Reduced Nondeterministic Finite Automata for Approximate String Matching

Jan Holub

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

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

Abstract. We will show how to reduce the number of states of nondeterministic finite automata for approximate string matching with k mismatches and nondeterministic finite automata for approximate string matching with k differences in the case when we do not need to know how many mismatches or differences are in the found string. Also we will show impact of this reduction on Shift-Or based algorithms.

Key words: finite automata, approximate string matching

1 Introduction

Problem of 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 article 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. Clearly the Hamming distance is a special case of the Levenshtein distance in which we submit only replace as an editing operation.

A nondeterministic finite automaton (NFA) is a quintuple $M = (Q, A, \delta, q_0, F)$, 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

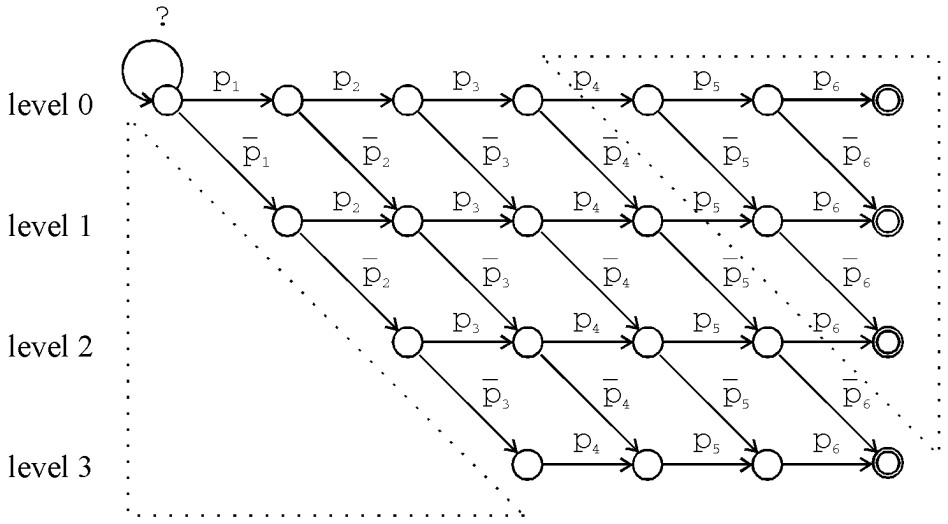

Figure 1: *NFA* for approximate string matching with k mismatches.

the set of final states. 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. A question mark ? will represent any character of the alphabet.

2 String Matching with k Mismatches

2.1 Nondeterministic Finite Automaton

A *NFA* for string matching with k mismatches for pattern P, as it was presented in [Me95] and [Da92], is shown in Figure 1. In this figure $m = 6$ and $k = 3$. The sequence of states of the level 0 of the *NFA* contains states that correspond to the given pattern without any mismatch, the sequence of states of the level 1 contains states that correspond to the given pattern with one mismatch, ...etc. At the end of each level there is a final state. This state says that the pattern was found with $0, 1, \ldots$ etc. mismatches, respectively.

There are three kinds of transition:

A transition representing matching character in pattern P and character in the text T, $t_i = p_j$. In Figure 1 this transition is marked by the arrow directed to the right. It leads to a state with the same number of mismatches as the old state, to a state of the same level.

A transition representing an editing operation *replace*, $t_i = \overline{p_j}$. In Figure 1 this transition is marked by the arrow directed to the right-down, it leads to a state with a number of mismatches one greater than the former state, to a state on one level lower.

A transition representing that the *NFA* always stays in the initial state. In Figure 1 this transition is marked by the self loop in the level 0.

This *NFA* accepts all strings having a postfix X such that $D_H(P, X) \leq k$.

The number of states of *NFA* for string matching with k mismatches is $(k+1)(m+1-\frac{k}{2})$.

2.2 Reducing the number of states

There can be some situations in which we want to know all occurrences of the given pattern in the input text with at most k mismatches but we are not interested in knowing the number of mismatches in the found string. The states that are needed only to recognize how many mismatches are in the found string, form a right angle triangle in upper right corner of the *NFA*, as marked by the dotted line in Figure 1. On the opposite side of the *NFA* there is the complement triangle of missing states. If we omit these two triangles we obtain a simplified *NFA* with $(k+1)(m+1-k)$ states.

Now the final states have been changed. If the *NFA* is in the final state at the end of level j it means that the pattern P can be found with at least j and at most k mismatches. A problem appears at the end of the input text. If less than $k-j$ characters remain in the input text then the final state of level j in the *NFA* does not mean that the pattern can be found with at least j and at most k mismatches, because the transition for either mismatch or replace needs to read one input character. So if the *NFA* is in final state at the end of level j and the position in input text i is at most $n-k+1$ it means that the pattern P can be found with at least j and at most k mismatches.

2.3 Shift-Or Algorithm

The initial version of this algorithm was presented in [BG92]. There are also modifications of this algorithm called Shift-Add [BG92] and Shift-And [WM92].

The Shift-Or algorithm uses m bit state vectors R^j which represent rows of the *NFA*, as it was presented in [Ho96], and a mask table D that for each character has an m bit vector in which bits corresponding to positions of the character in the pattern are set to 0 and other bits are set to 1. This table is used for operation matching. If the *NFA* is in a state (i,j), where i, $0 \leq i \leq m$, is a depth of state in the *NFA* and j, $0 \leq j \leq k$, is a level of state, then i-th bit of the vector R^j contains 0. If the *NFA* is not in a state (i,j), i-th bit of the vector R^j contains 1. The right angle triangle in lower left corner of the *NFA*, as marked by the dotted line in Figure 1, can be defined as the first j bits of each vector R^j and in the vectors R^j it is represented by 0s.

The vector R^0 is defined by formula $R^0_{i+1} = shl(R^0_i) or (D[t_{i+1}])$, where R^0_i is the old value and R^0_{i+1} is a new value of R^0 corresponding to position i in the text T, and represents exact string matching. *or* is bitwise operation OR and *shl* is bitwise operation left shift, that moves bits of the vector to the left and fills the last bit of the vector with an 0. Vectors for approximate pattern matching with k mismatches are defined by the formula $R^j_{i+1} = (shl(R^j_i) or (D[t_{i+1}])) and (shl(R^{j-1}_i))$, where j denotes the number of substitutions. At the beginning of the search the vectors R^j are filled up by 1s.

The fact, that the pattern has been found with at most j mismatches in position i, is detected by appearing 0 at the end of the vector R^j_i.

The Shift-Or algorithm computes new states of the *NFA* in a parallel way. It computes whole sequence of states of one level at once. The bitwise operation $shl(R^j_i)$ moves all the states of one level to the left and inserts 0 in first position. It represents the transition of matching, each state moves to the state corresponding to the next position in the pattern P. This operation is only for matching, so we have to eliminate states that do not match. That is performed by the bitwise operation $or(D[t_{i+1}])$,

that takes the mask vector corresponding to character t_{i+1} in the text T and replaces all 0s in mismatching positions by 1s. The result of this operation is that only states in matching positions have been moved to the next states of the same level.

The second item of the above formula is $shl(R_i^{j-1})$. The item represents the editing operation *replace*. It takes the sequence of states of level $j-1$ corresponding to $j-1$ mismatches, moves it one position to the left and makes the bitwise operation *and* between result of the first item of the formula and this sequence of states. Here, the bitwise operation $and(shl(R_i^{j-1}))$ adds states coming from the level corresponding to one less mismatches than in level j. It is clear that this item has no meaning for the states of the sequence of states without mismatches. Thus this item is present only in formulae for computing vectors R^j, where $0 < j \leq m$. To make Shift-Or algorithm faster this bitwise operation representing *replace* is performed even in the positions matching the input character. It simplifies the formula without change of behaviour of Shift-Or algorithm.

2.4 Simplified Shift-Or Algorithm

The reduced *NFA* can be described by vectors R^j, that are k bit shorter than the original ones. Of course, the formulae for computation of new vectors R^j have changed too. The new formulae are $R_{i+1}^0 = shl(R_i^0) or (D_j[t_{i+1}])$ for exact matching and $R_{i+1}^j = (shl(R_i^0) or (D_j[t_{i+1}])) and\ R_i^{j-1}$ for j mismatches. The mask table D_j is a part of mask table D being $m-k$ bit long and starting at position j in D. There are two ways how to represent mask tables D_j. The first way is to compute needed column of this mask table by shifting the column of the original mask table D whenever it is needed and another is to store shifted mask tables D_j, $0 \leq j \leq k$. The first way has higher time complexity and the second has higher space complexity.

A problem similar to the problem that appears at the end of the input text described above appears at the beginning of the input text. The previous problem has appeared because of omitting states in a *NFA* and this problem has appeared because of omitting first j bits of vector R^j. If the input text starts with a string that is equal to the last $m-k$ characters of the pattern then the vector R^k will report found pattern with at most k mismatches after reading $m-k$ input character, but it is clear that the pattern can be found after reading at least m input characters.

Because of the problems at the beginning of the input text and at the end of the input text we can say that the vector R_i^j can report that the pattern can be found with at most k mismatches only if $m-k+j \leq i \leq n-k+j$.

This simplification reduces the length of the state vectors R^j and simplifies the formula for computation of the state vectors with one or more mismatches. One operation shl is omitted, but on the other hand a new operation appears. This operation is shift of one column of the mask table D. This operation can be omitted too, if we accept higher space complexity of characteristic vector representation.

3 String Matching with k Differences

In string matching with k differences there are two new editing operations. The new editing operations are *insert* and *delete*. The operation *insert* puts some character in a text and the operation *delete* removes some character from a text. It is clear that

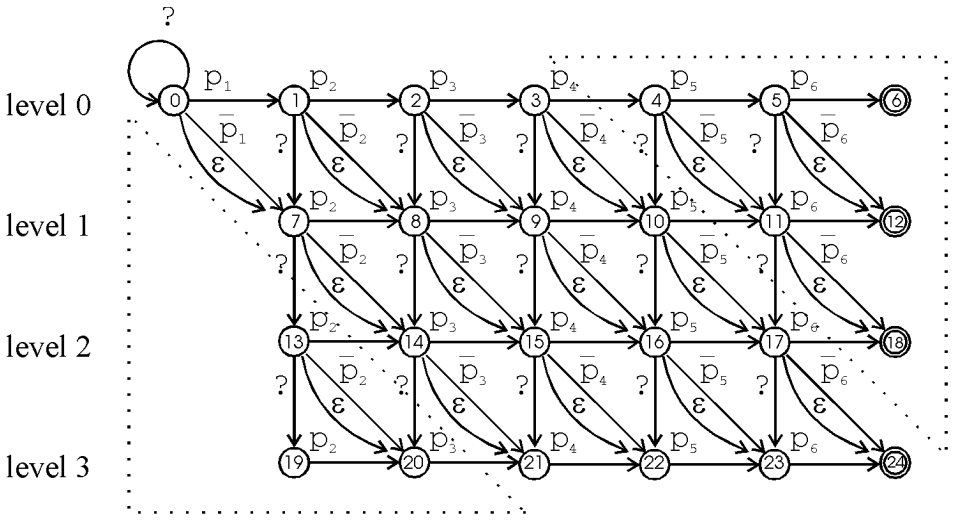

Figure 2: *NFA* for approximate string matching with k differences.

after adding these two editing operations into the set of editing operations, the string found with k differences need not be of the same length as the pattern P.

3.1 Nondeterministic Finite Automaton

A *NFA* for string matching with k differences for the pattern $P = p_1 p_2 \cdots p_m$, as it was presented in [Me96-1], is shown in Figure 2. In this figure $m = 6$ and $k = 3$. The sequence of states of level 0 of the *NFA* contains states that correspond to the given pattern without any differences, the sequence of states of level 1 contains states that correspond to the given pattern with one difference, ...etc. At the end of each level there is a final state. This state says that the pattern was found with 0,1,...etc. differences, respectively. There are two new arrows. The first is directed to the down and represents editing operation *insert*. The second one is directed to the right-down and represents ε transition of editing operation *delete*. This *NFA* accepts all strings having a postfix X such that $D_L(P, X) \leq k$.

The number of states of the *NFA* for string matching with k differences is $m * (k + 1) + 1$.

The initial state of *NFA* in Figure 2 is state 0, but also, as presented in [HU79], all states to which *NFA* can move from the initial state without reading any input character are also initial states. So initial state includes all states that are located on the diagonal starting from the state 0. Since *NFA* is all the time also in initial state it is in states 0, 7, 14 and 21. At the beginning of the *NFA* there are several states bordered by the dotted line. The *NFA* moves to all these states after reading the first $k - 1$ input characters. Then the *NFA* stays all the time also in these states but the *NFA* can move from these states only in initial states and so these states are redundant.

The *NFA* can move to these states by transitions representing editing operation *insert*. So these states represent situations that at most $k-1$ characters were inserted before the string and we do not care how many characters were inserted before the string.

If we denote editing operation *replace* r, *insert* i, *delete* d and matching m we can

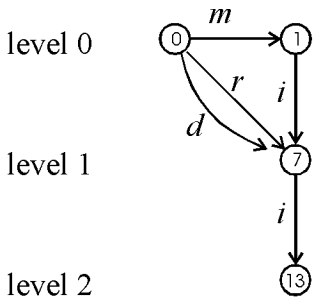

Figure 3: The transitions in *NFA* for approximate string matching with k differences.

describe ways how to move to these states by these four characters. For example we can move to the state numbered by 13 by following sequences of operations: $d + i$ or $r + i$ or $m + i + i$ as it is shown in Figure 3.

The sequence of operations *delete* and *insert* has the same result as operation *replace* so we can write $d + i = r$. The sequence of operations *replace* and *insert* has the same result as sequence of operations *insert* and *replace* so we can write $r + i = i + r$. The sequence of operations *matching* and *insert* leading from some of initial states has the same result as operations *insert* and *replace* so we can write $m + i = i + r$.

Now we can rewrite the sequences of operations how to move to state numbered by 13: $d + i = r$, $r + i = i + r$, $m + i + i = i + r + i = i + i + r$. If we look at these sequences of operations we can see that all the sequences of operations needed to move to the state numbered by 13 can be replaced by operation *replace* because we do not care about the characters inserted before the string so we do not need the state numbered by 13.

Such replacing of sequences of operations can be done for all states bordered by the dotted line so all states inside the bordered area are redundant and can be omitted.

The *NFA* for approximate string matching with k differences reduced by the way described above has the same number of states as the *NFA* for approximate string matching with k mismatches and it is $(k+1)(m+1-\frac{k}{2})$ states, as shown in section 2.1. It is clear that only reduced *NFA* for approximate string matching with k differences, that have $k > 1$, have less states then nonreduced *NFA*.

The *NFA* for approximate string matching with k differences reduced as described above is shown in Figure 4.

3.2 Reducing the number of states

The simplification described in section 2.2 for approximate string matching with k mismatches can be applied to approximate string matching with k differences as well.

The states that are needed only to recognize how many differences are in a found string, form also a right angle triangle in upper right corner of the *NFA*, as marked by the dotted line in Figure 4. On the opposite side of the *NFA* there is the complement triangle of states omitted because of the reduction. If we omit these two triangles we obtain a simplified *NFA* with $(k + 1)(m + 1 - k)$ states.

Now final states have also been changed. If the *NFA* is in final state at the end

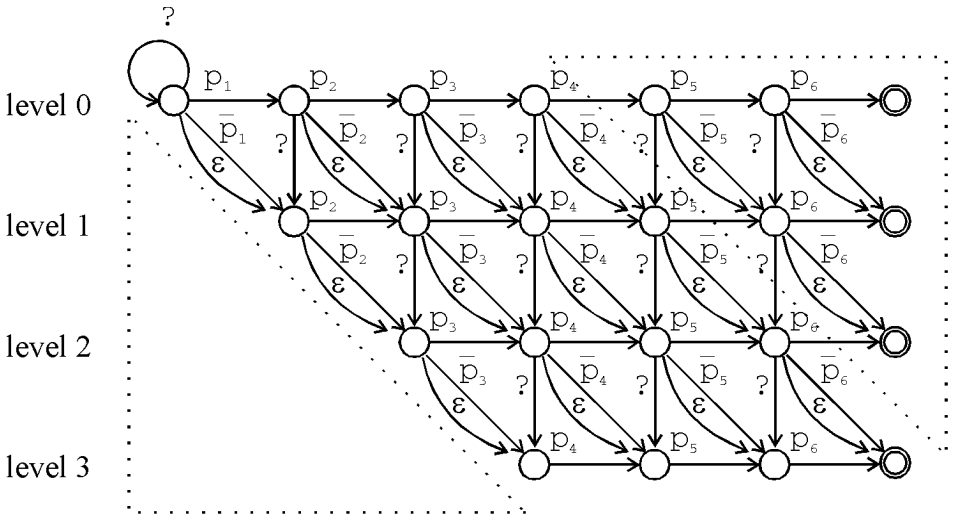

Figure 4: Reduced *NFA* for approximate string matching with k differences.

of level j it means that the pattern can be found with at least j and at most with k differences. The problem that appears at the end and at the beginning of the input text in approximate string matching with k mismatches does not exist in the case of approximate string matching with k differences. It is due to ε transitions that the *NFA*, to move from one state to another, does not need to read any input character. ε transitions represent editing operation *delete* so we can delete the first k characters or the last k characters of the pattern. That is why this simplification does not need the limits used in case of approximate string matching with k mismatches.

3.3 Shift-Or Algorithm

The Shift-Or algorithm for string matching with k differences has two new items in formulae for computing vectors R^j. The new items of the formula are $shl(R^{j-1}_{i+1})$, representing a transition of the editing operation *delete*, and R^{j-1}_i, representing a transition of editing operation *insert*. The formula for string matching with k differences is $R^j_{i+1} = (shl(R^j_i) or (D[t_{i+1}])) and (shl(R^{j-1}_i)) and (shl(R^{j-1}_{i+1})) and (R^{j-1}_i)$. It can be reduced to $R^j_{i+1} = (shl(R^j_i) or (D[t_{i+1}])) and (shl(R^{j-1}_i and\ R^{j-1}_{i+1})) and (R^{j-1}_i)$. At the beginning of searching the vectors R^j are filled up in such a way, that first j bits from the left contain 0s and other bits contain 1s.

In the case of approximate string matching with k differences we do not omit any 0s at the beginning of the table as in the case of approximate string matching with k mismatches.

The third item, $shl(R^{j-1}_{i+1})$, represents the editing operation *delete*. If there is some character deleted, we have to skip it and continue behind it. In the *NFA* in Figure 4, the skipping of the deleted character is marked by ε transition and continuation is marked by transition representing matching. In the Shift-Or algorithm there are those two operations in an inverted order. At first the operation representing matching is executed. It was already executed in computing of the vector R^{j-1}_{i+1}. By that we got to the next character in the pattern P. The following operation is ε transition. It is represented by operation shl that gets us to the next character in the pattern P and to the level of the states corresponding to differences one higher. It seems the

case of deleting the first character in the pattern P was not involved. But that is only an illusion. Such a case is covered by an initial setting of the vectors R^j. At the beginning of searching the vectors R^j are filled up in such a way that the first j bits from the left contain 0s and other bits contain 1s. The initial filling up by 0s is given by the ε transitions coming from the initial state with a self loop. It means that at the beginning of searching there are j deleted characters, $0 \leq j \leq k$.

The fourth item in the formula is R_i^{j-1}, that represents the editing operation *insert*. In Figure 4, the transition representing operation *insert* is marked by an arrow directed down. It means that the state in the *NFA* stays in the same depth but moves to the level of the states corresponding to one more differences.

3.4 Simplified Shift-Or Algorithm

The new *NFA* can be described also by vectors R^j, that are k bit shorter then the original ones. The new formulae are $R_{i+1}^0 = shl(R_i^j) or (D_j[t_{i+1}])$ for exact matching and $R_{i+1}^j = (shl(R_i^j) or (D_j[t_{i+1}])) and\ R_i^{j-1} and\ R_{i+1}^{j-1} and (shr\ R_i^{j-1})$ for j differences, where shr is bitwise operation right shift. The mask table D_j is also a part of mask table D being $m-k$ bit long and starting at position j in D. The ways how to represent mask tables D_j are the same as for approximate string matching with k mismatches.

This simplification reduces the length of the state vectors R^j but does not simplify the formula for computation of the state vectors with one or more differences as in previous section.

Conclusions

In this article we have shown that in the case that we are not interested in knowing the number of errors in the found string we can reduce *NFA* for aproximate string matching. In the case of approximate string matching with k mismatches this reduction not only reduces length of vectors of Shift-Or based algorithms but also simplifies formulae for computing these vectors. In the case of approximate string matching with k differences it only reduces length of the vectors.

Another way how to use *NFA* is to transform it into deterministic finite automaton. Decrease of states in reduced deterministic finite automata is described in [Me96-2].

References

[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.

[Da92] Darabont, T.: Approximate string matching with k mismatches. Master's thesis, Czech Technical University, 1992.

[Ho96] Holub, J.: Approximate string matching in text. Master's thesis, Czech Technical University, 1996.

[HU79] Hopcroft, J. E., Ullman, J. D.: Introduction to automata theory, languages, and computation. Addison-Wesley, Reading, Massachusetts.

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

[Me96-1] Melichar, B.: String matching with k differences by finite automata. Proceedings of the 13th ICPR, Vol. II, August 1996, pp. 256 – 260.

[Me96-2] Melichar, B.: Space Complexity of Linear Time Approximate String Matching. In this volume.

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