International Journal of Foundations of Computer Science

A FULLY COMPRESSED PATTERN MATCHING ALGORITHM FOR SIMPLE COLLAGE SYSTEMS

SHUNSUKE INENAGA* †
Department of Informatics, Kyushu University 33, Fukuoka 812-8581, Japan
shunsuke.inenaga@i.kyushu-u.ac.jp

AYUMI SHINOHARA‡
Graduate School of Information Sciences, Tohoku University, Sendai 980-8579, Japan
ayumi@ecei.tohoku.ac.jp

and

MASAYUKI TAKEDA
Department of Informatics, Kyushu University 33, Fukuoka 812-8581, Japan
SORST, Japan Science and Technology Agency (JST)
takeda@i.kyushu-u.ac.jp

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

We study the *fully compressed pattern matching problem* (*FCPM problem*): Given $\mathcal{T}$ and $\mathcal{P}$ which are descriptions of text T and pattern P respectively, find the occurrences of P in T *without decompressing* $\mathcal{T}$ *or* $\mathcal{P}$. This problem is rather challenging since patterns are also given in a compressed form. In this paper we present an FCPM algorithm for *simple collage systems*. Collage systems are a general framework representing various kinds of dictionary-based compressions in a uniform way, and simple collage systems are a subclass that includes LZW and LZ78 compressions. Collage systems are of the form $\langle \mathcal{D}, \mathcal{S} \rangle$, where $\mathcal{D}$ is a dictionary and $\mathcal{S}$ is a sequence of variables from $\mathcal{D}$. Our FCPM algorithm performs in $O(||\mathcal{D}||^2 + mn \log |\mathcal{S}|)$ time, where $n = |\mathcal{T}| = ||\mathcal{D}|| + |\mathcal{S}|$ and $m = |\mathcal{P}|$. This is faster than the previous best result of $O(m^2 n^2)$ time.

Keywords: string processing, text compression, fully compressed pattern matching, collage systems, algorithm

1. Introduction

* Main part of this research was done when the author was visiting the Department of Computer Science, the University of Helsinki, Finland.

† Supported by JSPS Research Fellowships for Young Scientists

‡ Main part of this research was done when the author was working for the Department of Informatics, Kyushu University, Japan, and PRESTO, Japan Science and Technology Agency (JST).

The *compressed pattern matching problem* (*CPM problem*) [1] is a challenging problem in Stringology such that, given compressed text $\mathcal{T}$ and uncompressed pattern P, find the pattern occurrences *without decompressing* $\mathcal{T}$. This problem has been intensively studied for a variety of text compression schemes, e.g. [2, 4, 3, 17].

An ultimate extension of the CPM problem is the *fully compressed pattern matching problem* (*FCPM problem*) [10] where *both* text T and pattern P are given in compressed forms $\mathcal{T}$ and $\mathcal{P}$ respectively, and the objective is to find all occurrences of P in T *without decompressing* $\mathcal{T}$ *or* $\mathcal{P}$. Miyazaki et al. [18] presented an algorithm to solve the FCPM problem for *straight line programs*, in $O(m^2n^2)$ time using $O(mn)$ space, where $m = |\mathcal{P}|$ and $n = |\mathcal{T}|$. For LZW compressed text $\mathcal{T}$ and pattern $\mathcal{P}$, Gąsieniec and Rytter [7] addressed a pattern matching algorithm running in $O((m+n)\log(m+n))$ time but this one *explicitly decompresses* part of $\mathcal{T}$ or $\mathcal{P}$ when the decompressed size does not exceed n. Hence their algorithm does not really solve the FCPM problem where pattern matching *without* any decompressing is strictly required. Therefore, the best known result for the FCPM problem on LZW is $O(m^2n^2)$ time and $O(mn)$ space by Miyazaki et al. [18].

In this paper, we consider the FCPM problem on *simple collage systems* which are a subclass of *collage systems* [11]. Collage systems are a general framework that represents various compression schemes such as LZ family [22, 20, 23, 21], run-length encoding, BPE [5], RE-PAIR [15], SEQUITUR [19], grammar transform [12, 14, 13], and straight line programs. A collage system is a pair $\langle \mathcal{D}, \mathcal{S} \rangle$ where $\mathcal{D}$ is a dictionary and $\mathcal{S}$ is a sequence of variables from $\mathcal{D}$. Simple collage systems [16] are a subclass of collage systems including LZ78 [23] and LZW [21]. Simple collage systems are very attractive in terms of accelerating CPM [16] despite of their generally weaker compression ratio.

In this paper, we present an efficient FCPM algorithm for simple collage systems, which runs in $O(\|\mathcal{D}\|^2 + mn\log|\mathcal{S}|)$ time with $O(\|\mathcal{D}\|^2 + mn)$ space, where $\|\mathcal{D}\|$ denotes the size of the dictionary $\mathcal{D}$, and $|\mathcal{S}|$ the length of the sequence $\mathcal{S}$. A preliminary version of this work has appeared in [8]. Although our algorithm requires more space than the algorithm by Miyazaki et al. [18], it consumes less time. In addition, since simple collage systems are a general framework, for our algorithm a text and a pattern may be compressed by different compression schemes. Namely, our algorithm is so flexible that it can deal with an LZ78-compressed text and an LZW-compressed pattern, and vice versa. Since it is natural to assume that a text and a pattern can be chosen from difference sources, this feature can be a practical advantage of our algorithm.

2. Preliminary

Let $\mathcal{N}$ be the set of natural numbers, and $\mathcal{N}^+$ be the set of positive integers. Let Σ be a finite *alphabet*. An element of Σ^* is called a *string*. The length of a string T is denoted by $|T|$. The i-th character of a string T is denoted by $T[i]$ for $1 \leq i \leq |T|$, and the substring of a string T that begins at position i and ends at position j is denoted by $T[i:j]$ for $1 \leq i \leq j \leq |T|$. A *period* of a string T is an integer p $(1 \leq p \leq |T|)$ such that $T[i] = T[i+p]$ for any $i = 1, 2, \ldots, |T| - p$.

Collage systems [11] are a general framework that enables us to capture the structure of different types of dictionary-based compressions. *Regular* collage systems, which are a subclass of collage systems, are pair $\langle \mathcal{D}, \mathcal{S} \rangle$ such that $\mathcal{D}$ is a sequence of assignments

$$X_1 = expr_1, X_2 = expr_2, \ldots, X_h = expr_h,$$

where X_k are variables and $expr_k$ are expressions of either of the form

$$\begin{array}{lll} a & \text{where } a \in (\Sigma \cup \varepsilon), & (\textit{primitive assignment}) \\ X_i X_j & \text{where } i, j < k, & (\textit{concatenation}) \end{array}$$

and $\mathcal{S}$ is a sequence of variables $X_{i_1}, X_{i_2}, \ldots, X_{i_s}$ obtained from $\mathcal{D}$. The size of $\mathcal{D}$ is h and is denoted by $\|\mathcal{D}\|$, and the size of $\mathcal{S}$ is s and is denoted by $|\mathcal{S}|$. The total size of the collage system $\langle \mathcal{D}, \mathcal{S} \rangle$ is $n = \|\mathcal{D}\| + |\mathcal{S}| = h + s$.

A regular collage system is said to be *simple* if, for any variable $X = X_\ell X_r$, either $|X_\ell| = 1$ or $|X_r| = 1$ [16]. LZW [21] and LZ78 [23] are simple collage systems formalized as follows.

LZW. $\mathcal{S} = X_{i_1}, X_{i_2}, \ldots, X_{i_s}$ and $\mathcal{D}$ is the following:

$$\begin{array}{l} X_1 = a_1;\ X_2 = a_2;\ \ldots;\ X_q = a_q; \\ X_{q+1} = X_{i_1} X_{\sigma(i_2)};\ X_{q+2} = X_{i_2} X_{\sigma(i_3)};\ \ldots;\ X_{q+s-1} = X_{i_{s-1}} X_{\sigma(i_s)}, \end{array}$$

where the alphabet is $\Sigma = \{a_1, a_2, \ldots, a_q\}$, $1 \leq i_1 \leq q$, and $\sigma(j)$ denotes the integer k $(1 \leq k \leq q)$ such that a_k is the first symbol of X_j.

LZ78. $\mathcal{S} = X_1, X_2, \ldots, X_s$ and $\mathcal{D}$ is the following:

$$X_0 = \varepsilon;\ X_1 = X_{i_1} b_1;\ X_2 = X_{i_2} b_2;\ \ldots;\ X_s = X_{i_s} b_s;$$

where b_j is a symbol in Σ.

In this paper, we study the *fully compressed pattern matching problem for simple collage systems*: Given two simple collage systems that are the descriptions of text T and pattern P, find all occurrences of P in T. Namely, we compute the following set:

$$Occ(T, P) = \{ i \mid T[i : i + |P| - 1] = P \}.$$

We emphasize that our goal is to solve this problem *without decompressing either of the two simple collage systems.* Our result is the following:

Theorem 1 *Given two simple collage systems $\langle \mathcal{D}, \mathcal{S} \rangle$ and $\langle \mathcal{D}', \mathcal{S}' \rangle$ that are the description of T and P respectively, $Occ(T, P)$ can be computed in $O(\|\mathcal{D}\|^2 + mn \log |\mathcal{S}|)$ time using $O(\|\mathcal{D}\|^2 + mn)$ space, where $n = \|\mathcal{D}\| + |\mathcal{S}|$ and $m = \|\mathcal{D}'\| + |\mathcal{S}'|$.*

3. Overview of algorithm

3.1. Translation to straight line programs

Consider a regular collage system $\langle \mathcal{D}, \mathcal{S} \rangle$. Note that $\mathcal{S} = X_{i_1}, X_{i_2}, \ldots, X_{i_s}$ can be translated in linear time to a sequence of assignments of size s. For instance, $\mathcal{S} = X_1, X_2, X_3, X_4$ can be rewritten to $X_5 = X_1X_2$; $X_6 = X_5X_3$; $X_7 = X_6X_4$, and $S = X_7$. Therefore, a regular collage system, which represents string $T \in \Sigma^*$, can be seen as a context free grammar of the Chomsky normal form that generates only T, and thus correspond to *straight line programs* (*SLPs*). In the sequel, for string $T \in \Sigma^*$, let $\mathcal{T}$ denote the SLP representing T. The size of $\mathcal{T}$ is denoted by $\|\mathcal{T}\|$, and $\|\mathcal{T}\| = \|\mathcal{D}\| + |\mathcal{S}| = h + s = n$.

Now we introduce *simple* straight line programs (*SSLP*) that correspond to simple collage systems.

Definition 1 *An SSLP $\mathcal{T}$ is a sequence of assignments such that*

$$X_1 = expr_1;\ X_2 = expr_2;\ \ldots;\ X_n = expr_n,$$

where X_i are variables and $expr_i$ are expressions of any of the form

$$\begin{array}{lll} a & \textit{where } a \in \Sigma & \text{(primitive)}, \\ X_\ell X' & \textit{where } \ell < i \textit{ and } X' = a & \text{(right simple)}, \\ X' X_r & \textit{where } r < i \textit{ and } X' = a & \text{(left simple)}, \\ X_\ell X_r & \textit{where } \ell, r < i & \text{(complex)}, \end{array}$$

and $\mathcal{T} = X_n$. Moreover, each type of variable satisfies the following properties:

- *For any right simple variable $X_i = X_\ell X'$, X_ℓ is either simple or primitive.*
- *For any left simple variable $X_i = X' X_r$, X_r is either simple or primitive.*
- *For any complex variable $X_i = X_\ell X_r$, X_r is either simple or primitive.*

An example of an SSLP $\mathcal{T}$ for string $T = \texttt{abaabababb}$ is as follows:

$$X_1 = \texttt{a}, X_2 = \texttt{b}, X_3 = X_1X_2, X_4 = X_1X_3, X_5 = X_3X_1, X_6 = X_2X_2,$$
$$X_7 = X_3X_4, X_8 = X_7X_5, X_9 = X_8X_6,$$

and $\mathcal{T} = X_9$. See also Fig. 1 that illustrates the derivation tree of $\mathcal{T}$.

X_1 and X_2 are primitive variables, X_3, X_4, X_5 and X_6 are simple variables, and X_7, X_8 and X_9 are complex variables.

For any simple collage system $\langle \mathcal{D}, \mathcal{S} \rangle$, let $\mathcal{T}$ be its corresponding SSLP. Let $\|\mathcal{D}\| = h$ and $|\mathcal{S}| = s$. Then the total number of primitive and simple variables in $\mathcal{T}$ is h, and the number of complex variables in $\mathcal{T}$ is s.

In the sequel, we consider computing $Occ(T, P)$ for given SSLPs $\mathcal{T}$ and $\mathcal{P}$. We use X and X_i for variables of $\mathcal{T}$, and Y and Y_j for variables of $\mathcal{P}$. When not confusing, X_i (Y_j, respectively) also denotes the string derived from X_i (Y_j, respectively). Let $\|\mathcal{T}\| = n$ and $\|\mathcal{P}\| = m$.

Proposition 1 *For any simple variable X, $|X| = \|X\|$, where $\|X\|$ denotes the number of variables in X.*

3.2. Basic idea of algorithm

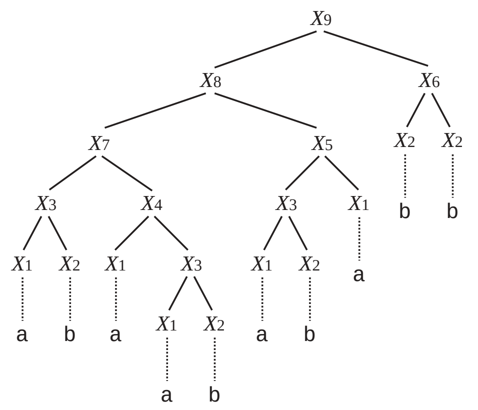

Fig. 1. Derivation tree of SSLP for string abaabababb.

In this section, we show a basis of our algorithm that outputs a compact representation of $Occ(T, P)$ for given SSLPs $\mathcal{T}, \mathcal{P}$.

For strings $X, Y \in \Sigma^*$ and integer $k \in \mathcal{N}$, we define the set of all occurrences of Y that cover or touch the position k in X by

$$Occ^{\uparrow}(X, Y, k) = \{i \in Occ(X, Y) \mid k - |Y| \leq i \leq k\}.$$

In the following, $[i, j]$ denotes the set $\{i, i+1, \ldots, j\}$ of consecutive integers. For a set U of integers and an integer k, we denote $U \oplus k = \{i + k \mid i \in U\}$ and $U \ominus k = \{i - k \mid i \in U\}$.

Observation 1 *For any strings $X, Y \in \Sigma^*$ and integer $k \in \mathcal{N}$,*

$$Occ^{\uparrow}(X, Y, k) = Occ(X, Y) \cap [k - |Y|, k].$$

Lemma 1 *For any strings $X, Y \in \Sigma^*$ and integer $k \in \mathcal{N}$, $Occ^{\uparrow}(X, Y, k)$ forms a single arithmetic progression.*

For positive integers $p, d \in \mathcal{N}^+$ and non-negative integer $t \in \mathcal{N}$, we define $\langle p, d, t\rangle = \{p + (i-1)d \mid i \in [1, t]\}$. Note that t denotes the cardinality of the set $\langle p, d, t\rangle$. By Lemma 1, $Occ^{\uparrow}(X, Y, k)$ can be represented as the triple $\langle p, d, t\rangle$ with the minimum element p, the common difference d, and the length t of the progression. By *'computing $Occ^{\uparrow}(X, Y, k)$'*, we mean to calculate the triple $\langle p, d, t\rangle$ such that $\langle p, d, t\rangle = Occ^{\uparrow}(X, Y, k)$.

Observation 2 *Assume each of sets A_1 and A_2 of integers forms a single arithmetic progression, and is represented by a triple $\langle p, d, t\rangle$. Then, the union $A_1 \cup A_2$ can be computed in constant time.*

Lemma 2 ([9]) *Let $\langle p, d, t\rangle = Occ^{\uparrow}(X, Y, k)$ for strings $X, Y \in \Sigma^*$ and integer $k \in \mathcal{N}$. If $t \geq 1$, then d is the shortest period of $X[p : q + |Y| - 1]$ where $q = p + (t-1)d$.*

Lemma 3 *For any strings $X, Y_1, Y_2 \in \Sigma^*$ and integers $k_1, k_2 \in \mathcal{N}$, the intersection $Occ^{\uparrow}(X, Y_1, k_1) \cap (Occ^{\uparrow}(X, Y_2, k_2) \ominus |Y_1|)$ can be computed in $O(1)$ time, provided that $Occ^{\uparrow}(X, Y_1, k_1)$ and $Occ^{\uparrow}(X, Y_2, k_2)$ are already computed.*

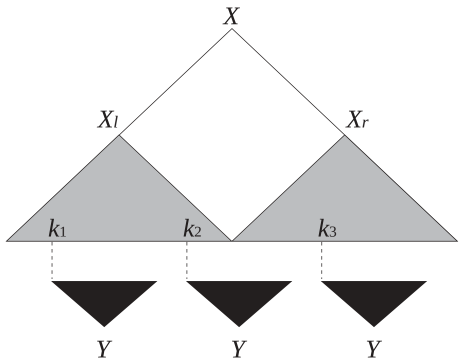

Fig. 2. $k_1, k_2, k_3 \in Occ(X, Y)$, where $k_1 \in Occ(X_\ell, Y)$, $k_2 \in Occ^\triangle(X, Y)$ and $k_3 \in Occ(X_r, Y)$.

For variables $X = X_\ell X_r$ and Y, we denote $Occ^\triangle(X, Y) = Occ^\uparrow(X, Y, |X_\ell|+1)$. The following observation is explained in Fig. 2.

Observation 3 ([18]) *For any variables $X = X_\ell X_r$ and Y,*

$$Occ(X, Y) = Occ(X_\ell, Y) \cup Occ^\triangle(X, Y) \cup (Occ(X_r, Y) \oplus |X_\ell|).$$

Observation 3 implies that $Occ(X_n, Y)$ can be represented by a combination of

$$\{Occ^\triangle(X_i, Y)\}_{i=1}^n = Occ^\triangle(X_1, Y), Occ^\triangle(X_2, Y), \ldots, Occ^\triangle(X_n, Y).$$

Thus, the desired output $Occ(T, P) = Occ(X_n, Y_m)$ can be expressed as a combination of $\{Occ^\triangle(X_i, Y_m)\}_{i=1}^n$ that requires $O(n)$ space. Hereby, computing $Occ(T, P)$ is reduced to computing $Occ^\triangle(X_i, Y_m)$ for every $i = 1, 2, \ldots, n$. In computing each $Occ^\triangle(X_i, Y_j)$ recursively, the same set $Occ^\triangle(X_{i'}, Y_{j'})$ might repeatedly be referred to, for $i' < i$ and $j' < j$. Therefore we take the dynamic programming strategy. We use an $m \times n$ table App where each entry $App[i, j]$ at row i and column j stores the triple for $Occ^\triangle(X_i, Y_i)$. We compute each $App[i, j]$ in a bottom-up manner, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$. In the following sections, we will show that the whole table App can be computed in $O(h^2 + mn \log s)$ time using $O(h^2 + mn)$ space, where h is the number of simple variables in $\mathcal{T}$ and s is the number of complex variables in $\mathcal{T}$. This leads to the result of Theorem 1.

4. Details of algorithm

In this section, we show how to compute each $Occ^\triangle(X_i, Y_j)$ efficiently. Our result is as follows:

Lemma 4 *For any variables X_i of $\mathcal{T}$ and Y_j of $\mathcal{P}$, $Occ^\triangle(X_i, Y_j)$ can be computed in $O(\log s)$ time, with extra $O(h^2 + mn)$ work time and space.*

The key to prove this lemma is, given integer k, to pre-compute $Occ^\uparrow(X_{i'}, Y_{j'}, k)$ for any $1 \le i' < i$ and $1 \le j' < j$. In case that X is simple, we have the following lemma:

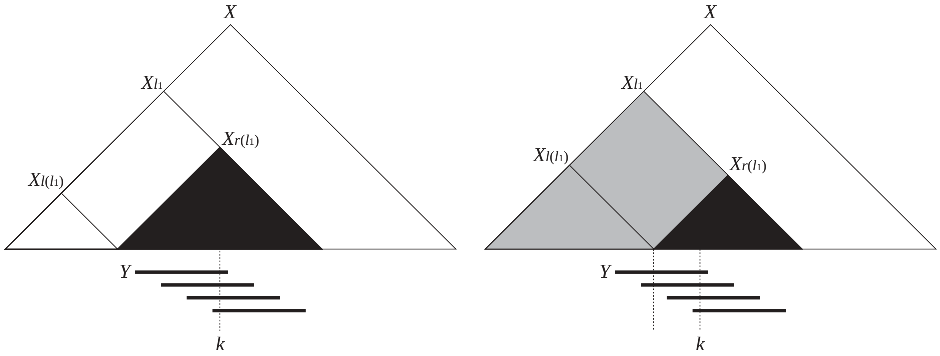

Fig. 3. In the left case, all the occurrences are covered by $Occ^{\uparrow}(X_{r(\ell_1)}, Y, k) \oplus |X_{\ell(\ell_1)}|$. In the right case, the first and second occurrences are covered by $Occ^{\triangle}(X_{\ell_1}, Y)$ and the third and fourth occurrences by $Occ^{\uparrow}(X_{r(\ell_1)}, Y, k) \oplus |X_{\ell(\ell_1)}|$.

Lemma 5 *Let X be any simple variable of $\mathcal{T}$ and Y be any variable of $\mathcal{P}$. Given integer $k \in \mathcal{N}$, $Occ^{\uparrow}(X, Y, k)$ can be computed in $O(1)$ time, with extra $O(h^2 + mh)$ work time and space.*

As a counterpart to Lemma 5, we have the following lemma for X to be complex:

Lemma 6 *Let X be any complex variable of $\mathcal{T}$ and Y be any variable of $\mathcal{P}$. Given integer $k \in \mathcal{N}$, $Occ^{\uparrow}(X, Y, k)$ can be computed in $O(\log s)$ time with extra $O(ms)$ work time and space.*

For any complex variable $X = X_\ell X_r$, let $range(X)$ denote the range $[r_1, r_2]$ such that $T[r_1, r_2] = X_r$. It is clear that for each complex variable its range is uniquely determined, since each complex variable appears in $\mathcal{T}$ exactly once. In proving Lemma 6 above, Lemma 7 and Lemma 8 below are used.

Lemma 7 *Let $X = X_\ell X_r$ be any complex variable of $\mathcal{T}$ and let Y be any variable of $\mathcal{P}$. Assume $Occ^{\uparrow}(X_\ell, Y, |X_\ell| - |Y| + 1)$ and $Occ^{\triangle}(X, Y)$ are already computed. Then $Occ^{\uparrow}(X, Y, |X| - |Y| + 1)$ can be computed in $O(1)$ time, with extra $O(ms)$ work space.*

Lemma 8 *Given integer $k \in \mathcal{N}$, we can retrieve in $O(\log s)$ time the complex variable X such that $range(X) = [r_1, r_2]$ and $r_1 \leq k \leq r_2$, after a preprocessing taking $O(s)$ time and space.*

Now the proof of Lemma 6 follows.

Proof. Let $A = Occ^{\uparrow}(X, Y, k)$. Let X_{ℓ_1} be the complex variable such that $k \in range(X_{\ell_1})$, and let $X_{\ell_1} = X_{\ell(\ell_1)} X_{r(\ell_1)}$. Let X_{ℓ_2} be the complex variable satisfying $k - |Y| \in range(X_{\ell_2})$, and let $X_{\ell_2} = X_{\ell(\ell_2)} X_{r(\ell_2)}$. There are the three following cases:

(1) when $k - |Y| \geq |X_{\ell(\ell_1)}| + 1$ and $k + |Y| - 1 \leq |X_{\ell_1}|$ (Fig. 3, left).
In this case, we have $A = Occ^{\uparrow}(X_{r(\ell_1)}, Y, k) \oplus |X_{\ell(\ell_1)}|$.

(2) when $k - |Y| < |X_{\ell(\ell_1)}| + 1$ and $k + |Y| - 1 \leq |X_{\ell_1}|$ (Fig. 3, right).

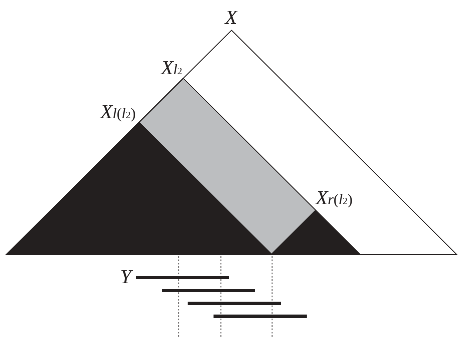

Fig. 4. In this case, the first and second occurrences are covered by $Occ^{\uparrow}(X_{\ell(\ell_2)}, Y, |X_{\ell(\ell_2)}| - |Y| + 1)$ and the third and fourth occurrences are covered by $Occ^{\triangle}(X_{\ell_2}, Y)$.

In this case, we have

$$A = (Occ^{\triangle}(X_{\ell_1}, Y) \cap [k - |Y|, X_{\ell(\ell_1)} + 1]) \cup (Occ^{\uparrow}(X_{r(\ell_1)}, Y, k) \oplus |X_{\ell(\ell_1)}|).$$

(3) when $k + |Y| - 1 > |X_{\ell_1}|$ (Fig. 4).
In this case, we have

$$\begin{aligned} A &= (Occ^{\uparrow}(X_{\ell(\ell_2)}, Y, |X_{\ell(\ell_2)}| - |Y| + 1) \cap [k - |Y|, |X_{\ell(\ell_2)}| - |Y| + 1]) \\ &\cup (Occ^{\triangle}(X_{\ell_2}, Y) \cap [|X_{\ell(\ell_2)}| - |Y| + 1, k]). \end{aligned}$$

Due to Lemma 8, X_{ℓ_1} and X_{ℓ_2} can be found in $O(\log s)$ time. Since $X_{r(\ell_1)}$ is simple, $Occ^{\uparrow}(X_{r(\ell_1)}, Y, k)$ of cases (1) and (2) can be computed in $O(1)$ time by Lemma 5. According to Lemma 7, $Occ^{\uparrow}(X_{\ell(\ell_2)}, Y, |X_{\ell(\ell_2)}| - |Y| + 1)$ of case (3) can be computed in $O(1)$ time. By Observation 2, the union operations can be done in $O(1)$ time. Thus, in any case $A = Occ^{\uparrow}(X, Y, k)$ can be computed in $O(\log s)$ time. By Lemma 7 and Lemma 8, the extra work time and space are $O(ms)$. This completes the proof. □

Now we have got Lemma 5 and Lemma 6 proved. Using these lemmas, we can prove Lemma 4 as follows:

Proof. Let $X_i = X_\ell X_r$ and $Y_j = Y_\ell Y_r$. Then, as seen in Fig. 5, we have

$$\begin{aligned} Occ^{\triangle}(X_i, Y_j) &= (Occ^{\triangle}(X_i, Y_\ell) \cap (Occ(X_r, Y_r) \oplus |X_\ell| \ominus |Y_\ell|)) \\ &\cup (Occ(X_\ell, Y_\ell) \cap (Occ^{\triangle}(X_i, Y_r) \ominus |Y_\ell|)). \end{aligned}$$

Let $A = Occ^{\triangle}(X_i, Y_\ell) \cap (Occ(X_r, Y_r) \oplus |X_\ell| \ominus |Y_\ell|)$ and $B = Occ(X_\ell, Y_\ell) \cap (Occ^{\triangle}(X_i, Y_r) \ominus |Y_\ell|)$. Since $Occ^{\triangle}(X_i, Y_j)$ forms a single arithmetic progression by Lemma 1, the union operation of $A \cup B$ can be done in constant time. Therefore, the key is how to compute A and B efficiently.

Now we show how to compute set A. Let $z = |X_\ell| - |Y_\ell|$. Let $\langle p_1, d_1, t_1 \rangle = Occ^{\triangle}(X_i, Y_\ell)$ and $q_1 = p_1 + (t_1 - 1)d_1$. Depending on the value of t_1, we have the following cases:

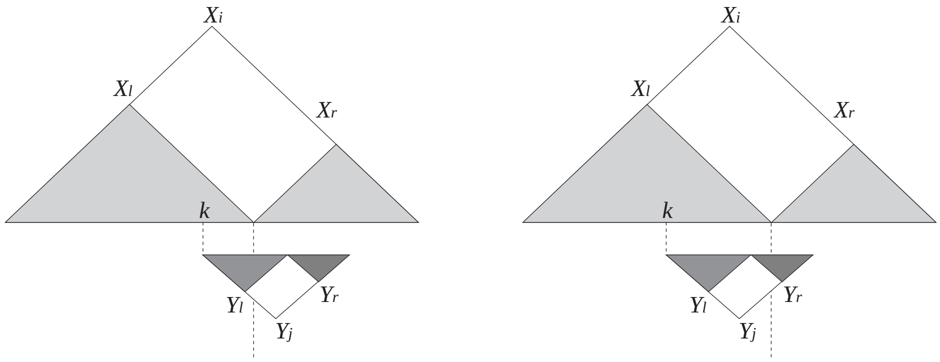

Fig. 5. $k \in Occ^{\triangle}(X, Y)$ if and only if either $k \in Occ^{\triangle}(X, Y_\ell)$ and $k + |Y_\ell| \in Occ(X, Y_r)$ (left case), or $k \in Occ(X, Y_\ell)$ and $k + |Y_\ell| \in Occ^{\triangle}(X, Y_r)$ (right case).

(1) when $t_1 = 0$.
In this case we have $A = \emptyset$.

(2) when $t_1 = 1$.
In this case, $Occ^{\triangle}(X_i, Y_\ell) = \{p_1\}$. It stands that

$$\begin{aligned}
A &= \{p_1\} \cap (Occ(X_r, Y_r) \oplus z) \\
&= (\{p_1 - z\} \cap Occ(X_r, Y_r)) \oplus z) \\
&= (\{p_1 - z\} \cap [p_1 - z - |Y_r|, p_1 - z] \cap Occ(X_r, Y_r)) \oplus z) \\
&= (\{p_1 - z\} \cap Occ^{\uparrow}(X_r, Y_r, p_1 - z)) \oplus z) \quad \text{(By Observation 1)} \\
&= \begin{cases} \{p_1\} & \text{if } p_1 - z \in Occ^{\uparrow}(X_r, Y_r, p_1 - z), \\ \emptyset & \text{otherwise.} \end{cases}
\end{aligned}$$

Since X_r is simple, $Occ^{\uparrow}(X_r, Y_r, p_1 - z)$ can be computed in constant time by Lemma 5. Checking whether $p_1 - z \in Occ^{\uparrow}(X_r, Y_r, p_1 - z)$ or not can be done in constant time since $Occ^{\uparrow}(X_r, Y_r, p_1 - z)$ forms a single arithmetic progression by Lemma 1.

(3) when $t_1 > 1$.

There are two sub-cases depending on the length of Y_r with respect to $q_1 - p_1 = (t_1 - 1)d_1 \geq d_1$, as follows.

- when $|Y_r| \geq q_1 - p_1$ (see the left of Fig. 6). By this assumption, we have $q_1 - |Y_r| \leq p_1$, which implies $[p_1, q_1] \subseteq [q_1 - |Y_r|, q_1]$. Thus

$$\begin{aligned}
A &= \langle p_1, d_1, t_1 \rangle \cap (Occ(X_r, Y_r) \oplus z) \\
&= (\langle p_1, d_1, t_1 \rangle \cap [p_1, q_1]) \cap (Occ(X_r, Y_r) \oplus z) \\
&= (\langle p_1, d_1, t_1 \rangle \cap [q_1 - |Y_r|, q_1]) \cap (Occ(X_r, Y_r) \oplus z) \\
&= \langle p_1, d_1, t_1 \rangle \cap ([q_1 - |Y_r|, q_1] \cap (Occ(X_r, Y_r) \oplus z))
\end{aligned}$$

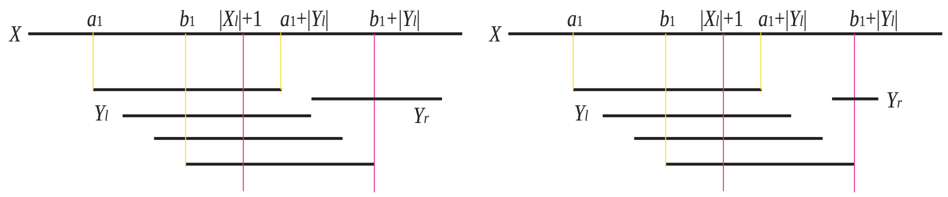

Fig. 6. Long case (left) and short case (right).

$$\begin{aligned} &= \langle p_1, d_1, t_1 \rangle \cap (([q_1 - |Y_r| - z, q_1 - z] \cap Occ(X_r, Y_r)) \oplus z) \\ &= \langle p_1, d_1, t_1 \rangle \cap (Occ^{\uparrow}(X_r, Y_r, q_1 - z) \oplus z), \end{aligned}$$

where the last equality is due to Observation 1. Since X_r is simple, due to Lemma 5, $Occ^{\uparrow}(X_r, Y_r, q_1 - z)$ can be computed in $O(1)$ time. By Lemma 3, $\langle p_1, d_1, t_1 \rangle \cap (Occ^{\uparrow}(X_r, Y_r, q_1 - z) \ominus |Y_\ell|)$ can be computed in constant time.

- when $|Y_r| < q_1 - p_1$ (see the right of Fig. 6). The basic idea is the same as the previous case, but computing $Occ^{\uparrow}(X_r, Y_r, q_1 - z)$ is not enough, since $|Y_r|$ is 'too short'. However, we can fill up the gap as follows.

$$\begin{aligned} A &= \langle p_1, d_1, t_1 \rangle \cap (Occ(X_r, Y_r) \oplus z) \\ &= (\langle p_1, d_1, t_1 \rangle \cap [p_1, q_1]) \cap (Occ(X_r, Y_r) \oplus z) \\ &= (\langle p_1, d_1, t_1 \rangle \cap ([p_1, q_1 - |Y_r| - 1] \cup [q_1 - |Y_r|, q_1])) \cap (Occ(X_r, Y_r) \oplus z) \\ &= \langle p_1, d_1, t_1 \rangle \cap (S \cup Occ^{\uparrow}(X_r, Y_r, q_1 - z)) \oplus z), \\ &\qquad \text{where } S = [p_1 - z, q_1 - z - |Y_r| - 1] \cap Occ(X_r, Y_r). \end{aligned}$$

By Lemma 2, d_1 is the shortest period of $X_i[p_1 : q_1 + |Y_\ell| - 1]$. For this string, we have

$$\begin{aligned} & X_i[p_1 : q_1 + |Y_\ell| - 1] \\ =\; & X_\ell[p_1 : |X_\ell|] X_r[1 : q_1 + |Y_\ell| - 1 - |X_\ell|] \\ =\; & X_\ell[p_1 : |X_\ell|] X_r[1 : q_1 - z - 1] \\ =\; & X_\ell[p_1 : |X_\ell|] X_r[1 : p_1 - z - 1] X_r[p_1 - z : q_1 - z - 1] \\ =\; & X_i[p_1 : p_1 + |Y_\ell| - 1] X_r[p_1 - z : q_1 - z - 1]. \end{aligned}$$

Therefore, $X_r[p_1 - z : q_1 - z - 1] = u^{t_1}$ where u is the suffix of Y_ℓ of length d_1. Thus,

$$S = \begin{cases} \langle p_1 - z, d_1, t' \rangle & \text{if } p_1 - z \in Occ(X_r, Y_r), \\ \emptyset & \text{otherwise}, \end{cases}$$

where t' is the maximum integer satisfying $p_1 - z + (t' - 1)d_1 \leq q_1 - z - |Y_r| - 1$. According to Observation 2, the union operation of $S \cup$

$Occ^{\uparrow}(Xr, Y_r, q_1 - z)$ can be done in constant time in both cases. By Observation 1, checking whether $p_1 - z \in Occ(X_r, Y_r)$ or not can be reduced to checking if $p_1 - z \in Occ^{\uparrow}(X_r, Y_r, p_1 - z)$. Since X_r is simple, it can be done in $O(1)$ time by Lemma 1 and Lemma 5. Finally, the intersection operation can be done in constant time by Lemma 3.

Therefore, in any case we can compute A in constant time.

Now we consider computing $B = Occ(X_\ell, Y_\ell) \cap (Occ^{\triangle}(X_i, Y_r) \ominus |Y_\ell|)$. Let $\langle p_2, d_2, t_2 \rangle = Occ^{\triangle}(X_i, Y_r)$. We have to consider how to compute $Occ^{\uparrow}(X_\ell, Y_\ell, p_2 - |Y_\ell|)$ efficiently. When X_ℓ is simple, we can use the same strategy as computing A. In case where X_ℓ is complex, $Occ^{\uparrow}(X_\ell, Y_\ell, p_2 - |Y_\ell|)$ can be computed in $O(\log s)$ time by Lemma 6.

Due to Lemma 5 and Lemma 6, the total extra work time and space are $O(h^2 + mh) + O(ms) = O(h^2 + m(h+s)) = O(h^2 + mn)$. This completes the proof. □

We have proven that each $Occ^{\triangle}(X, Y)$ can be computed in $O(\log s)$ time with extra $O(h^2 + mn)$ work time and space. Thus, the whole time complexity is $O(h^2 + mn) + O(mn \log s) = O(h^2 + mn \log s)$, and the whole space complexity is $O(h^2 + mn)$. This leads to the result of Theorem 1.

5. Conclusions

Miyazaki et al. [18] presented an algorithm to solve the FCPM problem for straight line programs in $O(m^2 n^2)$ time and with $O(mn)$ space. Since simple collage systems can be translated to straight line programs, their algorithm gives us an $O(m^2 n^2)$ time solution to the FCPM problem for simple collage systems. In this paper we developed an FCPM algorithm for simple collage systems which runs in $O(||\mathcal{D}||^2 + mn \log |\mathcal{S}|)$ time using $O(||\mathcal{D}||^2 + mn)$ space. Since $n = ||\mathcal{D}|| + |\mathcal{S}|$, the proposed algorithm is faster than the algorithm by Miyazaki et al. [18].

An interesting extension of this research is to consider the FCPM problem for *composition systems* [22]. Composition systems can be seen as collage systems without repetitions. Since it is known that LZ77 compression can be translated into a composition system of size $O(n \log n)$, an efficient FCPM algorithm for composition systems would lead to a better solution for the FCPM problem with LZ77 compression. We remark that the only known FCPM algorithm for LZ77 compression takes $O((n+m)^5)$ time [6], which is still very far from desired optimal time complexity.

References

1. A. Amir and G. Benson. "Efficient two-dimensional compressed matching," In *Proc. DCC'92*, page 279. IEEE Computer Society, 1992.
2. A. Amir, G. Benson, and M. Farach. "Let sleeping files lie: Pattern matching in Z-compressed files," *J. Computer and System Sciences*, 52(6):299–307, 1996.
3. T. Eilam-Tzoreff and U. Vishkin. "Matching patterns in strings subject to multi-linear transformations," *Theoretical Computer Science*, 60:231–254, 1988.
4. M. Farach and M. Thorup. "String matching in Lempel-Ziv compressed strings," *Algorithmica*, 20(4):388–404, 1998.

5. P. Gage. "A new algorithm for data compression," *The C Users Journal*, 12(2), 1994.

6. L. Gąsieniec, M. Karpinski, W. Plandowski, and W. Rytter. "Efficient algorithms for Lempel-Ziv encoding (extended abstract)," In *Proc. SWAT'96*, volume 1097 of *LNCS*, pages 392–403. Springer-Verlag, 1996.

7. L. Gąsieniec and W. Rytter. "Almost optimal fully LZW-compressed pattern matching," In *Proc. DCC'99*, pages 316–325. IEEE Computer Society, 1999.

8. S. Inenaga, A. Shinohara, and M. Takeda. "A fully compressed pattern matching algorithm for simple collage systems," In *Proc. PSC'04*, pages 98–113. Czech Technical University, 2004.

9. S. Inenaga, A. Shinohara, and M. Takeda. "An efficient pattern matching algorithm on a subclass of context free grammars," In *Proc. DLT'04*, volume 3340 of *LNCS*, pages 225–236. Springer-Verlag, 2004.

10. M. Karpinski, W. Rytter, and A. Shinohara. "An efficient pattern-matching algorithm for strings with short descriptions," *Nord. J. Comput.*, 4(2):172–186, 1997.

11. T. Kida, T. Matsumoto, Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa. "Collage system: a unifying framework for compressed pattern matching," *Theoretical Computer Science*, 298:253–272, 2003.

12. J. Kieffer and E. Yang. "Grammar-based codes: a new class of universal lossless source codes," *IEEE Trans. Inform. Theory*, 46(3):737–754, 2000.

13. J. Kieffer and E. Yang. "Grammar-based codes for universal lossless data compression," *Communications in Information and Systems*, 2(2):29–52, 2002.

14. J. Kieffer, E. Yang, G. Nelson, and P. Cosman. "Universal lossless compression via multilevel pattern matching," *IEEE Trans. Inform. Theory*, 46(4):1227–1245, 2000.

15. J. Larsson and A. Moffat. "Offline dictionary-based compression," In *Proc. DCC'99*, pages 296–305. IEEE Computer Society, 1999.

16. T. Matsumoto, T. Kida, M. Takeda, A. Shinohara, and S. Arikawa. "Bit-parallel approach to approximate string matching in compressed texts," In *Proc. SPIRE'00*, pages 221–228. IEEE Computer Society, 2000.

17. S. Mitarai, M. Hirao, T. Matsumoto, A. Shinohara, M. Takeda, and S. Arikawa. "Compressed pattern matching for SEQUITUR," In *Proc. DCC'01*, pages 469–480. IEEE Computer Society, 2001.

18. M. Miyazaki, A. Shinohara, and M. Takeda. "An improved pattern matching algorithm for strings in terms of straight line programs," *J. Discrete Algorithms*, 1(1):187–204, 2000.

19. C. Nevill-Manning and I. Witten. "Identifying hierarchical structure in sequences: a linear-time algorithm," *J. Artificial Intelligence Research*, 7:67–82, 1997.

20. J. A. Storer and T. G. Szymanski. "Data compression via textual substitution," *J. ACM*, 29(4):928–951, 1982.

21. T. Welch. "A technique for high performance data compression," *IEEE Comput. Magazine*, 17(6):8–19, 1984.

22. J. Ziv and A. Lempel. "A universal algorithm for sequential data compression," *IEEE Trans. Inform. Theory*, 23:337–343, 1977.

23. J. Ziv and A. Lempel. "Compression of individual sequences via variable length coding," *IEEE Trans. Inform. Theory*, 24:530–536, 1978.