

[BH67] M.Blum, C.Hewitt. Automata on a 2-dimensional tape, *8th IEEE Symp. on Switching and Automata Theory*, 1967, pp. 155-160

[Ho96] J.Holub. Reduced Nondeterministic Finite Automata for Approximate String Matching, *Proceedings of the Prague Stringologic Club Workshop*, 1996

[KM1] J.Kari, C.Moore. Rectangles and Squares Recognized by Two-dimensional Automata, submitted, 2002

[KM2] J.Kari, C.Moore. New results on alternating and non-deterministic two-dimensional finite-state automata, In: *Proc. of the Symposium on Theoretical Aspects of Computer Science (STACS)*, LNCS, 2001.

[LMN98] K.Lindgren, C.Moore, M.Nordahl. Complexity of Two-dimensional Patterns, In: *Journal of Statistical Physics* 91(5-6) (1998) 909-951.

[RS97] D. Giammarresi, A. Restivo. Two-Dimensional Languages, In: *Handbook of Formal Languages*, vol 3, G. Rowzenberg and A. Salomaa eds, Springer-Verlag, 1997, chapter 4, 215–267.

[Le66] V.I.Levenshtein. Binary codes capable of correcting deletions, insertions and reversals, *Soviet Physics-Doklady 10* (1966), 707-710.

[Ma99] Markl, V.: *Mistral: Processing Relational Queries using a Multidimensional Access Technique*, Ph.D. thesis, Technical University Munchen, `http://mistral.in.tum.de/results/publications/Mar99.pdf`, 1999

[Sa94] Sagan H. *Space-Filling Curves*, Springer-Verlag, 1994

[SKS02] Skopal T., Krátký M., Snášel V.: Properties of Space Filling Curves And Usage With UB-trees. Submitted to MIS 2002.