Multiple Pattern Matching Algorithms on Collage System

Takuya Kida, Tetsuya Matsumoto, Masayuki Takeda,
Ayumi Shinohara, and Setsuo Arikawa

Department of Informatics, Kyushu University 33
Fukuoka 812-8581, Japan
{kida,tetsuya,takeda,ayumi,arikawa}@i.kyushu-u.ac.jp

Abstract. Compressed pattern matching is one of the most active topics in string matching. The goal is to find all occurrences of a pattern in a compressed text without decompression. Various algorithms have been proposed depending on underlying compression methods in the last decade. Although some algorithms for multipattern searching on compressed text were also presented very recently, all of them are only for Lempel-Ziv family compressions. In this paper we propose two types of multipattern matching algorithms on collage system, which simulate the AC algorithm and a multipattern version of the BM algorithm, the most important algorithms for searching in uncompressed files. Collage system is a formal framework which is suitable to capture the essence of compressed pattern matching according to various dictionary based compressions. That is, we provide the model of multipattern matching algorithm for any compression method covered by the framework.

1 Introduction

The *compressed pattern matching problem* was first defined by Amir and Benson [2], and various compressed pattern matching algorithms have been proposed depending on underlying compression methods (see survey papers [19,23]).

In [7] we introduced a *collage system*, which is a formal system to represent a string by a pair of dictionary $\mathcal{D}$ and sequence $\mathcal{S}$ of phrases in $\mathcal{D}$. The basic operations are concatenation, truncation, and repetition. Collage systems give us a unifying framework of various dictionary-based compression methods, such as Lempel-Ziv family (LZ77, LZSS, LZ78, LZW), RE-PAIR [11], SEQUITUR [16], and the static dictionary based compression method. We also proposed in [7] the simple pattern matching algorithm on collage system, which simulates the move of the Knuth-Morris-Pratt automaton [10] running on the original text, by using the functions *Jump* and *Output*.

In this paper we address the *multiple pattern* matching problem on collage system. That is, given a set Π of patterns and a collage system $\langle\mathcal{D},\mathcal{S}\rangle$, we find all occurrences of any pattern in Π within the text represented by $\langle\mathcal{D},\mathcal{S}\rangle$. It is rather easy to extend *Jump* to the multipattern case. However, the extension of *Output* is not straightforward because the single pattern version utilizes some combinatorial properties on the period of the pattern. Although we have developed a multipattern searching algorithm for LZW compressed texts in [8], the same technique cannot be adopted to general collage systems. Nevertheless, we succeeded to develop an algorithm that runs in $O((\|\mathcal{D}\|+|\mathcal{S}|)\cdot \mathit{height}(\mathcal{D})+m^2+r)$ time with $O(\|\mathcal{D}\|+m^2)$ space, where $\|\mathcal{D}\|$ denotes the size of the dictionary $\mathcal{D}$, $\mathit{height}(\mathcal{D})$ denotes the maximum dependency of the operations in $\mathcal{D}$, $|\mathcal{S}|$ is the length of the sequence $\mathcal{S}$, m is the total length of patterns, and r is the number of pattern occurrences. Note that the time for decompressing-then-searching is linear with respect to the original text length, which can grow in proportion to $|S|\cdot 2^{\|\mathcal{D}\|}$ on the worst case. Therefore, the algorithm is more efficient than the decompress-then-search approach.

We also show an extension of the Boyer-Moore type algorithm presented in [21] to multiple patterns. The algorithm runs on the sequence $\mathcal{S}$, with skipping some tokens. It runs in $O((\mathit{height}(\mathcal{D})+m)|\mathcal{S}|+r)$ time after an $O(\|\mathcal{D}\|\cdot \mathit{height}(\mathcal{D})+m^2)$ time preprocessing with $O(\|\mathcal{D}\|+m^2)$ space. Moreover, we mention the parallel complexity of compressed pattern matching for a subclass of collage system in Section 8. Our result implies that the compressed pattern matching for regular collage system can be efficiently parallelized in principle.

2 Related Works

We presented in [7] a general pattern matching algorithm on collage system for a single pattern. The algorithm runs in $O((\|\mathcal{D}\|+|\mathcal{S}|)\cdot \mathit{height}(\mathcal{D})+m^2+r)$ time with $O(\|\mathcal{D}\|+m^2)$ space. For the subclass of collage system which contains no truncation, it runs in $O(\|\mathcal{D}\|+|\mathcal{S}|+m^2+r)$ time using $O(\|\mathcal{D}\|+m^2)$ space. We also presented a Boyer-Moore type algorithm in [21].

Independently, Navarro and Raffinot [14] developed a general technique for string matching on a text given as a sequence of blocks, which abstracts both LZ77 and LZ78 compressions, and gave bit-parallel implementations. The running time of these algorithms based on the bit-parallelism for LZW is $O(nm/w+m+r)$, where n is the text length and w is the length in bits of the machine word. If the pattern is short $(m<w)$, these algorithms are efficient in practice. A Boyer-Moore type algorithm for a single pattern on Ziv-Lempel compressed text is also developed [15].

3 Preliminaries

Let Σ be a finite set of characters, called an *alphabet*. A finite sequence of characters is called a *string*. We denote the length of a string u by $|u|$. The empty string is denoted by ε, that is, $|\varepsilon|=0$. Let Σ^* be the set of strings over Σ, and let $\Sigma^+=\Sigma^*\backslash\{\varepsilon\}$. Strings x, y, and z are said to be a *prefix*, *factor*, and *suffix* of the string $u=xyz$, respectively. A prefix, factor, and suffix of a string u is said to be *proper* if it is not u. Let $\mathit{Prefix}(u)$ be the set of prefixes of a string u, and let $\mathit{Prefix}(S)=\bigcup_{u\in S}\mathit{Prefix}(u)$ for a set S of strings. We also define the sets *Suffix* and *Factor* in a similar way. For a string $u, v \in \Sigma^*$, let

$$\begin{aligned} \mathit{lpf}_v(u) &= \text{the longest prefix of } u \text{ that is also in } \mathit{Factor}(v), \\ \mathit{lsf}_v(u) &= \text{the longest suffix of } u \text{ that is also in } \mathit{Factor}(v), \\ \mathit{lps}_v(u) &= \text{the longest prefix of } u \text{ that is also in } \mathit{Suffix}(v), \\ \mathit{lsp}_v(u) &= \text{the longest suffix of } u \text{ that is also in } \mathit{Prefix}(v). \end{aligned}$$

For a set Π of strings, let $\mathit{lpf}_\Pi(u)$ be the longest prefix of u that is also in $\mathit{Factor}(\Pi)$. We also define $\mathit{lsf}_\Pi(u)$, $\mathit{lps}_\Pi(u)$, and $\mathit{lsp}_\Pi(u)$ in a similar way.

The ith symbol of a string u is denoted by $u[i]$ for $1 \le i \le |u|$, and the factor of a string u that begins at position i and ends at position j is denoted by $u[i:j]$ for $1 \le i \le j \le |u|$. Denote by ${}^{[i]}u$ (resp. $u^{[i]}$) the string obtained by removing the length i prefix (resp. suffix) from u for $0 \le i \le |u|$. The concatenation of i copies of the same string u is denoted by u^i. The reversed string of a string u is denoted by u^R.

For a set A of integers and an integer k, let $A \oplus k = \{i + k \mid i \in A\}$ and $A \ominus k = \{i - k \mid i \in A\}$. For strings x and y, we denote the set of occurrences of x in y by $\mathit{Occ}(x, y)$. That is, $\mathit{Occ}(x, y) = \big\{i \bigm| |x| \le i \le |y|, x = y[i - |x| + 1 : i]\big\}$. For a set $\Pi \subset \Sigma^+$ of strings, $\mathit{Occ}(\Pi, y) = \bigcup_{x \in \Pi} \big\{\langle i, x\rangle \bigm| i \in \mathit{Occ}(x, y)\big\}$. Also denote by $\mathit{Occ}^\star(x, u \bullet v)$ the set of occurrences of x within the concatenation of two strings u and v which covers the boundary between u and v. That is, $\mathit{Occ}^\star(x, u \bullet v) = \big\{i \bigm| i \in \mathit{Occ}(x, uv), |u| < i < |u| + |x|\big\}$. For a set $\Pi \subset \Sigma^+$ of strings, $\mathit{Occ}^\star(\Pi, u \bullet v) = \bigcup_{x \in \Pi} \big\{\langle i, x\rangle \bigm| i \in \mathit{Occ}^\star(x, u \bullet v)\big\}$. We denote the cardinality of a set V by $|V|$.

A *period* of a string u is an integer p, $0 < p \le |u|$, such that $x[i] = x[i + p]$ for all $i \in \{1, \ldots, |x| - p\}$. The next lemma provides an important property on periods of a string.

Lemma 1 (Periodicity Lemma (see [3])). *Let p and q be two periods of a string x. If $p + q - \gcd(p, q) \le |x|$, then $\gcd(p, q)$ is also a period of x.*

The next lemma follows from the periodicity lemma.

Lemma 2. *Let x and y be strings. 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.*

4 Collage System and Text Compressions

A *collage system* [7] is a pair $\langle \mathcal{D}, \mathcal{S}\rangle$ defined as follows: $\mathcal{D}$ is a sequence of assignments $X_1 = \mathit{expr}_1;\ X_2 = \mathit{expr}_2;\ \cdots;X_\ell = \mathit{expr}_\ell$, where each X_k is a token and expr_k is any of the form

$$\begin{array}{rll} a & \text{for } a \in \Sigma \cup \{\varepsilon\}, & (\textit{primitive assignment}) \\ X_iX_j & \text{for } i, j < k, & (\textit{concatenation}) \\ {}^{[j]}X_i & \text{for } i < k \text{ and an integer } j, & (\textit{prefix truncation}) \\ X_i^{[j]} & \text{for } i < k \text{ and an integer } j, & (\textit{suffix truncation}) \\ (X_i)^j & \text{for } i < k \text{ and an integer } j. & (j\ \textit{times repetition}) \end{array}$$

Each token represents a string obtained by evaluating the expression as it implies. The strings represented by tokens are called *phrases*. The set of phrases is called *dictionary*. Denote by $X.u$ the phrase represented by a token X. For example, $\mathcal{D} : X_1 = a$; $X_2 = b$; $X_3 = c$; $X_4 = X_1 \cdot X_2$; $X_5 = X_3 \cdot X_2$; $X_6 = X_4 \cdot X_2$; $X_7 = (X_4)^3$; $X_8 = X_7^{[2]}$, then $X_4.u$, $X_5.u$, $X_6.u$, $X_7.u$, $X_8.u$ are ab, cb, abb, $ababab$, and $abab$, respectively. The *size* of $\mathcal{D}$ is the number ℓ of assignments and denoted by $\|\mathcal{D}\|$. Also denote by $F(\mathcal{D})$ the set of tokens which are defined in $\mathcal{D}$. That is, $\|\mathcal{D}\| = |F(\mathcal{D})| = \ell$. Define the *height* of a token X to be the height of the syntax tree whose root is X. The *height* of $\mathcal{D}$ is defined by $\mathit{height}(\mathcal{D}) = \max\{\mathit{height}(X) \mid X \text{ in } \mathcal{D}\}$. It expresses the maximum dependency of the tokens in $\mathcal{D}$.

On the other hand, $\mathcal{S} = X_{i_1}, \ldots, X_{i_n}$ is a sequence of tokens defined in $\mathcal{D}$. We denote by $|\mathcal{S}|$ the number k of tokens in $\mathcal{S}$. The collage system represents a string obtained by concatenating strings $X_{i_1}.u, \cdots, X_{i_n}.u$. Most text compression methods can be viewed as mechanisms to factorize a text into a series of phrases and to store a sequence of 'representations' of the phrases. In fact, various compression methods can be translated into corresponding collage systems (see [7]). Both $\mathcal{D}$ and $\mathcal{S}$ can be encoded in various ways. The compression ratios therefore depend on the encoding sizes of $\mathcal{D}$ and $\mathcal{S}$ rather than $\|\mathcal{D}\|$ and $|\mathcal{S}|$.

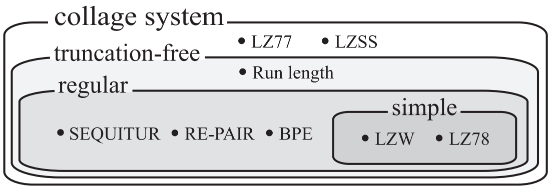

Fig. 1. Hierarchy of collage system.

A collage system is said to be *regular* if it contains neither repetition nor truncation. A regular collage system is said to be *simple* if, for every assignment $X = YZ$, $|Y.u| = 1$ or $|Z.u| = 1$. Through the collage systems, many dictionary-based compression methods can be categorized into some classes (see Fig. 1). Note that the collage systems for the SEQUITUR and the RE-PAIR are regular, and those for the LZW/LZ78 compressions are simple.

5 Main Result

Our main result is as follows.

Theorem 1. *The problem of compressed multiple pattern matching on a collage system* $\langle \mathcal{D}, \mathcal{S} \rangle$ *can be solved in* $O\big((\|\mathcal{D}\| + |\mathcal{S}|) \cdot \mathit{height}(\mathcal{D}) + m^2 + r\big)$ *time using* $O(\|\mathcal{D}\| + m^2)$ *space, where* m *is the total length of patterns in* Π*, and* r *is the* *number of pattern occurrences. If $\mathcal{D}$ contains no truncation, it can be solved in $O(\|\mathcal{D}\| + |\mathcal{S}| + m^2 + r)$ time.*

We developed in [7] an algorithm on collage system for a single pattern, which basically simulates the Knuth-Morris-Pratt (KMP) algorithm [10]. Although we devised a multipattern searching algorithm for LZ78/LZW in [9], the same technique cannot be applied directly to even the case of regular collage systems. One natural way of dealing with multiple patterns would be a simulation of the Aho-Corasick (AC) pattern matching machine. Now, we start with the definitions of $\mathit{Jump}_{\mathrm{AC}}$ and $\mathit{Output}_{\mathrm{AC}}$ which play a key role in our algorithm.

Let $\delta_{\mathrm{AC}} : Q \times \Sigma \to Q$ be the deterministic state transition function of the AC machine for Π obtained by eliminating the failure transitions (see [1]). The set Q of states has a one-to-one correspondence with *Prefix*(Π), and hence we identify Q with *Prefix*(Π) if no confusion occurs. We shall use the terms "state" and "string" interchangeably throughout the remainder of this paper. Fig. 3 is an example for $\Pi = \{aba, ababb, abca, bb\}$. Let $\mathit{Jump}_{\mathrm{AC}}$ be the state transition function. For a collage system $\langle \mathcal{D}, \mathcal{S} \rangle$ and Π, define the function $\mathit{Jump}_{\mathrm{AC}} : Q \times F(\mathcal{D}) \to Q$ by

$$\mathit{Jump}_{\mathrm{AC}}(q, X) = \delta_{\mathrm{AC}}(q, X.u).$$

We also define the set $\mathit{Output}_{\mathrm{AC}}(q, X)$ for any pair $\langle q, X \rangle$ in $Q \times F(\mathcal{D})$ by

$$\mathit{Output}_{\mathrm{AC}}(q, X) = \left\{ \langle |v|, \pi \rangle \;\middle|\; \begin{array}{l} v \text{ is a non-empty prefix of } X.u \text{ such} \\ \text{that } \pi \in \Pi \text{ is one of the outputs of} \\ \text{state } s = \delta_{\mathrm{AC}}(q, v). \end{array} \right\}$$

That is, $\mathit{Output}_{\mathrm{AC}}(q, X)$ stores all outputs emitted by the AC machine during the state transitions from the state q reading the string $X.u$. The proposed algorithm can be summarized as in Fig. 2. For example, Fig. 4 shows that the move of our algorithm on $\mathcal{S}$ for $\Pi = \{aba, ababb, abca, bb\}$, where $\mathcal{D}$ is the same as the example in Section 4 and $\mathcal{S} = X_4, X_3, X_8, X_5, X_4, X_6$.

Concerning the function $\mathit{Jump}_{\mathrm{AC}}(q, X)$, we can prove the next lemma in a similar way to [7] by regarding the string obtained by concatenating all patterns in Π as a single pattern. That is, for a set $\Pi = \{\pi_1, \pi_2, \cdots, \pi_s\}$ of patterns, we make a string $P = \pi_1\#\pi_2\#\cdots\#\pi_s$, where $\# \notin \Sigma$ is a separate character.

Lemma 3. *The function $\mathit{Jump}_{\mathrm{AC}}(q, X)$ can be realized in $O(\|\mathcal{D}\| \cdot \mathit{height}\ (\mathcal{D}) + m^2)$ time using $O(\|\mathcal{D}\| + m^2)$ space, so that it answers in $O(1)$ time. If $\mathcal{D}$ contains no truncation, the time complexity becomes $O(\|\mathcal{D}\| + m^2)$, where m is the total length of patterns in Π.*

On the other hand, the realization of $\mathit{Output}_{\mathrm{AC}}$ is not straightforward, and we need some additional efforts, which will be stated in the next section. Now, we have:

Input. A set Π of patterns and a collage system $\langle \mathcal{D}, \mathcal{S} \rangle$, where $\mathcal{S} = \mathcal{S}[1:n]$.
Output. All positions at which a pattern $\pi \in \Pi$ occurs in $\mathcal{S}[1].u \cdots \mathcal{S}[n].u$.

```
    /* Preprocessing */
    Perform the preprocessing required for $Jump_{\mathrm{AC}}$ and $Output_{\mathrm{AC}}$
    (The complexity of this part depends on $\Pi$ and $\mathcal{D}$. See Section 6);
    /* Text scanning */
    $\ell := 0$;
    $state := 0$;
    for $k := 1$ to $n$ do begin
        for each $\langle p, \pi \rangle \in Output_{\mathrm{AC}}(state, \mathcal{S}[k])$ do
            Report an occurrence of $\pi$ that ends at position $\ell + p$ ;
        $state = Jump_{\mathrm{AC}}(state, \mathcal{S}[k])$;
        $\ell := \ell + |\mathcal{S}[k].u|$
    end
```

Fig. 2. Pattern matching algorithm.

Lemma 4. *The procedure to enumerate the set* $Output_{\mathrm{AC}}\ (q, X)$ *can be realized in* $O(\|\mathcal{D}\| \cdot height(\mathcal{D}) + m^2)$ *time using* $O(\|\mathcal{D}\| + m^2)$ *space, so that it runs in* $O(height(X) + \ell)$ *time, where* ℓ *is the size of the set* $Output_{\mathrm{AC}}(q, X)$*. If* $\mathcal{D}$ *contains no truncation, it can be realized in* $O(\|\mathcal{D}\| + m^2)$ *time and space, so that it runs in* $O(\ell)$ *time.*

Theorem 1 follows from Lemma 3 and Lemma 4.

6 Realization of *Output*$_{\mathbf{AC}}$

Recall the definition of the set $Output_{\mathrm{AC}}(q, X)$. According to whether a pattern occurrence covers the boundary between the strings $q \in Prefix(\Pi)$ and $X.u$, we can partition the set $Output_{\mathrm{AC}}(q, X)$ into two disjoint subsets as follows.

$$Output_{\mathrm{AC}}(q, X) = Occ^{\star}(\Pi, q \bullet X.u) \ominus |q| \cup Occ(\Pi, X.u),$$

We consider mainly the subset $Occ(\Pi, X.u)$ below. It is easy to see that we can enumerate the set $Occ^{\star}(\Pi, q \bullet X.u)$ in $O(|Occ^{\star}(\Pi, q \bullet X.u)|)$ time if we can enumerate the set $Occ(\Pi, X.u)$ in $O(|Occ(\Pi, X.u)|)$ time, because the former is essentially the same as the problem of the concatenation case of the latter. Thus, we concentrate on proving the following lemma.

Lemma 5. *For a collage system* $\langle \mathcal{D}, \mathcal{S} \rangle$ *and a set* Π *of patterns, we can enumerate the set* $Occ(\Pi, X.u)$ *for* $X \in F(\mathcal{D})$ *in* $O(|Occ(\Pi, X.u)|)$ *time after* $O(m^2)$ *time and space preprocessing, assuming that the set* $Occ(\Pi, Y.u)$*,* $lpf_{\Pi}(Y.u)$*, and* $lsf_{\Pi}(Y.u)$ *are already computed for all* Y *such that* $\mathcal{T}(Y)$ *is a subtree of* $\mathcal{T}(X)$ *in the syntax tree.*

Now, we begin to consider the case of regular collage systems.

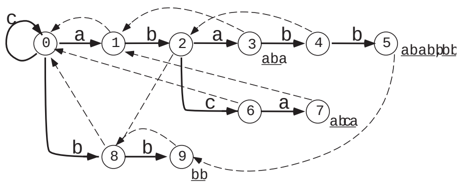

Fig. 3. Aho-Corasick machine for $\Pi = \{aba, ababb, abca, bb\}$.
The solid and the broken arrows represent the goto and the failure functions, respectively. The underlined strings adjacent to the states mean the outputs from them.

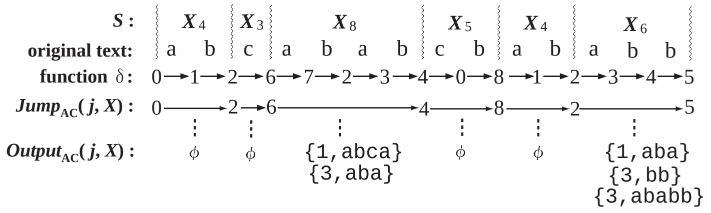

Fig. 4. Move of our algorithm.

6.1 For Regular Collage Systems

It is obvious if X is a primitive assignment. If X is a concatenation, i.e. $X = YZ$, we have $Occ(\Pi, X.u) = Occ(\Pi, Y.u) \cup Occ^{\star}(\Pi, Y.u \bullet Z.u) \cup Occ(\Pi, Z.u) \oplus |Y.u|$. Assume that $Occ(\Pi, W.u)$, $lpf_{\Pi}(W.u)$, and $lsf_{\Pi}(W.u)$ are already computed for all W such that $\mathcal{T}(W)$ is the subtree of $\mathcal{T}(X)$ in the syntax tree. Then, we need to enumerate the set $Occ^{\star}(\Pi, Y.u \bullet Z.u)$ in order to enumerate the set $Occ(\Pi, X.u)$. We can reduce the above problem to the following problem since $Occ^{\star}(\Pi, Y.u \bullet Z.u) = Occ^{\star}(\Pi, lsf_{\Pi}(Y.u) \bullet lpf_{\Pi}(Z.u))$.

Instance: A set Π of patterns and two factors x and y of Π.
Question: Enumerate the set $Occ^{\star}(\Pi, x \bullet y)$.

For the single pattern case, i.e. $\Pi = \{\pi\}$, it follows from Lemma 2 that the set $Occ^{\star}(\pi, x \bullet y)$ forms an arithmetic progression if it has more than two elements, where the step is the smallest period of π. Thus the $Occ^{\star}(\pi, x \bullet y)$ can be stored in $O(1)$ space as a pair of the minimum and the maximum values in it. The table storing those values can be computed in $O(m^2)$ time and space (see [7] for its detail).

For the multipattern case, however, we cannot apply the above technique directly to the enumeration of $Occ^{\star}(\Pi, x \bullet y)$. Now, we prove the next lemma.

Lemma 6. *For a set Π of patterns, we can enumerate the set $Occ^{\star}(\Pi, x \bullet y)$ for all pairs of $x \in Prefix(\Pi)$ and $y \in Suffix(\Pi)$ in $O(|Occ^{\star}(\Pi, x \bullet y)|)$ time, after $O(m^2)$ time and space preprocessing.*

Proof. For any $x \in Prefix(\Pi)$ and $y \in Suffix(\Pi)$, we can build in $O(m^2)$ time and space a table T that stores xy if $xy \in \Pi$, otherwise *nil*. Then, we can enumerate

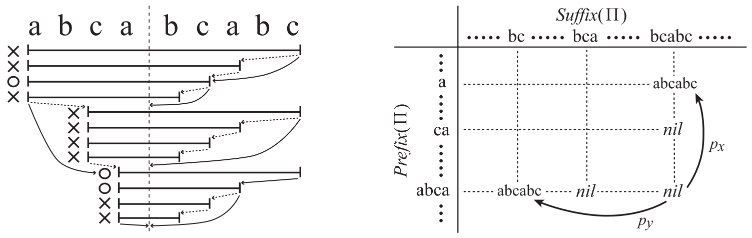

Fig. 5. Short-cut pointers and the table T for $\Pi = \{abcabc, cabb, abca\}$. In the left figure, $\circ$ indicates that $x'y'$ matches some pattern $\pi \in \Pi$ for $x' \in \mathit{Suffix}(x)$ and $y' \in \mathit{Prefix}(y)$, and $\times$ indicates that it does not match.

the set $\mathit{Occ}^\star(\Pi, x \bullet y)$ for all pairs of $x \in \mathit{Prefix}(\Pi)$ and $y \in \mathit{Suffix}(\Pi)$ by using such table T as the following manner: for each $x' \in \mathit{Suffix}(x) \cap \mathit{Prefix}(\Pi)$ and $y' \in \mathit{Prefix}(y) \cap \mathit{Suffix}(\Pi)$ in the descending order of their length, report the occurrence of the pattern $\pi = x'y'$ if $T(x', y') \neq \mathit{nil}$. However, the time complexity for the enumeration in this way becomes $O(m^2)$, not $O(|\mathit{Occ}^\star(\Pi, x \bullet y)|)$. Then, we add to each entry of the table T a pair of two short-cut pointers p_x and p_y in order to avoid increasing the time complexity, that is, for any pair of $x \in \mathit{Prefix}(\Pi)$ and $y \in \mathit{Suffix}(\Pi)$, p_x and p_y point to the longest proper suffix x' of x such that $\mathit{Occ}^\star(\Pi, x' \cdot y) \neq \emptyset$ or $x = \varepsilon$, and the longest proper prefix of y such that xy is a pattern in Π or $y = \varepsilon$, respectively. Fig. 5 shows an example of the pointers and the table T, where $x = abca$ and $y = bcabc$ for $\Pi = \{abcabc, cabb, abca\}$. Such pointers can be computed in $O(m^2)$ time by using the table T. Using these pointers, we can get the desired sequence of pairs of $x' \in \mathit{Suffix}(x)$ and $y' \in \mathit{Prefix}(y)$ in $O(|\mathit{Occ}^\star(\Pi, x \bullet y)|)$ time for any pair of $x \in \mathit{Prefix}(\Pi)$ and $y \in \mathit{Suffix}(\Pi)$. In the running example, the obtained sequence is $(abca, bcabc) \to (abca, bc) \to (abca, \varepsilon) \to (a, bcabc) \to (a, bca) \to (a, \varepsilon) \to (\varepsilon, \varepsilon)$. The proof is complete. □

We thus finished the proof of Lemma 5 restricted to the class of regular collage systems.

6.2 For Truncation-Free Collage Systems

We need to solve the following problem for dealing with repetitions.

Instance: A set Π of patterns, a factor x of Π, and an integer $k \geq 1$.
Question: Enumerate the set $\mathit{Occ}(\Pi, x^k)$.

For the single pattern case, i.e. $\Pi = \{\pi\}$, we presented a solution in [7] by using Periodicity Lemma (Lemma 1). However the same technique does not work for the multipattern case. Now, we need to prove the next lemma.

Lemma 7. *For a set Π of patterns, $x \in \mathit{Factor}(\Pi)$, and an integer $k \geq 1$, we can enumerate the set $\mathit{Occ}(\Pi, x^k)$ in $O(|\mathit{Occ}(\Pi, x^k)|)$ time after $O(m^2)$ time and* *space preprocessing, assuming that $Occ(\Pi, x)$, $lpf_\Pi(x)$, and $lsf_\Pi(x)$ are already computed.*

Proof. It is trivial for $k \leq 2$. Suppose $k > 2$. Note that we can enumerate the set $Occ^\star(\Pi, x \bullet x)$ in $O(|Occ^\star(\Pi, x \bullet x)|)$ time from Lemma 6, and that $lps_\Pi(x) = lps_\Pi(lpf_\Pi(x))$. We use a *generalized suffix trie* [5] for a set Π of strings (GST_Π for short) in order to represent the set of suffixes of the strings in Π. It is an extension of the suffix trie for a single string. Note that each node of the GST_Π corresponds to a string in $Factor(\Pi)$. The construction of the GST_Π takes $O(m^2)$ time and space.

Now, we have two cases to consider.

Case 1: $xx \notin Factor(\Pi)$. Any pattern in Π cannot cover more than three x's. We can answer in $O(1)$ time whether xx is in $Factor(\Pi)$ or not since $x \in Factor(\Pi)$ and the factor concatenation problem can be solved in $O(1)$ time. Moreover, we can obtain $Occ^\star(\Pi, x{\bullet}xx)$ since we can obtain $lpf_\Pi(xx)$ in $O(1)$ time (see [7]). Then, we can compute three sets $Occ(\Pi, x)$, $Occ^\star(\Pi, x \bullet x)$, and $Occ^\star(\Pi, x \bullet xx) \backslash Occ^\star(\Pi, x \bullet x)$. Therefore, the set $Occ(\Pi, x^k)$ can be enumerated in $O(|Occ(\Pi, x^k)|)$ time using these sets, $|x|$ and k.

Case 2: $xx \in Factor(\Pi)$. For the pattern occurrences which are within three x's, we can enumerate them in the same way as Case 1. Now, we concentrate on the enumeration of the pattern occurrences that are not within three x's. Suppose that a pattern π has such an occurrence. Then xx must be a factor of π. Since $|x|$ is a period of π and $2|x| \leq |\pi|$, it follows from Lemma 1 that $|x|$ is a multiple of the smallest period t of π and therefore the set $Occ(\pi, x^k)$ forms an arithmetic progression whose step is t. Thus the set can be enumerated in only linear time proportional to its size. However, some occurrences in the enumeration can be included entirely within three x's. In order to avoid reporting them twice, we omit p in $Occ(\pi, x^k)$ satisfying the inequation $|\pi| - (p - \lfloor p/|x| \rfloor \cdot |x|) > 2|x|$ in the enumeration. So, we can enumerate all the pattern occurrences that are not within three x's in time linearly proportional to the number of them, if we have the list of the patterns $\pi \in \Pi$ satisfying the conditions: (1) $xx \in Factor(\pi)$ and (2) $|x|$ is a period of π. The condition (2) can be replaced by the condition (2'): the smallest period of xx equals to that of π. We add a list of the patterns π that satisfy the conditions (1) and (2') to each node of GST_Π that represents a string $xx = x^2$, called a square. It is not so hard to check up on the conditions in $O(m^2)$ time for all nodes of GST_Π. Each list added to a node of GST_Π requires $O(|\Pi|) = O(m)$ space. The number of nodes representing squares is $O(m)$ (see [4]). Thus, the total space requirement is $O(m^2)$. Therefore, we can enumerate the set $Occ(\Pi, x^k)$ in $O(|Occ(\Pi, x^k)|)$ time with $O(m^2)$ time and space preprocessing.

The proof is complete. □

If $Y.u \notin Factor(\Pi)$, since any pattern in Π cannot cover more than two $Y.u$'s, it is not hard to see that $Occ(\Pi, X)$ can be enumerated in $O(|Occ(\Pi, X)|)$ time using $Occ(\Pi, Y.u)$, $Occ^{\star}(\Pi, Y.u \bullet Y.u)$, $|Y.u|$, and k. We thus finished the proof of Lemma 5 restricted to the class of truncation-free collage systems.

6.3 For General Collage Systems

For general collage systems, we must deal with truncation operations, in addition to concatenations and repetitions. Using the same technique of the single pattern case, we can see that Lemma 5 holds if $X = Y^{[k]}$, or $X = {}^{[k]}Y$ (see [7]), and that the time complexity increase by $height(\mathcal{D})$ as the single pattern case does. That is, the next lemma holds.

Lemma 8. *We can build in $O(\|\mathcal{D}\| \cdot height(\mathcal{D}) + m^2)$ time using $O(\|\mathcal{D}\| + m^2)$ space a data structure by which the enumeration of $Occ(\Pi, X.u)$ is performed in $O(height(X) + \ell)$ time, where $\ell = |Occ(\Pi, X.u)|$. If $\mathcal{D}$ contains no truncation, it can be built in $O(\|\mathcal{D}\| + m^2)$ time and space, and the enumeration requires only $O(\ell)$ time.*

Lemma 4 follows from the above. Although we need $lpf_{\Pi}(X.u)$ and $lsf_{\Pi}(X.u)$ for $X \in F(\mathcal{D})$, these can be computed in $O(\|\mathcal{D}\| \cdot height(\mathcal{D}) + m^2)$ time using $O(\|\mathcal{D}\| + m^2)$ space (see [7]).

7 On BM Type Algorithm for Multiple Patterns

We proposed in [21] a general, BM type algorithm for a single pattern on collage system. This algorithm is easily extensible to deal with multiple patterns if we use the techniques stated in Section 6. We give a brief sketch of the algorithm.

Recall that the BM algorithm on uncompressed texts performs the character comparisons in the right-to-left direction, and slides the pattern to the right using the so-called shift function when a mismatch occurs. Let $lpps_{\Pi}(w)$ denote the longest prefix of a string w that is also properly in $Suffix(\Pi)$. Note that the function $\delta_{\mathrm{AC}}^{\mathrm{rev}}$ is the state transition function of the (partial) automaton that accepts a set $\Pi^{\mathrm{R}} = \{x^{\mathrm{R}} | x \in \Pi\}$ of reversed patterns. Contrary to the case of AC machine, the set Q of states is $Suffix(\Pi)$. Define the functions $Jump_{\mathrm{MBM}}$ and $Output_{\mathrm{MBM}}$ as follows. For any state $q \in Suffix(\Pi)$ and any token $X \in F(\mathcal{D})$,

$$Jump_{\mathrm{MBM}}(q, X) = \begin{cases} lpps_{\Pi}(X.u), \text{ if } q = \varepsilon \text{ and } lpps_{\Pi}(X.u) \neq \varepsilon; \\ \delta_{\mathrm{AC}}^{\mathrm{rev}}(q, X.u), \text{ if } q \neq \varepsilon; \\ \text{undefined}, \quad \text{otherwise.} \end{cases}$$

$$Output_{\mathrm{MBM}}(q, X) = \{\pi \in \Pi \mid wq = \pi \text{ and } w \text{ is a proper suffix of } X.u\}.$$

The shift function is basically designed to shift the pattern to the right so as to align a text substring with its rightmost occurrence within the pattern. For a pattern π and a string w, let

$$rightmost\_occ_{\pi}(w) = \min\left\{ \ell > 0 \;\middle|\; \begin{array}{l} \pi[|\pi| - \ell - |w| + 1 : |\pi| - \ell] = w, \\ \text{or } \pi[1 : |\pi| - \ell] \text{ is a suffix of } w \end{array} \right\}.$$

```
/*Preprocessing for computing $Jump_{\mathrm{MBM}}(j,t)$, $Output_{\mathrm{MBM}}(j,t)$, and $Occ(t)$ */
    Preprocess the pattern $\pi$ and the dictionary $\mathcal{D}$;
/* Main routine */
    $focus :=$ an appropriate value;
    $focus := \lceil m/C \rceil$;
    while $focus \le n$ do begin
Step 1:    Report all pattern occurrences that are contained in the phrase $\mathcal{S}[focus].u$
             by using $Occ(t)$;
Step 2:    Find all pattern occurrences that end within the phrase $\mathcal{S}[focus].u$
             by using $Jump_{\mathrm{MBM}}(j,t)$ and $Output_{\mathrm{MBM}}(j,t)$;
Step 3:    Compute a possible shift $\Delta$ based on information gathered in Step 2;
           $focus := focus + \Delta$
    end
```

Fig. 6. Overview of BM type compressed pattern matching algorithm.

For a set Π of patterns, let $rightmost\_occ_{\Pi}(w) = \min_{\pi \in \Pi}\{rightmost\_occ_{\pi}\ (w)\}$, and let $Shift_{\mathrm{MBM}}(q, X) = rightmost\_occ_{\Pi}(X.u \cdot q)$. When we encounter a mismatch against a token X in state $q \in Suffix(\Pi)$, the possible shift Δ of the focus can be computed using $Shift_{\mathrm{MBM}}$ in the same way as [21]. Figure 6 gives an overview of our algorithm.

We can prove the next lemma by using the techniques similar to those stated in Section 6, and Theorem 2 follows from Lemma 9.

Lemma 9. *The functions $Jump_{\mathrm{MBM}}$, $Output_{\mathrm{MBM}}$, and $Shift_{\mathrm{MBM}}$ can be built in $O(height(\mathcal{D}) \cdot \|\mathcal{D}\| + m^2)$ time and $O(\|\mathcal{D}\| + m^2)$ space, so that they answer in $O(1)$ time, where m is the total length of patterns in Π. The factor $height(\mathcal{D})$ can be dropped if $\mathcal{D}$ contains no truncation.*

Thus, we have the following theorem.

Theorem 2. *The BM type algorithm for multiple pattern searching on collage system runs in $O(height(\mathcal{D})\cdot(\|\mathcal{D}\|+|\mathcal{S}|)+|\mathcal{S}|\cdot m+m^2+r)$ time, using $O(\|\mathcal{D}\|+m^2)$ space, where m is the total length of patterns in Π, and r is the number of pattern occurrences. If $\mathcal{D}$ contains no truncation, the time complexity becomes $O(\|\mathcal{D}\| + |\mathcal{S}| \cdot m + m^2 + r)$.*

8 Parallel Complexity of Compressed Pattern Matching

In this section, we consider the computational complexity of the following decision problem for a class $\mathcal{C}$ of collage systems:

Instance: A collage system $\langle \mathcal{D}, \mathcal{S} \rangle$ in $\mathcal{C}$ over Σ and a set $\Pi = \{\pi_1, \cdots, \pi_s\}$ of patterns.

Question: Is there any pattern $\pi_j \in \Pi$ that occurs in the text T represented by $\langle \mathcal{D}, \mathcal{S} \rangle$? That is, are there any i and j such that $T[i : i + |\pi_j| - 1] = \pi_j$ or not?

LogCFL is the class of problems logspace-reducible to a context-free language. An auxiliary pushdown automaton (*AuxPDA*) is a nondeterministic Turing machine with a read-only input tape, a space-bounded worktape, and a pushdown store that is not subject to the space-bound. The class of languages accepted by auxiliary pushdown automata in space $s(n)$ and time $t(n)$ is denoted by $\mathit{AuxPDA}(s(n), t(n))$. The next lemma is quite useful.

Lemma 10 ([22]). $\mathit{LogCFL} = \mathit{AuxPDA}(\log n, n^{O(1)})$.

We now show the following theorem.

Theorem 3. *Compressed pattern matching problem on regular collage system is in LogCFL.*

Proof. We show an auxiliary pushdown automaton M that accepts an input string if and only if there is some pattern $\pi_j \in \Pi$ that occurs in the text T represented by $\langle \mathcal{D}, \mathcal{S} \rangle$. We note that by using pushdown store, M can traverse the evaluation tree of any variable X_k and 'scan' the string $X_k.u$ from left to right that is the sequence of leaves in the tree. Moreover, by utilizing the nondeterminism, M can scan any substring of $X_k.u$.

M represents a position t of a pattern as a binary string in the worktape, and initializes it $t = 1$. For simplicity, we first consider the case that a pattern π_j occurs within the string $X_{i_k}.u$ for some X_{i_k}. M nondeterministically guesses such j and k, and nondeterministically goes down the evaluation tree of X_k from the root by pushing the traversed variables in the pushdown store. At a leaf, M confirms that the character $X_k.u[l]$ is equal to $\pi_j[t]$. Then M increments t by one by using the worktape, and proceeds to the next character $X_k.u[l+1]$ by using the pushdown store. M repeats this procedure until π_j is verified to occur in X_k at position l. Remark that l is not explicitly written in the worktape: it is impossible in general since $l = O(|X_k|) = O(2^{||\mathcal{D}||})$. However, on the other hand, since $t \leq |\pi_j|$ and patterns are explicitly written in the input tape, the space required by M is $O(\log |\pi_j|)$, that is logarithmic with respect to the input size. The computation time is clearly bounded by a polynomial, since the height of the evaluation tree is at most $||D||$. For a general case that a pattern π_j spreads over a region $X_{i_k} \cdot X_{i_{k+1}} \cdots X_{i_h}$, we can show that M verifies the occurrences in polynomial time using a log-space worktape in the same way. By Lemma 10, we complete the proof. □

Since it is known that $\mathit{LogCFL} \subseteq \mathbf{NC}^2$ [17,18], the above theorem implies that the compressed pattern matching for regular collage systems can be efficiently parallelized in principle. For general collage systems including repetitions and truncations, we have not succeeded to show that the problems are in **NC** nor **P**-complete yet.

9 Concluding Remarks

We proposed two types of multipattern matching algorithms on collage system. One is an AC-type algorithm, which runs in $O((||\mathcal{D}|| + |\mathcal{S}|) \cdot \mathit{height}(\mathcal{D}) + m^2 + r)$

time with $O(\|\mathcal{D}\| + m^2)$ space. Its running time becomes $O(\|\mathcal{D}\| + |\mathcal{S}| + m^2 + r)$ if a collage system contains no truncation. The other is a BM-type algorithm, which runs in $O((\mathit{height}(\mathcal{D}) + m)|\mathcal{S}| + r)$ time after an $O(\|\mathcal{D}\| \cdot \mathit{height}(\mathcal{D}) + m^2)$ time preprocessing with $O(\|\mathcal{D}\| + m^2)$ space. We also showed that compressed pattern matching on regular collage system is in $\mathit{LogCFL} \subseteq \mathbf{NC}^2$.

The compressed pattern matching usually aims to search in compressed files faster than a regular decompression followed by an ordinary search (Goal 1). A more ambitious goal is to perform a faster search in compressed files *in comparison with an ordinary search in the original files* (Goal 2). In this case, the aim of compression is not only to reduce disk storage requirement but also to speed up string searching task. In fact, we have achieved Goal 2 for the compression method called Byte Pair Encoding (BPE) [21,20,23].

In [6,12], approximate string matching algorithms over LZW/LZ78 compressed texts were proposed. Very recently, Navarro et al. proposed a practical solution for the LZW/LZ78 compressions and showed experimentally that it is up to three times faster than the trivial approach of uncompressing and searching [13]. The basic idea of the solution is to reduce the problem of approximate string searching to the problem of multipattern searching of a set of pattern pieces plus local decompression and direct verification of candidate text areas. Using the same technique, the result of this paper leads to speed-up of approximate string matching for various compression methods. In fact, we have verified that the suggested algorithm runs on BPE compressed texts faster than *Agrep*, known as the fastest pattern matching tool.

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. M. Crochemore and W. Rytter. *Text Algorithms.* Oxford University Press, New York, 1994.
4. A.S. Fraenkel and J. Simpson. How many squares can a string contain? *J. Combin. Theory Ser. A*, 82:112–120, 1998.
5. L.C.K. Hui. Color set size problem with application to string matching. In *Combinatorial Pattern Matching*, volume 644 of *Lecture Notes in Computer Science*, pages 230–243. Springer-Verlag, 1992.
6. J. Kärkkäinen, G. Navarro, and E. Ukkonen. Approximate string matching over Ziv-Lempel compressed text. In *Proc. 11th Ann. Symp. on Combinatorial Pattern Matching*, volume 1848 of *Lecture Notes in Computer Science*, pages 195–209. Springer-Verlag, 2000.
7. T. Kida, Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa. A unifying framework for compressed pattern matching. In *Proc. 6th International Symp. on String Processing and Information Retrieval*, pages 89–96. IEEE Computer Society, 1999.
8. T. Kida, M. Takeda, A. Shinohara, M. Miyazaki, and S. Arikawa. Multiple pattern matching in LZW compressed text. *Journal of Discrete Algorithms.* to appear (previous version in DCC'98 and CPM'99).

9. T. Kida, M. Takeda, A. Shinohara, M. Miyazaki, and S. Arikawa. Multiple pattern matching in LZW compressed text. In J. A. Storer and M. Cohn, editors, *Proc. Data Compression Conference '98*, pages 103–112. IEEE Computer Society, 1998.
10. D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM J. Comput*, 6(2):323–350, 1977.
11. N.J. Larsson and A. Moffat. Offline dictionary-based compression. In *Proc. Data Compression Conference '99*, pages 296–305. IEEE Computer Society, 1999.
12. T. Matsumoto, T. Kida, M. Takeda, A. Shinohara, and S. Arikawa. Bit-parallel approach to approximate string matching in compressed texts. In *Proc. 7th International Symp. on String Processing and Information Retrieval*, pages 221–228. IEEE Computer Society, 2000.
13. G. Navarro, T. Kida, M. Takeda, A. Shinohara, and S. Arikawa. Faster approximate string matching over compressed text. In *Proc. Data Compression Conference 2001*. IEEE Computer Society, 2001. to appear.
14. G. Navarro and M. Raffinot. A general practical approach to pattern matching over Ziv-Lempel compressed text. In *Proc. 10th Ann. Symp. on Combinatorial Pattern Matching*, volume 1645 of *Lecture Notes in Computer Science*, pages 14–36. Springer-Verlag, 1999.
15. G. Navarro and J. Tarhio. Boyer-Moore string matching over Ziv-Lempel compressed text. In *Proc. 11th Ann. Symp. on Combinatorial Pattern Matching*, volume 1848 of *Lecture Notes in Computer Science*, pages 166–180. Springer-Verlag, 2000.
16. C.G. Nevill-Manning, I.H. Witten, and D.L. Maulsby. Compression by induction of hierarchical grammars. In *Proc. Data Compression Conference '94*, pages 244–253. IEEE Press, 1994.
17. W. Ruzzo. Tree-size bounded alternation. *Journal of Computer and System Sciences*, 21(2):218–235, 1980.
18. W. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Sciences*, 22(3):365–383, 1981.
19. W. Rytter. Algorithms on compressed strings and arrays. In *Proc. 26th Ann. Conf. on Current Trends in Theory and Practice of Infomatics*. Springer-Verlag, 1999.
20. Y. Shibata, T. Kida, S. Fukamachi, M. Takeda, A. Shinohara, T. Shinohara, and S. Arikawa. Speeding up pattern matching by text compression. In *Proc. 4th Italian Conference on Algorithms and Complexity*, volume 1767 of *Lecture Notes in Computer Science*, pages 306–315. Springer-Verlag, 2000.
21. Y. Shibata, T. Matsumoto, M. Takeda, A. Shinohara, and S. Arikawa. A Boyer-Moore type algorithm for compressed pattern matching. In *Proc. 11th Ann. Symp. on Combinatorial Pattern Matching*, volume 1848 of *Lecture Notes in Computer Science*, pages 181–194. Springer-Verlag, 2000.
22. I. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of ACM*, 25:405–414, 1978.
23. M. Takeda, Y. Shibata, T. Matsumoto, T. Kida, A. Shinohara, S. Fukamachi, T. Shinohara, and S. Arikawa. Speeding up string pattern matching by text compression: The dawn of a new era. *Transactions of Information Processing Society of Japan*, 2001. to appear.