Space Complexity of Linear Time Approximate String Matching

Bořivoj Melichar

Department of Computer Science and Engineering,
Faculty of Electrical Engineering,
Czech Technical University,
Karlovo nám. 13,
121 35 Prague 2,
Czech Republic

e-mail: melichar@cs.felk.cvut.cz

Abstract. Approximate string matching is a sequential problem and therefore it is possible to solve it using finite automata. Nondeterministic finite automata are constructed for string matching with k mismatches and k differences. The corresponding deterministic finite automata are base for approximate string matching in linear time. Then the space complexity of both types of deterministic automata is calculated. Moreover, reduced versions of nondeterministic automata are taken into account and the space complexity of their deterministic equivalents is calculated.

Key words: approximate string matching, finite automata, space complexity

1 Introduction

Approximate string matching can be described in the following way:
Given a text string $T = t_1 t_2 \cdots t_n$, a pattern $P = p_1 p_2 \cdots p_m$, and an integer k, $k \leq m \leq n$, we are interested in finding all occurrences of a substring X in the text string T such that the distance $D(P, X)$ between the pattern P and the string X is less than or equal to k. In this paper we will consider two types of distances called Hamming distance and Levenshtein distance.

The Hamming distance, denoted by D_H, between two strings P and X of equal length is the number of positions with mismatching symbols in the two strings. We will refer to approximate string matching as *string matching with k mismatches* whenever D is the Hamming distance. The Levenshtein distance, denoted by D_L, or edit distance, between two strings P and X, not necessarily of equal length, is the minimal number of editing operations **insert**, **delete** and **replace** needed to convert P into X. We will refer to approximate string matching as *string matching with k differences* whenever D is the Levenshtein distance.

Approximate string matching is a sequential problem and therefore it is possible to solve it using finite automata. Two variants of nondeterministic finite automata are constructed for string matching with k mismatches and for string matching with k differences ([Me95], [Me96]).

There are two ways how to use these automata as a base for the matching algorithm:

1. To simulate the nondeterministic automaton in a deterministic way.

2. To construct an equivalent deterministic automaton.

Several known algorithms use simulation of nondeterministic automata in a deterministic way [BG92], [MW92], [Uk85], [WM92]. The simulation leads to the time complexity which is greater than linear. The only exception are SHIFT-OR based algorithms ([BG92], [WM92]) which simulate the nondeterministic automata in linear time for small m and k using bit vectors. The advantage of the simulation of nondeterministic automata is the low space complexity.

Use of deterministic finite automata leads to the linear time complexity for all m and k. The drawback of this approach is a high expected space complexity. Therefore we try to find the space complexity of deterministic finite automata for matching which is less pessimistic than in [Uk95].

A nondeterministic finite automaton (NFA) is a 5 - tuple $M = (Q, A, \delta, q_0, F)$, where Q is a finite set of states, A is a finite set of input symbols, δ is a state transition function from $Q \times (A \cup \{\varepsilon\})$ to the power set of Q, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states.

A finite automaton is deterministic (DFA) if $\delta(q, a)$ has exactly one element for any $q \in Q$ and $a \in A$ and $\delta(q, \varepsilon) = \emptyset$ for any $q \in Q$.

In the following, we will use the alphabet $A = \{s_1, s_2, \cdots, s_{|A|}\}$.
If $p \in A$ then $\bar{p}$ is the complement set $A - \{p\}$, in our case.

2 String Matching with k Mismatches

First, we construct a nondeterministic finite automaton M_H for a given pattern $P = p_1 p_2 \cdots p_m$, alphabet $A = \{s_1, s_2, \cdots, s_{|A|}\}$, and $k \leq m$. This automaton is depicted in Fig. 1.

Each state $q \in Q$ has a label (i, j), where $i, 0 \leq i \leq k$, is a level of q, and $j, 0 \leq j \leq m$, is a depth of q. In the automaton M_H, there are $k + 1$ levels of states sequences. Every level ends in one of the final states $(0, m), (1, m), \cdots, (k, m)$. These final states are accepting states of strings with $0, 1, 2, \cdots, k$ mismatching symbols, respectively. The sequence of states of the level 0 corresponds to the given pattern without any mismatch. Levels $1, 2, \cdots, k$ correspond to the strings with $1, 2, \cdots, k$ mismatching symbols, respectively. From each nonfinal state of level j, $0 \leq j < k$, there exists a transition to the state of the level $j + 1$, which means, that a mismatch occurs. Moreover, there is a self loop in the state $(0, 0)$ for every symbol of the alphabet A. This automaton accepts all strings having a postfix X such that $D_H(P, X) \leq k$. The number of states of the automaton M_H is
$(k + 1)(m + 1 - \frac{k}{2}) = (m + 1) + (m) + (m - 1) + \cdots + (m - k + 1)$.

Because this finite automaton is nondeterministic, it is necessary to construct an equivalent deterministic finite automaton (DFA_H) using the standard algorithm [AU71,2].

Let us use the number of items of the transition table of DFA_H as a measure of the space complexity of the algorithm of string matching with k mismatches. This

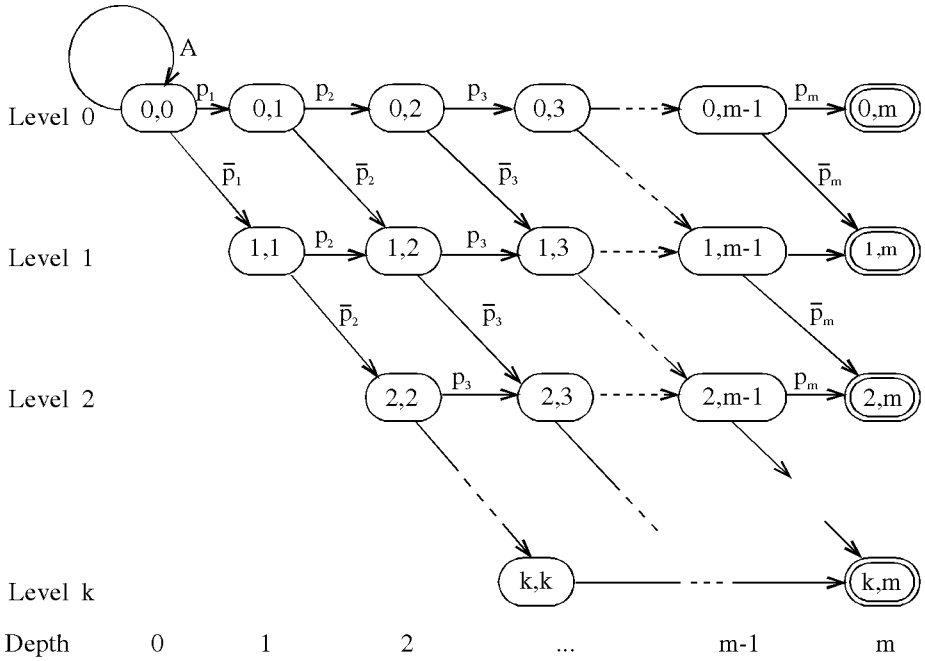

Figure 1: Nondeterministic finite automaton M_H.

number of items is the number of states (number of rows) of DFA_H multiplied by number of columns.

For the evaluation of the number of states of the DFA_H in question we will use the following lemma.

Lemma 1 *Let M_H be a nondeterministic finite automaton for given pattern $P = p_1p_2\cdots p_m$, $k \leq m$ (see Fig. 1). Let DFA_H be the deterministic finite automaton constructed by the standard algorithm for M_H. Then each state of the automaton DFA_H contains at most one state of M_H with depth j, $0 \leq j \leq m$.*

Proof. The standard construction of the deterministic finite automaton equivalent to the nondeterministic one is based on the "parallel simulation" of the nondeterministic automaton.

The assertion of the lemma can be formulated in this way:

(∗) The nondeterministic automaton M_H can reach at most one state at each depth during parallel simulation.

This assertion can be proven by induction on the length of input string l, $0 \leq l \leq n$. For $l = 0$ assertion (∗) holds, because M_H is in state $(0,0)$. Let us assume, that assertion (∗) is true for all $l \leq n'$. That means that M_H is in some number of states $(i_1,j_1),(i_2,j_2),\cdots,(i_q,j_q)$, where all $j_r, 0 \leq r \leq q$, are different. For the state (i_r,j_r), $j_r < m, i_r < k$, there are two possible transitions:

1. to the state (i_r, j_{r+1}) in case when the input symbol matches the symbol p_{r+1} of the pattern,

2. to the state (i_{r+1}, j_{r+1}) in case when no match occurs.

For the state $(0,0)$ there is moreover the selfloop. For the state (i_r, j_r), where $i_r = k$, there is possible only transition when match occurs. For the state (i_r, j_r), where $j_r = m$, there is no transition possible. From this follows that at most one state in the depth j_{r+1} will be reached from each state (i_r, j_r) and the assertion holds. This completes the proof of the lemma.

□

From the Lemma 1 follows the method of computation of the maximum number of states of the deterministic automaton DFA_H.

There are $p+1$ states in each depth p of the nondeterministic automaton M_H for $0 \leq p \leq k-1$. Moreover, there are $k+1$ states in M_H for each depth p, $k \leq p \leq m$.

The number of subsets of these states can be computed as a product of numbers of states of all depths between 1 and m increased by one, for the case, when no state of particular depth is present in the subset. Therefore the maximum number of states of the deterministic automaton DFA_H is

$$3 * 4 * \cdots * (k+1) * (k+2)^{m-k+1} = \frac{(k+1)!}{2} * (k+2)^{m-k+1}.$$

The number of states of DFA_H is

$$\mathcal{O}((k+1)! * (k+2)^{m-k+1}).$$

For the computation of the number of columns of the transition table of the DFA the following lemma is useful.

Lemma 2 *Let $P = p_1 p_2 \cdots p_m$ be a pattern. Let $M_H = (Q, A, \delta, q_0, F)$ be nondeterministic automaton for P and $k \geq 0$. Let $X = \{x : x \in A, x \neq p_i, 1 \leq i \leq m\}$. Then for a deterministic automaton $DFA_H = (Q_D, A, \delta_D, q_{0D}, F_D)$ constructed for M_H holds: for all $q \in Q_D$ exists $p \in Q_D$ such that $\delta_D(q, x) = p$ for all $x \in X$.*

Proof. The set X is a subset of A containing symbols not used in the pattern P. The automaton M_H has for all symbols $x \in X$ identical columns in the transition table. Thus $\delta(q, x) = \{p_1, p_2, \cdots, p_r\}$ and $\delta(q, y) = \{p_1, p_2, \cdots, p_r\}$ holds for all $q \in Q$ and all pairs $x, y \in X$. Due to the construction of the deterministic automaton, for $q \in Q_D$ and $q = \{q_1, q_2, \cdots, q_S\}$, it holds

$$\delta_D(\{q_1, q_2, \cdots, q_S\}, x) = \bigcup_{i=1}^{s} \delta(q_i, x)$$

and

$$\delta_D(\{q_1, q_2, \cdots, q_S\}, y) = \bigcup_{i=1}^{s} \delta(q_i, y).$$

Because
$\delta(q_i, x) = \delta(q_i, y)$, $1 \leq i \leq S$, then $\delta_D(\{q_1, q_2, \cdots, q_S\}, x) = \delta_D(\{q_1, q_2, \cdots, q_S\}, y)$.

□

From this lemma the consequence follows: If the pattern has length m then no more than m different symbols from an alphabet A may appear in it. For all other symbols, both deterministic and nondeterministic automata behave in the same way. It means, that the subset $X \subset A$ of symbols not present in the pattern may be replaced by some $x \in X$ and the size of alphabet will be $m+1$.

From it follows, that the space complexity of the deterministic automaton DFA does not depend on the size of alphabet if it is large enough.

It follows from this discussion, that the number of columns of the transition table of DFA is $a = min(|A|, m+1)$.

The total space complexity of deterministic automaton DFA_H is:

$$\mathcal{O}((k+1)!(k+2)^{m-k+1} * min(|A|, m+1)).$$

The nondeterministic automaton M_H can be reduced as described in [Ho96]. This reduction leads to the nondeterministic automaton RM_H having just $(m+1-k)$ states at each level. States

$$\begin{array}{l}(0, m-k), (0, m-k+1), \cdots, (0, m),\\ (1, m-k+1), (1, m-k+2), \cdots, (1, m),\\ \vdots\\ (k-1, m)\end{array}$$

can be removed when we need not know the number of mismatches in the found string. Moreover states $(0, m-k+1), (1, m-k), \cdots, (k, m)$ will be final states.

Because Lemma 1 is valid for RM_H as well as for M_H, it is possible to use similar approach for computation of maximum number of states of deterministic finite automaton $RDFA_H$ constructed for RD_H. Let us assume $k \leq \frac{m}{2}$. In this case the automaton RM_H has $p+1$ states in each depth p for $0 \leq p \leq k+1$. There are $k+1$ states in each depth $p, k \leq p \leq m-k-1$. Moreover, there are $m-p+1$ states in RM_H for each depth p, $m-k+1 \leq p \leq m$.

From this follows, that the maximum number of states of the reduced deterministic automaton $RDFA_H$ is

$$NS(RDFA_H) = 3 * 4 * \cdots * (k+1) * (k+2)^{m-2k+1} * (k+1) * \cdots * 3 * 2 =$$

$$= \frac{1}{2}((k+1)!)^2 * (k+2)^{m-2k+1}..$$

If $\frac{m}{2} < k < m$ then the situation is different. In this case the maximal number of states of RM_H in one depth is lower than $k+1$ and it is equal to $m-k+1$. The expression $NS'(RDFA_H)$ for the evaluation of number of states of deterministic automaton has the form:

$$NS'(RDFA_H) = 3 * 4 * \cdots * (m-k+1) * (m-k+2)^{2k-m+1}(m-k+1) * \cdots * 3 * 2 =$$

$$= \frac{1}{2}((m-k+1)!)^2 * (m-k+2)^{2k-m+1}$$

Because Lemma 2 is also valid for reduced automaton RM_H, the space complexity of the reduced deterministic finite automaton $RDFA_H$ is:

$$\mathcal{O}(((k+1)!)^2 * (k+2)^{m-2k+1} * min(|A|, m+1)),$$

when $k < \frac{m}{2}$ and

$$\mathcal{O}(((m-k+1)!)^2 * (m-k+2)^{2m-k+1} * min(|A|, m+1))$$

when $\frac{m}{2} < k < m$.

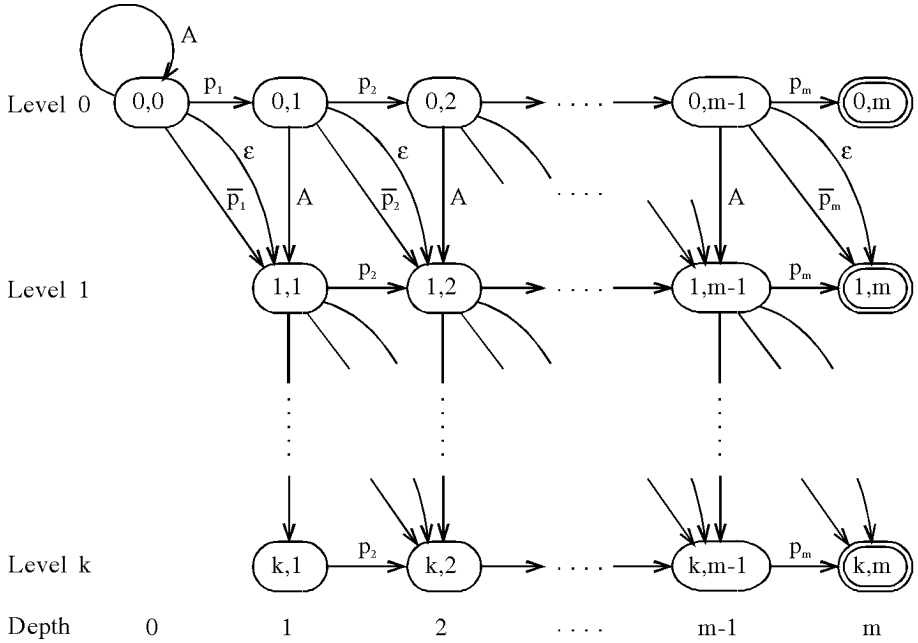

Figure 2: Nondeterministic finite automaton M_L.

3 String Matching with k Differences

We will construct a nondeterministic finite automaton M_L for a given pattern $P = p_1 p_2 \cdots p_m$, alphabet $A = \{s_1, s_2, \cdots, s_{|A|}\}$, and $k \leq m$. This automaton is depicted in Fig. 2. Each state $q \in Q$ has a label (i, j), where $i, 0 \leq i \leq k$, is the level of q, and $j, 0 \leq j \leq m$, is the depth of q.

The automaton is composed of $k+1$ levels of state sequences. Every level ends in one of the final states $(0, m), (1, m), \cdots, (k, m)$ which accept strings with $0, 1, \cdots, k$ differences, respectively. In each level, with exception of the level 0, there are m states with depth $1, 2, \cdots, m-1, m$, where the depth of a state is its "distance" from the state $(0, 0)$ of level 0. In the level 0, there are $m+1$ states and the state $(0, 0)$ has the depth equal to 0.

The transitions between adjacent levels correspond to the edit operations **insert**, **replace** and **delete** in the following way:

1. The transitions corresponding to the operation **insert** are "vertical" transitions from each nonfinal state of level j, $0 \leq j < k$, with the exception of the initial state, to the state of level $j+1$ with the same depth for all symbols of the alphabet A.

2. The transitions corresponding to the operation **replace** are "diagonal" transitions from each nonfinal state (i, j) of level j, $0 \leq j < k$ to the state $(i+1, j+1)$ of level $j+1$. The label of such transition is the complement of the label of transition from state (i, j) to state $(i, j+1)$.

3. The transitions corresponding to the operation **delete** are "diagonal" ε-transitions from each nonfinal state (i, j) of level j, $0 \leq j < k$, and depth less than m to the state $(i+1, j+1)$ of the level $j+1$.

Finally, there are self loops in the state $(0,0)$ for all symbols of the alphabet A. This automaton accepts all strings with postfix X such that $D_L(P,X) \leq k$. The automaton has $m(k+1)+1$ states.

As in the case of the automaton for mismatching problem, we will construct an equivalent deterministic automaton DFA_L for M_L.

The approach we use for evaluation of the space complexity of the DFA_L is based on the notion of ε-diagonals. Because we can leave out ε-diagonals below the initial ε-diagonal starting with state $(0,0)$ for reasons described in [Ho96], the nondeterministic automaton M_L contains $(m+1)$ ε-diagonals containing the following states:

number of diagonal	set of states	number of states
0	$(0,0),(1,1),\cdots,(k,k)$	$k+1$
1	$(0,1),(1,2),\cdots,(k,k+1)$	$k+1$
2	$(0,2),(1,3),\cdots,(k,k+2)$	$k+1$
$\vdots$		
$m-k$	$(0,m-k),(1,m-k+1),\cdots,(k,m)$	$k+1$
$\vdots$		
$m-2$	$(0,m-2),(1,m-1),(2,m)$	3
$m-1$	$(0,m-1),(1,m)$	2
m	$(0,m)$	1

For the computation of number of states of the equivalent deterministic automaton DFA_L for M_L we will use the following lemma.

Lemma 3 *Let M_L be a nondeterministic finite automaton for given pattern $P = p_1p_2\cdots p_m$, $k \leq m$ (see Fig. 2). Let DFA_L be the deterministic finite automaton constructed by the standard algorithm for M_L. Then each state of the automaton DFA_L contains at most one state of M_L from each ε-diagonal.*

Proof. The proof is based on the observation, that if automaton M_L reaches some state (p,q) at the ε-diagonal d, then, due to ε-transitions, it reaches all next states $(p+1,q+1),(p+2,q+2),\cdots,(p+k,q+k)$ of the same ε-diagonal. Therefore we can select such state (p,q) of each diagonal, where the level p is minimal, as a "representative" of the set of all next states at the same ε-diagonal.

□

The number of states of deterministic automaton DFA_L for M_L we can compute as a product of the number of states at diagonals $1,2,\cdots,m$ increased by one because some states of DFA_L may contain no state from particular diagonal. The diagonal 0 plays special role, because automaton M_L is always in the state $(0,0)$ due to the selfloop in the initial state.

Because the number of "full" ε-diagonals having length $k+1$ others than diagonal 0 is $m-k$ and there is k "short" diagonals having length $k,k-1,\cdots,1$, respectively, the number of different subsets of the representatives (the maximum number of states of DFA_L) is given by

$$NS(DFA_L) = (k+2)^{m-k} * \frac{(k+1)!}{2}$$

Using the previous result on number of rows of transition table we can express the space complexity of the deterministic finite automaton for approximate string matching with k-differences as:

$$\mathcal{O}((k+2)^{m-k} * (k+1)! * min(|A|, m+1)).$$

The nondeterministic automaton M_L can also be reduced as described in [Ho96]. This reduction leads to the reduced automaton RM_L having "full" ε-diagonals only. The number of states of reduced deterministic automaton $RDFA_L$ we can express as

$$NS(RDFA_L) = (k+2)^{m-k}$$

and the space complexity of the reduced deterministic automaton for approximate string matching with k-differences is

$$\mathcal{O}((k+2)^{m-k} * min(|A|, m+1)).$$

4 Conclusion

The main result presented here is upper bound of space complexity of four variants of deterministic finite automata for approximate string matching. While the number of states of nondeterministic finite automata is $O(k * m)$ in all cases, the number of states of corresponding deterministic automata is much lower then $O(2^{k*m})$. In the case of string matching with k differences, the presented space complexity is still pessimistic and the computing of more realistic upper bound is open problem.

References

[AU71,2] Aho, A., Ullman, J,: The theory of parsing, translation and compiling. Vol. I: Parsing, Prentice Hall, Englewood Cliffs, New York 1972.

[BG92] Baeza-Yates, R., Gonnet, G. H.: A new approach to text searching. Communications of the ACM, October 1992, Vol. 35, No. 10, pp. 74 – 82.

[Ho96] Holub, J.: Reduced nondeterministic finite automata for approximate string matching. In this volume.

[Me95] Melichar, B.: Approximate string matching by finite automata. Computer Analysis of Images and Patterns, LNCS 970, Springer, Berlin 1995, pp. 342 – 349.

[Me96] Melichar, B.: String matching with k differences by finite automata. Proceedings of the 13th ICPR, Vol. II, August 1996, pp. 256 – 260.

[MW92] Manber, U., Wu, S.: Approximate pattern matching. BYTE, November 1992, pp. 281 – 292.

[Uk85] Ukkonen, E.: Finding approximate patterns in strings. Journal of algorithms 6, 1985, pp. 132 – 137.

[WF74] Wagner, R., A., Fischer, M., J.: The string-to-string correction problem. Journal of the ACM, January 1974, Vol. 21, No. 1, pp. 168 – 173.

[WM92] Wu, S., Manber, U.: Fast text searching allowing errors. Communications of the ACM, October 1992, Vol. 35, No. 10, pp. 83 – 91.