EFFICIENT STRING MATCHING WITH k MISMATCHES

Gad M. Landau
Uzi Vishkin

June 1985
Technical Report #167

NEW YORK UNIVERSITY

Department of Computer Science
Courant Institute of Mathematical Sciences
251 Mercer Street, New York, N.Y. 10012
EFFICIENT STRING MATCHING WITH $k$ MISMATCHES

Gad M. Landau
Uzi Vishkin

June 1985
Technical Report #167

1. Department of Computer Science, School of Mathematical Sciences, Tel. Aviv University
2. Department of Computer Science, Courant Institute of Mathematical Sciences, New York University and (present address) Department of Computer Science, School of Mathematical Sciences, Tel. Aviv University. The research of this author was supported by NSF grant NSF-DCR-8318874 and ONR grant N00014-85-K-0046.
EFFICIENT STRING MATCHING WITH \( k \) MISMATCHES

Gad M. Landau\(^1\)
Uzi Vishkin\(^2\)

ABSTRACT

Given a text of length \( n \), a pattern of length \( m \) and an integer \( k \), we present an algorithm for finding all occurrences of the pattern in the text, each with at most \( k \) mismatches. The algorithm runs in \( O(k(\log m + n)) \) time.

1. INTRODUCTION

The problem of **string matching with \( k \) mismatches** is defined as follows. Suppose we are given a text of length \( n \), a pattern of length \( m \) and an integer \( k \). Find all occurrences of the pattern in the text with at most \( k \) location in which the text and the pattern have different symbols. Note that the case \( k = 0 \) is the extensively studied string matching problem. Let us mention a few notable algorithms for the string matching problem: linear time serial algorithms - [BM], [GS], [KMP], [KR] (a randomized algorithm) and [V], parallel algorithms [G] and [V]. The problem has a strong pragmatical flavor. In practice, we often need to analyze situations where the data is not completely reliable. Specifically, consider a situation where the strings which are the input for our problem contain errors and we still need to find all possible occurrences of the pattern in the text as in reality. Assuming some bound on the number of errors would clearly imply our problem.

We present an algorithm for string matching with \( k \) mismatches which runs in time \( O(k(\log m + n)) \) on a random-access-machine (RAM) [AHU].

After all the results in the present paper have been achieved, A. Slisenko has brought to our attention the paper [I] in which another algorithm for the
same problem has been given. Ivanov claims that his algorithm runs in time 
$O(f(k)(n+m))$, where $f(k)$ is a function of $k$. $f(k)$ is described by a combi-
notation of two intricate recursive inequalities. No additional hints regarding the
behaviour of $f(k)$ were found in his paper. We were unable to solve these inequali-
ties. However, we managed to show that $f(k)$ is bounded from below by $2^k$ for
every positive integer $k$. It might be that $f(k)$ grows even substantially faster
than $2^k$. His algorithm runs faster than ours only when $k$ is very small and $m$
and $n$ are almost of the same order of magnitude. In all other cases, our algo-
rum is faster. An even more important advantage of our algorithm is that it is
simple and intuitive while Ivanov's algorithm is very complicated. (Its descrip-
tion needed over 40 journal pages).

II. ANALYSIS OF THE TEXT

Our algorithm has two parts. In the first part the pattern is analyzed. The
outcome of this analysis is used in the second part for analyzing the text. The
next section describes the first part. The present section is devoted to the
second part. We show how to use the results of the pattern analysis in order to
find all occurrences of the pattern in the text with at most $k$ mismatches.

The input to the text analysis consists of the following:

a) The pattern. An array $A = a_1, \ldots, a_m$.
b) The text. An array $T = t_1, \ldots, t_n$.
c) The output of the pattern analysis. A two dimensional array
$\textit{PAT-MISMATCH}[1, \ldots, m-1; 1, \ldots, 2k+1]$. Where, row $i$ of the array
$(\textit{PAT-MISMATCH}(i,1), \ldots, \textit{PAT-MISMATCH}(i,2k+1))$, contains the $2k+1$
first locations in which $a_{i+1}, \ldots, a_m$ has different symbols than
$a_1, \ldots, a_{m-i}$. $(\textit{PAT-MISMATCH}(i,v) = f$ means that $a_{i+f} \neq a_f$ and $f$ is
the mismatch number $v$ from left to right).

If there are only $c < 2k+1$ mismatches between $a_1, \ldots, a_m$ and
$a_{1}, \ldots, a_{m-i}$ we enter the default value $m+1$ from location $c+1$ on. That is,
$\textit{PAT-MISMATCH}(i,c+1) = \ldots, \textit{PAT-MISMATCH}(i,2k+1) = m+1$.

The text is analyzed into the array $\textit{TEXT-MISMATCH}[0, \ldots, n-m; 1, \ldots, k+1]$.
Following the text analysis, row $i$ of the array $(\textit{TEXT-MISMATCH}(i,1), \ldots,$
$\textit{TEXT-MISMATCH}(i,k+1))$, contains the $k+1$ first mismatches between the
strings $t_{i+1}, \ldots, t_{i+m}$ and $a_1, \ldots, a_m$. $(\textit{TEXT-MISMATCH}(i,v) = f$ means that
If $t_{i+1} \neq a_j$ and this is mismatch number $v$ from left to right). If there are only $c < k + 1$ mismatches between $t_{i+1}, \ldots, t_{i+m}$ and $a_1, \ldots, a_m$ then we enter the default value $m + 1$ from location $c + 1$ on. That is,

$$TEXT-MISMATCH(i, c + 1) = \ldots, TEXT-MISMATCH(i, k + 1) = m + 1.$$

**Remark.** This solves our problem since $TEXT-MISMATCH(i, k + 1) = m + 1$ means that there is an occurrence of the pattern which starts at $t_{i+1}$ with at most $k$ mismatches.

We start with a very high-level specification of the algorithm. It is explained by the verbal and illustrative descriptions that follow.

**TEXT-ANALYSIS**

**Initialize:** 

$TEXT-MISMATCH[0, \ldots, n-m, i, \ldots, k + 1] := m + 1$;

$r := 0; j := 0$;

for $i := 0$ to $n-m$ do

begin

$b := 0$;

if $i < j$

then $MERGE(i, r, j, b)$;

if $b < k + 1$

then $r := i$ ; $EXTEND(i, j, b)$

end

The for loop is responsible for "sliding" the pattern to the right one place at a time. At iteration $i$, we check if an occurrence of the pattern starts at $t_{i+1}$. Suppose that $r$ is an iteration prior to $i$, $0 \leq r < i$, that maximizes $j = r + TEXT-MISMATCH(r, k + 1)$. Namely, $j$ is the rightmost index of the text to which we arrived at previous iterations of the loop. Each iteration consists of calling procedure $MERGE$, (if $i < j$), and possibly procedure $EXTEND$, (Note, that at the begining $i = 0, j = 0$, and therfore $MERGE$ is not invoked, at the first iteration). $MERGE$ finds mismatches between $t_{i+1}, \ldots, t_j$ and $a_1, \ldots, a_{j-1}$ and reports in $b$ the number of mismatches found. If $b \geq k + 1$ we proceed to the next iteration. Otherwise, $EXTEND$ scans the text from $t_{j+1}$ on till it either finds $k + 1$ mismatches or till it hits $t_{i+m}$ and finds that there is an occurrence of the pattern which starts at $t_{i+1}$ with at most $k$ mismatches. The situation is illustrated in Figure 1.

Let us explain the role that procedure $MERGE$ plays at iteration $i$ of the **TEXT-ANALYSIS**. In the previous paragraph we stated that $MERGE$ finds
mismatches between \( t_{i+1}, \ldots, t_j \) and \( a_1, \ldots, a_{j-1} \) and reports in \( b \) the number of mismatches found. That is, MERGE computes \( \text{TEXT-MISMATCH}[i;1,\ldots,b] \), \( (b \leq k+1) \). MERGE uses two kinds of data that were computed in iterations prior to \( i \) of TEXT-ANALYSIS.

(a) The mismatches with respect to (in short, w.r.t.) \( r+1 \) in the text. Obviously, such mismatches which occur in locations \( <i+1 \) in the text are irrelevant for checking whether there is an occurrence of the pattern that starts at \( t_{i+1} \). Let \( q \) be the smallest integer satisfying \( \text{TEXT-MISMATCH}[r;q] \) is greater than \( i-r \). Thus, MERGE uses \( \text{TEXT-MISMATCH}[r;q,\ldots,k+1] \). (Fig. 1(b)).

(b) \( \text{PAT-MISMATCH}[i-r;1,\ldots,s] \), where \( s \) is the rightmost mismatch in \( \text{PAT-MISMATCH}[i-r;1,\ldots,2k+1] \) such that \( \text{PAT-MISMATCH}(i-r,s) \) is less than \( (j-i+1) \). (Fig. 1(c)).

We apply a case analysis in order to understand how to use these previously computed data. We need the following two conditions for the case analysis. Consider any location \( z \) of the text, \( i+1 \leq z \leq j \). We define two conditions on \( z \).

**Condition 1.** \( z \) falls under a mismatch w.r.t. \( r \). That is, \( t_z \neq a_{z-r} \) and for some \( d \), \( (q \leq d \leq k+1) \), \( z-r = \text{TEXT-MISMATCH}(r,d) \). (This correspond to a mismatch between two locations one from the bottom line and the other from the middle line in Fig. 1(d)).

Consider laying one copy of the pattern starting at \( t_{r+1} \) and another copy starting at \( t_{r+1} \). (The upper and middle lines in Fig. 1(d)).

**Condition 2.** \( z \) falls under a mismatch between these two copies of the pattern. That is \( a_{z-r} \neq a_{z-1} \). Also, \( z-i = \text{PAT-MISMATCH}(i-r,f) \) for some \( f \), \( (1 \leq f \leq s) \).

Location \( z \) may satisfy either both conditions or any one of them or none.

We are ready now to present the case analysis for any location of the text \( z \), \( i+1 \leq z \leq j \), and how it affects the question: \( t_z = a_{z-i} \) ? (In words, does location \( z \) of the text match location \( z-i \) of the pattern?)

**Case 0.** \( z \) does not satisfy Condition 1 and \( z \) also does not satisfy Condition 2. Location \( z \) of the text must match location \( z-i \) of the pattern (\( t_z = a_{z-i} \)) and we need not bother to compare \( t_z \) and \( a_{z-i} \). (A similar argument is used in the algorithm of [KMP]).
Case 1. \( x \) satisfies one of the two conditions and does not satisfy the other. Let us justify why \( t_x \neq a_{x-i} \) in any of these two possibilities.

If Condition 1 holds and Condition 2 does not hold then \( t_x \neq a_{x-r} \) and \( a_{x-r} = a_{x-i} \). Therefore, \( t_x \neq a_{x-i} \). If Condition 1 does not hold and Condition 2 holds then \( t_x = a_{x-r} \) and \( a_{x-r} \neq a_{x-i} \). Therefore, \( t_x \neq a_{x-i} \).

So, we know that there must be a mismatch at location \( x \) and again we dispense with comparing \( t_x \) and \( a_{x-i} \). However, we do need to increase the counter of mismatches \( b \) by one and update \( TEXT-MISMATCH(i,b) \).

Case 2. \( x \) satisfies both conditions. Here we are unable to reason whether \( t_x = a_{x-i} \) or not. So, we compare these two symbols. If they are different we update \( b \) and \( TEXT-MISMATCH(i,b) \) as in Case 1.

Specifically, procedure MERGE operates as if it merges the increasing sequence of \( \leq k+1 \) locations

\[
\tau + TEXT-MISMATCH(\tau,q),...,\tau + TEXT-MISMATCH(\tau,k+1)
\]

and the increasing sequence of \( \leq 2k+1 \) locations

\[
i + PAT-MISMATCH(i-r,1),...,i + PAT-MISMATCH(i-r,s)
\]

into one increasing sequence. However, instead of explicitly merging the two sequences MERGE checks whether each location satisfies Case 1 or Case 2 and treats the location according to the case analysis given above.

Procedure MERGE \((i,r,j,b)\)

```text
Input: 1) TEXT-MISMATCH[\tau:q,...,k+1]  
2) PAT-MISMATCH[i-r:1,...,s]

Initialize: \( d := q \); \( f := 1 \)
(* The variable \( d \) will be used in the form TEXT-MISMATCH\((\tau,d)\). Initially it is \( q \) and then it is increased by one at a time. The variable \( f \) will be used in the form PAT-MISMATCH\((i-r,f)\). Initially it is \( 1 \) and then it is increased by one at a time. *)

while not [Case a or Case b or Case c] do
(* We stop iterating the while loop, and return control to TEXT-ANALYSIS, in any of the following cases:

Case a. \( b = k+1 \). This means that we have already found \( k+1 \) mismatches with respect to \( i \).
Case b. \( d = k+2 \). When \( d \) was assigned with \( k+1 \) then in the middle line we were exactly over location \( j \) of the bottom line. A careful observation at the way in which \( d \) is updated in procedure MERGE reveals that the fact that \( d \) was increased to \( k+2 \) implies that in the middle line we must have also passed location \( j \) of the bottom line, and therefore it is time to return control to TEXT-ANALYSIS and continue the search for mismatches by procedure EXTEND.
```

```
Case c. \([i + \text{PAT-MISMATCH}(i - r, f) > j\) and \(\text{TEXT-MISMATCH}(r, d) = m + 1\). The first conjunct means that in the upper line of Fig 1(d) we have already passed location \(j\) of the bottom line. The second conjunct means that there were an occurrence of the pattern at \(i_{r+1}\) with \(d - 1\) mismatches and in the middle line of Fig 1(d) we have also already passed the location \(j\) of the bottom line.\]

\[
\begin{align*}
\text{begin} \\
\text{if } i + \text{PAT-MISMATCH}(i - r, f) > r + \text{TEXT-MISMATCH}(r, d) \\
\text{(* Case 1: Condition 1 is satisfied*)} \\
\text{then} \\
b := b + 1; \\
\text{TEXT-MISMATCH}(i, b) := \text{TEXT-MISMATCH}(r, d) - (i - r); \\
d := d + 1; \\
\text{else} \\
\text{if } i + \text{PAT-MISMATCH}(i - r, f) < r + \text{TEXT-MISMATCH}(r, d) \\
\text{(* Case 1: Condition 2 is satisfied*)} \\
\text{then} \\
b := b + 1; \\
\text{TEXT-MISMATCH}(i, b) := \text{PAT-MISMATCH}(i - r, f); \\
f := f + 1; \\
\text{else} \\
\text{(* i + PAT-MISMATCH(i - r, f) = r + TEXT-MISMATCH(r, d) *)} \\
\text{(*Case 2 *)} \\
\text{if } a \text{PAT-MISMATCH}(i - r, f) \neq b + \text{PAT-MISMATCH}(i - r, f) \\
\text{then} \\
b := b + 1; \\
\text{TEXT-MISMATCH}(i, b) := \text{PAT-MISMATCH}(i - r, f); \\
f := f + 1; d := d + 1 \\
\text{end}
\end{align*}
\]

Correctness of procedure MERGE. Consider iteration \(i\).

Claim. If there are \(\geq k + 1\) mismatches in locations \(\leq j\) then MERGE finds the first \(k + 1\) of them. If there are \(< k + 1\) mismatches in locations \(\leq j\) then MERGE finds all of them.

Proof of claim. Condition 1 holds for \(\leq k + 1\) locations, which are \(> i\) and \(\leq j\). Let \(y\) be the number of locations in this range for which Condition 2 holds. We do not know anything about \(y\). Suppose \(\text{PAT-MISMATCH}(i - r, 1), \ldots, \text{PAT-MISMATCH}(i - r, y)\) had had included all mismatches between two copies of the pattern which are \(i - r\) apart. Then, by our case analysis, MERGE could have found all mismatches in the range between \(i + 1\) and \(j\). But \(\text{PAT-MISMATCH}(i - r, 1), \ldots, 2k + 1\) contains no more than \(2k + 1\) mismatches. We have to show that we never need more than this for the Claim to hold. If \(\text{PAT-MISMATCH}(i - r, 2k + 1) \geq j - i\) then we have all mismatches between the two
patterns for which Condition 2 holds for locations $\leq j$ in the text and the claim follows. The remaining case is when $PAT-MISMATCH(i-r,2k+1)<j-i$. This gives $2k+1$ locations, which are $>i$ and $<j$, for which Condition 2 holds. Recall that Condition 1 holds for $\leq k$ locations in this range. Therefore, there are $\geq k+1$ locations, which are $>i$ and $<j$, for which Condition 2 holds and Condition 1 does not hold. All these locations satisfy Case 1. Therefore, they suffice to establish that there is no occurrence with $\leq k$ mismatches starting at $t_{i+1}$ and the claim follows.

Procedure EXTEND finds mismatches between $t_{j+1}, \ldots, t_{i+m}$ and $a_{j-i+1}, \ldots, a_m$, by comparing proper pairs of symbols from the pattern and the text in the naive way. EXTEND stops once it finds the $k+1$-st mismatch. If there is an occurrence of the pattern with at most $k$ mismatches which starts at $t_{i+1}$ then EXTEND stops at $t_{i+m}$ after it finishes verifying this fact.

Procedure EXTEND $(i,j,b)$
while $(b < k+1)$ and $(j-i < m)$ do
begin
  $j := j + 1$
  if $t_j \neq a_{j-i}$
  then $b:=b+1$; TEXT-MISMATCH$(i,b):=j-i$
end

Complexity. The running time of TEXT-ANALYSIS is $O(nk)$. For each iteration $i$ ($0 \leq i \leq n-m$) the operations in TEXT-ANALYSIS excluding MERGE and EXTEND take $O(1)$ time. MERGE treats entries of the form $PAT-MISMATCH[j-r:1,2k+1]$ (whose number is $2k+1$) and entries of the form $TEXT-MISMATCH[r:1,k+1]$ (whose number is $k+1$). Each of the operations of MERGE can be charged to one of these $3k+2$ entries in such a way that each entry is being charged by $O(1)$ operations. Therefore, MERGE requires $O(k)$ time. The total number of operations performed by EXTEND throughout all the iterations is $O(n)$ since it scans each symbol of the text at most once. So, we get in total $O(n(k+1)) = O(nk)$.

III. ANALYSIS OF THE PATTERN

In this section we describe the pattern analysis, in which $PAT-MISMATCH[1,\ldots,m-1;1,\ldots,2k+1]$ is computed.

Let $[1,\ldots,m-1]$ be the set of $m-1$ rows of $PAT-MISMATCH$. Assume, w.l.g., that $m$ is some power of 2. The algorithm uses a partition of this set into $\log_2 m$
sets as follows:

\[ [1],[2,3],[4,5,6,7],[8\ldots 15],[m/2\ldots m-1]. \]

The pattern analysis has \( \log m \) stages:

**Stage** \( l \), \( 1 \leq l \leq \log m \). Compute \( PAT-MISMATCH \) for the rows of set \( l \). (Where, set \( l \), \( 1 \leq l \leq \log m \), is \( [2^{l-1}\ldots 2^l-1] \).)

We describe in more detail the last stage (stage \( \log m \)) and discuss briefly later how to extend the same technique for the earlier stages. Essentially, we apply the text analysis algorithm of the previous section. In order to keep this presentation short, we overview the similarities to the text analysis and elaborate only on the differences.

The input to stage \( \log m \) of the pattern analysis consists of the following:

a) The array \( a_1, \ldots, a_{m/2} \), which plays the role of the pattern (in the text analysis).

b) The array \( a_{m/2+1}, \ldots, a_m \), which plays the role of the text.

c) The two dimensional array \( PAT-MISMATCH[1\ldots m/2-1;1\ldots 2k+1] \), which is the output of the previous \( \log m - 1 \) stages of the pattern analysis.

The output of stage \( \log m \) is \( PAT-MISMATCH[m/2\ldots m-1;1\ldots 2k+1] \).

Below, we give a very high-level specification of stage \( \log m \) of the pattern analysis.

**Initialize:** \( PAT-MISMATCH[m/2\ldots m-1;1\ldots 2k+1] = m+1 \);

\[ r := m/2 ; j := m/2 ; \]

**for** \( i := m/2 \) **to** \( m-1 \) **do**

**begin**

\[ b := 0 ; \]

if \( i < j \)

**then** \( MERGE(i,r,j,b) ; \)

if \( b < 2k + 1 \)

**then** \( r := i ; EXTEND(i,j,b) \)

**end**

One important difference with respect to the text analysis needs to be emphasized:

In TEXT-ANALYSIS we were after the \( k+1 \) first mismatches for each location of the text, while here we want to find the \( 2k+1 \) first mismatches. The correctness proof of iteration \( i \) of procedure \( MERGE \), in the previous
section, needed the first \(2k + 1\) locations for which Condition 2 holds in order to find these first \(k + 1\) mismatches. A careful check of the proof will show that the first \(4k + 1\) locations were Condition 2 does not hold would have sufficed for finding the first \(k + 1\) mismatches, as required here. This explains item c) in the input for stage \(\log m\).

Next, we describe briefly stage \(l\), \((1 \leq l < \log m)\), by emphasizing the differences with respect to stage \(\log m\) which was described above.

The input to stage \(l\) of the pattern analysis consists of the following:

a) The array \(a_1, \ldots, a_m - 2^{l-1}\), which plays the role of the pattern (in the text analysis).

b) The array \(a_{2^{l-1}+1}, \ldots, a_m\), which plays the role of the text.

c) The two-dimensional array

\[
\text{PAT-MISMATCH}[1, \ldots, 2^i - 1; 1, \ldots, \min(2^{\log m} - i, 4k + 1, m - 2^i - 1)],
\]

which is the output of the previous \(l-1\) stages of the pattern analysis.

The output of iteration \(i\) at stage \(l\) \((2^{l-1} \leq i \leq 2^l - 1)\) is

\[
\text{PAT-MISMATCH}[i; 1, \ldots, \min(2^{\log m} - i, 2k + 1, m - i)]
\]

We note three differences in this stage, with respect to stage \(\log m\):

a) At stage \(l\) the for loop is for \(i = 2^{l-1}\) to \(2^l - 1\). At each iteration \(i\), we look for the mismatches between \(a_{i+1}, \ldots, a_m\) and \(a_1, \ldots, a_{m-i}\), \((2^{l-1} \leq i \leq 2^l - 1)\).

b) Iteration \(i\) of stage \(l\) looks for \((\min(2k 2^{\log m - i} + 1, m - i))\) mismatches.

c) The output of stages \(1 \ldots l-1\) must give the first \((\min(4k 2^{\log m - i} + 1, m - 2^i - 1))\) mismatches.

Complexity. For each iteration \(i\) at stage \(l\) \((2^{l-1} \leq i \leq 2^l - 1)\), \((1 \leq l \leq \log m)\) the operations in the "main program" excluding MERGE and EXTEND take \(O(1)\) time. As in the previous section MERGE requires \(O(\text{"number of mismatches we look for"})\) time. Here it means \(O(2k 2^{\log m - i})\) time. The total number of operations performed by EXTEND throughout all iterations of stage \(l\) is \(O(m)\). Stage \(l\) has \(2^l - 1\) iterations, therefore it takes \(O(m + \sum (2k 2^{2^l - 1}) = O(km)\) time. So, the running time of the pattern analysis is \(O(\sum_{i=1}^{\log m} km) = O(km \log m)\).
ACKNOWLEDGEMENT. [1] is written in Russian, a language which is unknown to us. Slisenko's kind help in identifying and deciphering the main points in this paper is gratefully acknowledged.

REFERENCES


iteration \( i \)

TEXT-ANALYSIS checks whether there are \( > k \) mismatches between the following strings:

\[
\begin{array}{c|c}
\text{a}_1 & \text{a}_m \\
\hline
\text{t}_{i+1} & \text{t}_{i+m}
\end{array}
\]

Figure 1(a)

\( \text{TEXT-MISMATCH}[r; q, \ldots, k + 1] \) gives all the mismatches between the following strings:

\[
\begin{array}{c|c|c}
\text{a}_1 & \text{a}_{i-r+1} & \text{a}_{j-r} \\
\hline
\text{t}_{r+1} & \text{t}_{i+1} & \text{t}_j
\end{array}
\]

Figure 1(b)

\( \text{PAT-MISMATCH}[i-r; i, \ldots, s] \) gives all, \( \leq 2k + 1 \), the mismatches between the following strings:

\[
\begin{array}{c|c|c|c|c}
\text{a}_1 & \text{a}_{j-1} & \text{a}_{m-(i-r)} \\
\hline
\text{a}_1 & \text{a}_{i-r+1} & \text{a}_{j-r} & \text{a}_m
\end{array}
\]

Figure 1(c)

MERGE uses the information in Fig. 1(b) and 1(c) to compute \( \text{TEXT-MISMATCH}[i; i, \ldots, k + 1] \). If MERGE is unable to complete this job then EXTEND completes it.

\[
\begin{array}{c|c|c|c|c|c|c|c}
\text{a}_1 & \text{a}_{j-1} & \text{a}_{j-1+1} & \text{a}_{j-1+2} & \ldots & \text{a}_m
\end{array}
\]

\[
\begin{array}{c|c}
\hline
\text{a}_1 & \text{a}_{i-r+1} & \text{a}_{j-r} \\
\text{t}_{r+1} & \text{t}_{i+1} & \text{t}_j & \text{t}_{i+1} & \text{t}_{i+m}
\end{array}
\]

\[
\begin{array}{c|c|c}
\hline
\text{MERGE} & \text{EXTEND}
\end{array}
\]

Figure 1(d)

Figure 1
Efficient string matching with $k$