On Two-Dimensional Pattern Matching by Finite Automata

Conference Paper · March 2006

DOI: 10.1007/11605157_28

CITATIONS
4

READS
59

2 authors, including:

Borivoj Melichar
Czech Technical University in Prague
82 PUBLICATIONS **397** CITATIONS

SEE PROFILE

On Two-Dimensional Pattern Matching by Finite Automata*

Jan Žďárek and Bořivoj Melichar

Department of Computer Science and Engineering,
Faculty of Electrical Engineering, Czech Technical University in Prague,
Karlovo náměstí 13, 121 35 Praha 2, Czech Republic
{melichar, zdarekj}@fel.cvut.cz

Abstract. This paper presents a general concept of two-dimensional pattern matching using conventional (one-dimensional) finite automata. Then two particular models and methods, implementations of the general principle, are presented. The first of these two models presents an automata based version of the Bird and Baker approach with lower space complexity than the original algorithm. The second introduces a new model for two-dimensional approximate pattern matching using the two-dimensional Hamming distance.

1 Introduction

In recent years there has been unceasing interest in two and more dimensional pattern matching problems. Such interest is substantiated by the growing computational strength of our computers allowing multidimensional data, e.g. NMR scans, photographs, etc., to be processed.

In this paper the idea of dimensional (linear) reduction is used to provide a generic algorithm of 2D pattern matching using finite automata, *FA* for short. This has been known for a very long time and is widely used (for a nice survey in the area of 2D matching see [1]). In our approach, by the dimensional reduction of the problem we obtain a mapping between final states and one-dimensional strings of the d dimensional pattern and a new preprocessed $d-1$ dimensional text array that is over an alphabet of automaton final state labels. Then linear reduction is used again and after $d-1$ steps we finally obtain the one-dimensional problem. From now on let us restrict our deliberation to describing the most practical case, i.e. 2D pattern matching by finite automata.

Based on the generic algorithm, a couple of automata based models and algorithms for 2D exact and 2D approximate pattern matching using a 2D Hamming distance are presented. For this purpose some of the wide scale of classical FA solving one-dimensional exact and approximate pattern matching problems [2,3,4] are reused. The proposed methods in fact generalize the one-dimensional pattern matching approach based on finite automata.

* This research is partially supported by the MŠMT under research program MSM 6840770014.

J. Farré, I. Litovsky, and S. Schmitz (Eds.): CIAA 2005, LNCS 3845, pp. 329–340, 2006.

1.1 Basic Notions

Let A be a finite alphabet and its elements are called symbols. A set of strings over A is denoted by A^* and A^ℓ is a set of strings of length ℓ. The empty string is denoted by ε. Language L is any subset of A^*, $L \subseteq A^*$. Let $P \in A^m$ and $T \in A^n$ be a pattern and a text, respectively, $m \leq n$. An exact occurrence of P in T is index i, such that $P[1,\ldots,m] = T[i,\ldots,i+m-1]$, $i+m-1 \leq n$. If some string R is a substring of T and the relevant edit distance is $D(P,R) \leq k$ then R is an approximate occurrence of P in T with at most k errors.

An array (picture, 2D string) is a rectangular arrangement PA of symbols taken from a finite alphabet. The set of all arrays over alphabet A is denoted by A^{**} and a 2D language over A is thus any subset of A^{**}. The set of all arrays of size $(n \times n')$ over A, where $n, n' > 0$, is denoted by $A^{n \times n'}$. ([5] discusses the theory of 2D languages in detail.) The size of an array is the size of its rectangular shape, denoted by $|PA|$ or $(x \times y)$, and its numerical value is the product of its $\boldsymbol{x}$ and $\boldsymbol{y}$ components, $|PA| = xy$. A 2D exact occurrence of $PA \in A^{m \times m'}$ in $TA \in A^{n \times n'}$ is a pair (i,j), such that $PA[1,\ldots,m;\ 1,\ldots,m'] = TA[i,\ldots,i+m-1;\ j,\ldots,j+m'-1]$. If for some sub-array X of TA and a relevant 2D edit distance *2D-dist*$(PA, X) \leq k$ then X is a 2D approximate occurrence of PA in TA with at most k errors.

A finite automaton (FA) is a quintuple (Q, A, δ, I, F). Q is a finite set of states, A is a finite input alphabet, $F \subseteq Q$ is a set of final states. If FA is nondeterministic (NFA), then δ is a mapping $Q \times (A \cup \{\varepsilon\}) \mapsto \mathcal{P}(Q)$ and $I \subseteq Q$ is a set of initial states. A deterministic FA (DFA) is (Q, A, δ, q_0, F), where δ is a (partial) function $Q \times A \mapsto Q$; $q_0 \in Q$ is the only initial state.

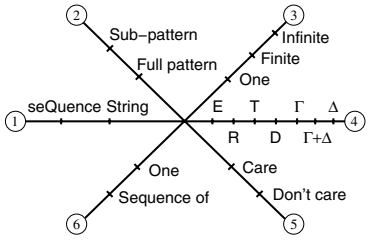
By custom the term finite automaton is used where a pattern matching automaton (PMA) would be more appropriate. The PMA is a program based on a run of an FA, e.g. the AC automaton (or machine) uses "forward" δ and "backward" *fail* functions [6]. A PMA may be able to do some additional operations, e.g. it may have some "actions" assigned to some or all of its transitions or states (a generalization of a transducer).

1.2 Types of Pattern Matching Automata

In 1997 Melichar and Holub [4] showed that 1D pattern matching problems are sequential problems and therefore it is possible to solve them using FA. Moreover, they presented a six-dimensional classification of all 192 then known 1D pattern matching problems for an alphabet of finite size.

The classification criteria (see Fig. 1) are: 1. nature of the pattern; 2. integrity of the pattern; 3. number of patterns; 4. way of matching (exact or approximate using various distances); 5. importance of symbols in the pattern; 6. number of instances of the pattern.

The original model has been updated recently [7] in its fourth dimension with distances used in the area of musicology (Δ, Γ) [8]. Those distances were not known at the time of publication of [4] and as a consequence, the number of problems described by this classification has risen from 192 to 336. Together, these criteria allow us to classify conveniently all pattern matching problems.



Criterion	1	2	3	4	5	6
	S	F	O	E	C	O
	Q	S	F	R	D	S
			I	T		
Abbrev.				D		
				$G(\Delta)$		
				$L(\Gamma)$		
				$H(\Gamma+\Delta)$		

Fig. 1. Updated classification of one-dimensional pattern matching problems

Example 1. A common problem in text editors is to find a mistyped word. This involves using approximate string matching of one pattern using the Levenshtein distance. This problem can be simply referred to as an *SFODCO* problem.

One can use the notion of a family of pattern matching problems. In this case symbol "?" is used instead of a particular letter. For example *SFF*??? describes the family of all problems concerning full pattern matching of multiple strings.

2 Generic Algorithm

Reusing finite automata from 1D pattern matching into 2D pattern matching has several advantages: the same formal method of modelling pattern matching algorithms in both cases and a description of all problems using a unified view. Furthermore, automata process a text in a time proportional to the text's length, for fixed alphabet A in linear time. (Otherwise the time complexity should be multiplied by a $\log|A|$ factor.) There are known simulation methods of NFA [9, 10,11], and for some types of problems there are known algorithms constructing appropriate DFA directly, some of them even in linear time [2].

The idea of solving multidimensional pattern matching problems using PMA is simple: multiple automata should be used passing their results among them, reducing the dimension of the problem by one in each step. In the last step classical pattern matching can be used. (A preliminary version of Alg. 1 along with some of our ideas presented below has appeared in [7].)

3 2D Exact Pattern Matching

Let pattern array PA be viewed as a sequence of strings. Without loss of generality let these strings be its columns. To locate columns of the pattern array within columns of the text array requires searching for several strings. They are contained in set of strings PS and its cardinality may be less than the number of columns of PA, which can be used to reduce the state complexity of the matching automaton. Testing for identical columns in PA can be done in $\mathcal{O}(|PA|)$ time by a trie construction algorithm (the trie construction algorithm is given in [2] or [6]).

Algorithm 1. A generic algorithm of the 2D pattern matching using pattern matching automata
Method:
1: Dictionary matching automaton $M(PS)$ (of type *SFF?CO*, see Tab. 1) is constructed.
2: Automaton $M(PS)$ is applied to each column of TA and new array TA' is generated.
3: For further matching a one-dimensional representation of the pattern array PA should be computed (*representing string* R).
4: String matching automaton M' (*SFO?CO*) searching for R with at most k errors is built; $k = 0$ in case of the exact matching (*SFOECO*).
5: Automaton M' locates string R inside the rows of TA', reporting eventual (1D) occurrences of R in TA' and therefore also 2D occurrences of PA in the original TA.

3.1 A Finite Automata Model of 2D Exact Pattern Matching

Let m be the number of columns and let m' be the number of rows of pattern array PA, and let m_d be the number of distinct columns of PA, $m_d \leq m$. In the following text all steps of Alg. 1 will be implemented describing the idea of 2D exact pattern matching using pattern matching automata.

According to step 1 of Alg. 1, the $M(PS)$ automaton of type *SFF?CO* should be used. For the purposes of 2D exact pattern matching it is a dictionary matching automaton (*SFFECO*), because the location of the exact occurrences of the individual patterns should be found (Alg. 1, step 2).

SFFECO automaton $M(PS) = (Q, A, \delta, \{q_0\}, F)$, accepts a language $L(M)$, $L(M) = A^*P$, where $P \in PS$. The construction of automaton $M(PS)$ for searching for set PS, $PS = \{P_1, P_2, \ldots, P_{m_d}\}$, consists of a trie construction of m_d patterns and adding a selfloop at its initial state q_0. The construction of NFA $M(PS)$ is given in [3].

A deterministic version of $M(PS)$ and how it works over TA is given in Sec. 3.2.

The Representing String. In step 3 of Alg. 1 construction of the *representing string* R of pattern array PA is required. String R represents the original 2D pattern array in such a way that every column becomes its representing symbol in R. This representing string is a result of linear reduction, and it is a one-dimensional representation of a 2D entity in symbols of a new alphabet. Here these symbols are labels of final states of $M(PS)$.

Example 2. Let $PA \in A^{3\times 3}$, $PA = \begin{array}{|c|c|c|}\hline a & b & a \\ \hline c & a & c \\ \hline c & d & c \\ \hline \end{array}$ and let columns $PA[1] = PA[3] = acc$ and $PA[2] = bad$ be accepted by states f_1 and f_2, respectively, and let $f_1, f_2 \in F$ be final states of automaton $M(PS)$. Then the representing string R of PA in rows of TA' will be 121.

Searching for 2D Exact Occurrences Using the Exact Pattern Matching Automaton. After application of $M(PS)$ on every column of TA the text array TA' is prepared for application of some one-dimensional pattern matching method, e.g. PMA for exact pattern matching.

In this step automaton M' of type $SFO??O$ will be used for matching in the rows of TA'. For the purposes of 2D exact pattern matching this is the simplest automaton, the *SFOECO* automaton.

SFOECO automaton M' is constructed over the alphabet $F \cup \{0\}$, i.e. over the set of final state labels of automaton $M(PS)$, united with at least one symbol that represents the rest of the states of $M(PS)$ (recall the idea of a reduced alphabet). In detail, $M' = (Q', F \cup \{0\}, \delta', \{q_0'\}, F')$ and accepts language $L(M) = (F \cup \{0\})^* R$, where $R \in F^m$, $F = \{s_1, s_2, \ldots, s_{m_d}\}$, $|F| = m_d = |PS|$, and m is the length of the rows of PA.

NFA M' searches for pattern R of length m in each row of TA' individually (Alg. 1, step 5). M' searches for occurrences of R and therefore it can report also 2D occurrences of PA, because at the moment R is found it has been verified that all columns of PA are found in the appropriate place and order.

3.2 Deterministic Finite Automata for 2D Exact Pattern Matching

The Bird and Baker algorithm uses for the 2D exact pattern matching the well known algorithms of Aho-Corasick and Knuth-Morris-Pratt [12], and their method works in linear time. This fact is a strong motivation for us to achieve linear time with our method, too.

Our model of 2D exact pattern matching requires two finite automata: the $M(PS)$ automaton for preprocessing of TA is the *SFFECO* automaton; the M' automaton for searching for string R is the *SFOECO* automaton.

These automata have a neat structure, so they are useful as a model, but they are also nondeterministic. There is no problem with this fact, as these NFA's can be either simulated or determinised before use. However, in order to achieve linear time complexity for 2D exact pattern matching using our method, direct constructions of equivalent deterministic finite automata should be used. Indeed, this is possible in this case (see [2]): DFA $M(PS)$ can be constructed as a *linear dictionary-matching automaton* with time complexity $\mathcal{O}(\log|A||PA|) = \mathcal{O}(mm')$, supposing alphabet A is fixed. Its space complexity is $\mathcal{O}(\log|A||PA|) = \mathcal{O}(mm')$,

Elements of TA' are computed as follows: let $M(PS) = (Q, A, \delta, q_0, F)$ be DFA, $q \in Q$, q is the active state after reading a symbol from the element $TA[i,j]$, then $TA'[i,j] = q\,;\ q \in Q,\ \forall i,j\ \ 1 \le i \le n,\ 1 \le j \le n'$.

DFA M' can be constructed as a *linear string-matching automaton* with time and space complexity $\mathcal{O}(|R|) = \mathcal{O}(m)$.

Theorem 1. *The presented method of 2D exact pattern matching using the direct construction of DFA's has asymptotic time complexity* $\mathcal{O}(|TA|)$.
(Proof omitted.)

This time complexity is the same as in the Bird and Baker solution, i.e. it is linear with the size of a given text array.

The space complexity is designated as $\mathcal{O}(|TA|)$, and it depends on the space for temporary data (and on the sizes of the pattern matching automata, which are smaller).

3.3 Optimized 2D Exact Pattern Matching

The new text array TA' has the same size as TA, and because we are dealing with pictures, it is clear that its size can be very large. Hence a significant saving can be achieved if we could avoid using it.

We can use some natural properties of finite automata to reduce the space complexity of 2D exact pattern matching. To be able to restart a run of a deterministic finite automaton only its transition function, active state and current position in the input text are required. Automaton $M(PS)$ matches in all columns of TA individually, so $\mathcal{O}(n)$ extra space is needed to store all active states of $M(PS)$ in each row of TA. Once one row of TA' is computed by $M(PS)$, it is possible to do matching in it using automaton M' to find possible 2D occurrences. Then the space of the row can be reused, efficiently eliminating the need to store the whole array TA'.

Theorem 2. *Two-dimensional exact pattern matching by finite automata has asymptotic space complexity* $\mathcal{O}(|PA| + n)$.

Proof. The space complexity of automata $M(PS)$ and M', $\mathcal{O}(|PA|)$ and $\mathcal{O}(m)$, respectively, remains unchanged. Since $M(PS)$ treats each column and M' each row individually, to be able to restart the preprocessing phase it is required to store the active states of $M(PS)$ only. The extra space required is $\mathcal{O}(n)$, the number of columns of TA.

Steps 2 and 5 of Alg. 1 are now "interleaved", therefore both automata should be stored in memory at the same time. The asymptotic space complexity of the 2D exact pattern matching is then $\mathcal{O}(|PA|) + \mathcal{O}(m) + \mathcal{O}(n) = \mathcal{O}(|PA| + n)$. □

Note 1. To save some processing time, it is sufficient to start matching for 2D occurrences (Alg. 1, step 5) in row $TA'[x, m']$, $1 \leq x \leq n$, where the topmost occurrences may be located.

4 2D Approximate Pattern Matching

In this section a new automata-based method of 2D approximate pattern matching using a 2D Hamming distance is introduced. The 2D Hamming distance (2D matching with mismatches) D_{2H} is analogous to the Hamming distance D_H in 1D matching [3,13].

$D_H(v, w)$ between two strings $v, w \in A^*$, $|v| = |w|$ is the minimum number of edit operations *replace* (change of symbol), needed to convert string v to w. Distance $D_{2H}(P, R)$ between two arrays P and R is the minimum number of edit operations *replace* needed to convert array P to R, $P, R \in A^{**}$, $|P| = |R|$.

In 2D exact matching a simple dimensional reduction was sufficient to do the task, but here more information about each prefix is needed: not only that some

prefix of one or more strings of PS can end at the actual element, but that there can be a certain number of (1D) mismatches in it.

Let us refer again to the generic algorithm. First, an approximate dictionary matching automaton $M(PS)$ for the approximate matching of a set of strings using the Hamming distance should be built (*SFFRCO*). $M(PS)$ accepts a language $L(M) = A^* H_k(PS)$, where

$$H_k(PS) = \{X;\ X \in A^*, D_H(X,P) \leq \min(k, m'-1)\ \wedge\ P \in PS\}, \qquad (1)$$

k is a given number of allowed 2D mismatches in the 2D occurrence of pattern array PA and m' is the length of the columns of PA (supposing we start the processing vertically).

Since $M(PS)$ searches for a set of strings, it consists of *SFORCO* sub-automata for approximate pattern matching using the Hamming distance. (*SFORCO* automata and their usage are discussed for example in [3].)

The reason why the distance $\min(k, m'-1)$ in formula (1) should be used is that the *SFORCO* sub-automaton for pattern of length m' can find its occurrences with at most $m'-1$ mismatches in every column of TA, while $k \leq mm'-1$.

The final states of $M(PS)$ indicate which one of the (1D) patterns was found, and the amount of mismatches found in a particular occurrence within the columns of TA.

Let $M(PS) = (Q, A, \delta, I, F)$ be the *SFFRCO* automaton for the approximate matching of a set of patterns using the Hamming distance.

Proposition 1. SFFRCO *automaton $M(PS)$ may have after each reading of the input symbol and executing all subsequent transitions*

1. *at most m_d final states active,*
2. *among these active final states at most one final state indicating the exact occurrence of pattern P_i of PS in the text array,*
3. *at most m_d active final states, indicating occurrences of m_d different patterns with l mismatches, $0 < l \leq k'$, where $k' = \min(k, |P| - 1)$, $P \in PS$.*

(Proof omitted.)

As a consequence of Proposition 1, symbols of a secondary alphabet will have $|PS|$ parts, each $\lceil \log_2(k'+1) \rceil$ bits long, representing the number of mismatches found (k') and one "no match" situation in each string of PS.

Let $M(PS) = (Q, A, \delta, I, F)$, $q \in Q$ be the active state after reading the symbol from element $TA[i,j]$. Let an ordered couple (s,x) be the label of a final state of $M(PS)$, where s is a string identification (number of its matching *SFORCO*), to which a final state belongs, and x is the number of mismatches found in it. Let $\oslash \notin Q$ be a special symbol not among the labels of states.

Then it holds for $\forall i, j, s\ 1 \leq i \leq n, 1 \leq j \leq n', 1 \leq s \leq |PS|$:

$$TA'[i,j,s] = \begin{cases} x\ ;\ q \text{ active}, q \in F, q = (s,x), \\ \oslash\ ;\ s^{th} \text{ sub-automaton has no active final state.} \end{cases} \qquad (2)$$

Let l_s be the number of mismatches for each pattern number s and $k' = \min(k, |P_s| - 1)$, $P_s \in PS$. Every element of TA' provides the following information about the potential occurrence of a column of PA ending at a particular element in the original array TA:

1. exact match, $l_s = 0$,
2. approximate match with l_s mismatches, $0 < l_s \leq k'$,
3. no match, $TA'[i, j, s] = \oslash$ ($l_s > k'$ mismatches found).

String R, representing the original pattern array PA in the rows of TA', should be constructed from "s" parts of (s, x) labels of final states, which are the same in each *SFORCO* sub-automaton M_s of $M(PS)$.

Searching for 2D Approximate Occurrences Using the 2D Hamming Distance. Here the *SFOLCO* automaton should be used, which is a PMA able to match pattern R using Γ distance. It will search for the representing string R and also count the numbers of errors found in each column of a possible 2D occurrence. According to [8], let A' be an ordered alphabet and a, b be two symbols of alphabet A', then Γ *distance* $D_\Gamma(v, w)$ between two strings $v, w \in A^*$, $|v| = |w|$, is $\sum_{i=1}^{|v|} |v[i] - w[i]|$.

Let F in this section denote the set of final states of finite automaton $M(PS)$ and let secondary alphabet A' be set F extended with a special symbol "$\oslash$": $A' = F \cup \{\oslash\}$. Symbols of alphabet A' denote the number of errors found at a given element of TA in the particular pattern from PS. Symbol $\oslash$ represents the "no match" situation, where the number of errors in a particular string at the given element is greater than the number of mismatches k' allowed in it.

Let M' be $M' = (Q', A', \delta', \{q'_{0,0}\}, F')$, δ' is a mapping $Q' \times (A')^m \mapsto \mathcal{P}(Q')$, where Q' are states of M' and $m = |PS|$. The active final state of M' indicates that an approximate 2D occurrence of PA in TA has been found. Furthermore, it shows the total number of 2D mismatches found in a particular 2D occurrence, up to the given maximum k.

Symbols of A' are ordered using the second part x of the (s, x) couple, i.e. the number of mismatches found in string $P_s \in PS$. Let the distances between two symbols, final state labels of $M(PS)$, be defined as follows:

$$|(s, x) - (t, y)| = \begin{cases} |x - y| \; ; s = t, \text{ if } (x \vee y) = \oslash, \text{ then } \oslash = m' \\ m' \quad ; s \neq t\,. \end{cases} \tag{3}$$

Example 3. To illustrate formula (3) let us compare symbols: $|(1, 0) - (1, 3)| = 3$, $|(2, 3) - (5, 0)| = m'$, $|(1, 0) - (1, \oslash)| = m'$, $|(3, \oslash) - (3, \oslash)| = 0$.

These comparisons are used in the error counting in TA'. If symbol (i, j) is found in a position where $(i, 0)$ is expected, then

$$l = \begin{cases} j \; ; j \neq \oslash, \\ m' \; ; j = \oslash. \end{cases} \tag{4}$$

l mismatches is added to the current value of err, if $err + l \leq k$. Let err be the current value of mismatches found in a particular occurrence of the representing

string. If $j = \oslash$, $k \geq m'$ errors are allowed and $err + m' \leq k$, m' mismatches are added to the current value, because a column of PA of length m' was not found at its expected position.

Let k be the number of mismatches allowed, $k < |P_i|$, $P_i \in PS$. In this case the number of 2D mismatches in any occurrence is lower than the length of the columns of PA. Then the *SFOLCO* automaton has a slightly simpler form than in the general case, as it allows at most $|P_i| - 1 = k'$ mismatches while searching for the representing string.

A slightly more complicated case is when $k \geq m'$ is given. This means that the number of 2D errors is greater than the number of mismatches that $M(PS)$ is able to find in one dimension. To be able to count 2D errors by one-dimensional means, $\oslash$ transitions are introduced in automaton M', representing $m' = |R|$ errors, $R \in PS$. Hence each state may have at most $m' + 1$ outgoing transitions, only initial state $q'_{0,0}$ has more, because of the selfloop.

Construction of the *SFOLCO* automaton M' is shown in Alg. 2.

Algorithm 2. Construction of NFA for the approximate pattern matching of string R over the set of identifiers of final states of NFA $M(PS)$

Input: Pattern R over the set of final states of $M(PS)$, $|R| = m$, and m' be the length of patterns in set PS. The maximum number of 2D errors allowed k, $k \leq mm'$.

Output: NFA M', $M' = (Q', A', \delta', \{q'_{0,0}\}, F')$, accepting language $L(M') = A'^{*}\Gamma(R) = \{wx;\ w, x \in A'^{*}, D_\Gamma(x, R) \leq k\}$.

Sets Q', F' and mapping δ' are constructed in the following way:

Method:

$Q' \leftarrow \{q'_{0,0}\}$ { the initial state }
for $i \leftarrow 0$ **to** k
  first $\leftarrow \left\lfloor \frac{|i-1|}{m'} \right\rfloor + 1$ { "depth" of the first non-initial state in the i^{th} row }
  for $j \leftarrow$ *first* **to** m
    $Q' \leftarrow Q' \cup \{q'_{j,i}\}$ { add a new state }
    if $j = m$ **then**
      $F' \leftarrow F' \cup \{q'_{j,i}\}$ { add a new final state }
      Assign an alias $q'_{j,i} \leftarrow i$. { the number of errors found }
    end
    if $j = 1$ **then**
      if $i < m'$ **then** $\delta'\left(q'_{j-1,0}, (R[j], i)\right) \leftarrow q'_{j,i}$
        else $\delta'\left(q'_{j-1,0}, (R[j], \oslash)\right) \leftarrow q'_{j,i}$ { if $i = m'$ }
    else
      $x \leftarrow \max(0, i - m')$
      if $j =$ *first* **then** *last* $\leftarrow i \pmod{m'}$
        else *last* $\leftarrow \min(i, m')$
      for $l \leftarrow 0$ **to** *last*
        if $(i \geq m') \wedge (l = 0)$ **then** $\delta'\left(q'_{j-1,x}, (R[j], \oslash)\right) \leftarrow q'_{j,i}$
          else $\delta'\left(q'_{j-1,x+l}, (R[j], i - x - l)\right) \leftarrow q'_{j,i}$
      end
    end
end end
$\delta'(q'_{0,0}, a) \leftarrow \delta'(q'_{0,0}, a) \cup \{q'_{0,0}\},\ \forall a \in A'$ { the selfloop of the initial state }

4.1 Practical 2D Pattern Matching Using the 2D Hamming Distance

Our model of this type of 2D approximate pattern matching requires two PMA: *SFFRCO* and *SFOLCO*. These are nondeterministic and there are no known direct methods for constructing their deterministic versions (in contrast to 2D exact matching).

In such a situation we can either transform NFA to DFA using the standard subset construction [14] or simulate a run of the NFA. The former method may result in quite high space complexity, hence in the rest of this section the latter method is used: a simulation of a run of $M(PS)$ and M'.

For the simulation of the run of $M(PS)$ we use dynamic programming for pattern matching using the Hamming distance [3,9,15]. This method for string matching computes matrix D of size $(m+1) \times (n+1)$ for a pattern of length m and a text of length n. Each element of D usually contains the edit distance between the string ending at a current position in text T and the prefix of pattern P. The advantages of this method are that it is very simple and can be implemented memory-efficiently, and for a pattern of length m it requires only $\mathcal{O}(m)$ space. It works in time $\mathcal{O}(mn)$ for a text of length n, and this complexity is independent of the number of errors. Ukkonen [15] improved the expected time of the standard dynamic programming for approximate string matching to $\mathcal{O}(nk')$, by computing only the zone of the dynamic programming matrix consisting of the prefix of each column ending with the last k' in the column, where k' is the maximum number of mismatches in the occurrence. This improvement does not help much in our case, because for 2D approximate matching we usually need more than m errors to be allowed and then we have $k' = m - 1$.

From the construction of $M(PS)$ we see that there is at most m_d sub-automata for the approximate pattern matching of $m_d \leq m$ strings of PA. These sub-automata can be simulated by the dynamic programming in time $\mathcal{O}(mm'nn')$, which is independent of the number of 2D errors.

Automaton M' has a special structure and can be easily simulated in time $\mathcal{O}(mnn' - mm'n)$. Its simulation algorithm has appeared in [7].

Theorem 3. *The described realization of our method of 2D approximate pattern matching with finite automata using the 2D Hamming distance has asymptotic time complexity* $\mathcal{O}(|TA||PA|)$. *(Proof omitted.)*

Proposition 2. *The space complexity of the basic version of 2D approximate pattern matching using the 2D Hamming distance presented above is* $\mathcal{O}(m|TA|)$. *(Proof omitted.)*

4.2 Optimized 2D Approximate Pattern Matching Using the 2D Hamming Distance

According to the idea from Sec. 3.3, in order to reduce the space complexity we have to be able to restart a run of finite automaton $M(PS)$ in columns of TA. We use dynamic programming to simulate sub-automata of $M(PS)$, so $\mathcal{O}(mm')$space is needed to store the current state of the dynamic programming

(state of $M(PS)$) in each row of TA. Once one row of TA' is computed, it is possible to do the matching in it using the simulation algorithm of automaton M' to find possible 2D occurrences. Then the space of the row can be reused, effectively eliminating the need to store the whole array TA'.

Theorem 4. *The space complexity of the optimized version of 2D approximate pattern matching using the 2D Hamming distance is* $\mathcal{O}(n|PA|)$.

Proof. Once again following steps of Alg. 1: (1-2) We have n columns of TA, and in each of them there run m_d, $m_d \leq m$, dynamic programming simulations of $M(PS)$ sub-automata in $\mathcal{O}(mm')$ space each, that is $\mathcal{O}(mm'n)$. To store the results of simulation in each column $\mathcal{O}(m)$ space is needed, that is $\mathcal{O}(mn)$. (3) We also have to store an assignment of PA columns to the sub-automata of $M(PS)$ in the representing string with the space complexity $\mathcal{O}(m)$. (4-5) The simulation algorithm of M' has asymptotic space complexity $\mathcal{O}(m)$.

Since steps 2 and 5 of Alg. 1 are interleaved, all space requirements should be summed up and the resulting asymptotic space complexity is $\mathcal{O}(mm'n + mn + 2m) = \mathcal{O}(n|PA|)$. □

5 Conclusion

The main contribution of this work is a general finite automata based approach to modelling of two-dimensional pattern matching problems. Based on the generic algorithm, two particular methods have been presented, one for 2D exact pattern matching and one for 2D approximate pattern matching using the 2D Hamming distance. In practice these are the most important kinds of 2D pattern matching, because most of the pictures stored in and by computers are rectangular in shape. However, there are two-dimensional edit distances, like R, C, L, RC [16], and we wish to find suitable automata models for them, too.

Beside automata based models we have dealt with issues in implementing them. In general it is impossible to use a simulation of nondeterministic models and obtain linear time complexity. Yet, there exist direct construction methods of equivalent deterministic pattern matching automata, and with the use of these our method is able to work in linear time, too. Moreover, we have shown a way to reduce the space needed to the size of only one row of the text array.

Then we presented the model of two-dimensional approximate pattern matching using the 2D Hamming distance. It has no known direct deterministic implementation, but it is possible to simulate its pattern matching automata.

The main point of our work is that it reuses a great deal of pattern matching automata in a new area of application. We offer a systematic approach for describing two-dimensional pattern matching.

References

1. Amir, A.: Theoretical issues of searching aerial photographs: a bird's eye view. In Balík, M., Holub, J., Šimánek, M., eds.: Proceedings of the Prague Stringology Conference 2004, Czech Technical University in Prague, Czech Republic (2004) 1–23

2. Crochemore, M., Hancart, C.: Automata for matching patterns. In Rozenberg, G., Salomaa, A., eds.: Handbook of Formal Languages. Springer-Verlag, Berlin (1997) 399–462
3. Melichar, B.: Approximate string matching by finite automata. In Hlaváč, V., Šára, R., eds.: Computer Analysis of Images and Patterns. Number 970 in Lecture Notes in Computer Science, Springer-Verlag, Berlin (1995) 342–349
4. Melichar, B., Holub, J.: 6D classification of pattern matching problems. In Holub, J., ed.: Proceedings of the Prague Stringology Club Workshop '97, Czech Technical University in Prague, Czech Republic (1997) 24–32
5. Giammarresi, D., Restivo, A.: Two-dimensional languages. In: Handbook of Formal Languages. Volume III (Beyond Words). Springer-Verlag, Heidelberg (1997) 216–267
6. Aho, A.V., Corasick, M.J.: Efficient string matching: an aid to bibliographic search. Commun. ACM **18** (1975) 333–340
7. Žďárek, J., Melichar, B.: Finite automata and two-dimensional pattern matching. In Heričko, M., Rozman, I., Jurič, M.B., Rajkovič, V., Urbančič, T., Bernik, M., Bučar, M., Brodnik, A., eds.: Proceedings of the 7th International Multiconference Information Society IS'2004. Volume D., Ljubljana, Slovenia, Institut "Jožef Stefan" (2004) 185–188
8. Cambouropoulos, E., Crochemore, M., Iliopoulos, C.S., Mouchard, L., Pinzon, Y.J.: Algorithms for computing approximate repetitions in musical sequences. In Raman, R., Simpson, J., eds.: Proceedings of the 10th Australasian Workshop On Combinatorial Algorithms, Perth, WA, Australia (1999) 129–144
9. Sellers, P.H.: The theory and computation of evolutionary distances: Pattern recognition. J. Algorithms **1** (1980) 359–373
10. Wu, S., Manber, U.: Fast text searching: allowing errors. Commun. ACM **35** (1992) 83–91
11. Holub, J.: Simulation of nondeterministic finite automata in pattern matching. Dissertation thesis, Czech Technical University in Prague, Czech Republic (2000)
12. Knuth, D.E., Morris, Jr, J.H., Pratt, V.R.: Fast pattern matching in strings. SIAM J. Comput. **6** (1977) 323–350
13. Hamming, R.W.: Error detecting and error correcting codes. The Bell System Technical Journal **29** (1950) 147–160
14. Hopcroft, J.E., Ullman, J.D.: Introduction to automata theory, languages and computations. Addison-Wesley, Reading, MA (1979)
15. Ukkonen, E.: Finding approximate patterns in strings. J. Algorithms **6** (1985) 132–137
16. Baeza-Yates, R.A., Navarro, G.: New models and algorithms for multidimensional approximate pattern matching. J. Discret. Algorithms **1** (2000) 21–49