DOI-TR-157

DOI Technical Report

Pattern Matching in Text Compressed by Using Antidictionaries

by

Y. Shibata and M. Takeda and A. Shinohara and S. Arikawa

February 5, 1999

Department of Informatics
Kyushu University
Fukuoka 812-81, Japan

Email: yusuke@i.kyushu-u.ac.jp Phone: +81-92-642-2697

Pattern Matching in Text Compressed by Using Antidictionaries

Yusuke Shibata Masayuki Takeda
Ayumi Shinohara Setsuo Arikawa
{ yusuke, takeda, ayumi, arikawa} @i.kyushu-u.ac.jp
Department of Informatics, Kyushu University 33
Fukuoka 812-8581, Japan

Abstract

In this paper we focus on the problem of compressed pattern matching for the text compression using antidictionaries, which is a new compression scheme proposed recently by Crochemore et al. (1998). We show an algorithm which preprocesses a pattern of length m and an antidictionary M in $O(m^2 + \|M\|)$ time, and then scans a compressed text of length n in $O(n + r)$ time to find all pattern occurrences, where $\|M\|$ is the total length of strings in M and r is the number of the pattern occurrences.

1 Introduction

Compressed pattern matching is one of the most interesting topics in the combinatorial pattern matching, and many studies have been undertaken on this problem for several compression methods from both theoretical and practical viewpoints. See Table 1. One important goal of compressed pattern matching is to achieve a linear time complexity that is proportional not to the original text length but to the compressed text length.

Recently, Crochemore *et al.* proposed a new compression scheme: *text compression using antidictionary* [8]. Contrary to the compression methods that make use of dictionaries, which are particular sets of strings occurring in texts, the new scheme exploits an *antidictionary* that is a finite set of strings that do not occur as factors in text, i.e. that are *forbidden*. Let $a_1 \dots a_n \in \{0,1\}^+$ be the text to be compressed. Suppose we have read a prefix $a_1 \dots a_j$ at a certain moment. If the string $a_i \dots a_j b$ ($i \leq j$, $b \in \{0,1\}$) is a forbidden word, namely, is in the antidictionary, then the next symbol a_{j+1} cannot be b. In other words, the next symbol a_{j+1} is predictable. Based on this idea, the compression method removes such predictable symbols from the text. The compression and the decompression are performed by using the automaton accepting the set of strings in which no forbidden words occur as factors.

In this paper we focus on the problem of compressed pattern matching for the text compression using antidictionaries. We present an algorithm that solves the problem

Table 1: Compressed pattern matching.

compression method	compressed pattern matching
run-length	Eilam-Tzoreff and Vishkin [9]
run-length (two dim.)	Amir, Landau, and Vishkin [6]; Amir and Benson [2, 3]; Amir, Benson, and Farach [5]
LZ77	Farach and Thorup [10]; Gąsieniec, Karpinski, Plandowski, and Rytter [12]
LZW	Amir, Benson, and Farach [4]; Kida, Takeda, Shinohara, Miyazaki, and Arikawa [15]
straight-line program	Karpinski, Rytter, and Shinohara [13]; Miyazaki, Shinohara, and Takeda [17]
Huffman	Fukamachi, Shinohara, and Takeda [11]
finite state encoding	Takeda [19]
others	Manber [16]; Shibata [18]

in $O(m^2 + \|M\| + n + r)$ time using $O(m^2 + \|M\| + n)$ space, where m and n are the pattern length and the compressed text length, respectively, $\|M\|$ denotes the total length of strings in antidictionary M, and r is the number of pattern occurrences. Since M is a part of the compressed representation of text, the text scanning time is $O(\|M\|+n+r)$, which is linear in the compressed text length $\|M\|+n$, when ignoring r. Moreover, in the case where a set of text files share a common antidictionary [8], we can regard the $O(\|M\|)$ time processing of M as a preprocessing. Then the $O(n + r)$ time text scanning will be fast in practice. The proposed algorithm thus has desirable properties.

2 Preliminaries

Strings x, y, and z are said to be a *prefix*, *factor*, and *suffix* of the string $u = xyz$, respectively. The sets of prefixes, factors, and suffixes of a string u are denoted by $\mathit{Prefix}(u)$, $\mathit{Factor}(u)$, and $\mathit{Suffix}(u)$, respectively. A prefix, factor, and suffix of a string u is said to be *proper* if it is not u. The length of a string u is denoted by $|u|$. The empty string is denoted by ε, that is, $|\varepsilon| = 0$. The ith symbol of a string u is denoted by $u[i]$ for $1 \leq i \leq |u|$, and the factor of a string u that begins at position i and ends at position j is denoted by $u[i:j]$ for $1 \leq i \leq j \leq |u|$. The reversed string of a string u is denoted by u^R. The total length of strings of a set S is denoted by $\|S\|$. For strings x and y, denote by $\mathit{Occ}(x, y)$ the set of occurrences of x in y. That is,

$$\mathit{Occ}(x, y) = \{|x| \leq i \leq |y| \mid x = y[i - |x| + 1 : i]\}.$$

The next lemma follows from the periodicity lemma.

Lemma 1 *If $\mathit{Occ}(x, y)$ has more than two elements and the difference of the maximum and the minimum elements is at most $|x|$, then it forms an arithmetic progression, in which the step is the smallest period of x.*

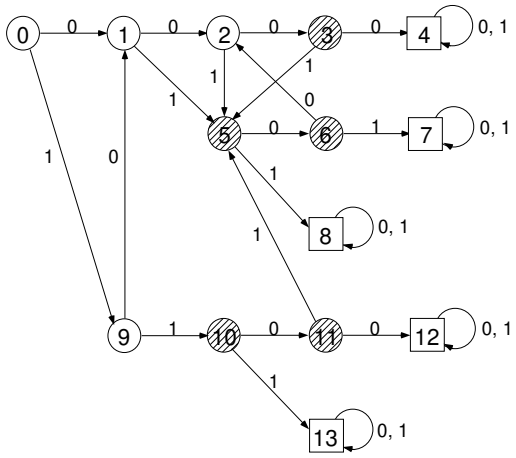

Figure 1: Automaton $\mathcal{A}(M)$ for $M = \{0000, 111, 011, 0101, 1100\}$. Circles and squares denote the final and the nonfinal states, respectively. Shaded circles denote the predict states.

3 Text compression using antidictionary

This section describes the text compression scheme recently proposed by Crochemore *et al.* [8].

3.1 Method

Let $B = \{0, 1\}$. Suppose that $\mathcal{T} \in B^+$ be the text to be compressed. A *forbidden word* for $\mathcal{T}$ is a string $u \in B^+$ that is not a factor of $\mathcal{T}$. A forbidden word is said to be *minimal* if it has no proper factor that is forbidden. An *antidictionary* for $\mathcal{T}$ is a set of minimal forbidden words for $\mathcal{T}$.

Let M be an antidictionary for $\mathcal{T}$. Then the text $\mathcal{T}$ is in the set $B^* \backslash B^* M B^*$. The automaton accepting the set $B^* \backslash B^* M B^*$ can be built from M in $O(\|M\|)$ time in a similar way to the construction of the Aho-Corasick pattern matching machine [1]. We denote the automaton by

$$\mathcal{A}(M) = (Q, B, \delta, \varepsilon, M),$$

where $Q = \mathit{Prefix}(M)$ is the set of states; B is the alphabet; δ is the state transition function from $Q \times B$ to Q defined as

$$\delta(u, a) = \begin{cases} u, & \text{if } u \in M; \\ \text{longest string in } Q \cap \mathit{Suffix}(ua), & \text{otherwise;} \end{cases}$$

ε is the initial state; M is the set of final states. Figure 1 shows the automaton $\mathcal{A}(M)$ for $M = \{0000, 111, 011, 0101, 1100\}$, which is an antidictionary for text $\mathcal{T} = 11010001$.

The encoder and the decoder in this compression scheme are obtained directly from the automaton $\mathcal{A}(M)$. The encoder $\mathcal{E}(M)$ is a generalized sequential machine based

input:		1		1		0		1		0		0		0		1		
state:	0	$\rightarrow$	9	$\rightarrow$	10	$\rightarrow$	11	$\rightarrow$	5	$\rightarrow$	6	$\rightarrow$	2	$\rightarrow$	3	$\rightarrow$	5	
output:		1		1		ε		ε		ε		ε		0		ε		

Figure 2: Move of encoder $\mathcal{E}(M)$ for $\mathcal{T} = 11010001$.

on $\mathcal{A}(M)$ with output function $\lambda : Q \times B$ defined by

$$\lambda(u, a) = \begin{cases} a, & \text{if } Deg(u) = 2; \\ \varepsilon, & \text{otherwise,} \end{cases}$$

where $Deg(u) = |\{a \in B | \delta(u, a) \notin M\}|$. The decoder $\mathcal{D}(M)$ is a generalized sequential machine obtained by swapping the input label and the output label on each arc of the encoder $\mathcal{E}(M)$. Figure 2 illustrates the move of the encoder $\mathcal{E}(M)$ based on $\mathcal{A}(M)$ of Fig. 1 which takes as input $\mathcal{T} = 11010001$ and emits 110. It should be noted that, any prefix of 1101000100 with length greater than 6 is compressed into the same string 110. For a decompression we therefore need the length of $\mathcal{T}$ together with the encoded string itself. Formally, the compressed representation of $\mathcal{T}$ is a triple $\langle M, b_1, \ldots b_n, N \rangle$, where M is an antidictionary, $b_1 \ldots b_n$ is output from the encoder, and N is the length of $\mathcal{T}$.

Let us denote by $MF(\mathcal{T})$ the set of all minimal forbidden words for $\mathcal{T}$. In the case of binary alphabet we have $|MF(\mathcal{T})| \leq 2 \cdot |\mathcal{T}| - 2$ as shown in [7]. To shorten the representation size of the above triple, we need a way to build a 'good' antidictionary as a subset of $MF(\mathcal{T})$. Crochemore *et al.* presented in [8] a simple method in which antidictionary is the set of forbidden words of length at most k, where k is a parameter. It is reported in [8] that the compression ratio in practice is comparable to `pkzip`.

3.2 Decoder without ε-moves

Note that the decoder $\mathcal{D}(M)$ mentioned above has ε-moves. For a simple presentation of our algorithm, we shall define a generalized sequential machine $\mathcal{G}(M)$ obtained by eliminating the ε-moves from the decoder $\mathcal{D}(M)$.

Let us partition the set Q into four disjoint subsets M, Q_0, Q_1, and Q_2 by

$$Q_i = \{u \in Q \backslash M \mid Deg(u) = i\} \qquad (i = 0, 1, 2).$$

A state p in Q_1 is called a *predict state* because of the uniqueness of outgoing arc when ignoring the arcs into states in M. Namely, there exists exactly one symbol a such that $\delta(p, a) \notin M$. We denote such symbol a by $NextSymbol(p)$, and denote by $NextState(p)$ the state $\delta(p, a)$.

Consider, for $p \in Q_1$, the sequence $p_1, p_2, \ldots$ of states in Q_1 defined by $p_1 = p$ and $p_{i+1} = NextState(p_i)$ $(i = 1, 2, \ldots)$. There are two cases: One is the case that there exists an integer $m > 0$ such that, for $i = 1, 2, \ldots, m-1$, $p_i \in Q_1$, and $p_m \in Q_0 \cup Q_2$. The other is the case of no such integer m, namely, the sequence continues infinitely. Let us call the sequence the *predict path* of p, and denote by $Terminal(p)$ the last state p_m. In the infinite case, let $Terminal(p) = \perp$, where $\perp$ is a special state not in Q. (Therefore, $Terminal(p) \in Q_0 \cup Q_2 \cup \{\perp\}$.) The finite/semi-infinite string spelled out by the predict path of $p \in Q_1$ is denoted by $Sequence(p)$. It is easy to see that:

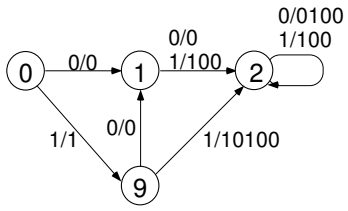

Figure 3: Decoder $\mathcal{G}(M)$ for $M = \{0000, 111, 011, 0101, 1100\}$.

Lemma 2 *For any $p \in Q_1$, there exist $u, v \in B^*$ with $|uv| < |Q_1|$ such that*

$$Sequence(p) = u\ v\ v\ \cdots.$$

Now we are ready to define a generalized sequential machine $\mathcal{G}(M)$, where the set of states is $Q_0 \cup Q_2 \cup \{\perp\}$; the state transition function is $\delta_{\mathcal{G}} : Q_2 \times B \to Q_0 \cup Q_2 \cup \{\perp\}$ defined by

$$\delta_{\mathcal{G}}(u,a) = \begin{cases} Terminal(\delta(u,a)), & \delta(u,a) \in Q_1; \\ \delta(u,a), & \text{otherwise;} \end{cases}$$

the output function is $\lambda_{\mathcal{G}} : Q_2 \times B \to B^+ \cup B^\infty$ defined by

$$\lambda_{\mathcal{G}}(u,a) = \begin{cases} a \cdot Sequence(\delta(u,a)), & \delta(u,a) \in Q_1; \\ a, & \text{otherwise,} \end{cases}$$

where B^∞ denotes the set of semi-infinite strings over B. Figure 3 shows the encoder $\mathcal{G}(M)$ obtained in this way from the automaton $\mathcal{A}(M)$ of Fig. 1.

Decompression algorithm using $\mathcal{G}(M)$ is shown in Fig. 4. It should be emphasized that, if the decoder $\mathcal{G}(M)$ enters a state q and then reads a symbol a such that $\lambda_{\mathcal{G}}(q,a)$ is semi-infinite, the symbol is the last symbol of the output from the encoder $\mathcal{E}(M)$. In this case the decoder $\mathcal{G}(M)$ halts after emitting an appropriate length prefix of $\lambda_{\mathcal{G}}(q,a)$ according to the value of N.

4 Main result

Generally, most of text compression methods can be recognized as mechanisms to factorize a text into several blocks as $\mathcal{T} = u_1 u_2 \dots u_n$ and to store a sequence of 'representations' of blocks u_i. In the LZW compression, for example, the representation of a block u_i is just an integer which indicates the node of dictionary trie representing the string u_i. In the case of the compression using antidictionaries, the way of representation of block is slightly complicated.

Consider how to simulate the move of the KMP automaton for a pattern $\mathcal{P}$ running on the uncompressed text $\mathcal{T}$. Let $\delta_{\mathrm{KMP}} : \{0, 1, \dots, m\} \times B \to \{0, 1, \dots, m\}$ be the state transition function of the KMP automaton for $\mathcal{P} = \mathcal{P}[1:m]$. We extend δ_{KMP} to the domain $\{0, 1, \dots, m\} \times B^*$ in the standard manner. We also define the function λ_{KMP} on $\{0, 1, \dots, m\} \times B^*$ by

$$\lambda_{\mathrm{KMP}}(j,u) = \{1 \le i \le |u| \mid \mathcal{P} \text{ is a suffix of string } \mathcal{P}[1:j] \cdot u[1:i]\ \}.$$

```
Input.   A compressed representation ⟨M, b_1...b_n, N⟩ of a text T = T[1 : N].
Output.  Text T.
begin
    ℓ := 0;
    q := ε;
    for i := 1 to n − 1 do begin
        u := λ_G(q, b_i);
        q := δ_G(q, b_i);
        ℓ := ℓ + |u|;
        print u
    end;
    u := λ_G(q, b_n);
    print the prefix of u with length N − ℓ
end.
```

Figure 4: Decompression by $\mathcal{G}(M)$.

We want a pattern matching algorithm which takes as input a sequence of representations of blocks $u_1, u_2, \ldots, u_n$ of $\mathcal{T}$ and reports all occurrences of $\mathcal{P}$ in $\mathcal{T}$ in $O(n+r)$ time, where $r = |Occ(\mathcal{P}, \mathcal{T})|$. Then we need a mechanism for obtaining in $O(1)$ time the value of $\delta_{\mathrm{KMP}}(j, u)$ and a linear size representation of the set $\lambda_{\mathrm{KMP}}(j, u)$. In the case of the LZW compression such mechanism can be realized in $O(m^2+n)$ time using $O(m^2+n)$ space as stated in [4] and [15]. Similar idea can also be applied to the case of text compression by antidictionary, except that block u_i, which will be an input to the second arguments of δ_{KMP} and λ_{KMP}, is represented in a different manner.

In our case a block u_i is represented as a pair of the current state q of $\mathcal{G}(M)$ and the first symbol b_i of u_i. Therefore we have to keep the state transitions of $\mathcal{G}(M)$. An overview of the intended algorithm is shown in Fig. 5. The algorithm makes $\mathcal{G}(M)$ run on $b_1 \ldots b_n$ to know inputs $u_1, u_2, \ldots, u_n$ to the KMP automaton being simulated. Figure 6 illustrates the move of the algorithm searching the compressed text 110 for the pattern $\mathcal{P} = 0001$.

We have the following theorems which will be proved in the next section.

Theorem 1 *Function which takes as input $(q, a) \in Q_2 \times B$ and returns in $O(1)$ time the value of $\delta_{\mathcal{G}}(q, a)$, can be realized in $O(\|M\|)$ time using $O(\|M\|)$ space.*

Theorem 2 *Function which takes as input a triple $(j, q, a) \in \{0, \ldots, m\} \times Q_2 \times B$ and returns in $O(1)$ time the value of*

$$\delta_{\mathrm{KMP}}(j, u) \qquad (u = \lambda_{\mathcal{G}}(q, a)),$$

can be realized in $O(\|M\| + m^2)$ time using $O(\|M\| + m^2)$ space.

Theorem 3 *Function which takes as input a triple $(j, q, a) \in \{0, \ldots, m\} \times Q_2 \times B$ and returns in $O(1)$ time a linear size representation of the set*

$$\lambda_{\mathrm{KMP}}(j, u) \qquad (u = \lambda_{\mathcal{G}}(q, a)),$$

can be realized in $O(\|M\| + m^2)$ time using $O(\|M\| + m^2)$ space.

Input. A compressed representation $\langle M, b_1 b_2 ... b_n, N \rangle$ of a text $\mathcal{T} = \mathcal{T}[1:N]$, and a pattern $\mathcal{P} = \mathcal{P}[1:m]$.
Output. All positions at which $\mathcal{P}$ occurs in $\mathcal{T}$.
begin
/* *Preprocessing* */
Construct the KMP automata and the suffix tries for $\mathcal{P}$ and $\mathcal{P}^R$;
Construct the automaton $\mathcal{A}(M)$ from M;
Construct the predict path graph from $\mathcal{A}(M)$;
Perform the processing required for $\delta_{\mathcal{G}}$, δ_{KMP}, and λ_{KMP} (See Section 5.);
/* *Text scanning* */
$\ell := 0$;
$q := \varepsilon$;
$state := 0$;
for $i := 1$ **to** $n - 1$ **do begin**
$u := \lambda_{\mathcal{G}}(q, b_i)$;
$q := \delta_{\mathcal{G}}(q, b_i)$;
for each $p \in \lambda_{\mathrm{KMP}}(state, u)$ **do**
Report a pattern occurrence that ends at position $\ell + p$;
$state := \delta_{\mathrm{KMP}}(state, u)$;
$\ell := \ell + |u|$
end;
$u := \lambda_{\mathcal{G}}(q, b_n)$;
for each $p \in \lambda_{\mathrm{KMP}}(state, u)$ such that $\ell + p \leq N$ **do**
Report a pattern occurrence that ends at position $\ell + p$
end.

Figure 5: Pattern matching algorithm.

input :		1		1		0	
state of $\mathcal{G}(M)$:	0	$\longrightarrow$	9	$\longrightarrow$	2	$\longrightarrow$	2
u :		1		10100		0100	
state of KMP automaton :	0	$\longrightarrow$	0	$\longrightarrow$	2	$\longrightarrow$	2
output :		$\emptyset$		$\emptyset$		$\{8\}$	

Figure 6: Move of pattern matching algorithm when $\mathcal{T} = 110100010$ and $\mathcal{P} = 0001$.

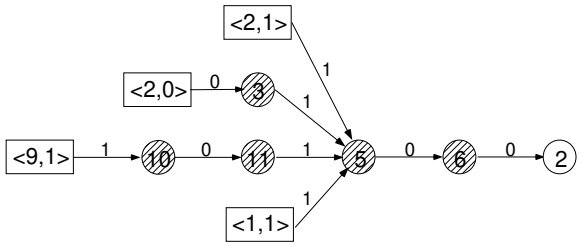

Figure 7: Predict path graph. Rectangles denote the auxiliary nodes.

Then we have the following result.

Theorem 4 *Problem of compressed pattern matching for the text compression using antidictionaries can be solved in* $O(\|M\|+n+m^2+r)$ *time using* $O(\|M\|+m^2)$ *space.*

5 Algorithm in detail

This section gives a detailed presentation of the algorithm to prove Theorems 1, 2, and 3.

5.1 Proof of Theorem 1

For a realization of $\delta_{\mathcal{G}}$, we have to find, for each $q \in Q_0 \cup Q_2 \cup \{\perp\}$, the pairs $(p,b) \in Q_2 \times B$ such that $\delta(p,b) = p' \in Q_1$ and $\mathit{Terminal}(p') = q$. First of all, we mention the graph consisting of the predict paths, which plays an important role in this proof.

Consider the subgraph of $\mathcal{A}(M)$ in which the arcs are limited to the outgoing arcs from predict nodes. We add auxiliary nodes $v = \langle p, b\rangle$ and new arcs labelled b from v to $q \in Q_1$ such that $p \in Q_2$, $b \in B$, and $\delta(p,b) = q$ to the subgraph. We call the resulting graph *predict path graph*. Figure 7 shows the predict path graph obtained from $\mathcal{A}(M)$ of Fig. 1.

The predict path graph illustrates, for $(p,b) \in Q_2 \times B$, the string $\lambda_{\mathcal{G}}(p,b)$ as a path which starts at the auxiliary node $\langle p, b\rangle$, passes through nodes in Q_1, and either finally encounters a node in $Q_0 \cup Q_2$, or flows into a loop consisting only of nodes in Q_1. A connected component of the predict path graph falls into two classes: (a) a tree which has as root a node in $Q_0 \cup Q_2$ and has as leaves auxiliary nodes, and (b) a loop with trees, each of which has as root a node on the loop and has leaves auxiliary nodes. See Fig. 8.

Now we are ready to prove Theorem 1. Construction of $\delta_{\mathcal{G}}$ is as follows: First, we set $\delta_{\mathcal{G}}(p,b) = \delta(p,b)$ for every $(p,b) \in Q_2 \times B$ with $\delta(p,b) \in Q_0 \cup Q_2$. Next, for every node $q \in Q_0 \cup Q_2$ of the predict path graph, we traverse the tree that has q as root. Note that the leaves of the tree are auxiliary nodes $\langle p, b\rangle$ such that $\mathit{Terminal}(\delta(p,b)) = q$, and we can set $\delta_{\mathcal{G}}(p,b) = q$. Finally, for every node q on loops of the predict path graph, we traverse the tree that has q as root. The leaves of the tree are auxiliary nodes $\langle p, b\rangle$ such that $\mathit{Terminal}(\delta(p,b)) = \perp$, and hence we set

(a) (b)

Figure 8: Connected components of predict path graph.

$\delta_{\mathcal{G}}(p,b) = \perp$. The total time complexity is linear in the number of nodes of the predict path graph, i.e. $O(\|M\|)$. The proof is now complete.

5.2 Proof of Theorem 2

In the following discussions, we are frequently faced with the need to get some value as a function of u, the strings that are spelled out by the paths from auxiliary nodes. Even when the value for each path can be computed in time proportional to the path length, the total time complexity is not $O(\|M\|)$ since more than one path can share common arcs.

Suppose that the value for each path can be computed by making an automaton run on the path in the reverse direction. Then, we can compute the values for such paths by traversing every tree in the depth-first-order using a stack. Since this method enables us to 'share' the computation for a common suffix of two strings, the total time complexity is linear in the number of arcs, i.e. $O(\|M\|)$. This technique plays a key role in the following proofs.

For an integer j with $0 \leq j \leq m$ and for a factor u of $\mathcal{P}$, let us denote by $N_1(j,u)$ the largest integer k with $0 \leq k \leq j$ such that $\mathcal{P}[j-k+1:j] \cdot u$ is a prefix of $\mathcal{P}$. Let $N_1(j,u) = nil$, if no such integer exists. Then, we have:

$$\delta_{\mathrm{KMP}}(j,u) = \begin{cases} N_1(j,u) + |u|, & \text{if } u \text{ is a factor of } \mathcal{P} \text{ and } N_1(j,u) \neq nil; \\ \delta_{\mathrm{KMP}}(0,u), & \text{otherwise.} \end{cases}$$

We assume that the second argument u of N_1 is given as a node of the suffix trie for $\mathcal{P}$. Amir *et al.* [4] showed the following fact.

Lemma 3 (Amir et al. 1996) *Function which takes as input $(j,u) \in \{0,\ldots,m\} \times Factor(\mathcal{P})$ and returns the value of $N_1(j,u)$ in $O(1)$ time, can be realized in $O(m^2)$ time using $O(m^2)$ space.*

We have also the next lemma.

Lemma 4 *Function which takes as input $(q,a) \in Q_2 \times B$ and returns $u = \lambda_{\mathcal{G}}(q,a)$ as a node of the suffix trie for $\mathcal{P}$ when $u \in Factor(\mathcal{P})$, can be realized in $O(\|M\| + m^2)$ time using $O(\|M\| + m^2)$ space.*

Proof. We use the technique mentioned above. We can ignore the infinite strings. That is, we can ignore the trees in which a root is on a loop. Consider the problem

of determining whether u^R is a factor of $\mathcal{P}^R$. It can be solved in $O(\min\{|u|, m\})$ time using the suffix trie for $\mathcal{P}^R$. If u^R is a factor of $\mathcal{P}^R$, the node u of the suffix trie for $\mathcal{P}$ can be determined directly from the node u^R of the suffix trie for $\mathcal{P}^R$ assuming a trivial one-to-one mapping between the two suffix tries, which can be computed in $O(m^2)$ time. □

Lemma 5 *Function which takes as input $(q, a) \in Q_2 \times B$ such that $u = \lambda_{\mathcal{G}}(q, a)$ is finite and returns in $O(1)$ time the value of $\delta_{\mathrm{KMP}}(0, u)$, can be realized in $O(\|M\| + m)$ time using $O(\|M\| + m)$ space.*

Proof. We use the technique mentioned above again. We have to consider the problem of finding the length of longest suffix of u that is also a prefix of $\mathcal{P}$. This is equivalent to finding the length of longest prefix of u^R that is also a suffix of $\mathcal{P}^R$. It is solved in $O(\min\{|u|, m\})$ time using the suffix tree for $\mathcal{P}^R$. We can ignore the trees in which a root is on a loop. □

Theorem 2 follows from the lemmas above.

5.3 Proof of Theorem 3

According to whether a pattern occurrence covers the boundary between the strings $\mathcal{P}[1 : j]$ and u, we can partition the set $\lambda_{\mathrm{KMP}}(j, u)$ into two disjoint subsets as follows.

$$\lambda_{\mathrm{KMP}}(j, u) = \lambda_{\mathrm{KMP}}(j, \tilde{u}) \cup X(u),$$

where

$$X(u) = \{|\mathcal{P}| \leq i \leq |u| \mid \mathcal{P} \text{ is a suffix of } u[1 : i]\},$$

and $\tilde{u}$ is the longest prefix of u that is also a proper suffix of $\mathcal{P}$. Let

$$Y(j, \ell) = Occ(\mathcal{P}, \mathcal{P}[1 : j] \cdot \mathcal{P}[m - \ell + 1 : m]) \ominus j,$$

where $\ominus$ denotes the element-wise subtraction. It is easy to see $\lambda_{\mathrm{KMP}}(j, \tilde{u}) = Y(j, |\tilde{u}|)$. It follows from Lemma 1 that the set $Y(j, \ell)$ has the following property:

Lemma 6 *If $Y(j, \ell)$ has more than two elements, it forms an arithmetic progression, where the step is the smallest period of $\mathcal{P}$.*

Lemma 7 *Function which takes as input $(j, \ell) \in \{0, \ldots, m\} \times \{0, \ldots, m\}$ and returns in $O(1)$ time an $O(1)$ space representation of the set $Y(j, \ell)$, can be realized in $O(m^2)$ time using $O(m^2)$ space.*

Proof. It follows from Lemma 6 that $Y(j, \ell)$ can be stored in $O(1)$ space as a pair of the minimum and the maximum values in it. The table storing the minimum values of $Y(j, \ell)$ for all (j, ℓ) can be computed in $O(m^2)$ time as stated in [4]. (Table N_2 defined in [4] satisfies $\min(Y(j, \ell)) = m - N_2(j, \ell)$.) By reversing the pattern $\mathcal{P}$, the table for the maximum values is also computed in $O(m^2)$ time. The smallest period of $\mathcal{P}$ is computed in $O(m)$ time. □

Lemma 8 *Function which takes as input $(q,a) \in Q_2 \times B$ and returns in $O(1)$ time the value of $|\tilde{u}|$ with $u = \lambda_{\mathcal{G}}(q,a)$, can be realized in $O(\|M\|+m)$ time using $O(\|M\|+m)$ space.*

Proof. We shall consider the problem of finding the length of longest suffix of u^R that is also a proper prefix of $\mathcal{P}^R$. This can be solved by using the KMP automaton for $\mathcal{P}^R$. But we have to consider the case where u is semi-infinite. In the finite string case, we make the automaton start at the root of tree with initial state. But in the infinite string case, we must change the value of the initial state. Let v be the string spelled out by the loop starting at the root of the tree being considered. We must pay attention to the case where a pattern suffix is also a prefix of the string v^ℓ with $\ell > 0$. To determine the correct value of the initial state at the root node, we make the automaton go around the loop exactly ℓ times and stop it at the root node that is the starting point, where ℓ is the smallest integer with $\ell \cdot |v| > |\mathcal{P}|$. The state of the automaton at that moment is the desired value. □

Lemma 9 *Function which takes as input $(q,a) \in Q_2 \times B$ and returns in $O(1)$ time a linear size representation of the set $X(u)$ with $u = \lambda_{\mathcal{G}}(q,a)$, can be realized in $O(\|M\|+m)$ time using $O(\|M\|+m)$ space.*

Proof. By using the KMP automaton for the reversed pattern, we mark the predict nodes at which the pattern begins. Suppose that every predict node has a pointer to the nearest proper ancestor that is marked. Such pointers are realized using $O(\|M\|)$ time and space. This enables us to get the elements of $X(u)$ in $O(|X(u)|)$ time. □

Theorem 3 follows from the lemmas above.

6 Concluding remarks

In this paper we focused on the problem of compressed pattern matching for the text compression using antidictionaries proposed recently Crochemore *et al.* [8]. We presented an algorithm which has a linear time complexity proportional to the compressed text length, when we exclude the pattern preprocessing. We are now implementing the algorithm to evaluate its performance from practical viewpoints. In [14] we showed that the Shift-And approach is effective in the compressed pattern matching for the LZW compression. We think that the Shift-And approach will be substituted for the KMP automaton approach presented in this paper and show a good performace in practice when the pattern length m is not so large, say $m \leq 32$.

For a long pattern we can also consider the following method. Let k be the length of the longest forbidden word in the antidictionary. By using the syncronizing property [8], we obtain:

Lemma 10 *If $|\mathcal{P}| \geq k-1$, then $\delta(u,\mathcal{P}) = \delta(\varepsilon,\mathcal{P})$ for any state u in Q such that $\delta(u,\mathcal{P}) \notin M$.*

Let $p = \delta(\varepsilon, \mathcal{P})$. Since $p \in M$ implies that $\mathcal{P}$ cannot occur in $\mathcal{T}$, we can assume $p \notin M$. If p is in Q_1, then let $q = \mathit{Terminal}(p)$. Otherwise, let $q = p$. We can monitor whether the state of $\mathcal{A}(M)$ is in state p by using the function $\delta_{\mathcal{G}}$ to check $\mathcal{G}(M)$ is in state q. If so, we shall confirm it. Our preliminary experiments suggest that this search method is efficient in practice.

References

[1] A. V. Aho and M. Corasick. Efficient string matching: An aid to bibliographic search. *Comm. ACM*, 18(6):333–340, 1975.

[2] A. Amir and G. Benson. Efficient two-dimensional compressed matching. In *Proc. Data Compression Conference*, page 279, 1992.

[3] A. Amir and G. Benson. Two-dimensional periodicity and its application. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 440–452, 1992.

[4] A. Amir, G. Benson, and M. Farach. Let sleeping files lie: Pattern matching in Z-compressed files. *Journal of Computer and System Sciences*, 52:299–307, 1996.

[5] A. Amir, G. Benson, and M. Farach. Optimal two-dimensional compressed matching. *Journal of Algorithms*, 24(2):354–379, August 1997.

[6] A. Amir, G. M. Landau, and U. Vishkin. Efficient pattern matching with scaling. *Journal of Algorithms*, 13(1):2–32, 1992.

[7] M. Crochemore, F. Mignosi, and A. Restivo. Minimal forbidden words and factor automata. In L. Brim, J. Gruska, and J. Zlatuska, editors, *Proceedings of the 23rd Internationial Symposium on Mathematical Foundations of Computer Science*, volume 1450 of *Lecture Notes in Computer Science*, pages 665–673. Springer-Verlag, 1998.

[8] M. Crochemore, F. Mignosi, A. Restivo, and S. Salemi. Text compression using antidictionaries. Technical Report IGM-98-10, Institut Gaspard-Monge, 1998.

[9] T. Eilam-Tzoreff and U. Vishkin. Matching patterns in strings subject to multi-linear transformations. *Theoretical Computer Science*, 60(3):231–254, 1988.

[10] M. Farach and M. Thorup. String-matching in Lempel-Ziv compressed strings. In *27th ACM STOC*, pages 703–713, 1995.

[11] S. Fukamachi, T. Shinohara, and M. Takeda. String pattern matching for compressed data using variable length codes. Submitted, 1998.

[12] L. Gąsieniec, M. Karpinski, W. Plandowski, and W. Rytter. Efficient algorithms for Lempel-Ziv encoding. In *Proc. 4th Scandinavian Workshop on Algorithm Theory*, volume 1097 of *Lecture Notes in Computer Science*, pages 392–403. Springer-Verlag, 1996.

[13] M. Karpinski, W. Rytter, and A. Shinohara. An efficient pattern-matching algorithm for strings with short descriptions. *Nordic Journal of Computing*, 4:172–186, 1997.

[14] T. Kida, M. Takeda, A. Shinohara, and S. Arikawa. Shift-And approach to pattern matching in LZW compressed text. Technical Report DOI-TR-CS-156, Department of Informatics, Kyushu University, January 1999.

[15] T. Kida, M. Takeda, A. Shinohara, M. Miyazaki, and S. Arikawa. Multiple pattern matching in LZW compressed text. In J. A. Atorer and M. Cohn, editors, *Proceedings of Data Compression Conference '98*, pages 103–112. IEEE Computer Society, 1998.

[16] U. Manber. A text compression scheme that allows fast searching directly in the compressed file. In *Proc. Combinatorial Pattern Matching*, volume 807 of *Lecture Notes in Computer Science*, pages 113–124. Springer-Verlag, 1994.

[17] M. Miyazaki, A. Shinohara, and M. Takeda. An improved pattern matching algorithm for strings in terms of straight-line programs. In *Proc. Combinatorial Pattern Matching*, volume 1264 of *Lecture Notes in Computer Science*, pages 1–11. Springer-Verlag, 1997.

[18] Y. Shibata. Speeding-up string pattern matching by text compression using byte pair encoding. Diploma thesis (in Japanese), Kyushu Instutute of Technology, 1998.

[19] M. Takeda. Pattern matching machine for text compressed using finite state model. Technical Report DOI-TR-CS-142, Department of Informatics, Kyushu University, October 1997.