A FULLY COMPRESSED PATTERN MATCHING ALGORITHM FOR SIMPLE COLLAGE SYSTEMS

SHUNSUKE INENAGA\textsuperscript{1}
Department of Informatics, Kyushu University 83, Fukuoka 812-8581, Japan
shunsuke.inenaga@i.kyushu-u.ac.jp

AYUMI SHINOHARA\textsuperscript{1}
Graduate School of Information Sciences, Tohoku University, Sendai 980-8579, Japan
ayumi@cei.tohoku.ac.jp

and

MASAYUKI TAKEDA
Department of Informatics, Kyushu University 83, 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 $T$ and $P$ which are descriptions of text $T$ and pattern $P$ respectively, find the occurrences of $P$ in $T$ without decompressing $T$ or $P$. 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 $(D,S)$, where $D$ is a dictionary and $S$ is a sequence of variables from $D$. Our FCPM algorithm performs in $O(\|D\|^2 + m\log|S|)$ time, where $n = |T|$ = $\|D\| + |S|$ and $m = |P|$. This is faster than the previous best result of $O(n^2 \log^2 n)$ time.

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

1. Introduction

\textsuperscript{1} Main part of this research was done when the author was visiting the Department of Computer Science, the University of Helsinki, Finland.
\textsuperscript{1} Supported by JSPS Research Fellowships for Young Scientists
\textsuperscript{1} 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 \( T \) and uncompressed pattern \( P \), find the pattern occurrences without decompressing \( T \). 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 \( \hat{T} \) and \( \hat{P} \) respectively, and the objective is to find all occurrences of \( P \) in \( T \) without decompressing \( T \) or \( P \). 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 \( m = |P| \) and \( n = |T| \). For LZW compressed text \( T \) and pattern \( P \), 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 \( T \) or \( P \) 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 transformation [12, 14, 13], and straight line programs. A collage system is a pair \((D, S)\) where \( D \) is a dictionary and \( S \) is a sequence of variables from \( D \). 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 \( O(|D|^2 + mn \log |S|) \) time with \( O(|D|^2 + mn) \) space, where \(|D|\) denotes the size of the dictionary \( D \), and \( |S| \) the length of the sequence \( S \). 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 different sources, this feature can be a practical advantage of our algorithm.

2. Preliminary

Let \( \mathbb{N} \) be the set of natural numbers, and \( \mathbb{N}^+ \) be the set of positive integers. Let \( \Sigma \) be a finite alphabet. An element of \( \Sigma^* \) 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

$$a \quad \text{where } a \in (\Sigma \cup \varepsilon), \quad \text{(primitive assignment)}$$

$$X_i X_j \quad \text{where } i, j < k, \quad \text{(concatenation)}$$

and $\mathcal{S}$ is a sequence of variables $X_{i_1}, X_{i_2}, \ldots, X_{i_q}$ 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 $X = X_{i_1}X_{i_2}$, either $|X| = 1$ or $|X'| = 1$ [16]. LZW [21] and LZ 78 [23] are simple collage systems formalized as follows.

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

$$X_1 = a_{i_1}; \quad X_2 = a_{i_2}; \quad \ldots; \quad X_q = a_{i_q};$$

$$X_{q+1} = X_{i_1}X_{\sigma(i_2)}; \quad X_{q+2} = X_{i_2}X_{\sigma(i_3)}; \quad \ldots; \quad X_{q+s-1} = X_{i_{q-1}}X_{\sigma(i_s)};$$

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 $X_j$.

**LZ 78.** $\mathcal{S} = X_1, X_2, \ldots, X_s$ and $\mathcal{D}$ is the following:

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

where $b_j$ is a symbol in $\Sigma$.

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:

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

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, $\text{Occ}(T, P)$ 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 \( \langle D, S \rangle \). Note that \( S = X_{i_1}, X_{i_2}, \ldots, X_{i_k} \) can be translated in linear time to a sequence of assignments of size \( s \). For instance, \( S = X_1, X_2, X_3, X_4 \) can be rewritten to \( X_5 = X_1X_2; X_6 = X_3X_3; X_7 = X_5X_4 \), 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 \textit{straight line programs (SLPs)}. In the sequel, for string \( T \in \Sigma^* \), let \( T \) denote the SLP representing \( T \). The size of \( T \) is denoted by \( |T| \), and \( |T| = |D| + |S| = h + s = n \).

Now we introduce \textit{simple} straight line programs (SSLP) that correspond to simple collage systems.

**Definition 1** An SSLP \( T \) is a sequence of assignments such that

\[
X_1 = \text{expr}_1; \ X_2 = \text{expr}_2; \ldots; \ X_n = \text{expr}_n,
\]

where \( X_i \) are variables and \( \text{expr}_i \) are expressions of any of the form

- \( a \) where \( a \in \Sigma \) (primitive),
- \( X_{i'}X' \) where \( i < i' \) and \( X' = a \) (right simple),
- \( X'X_r \) where \( i < i' \) and \( X' = a \) (left simple),
- \( X_{i'}X_r \) where \( i < i' \) (complex),

and \( T = X_n \). Moreover, each type of variable satisfies the following properties:

- For any right simple variable \( X_i = X_{i'}X' \), \( X_i \) is either simple or primitive.
- For any left simple variable \( X_i = X'X_r \), \( X_i \) is either simple or primitive.
- For any complex variable \( X_i = X_{i'}X_r \), \( X_i \) is either simple or primitive.

An example of an SSLP \( T \) for string \( T = abaabababbb \) is as follows:

\[
X_1 = a, \ X_2 = b, \ X_3 = X_1X_2, \ X_4 = X_1X_3, \ X_5 = X_1X_1, \ X_6 = X_2X_2, \ X_7 = X_3X_1, \ X_8 = X_7X_3, \ X_9 = X_8X_6,
\]

and \( T = X_9 \). See also Fig. 1 that illustrates the derivation tree of \( T \).

\( X_1 \) 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 \( \langle D, S \rangle \), let \( T \) be its corresponding SSLP. Let \( |D| = h \) and \( |S| = s \). Then the total number of primitive and simple variables in \( T \) is \( h \), and the number of complex variables in \( T \) is \( s \).

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

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

3.2. Basic idea of algorithm
In this section, we show a basis of our algorithm that outputs a compact representation of \( \text{Occ}(T, P) \) for given SLPs \( T, P \).

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

\[
\text{Occ}^k(X, Y, k) = \{ i \in \text{Occ}(X, Y) \mid k - |Y| \leq i \leq k \}.
\]

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 \( U \uplus k = \{ i + k \mid i \in U \} \) and \( U \ominus k = \{ i - k \mid i \in U \} \).

**Observation 1** For any strings \( X, Y \in \Sigma^* \) and integer \( k \in \mathbb{N} \),

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

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

For positive integers \( p, d \in \mathbb{N}^* \) and non-negative integer \( t \in \mathbb{N} \), 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, \( \text{Occ}^k(X, Y, k) \) can be represented as the triple \( \langle p, d, t \rangle \) with the minimal element \( p \), the common difference \( d \), and the length \( t \) of the progression. By ‘computing \( \text{Occ}^k(X, Y, k) \)’, we mean to calculate the triple \( \langle p, d, t \rangle \) such that \( \langle p, d, t \rangle = \text{Occ}^k(X, Y, k) \).

**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 \( \langle p, d, t \rangle = \text{Occ}^k(X, Y, k) \) for strings \( X, Y \in \Sigma^* \) and integer \( k \in \mathbb{N} \). 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 \( k_1, k_2 \in \mathbb{N} \), the intersection \( \text{Occ}^k(X, Y_1, k_1) \cap (\text{Occ}^k(X, Y_2, k_2) \cap |Y_2|) \) can be computed in \( O(1) \) time, provided that \( \text{Occ}^k(X, Y_1, k_1) \) and \( \text{Occ}^k(X, Y_2, k_2) \) are already computed.
For variables $X = X_1 X_r$ and $Y$, we denote $\text{Occ}^\Delta( X, Y ) = \text{Occ}^\Delta( X, Y, |X_d| + 1 )$.

The following observation is explained in Fig. 2.

**Observation 3 ([18])** For any variables $X = X_1 X_r$ and $Y$,\[ \text{Occ}( X, Y ) = \text{Occ}( X_1, Y ) \cup \text{Occ}^\Delta( X, Y ) \cup ( \text{Occ}( X_r, Y ) \oplus |X_d| ). \]

Observation 3 implies that $\text{Occ}( X_n, Y )$ can be represented by a combination of \[ \{ \text{Occ}^\Delta( X_i, Y ) \}_{i=1}^n = \text{Occ}^\Delta( X_1, Y ), \text{Occ}^\Delta( X_2, Y ), \ldots, \text{Occ}^\Delta( X_n, Y ). \]

Thus, the desired output $\text{Occ}( T, P, T ) = \text{Occ}( X_n, Y_m )$ can be expressed as a combination of $\{ \text{Occ}^\Delta( X_i, Y_m ) \}_{i=1}^n$ that requires $O(n)$ space. Hereby, computing $\text{Occ}( T, P, T )$ is reduced to computing $\text{Occ}^\Delta( X_i, Y_m )$ for every $i = 1, 2, \ldots, n$. In computing each $\text{Occ}^\Delta( X_i, Y_j )$ recursively, the same set $\text{Occ}^\Delta( X_i, Y_j )$ 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 $\text{App}$ where each entry $\text{App}[i, j]$ at row $i$ and column $j$ stores the triple for $\text{Occ}^\Delta( X_i, Y_j )$. We compute each $\text{App}[i, j]$ 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 $\text{App}$ 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 $T$ and $s$ is the number of complex variables in $T$. This leads to the result of Theorem 1.

4. Details of algorithm

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

**Lemma 4** For any variables $X_i$ of $T$ and $Y_j$ of $P$, $\text{Occ}^\Delta( X_i, Y_j )$ 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 $\text{Occ}^\Delta( X_i', Y_j', k )$ for any $1 \leq i' < i$ and $1 \leq j' < j$. In case that $X$ is simple, we have the following lemma:
Lemma 5 Let $X$ be any simple variable of $\mathcal{T}$ and $Y$ be any variable of $\mathcal{P}$. Given integer $k \in N$, $\text{Occ}^\Delta(X, Y, k)$ 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 $k \in N$, $\text{Occ}^\Delta(X, Y, k)$ can be computed in $O(\log s)$ time with extra $O(ms)$ work time and space.

For any complex variable $X = X_t X_r$, let $\text{range}(X)$ 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 $X = X_t X_r$ be any complex variable of $\mathcal{T}$ and let $Y$ be any variable of $\mathcal{P}$. Assume $\text{Occ}^\Delta(X_t, Y, |X_t| + |Y| + 1)$ and $\text{Occ}^\Delta(X, Y)$ are already computed. Then $\text{Occ}^\Delta(X, Y, |X| + |Y| + 1)$ can be computed in $O(1)$ time, with extra $O(ms)$ work space.

Lemma 8 Given integer $k \in N$, we can retrieve in $O(\log s)$ time the complex variable $X$ such that $\text{range}(X) = [r_1, r_2]$ 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 $A = \text{Occ}^\Delta(X, Y, k)$. Let $X_{t_1}$ be the complex variable such that $k \in \text{range}(X_{t_1})$, and let $X_{t_1} = X_{t_1} X_{r_{t_1}}$. Let $X_{t_2}$ be the complex variable satisfying $k - |Y| \in \text{range}(X_{t_2})$, and let $X_{t_2} = X_{t_2} X_{r_{t_2}}$. There are the three following cases:

1. when $k - |Y| \geq |X_{r_{t_1}}| + 1$ and $k + |Y| - 1 \leq |X_{t_1}|$ (Fig. 3, left).

In this case, we have $A = \text{Occ}^\Delta(X_{r_{t_1}}, Y, k) \oplus |X_{r_{t_1}}|$.

2. when $k - |Y| < |X_{r_{t_1}}| + 1$ and $k + |Y| - 1 \leq |X_{t_1}|$ (Fig. 3, right).

3. when $k - |Y| < |X_{r_{t_1}}| + 1$ and $k + |Y| - 1 > |X_{t_1}|$. (Fig. 3, right).
In this case, we have

\[ A = (\text{Occ}^{\Delta}(X_{\ell_1}, Y) \cap [k - |Y|, X_{\ell_1} + I]) \cup (\text{Occ}^{\delta}(X_{r(\ell_1)}, Y, k) \oplus [X_{\ell_1}]). \]

(3) when \( k + |Y| - 1 > |X_{\ell_1}| \) (Fig. 4).

In this case, we have

\[ A = \left( \text{Occ}^{\Delta}(X_{\ell_1}, Y) \cap [X_{\ell_1} - |Y| + I] \cap [k - |Y|, X_{\ell_1} - |Y| + I] \right) \cup (\text{Occ}^{\Delta}(X_{\ell_1}, Y) \cap [X_{\ell_1}]). \]

Due to Lemma 8, \( X_{\ell_1} \) and \( X_{\ell_2} \) can be found in \( O(\log s) \) time. Since \( X_{r(\ell_1)} \) is simple, \( \text{Occ}^{\delta}(X_{r(\ell_1)}, Y, k) \) of cases (1) and (2) can be computed in \( O(1) \) time by Lemma 5. According to Lemma 7, \( \text{Occ}^{\delta}(X_{\ell_1}, Y, X_{\ell_2} - |Y| + I) \) 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 \( A = \text{Occ}^{\delta}(X, Y, k) \) 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 \( X_i = X_{\ell_1}X_{r_1} \) and \( Y_j = Y_{\ell_1}Y_r \). Then, as seen in Fig. 5, we have

\[ \text{Occ}^{\Delta}(X_i, Y_j) = \left( \text{Occ}^{\Delta}(X_i, Y_j) \cap (\text{Occ}(X_r, Y_r) \oplus [X_i] \cap [Y_j]) \right) \cup (\text{Occ}(X_r, Y_r) \cap (\text{Occ}^{\Delta}(X_i, Y_j) \oplus [Y_j])). \]

Let \( A = \text{Occ}^{\Delta}(X_i, Y_j) \cap (\text{Occ}(X_r, Y_r) \oplus [X_i] \cap [Y_j]) \) and \( B = \text{Occ}(X_i, Y_j) \cap (\text{Occ}^{\Delta}(X_i, Y_j) \oplus [Y_j]). \) Since \( \text{Occ}^{\Delta}(X_i, Y_j) \) 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 \( z = |X_i| - |Y_j| \). Let \( \langle p_1, d_1, t_1 \rangle = \text{Occ}^{\Delta}(X_i, Y_j) \) and \( q_1 = p_1 + (t_1 - 1)d_1 \). Depending on the value of \( t_1 \), we have the following cases:

\[ \text{Occ}^{\Delta}(X_i, Y_j) = \left\{ \begin{array}{ll}
\end{array} \right. \]
Fig. 5. \( k \in \operatorname{Occ}^\Delta(X, Y) \) if and only if either \( k \in \operatorname{Occ}^\Delta(X, Y_r) \) and \( k + |Y_z| \in \operatorname{Occ}(X, Y_r) \) (left case), or \( k \in \operatorname{Occ}(X, Y_s) \) and \( k + |Y_z| \in \operatorname{Occ}^\Delta(X, Y_r) \) (right case).

(1) when \( t_1 = 0 \).

In this case we have \( A = \emptyset \).

(2) when \( t_1 = 1 \).

In this case, \( \operatorname{Occ}^\Delta(X_r, Y_r) = \{ p_1 \} \). It stands that

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

Since \( X_r \) is simple, \( \operatorname{Occ}^\Delta(X_r, Y_r, p_1 - z) \) can be computed in constant time by Lemma 5. Checking whether \( p_1 - z \in \operatorname{Occ}^\Delta(X_r, Y_r, p_1 - z) \) or not can be done in constant time since \( \operatorname{Occ}^\Delta(X_r, Y_r, p_1 - z) \) 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 \( q_1 - p_1 = (t_1 - 1)d_1 = d_1 \), 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

\[
A = \{ p_1 \} \cap \{ \operatorname{Occ}(X_r, Y_r) \oplus z \} \\
= ([p_1 - z] \cap \operatorname{Occ}(X_r, Y_r)) \oplus z \\
= ([p_1 - z] \cap [q_1 - |Y_r|, q_1] \cap \operatorname{Occ}(X_r, Y_r) \oplus z) \\
= \{ p_1 \} \cap ([q_1 - |Y_r|, q_1] \cap \operatorname{Occ}(X_r, Y_r) \oplus z) \\
= \emptyset
\]

9
Fig. 6. Long case (left) and short case (right).

\[
\begin{align*}
&= \langle p_1, d_1, t_1 \rangle \cap ([Y_r - z, q_1 - z] \cap \text{Occ}(X_r, Y_r)) \odot z) \\
&= \langle p_1, d_1, t_1 \rangle \cap (\text{Occ}^3(X_r, Y_r, q_1 - z) \odot z),
\end{align*}
\]

where the last equality is due to Observation 1. Since \(X_r\) is simple, due to Lemma 5, \(\text{Occ}^3(X_r, Y_r, q_1 - z)\) can be computed in \(O(1)\) time. By Lemma 3, \(\langle p_1, d_1, t_1 \rangle \cap (\text{Occ}^3(X_r, Y_r, q_1 - z) \odot [Y_d])\) 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 \(\text{Occ}^3(X_r, Y_r, q_1 - z)\) is not enough, since \([Y_r]\) is ‘too short’. However, we can fill up the gap as follows.

\[
A = \langle p_1, d_1, t_1 \rangle \cap \text{Occ}(X_r, Y_r) \odot z
\]

\[
= \langle p_1, d_1, t_1 \rangle \cap [p_1, q_1] \cap \text{Occ}(X_r, Y_r) \odot z
\]

\[
= \langle p_1, d_1, t_1 \rangle \cap ([p_1, q_1 - [Y_r] - 1] \cup [q_1 - [Y_r], q_1]) \cap \text{Occ}(X_r, Y_r) \odot z
\]

\[
= \langle p_1, d_1, t_1 \rangle \cap (S \cup \text{Occ}^3(X_r, Y_r, q_1 - z) \odot z),
\]

where \(S = [p_1 - z, q_1 - z - [Y_r] - l] \cap \text{Occ}(X_r, Y_r)\).

By Lemma 2, \(d_1\) is the shortest period of \(X_r[p_1 : q_1 + [Y_r] - 1]\). For this string, we have

\[
X_r[p_1 : q_1 + [Y_r] - 1] = X_r[p_1 : [X_r]_r[X_r[1 : q_1 + [Y_r] - 1] - [X_r]]
\]

\[
= X_r[p_1 : [X_r]_r[X_r[1 : q_1 - z - 1]
\]

\[
= X_r[p_1 : X_r[1 : p_1 - z - l] X_r[p_1 - z : q_1 - z - l]
\]

\[
= X_r[p_1 : p_1 + [Y_r] - l] X_r[p_1 - z : q_1 - z - l].
\]

Therefore, \(X_r[p_1 - z : q_1 - z - l] = u^l\) where \(u\) is the suffix of \(Y_r\) of length \(d_1\). Thus,

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

where \(t'\) is the maximum integer satisfying \(p_1 - z + (t' - l)d_1 \leq q_1 - z - [Y_r] - l\). According to Observation 2, the union operation of \(S \cup\)
$\text{Occ}_{\Delta}^{3}(X_r, Y_r, q_1 - z)$ can be done in constant time in both cases. By Observation 1, checking whether $p_1 - z \in \text{Occ}(X_r, Y_r)$ or not can be reduced to checking if $p_1 - z \in \text{Occ}_{\Delta}^{3}(X_r, Y_r, p_2 - z)$. 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 $B = \text{Occ}(X_t, Y_t) \cap (\text{Occ}_{\Delta}^{3}(X_t, Y_t) \cap |Y_t|)$. Let $(p_2, d_2, t_2) = \text{Occ}_{\Delta}^{3}(X_t, Y_t)$. We have to consider how to compute $\text{Occ}_{\Delta}^{3}(X_t, Y_t, p_2 - |Y_t|)$ efficiently. When $X_t$ is simple, we can use the same strategy as computing $A$. In case where $X_t$ is complex, $\text{Occ}_{\Delta}^{3}(X_t, Y_t, p_2 - |Y_t|)$ 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 $O(h^2 + m h) + O(m s) = O(h^2 + m h + s) = O(h^2 + mn)$. This completes the proof. $\blacksquare$

We have proven that each $\text{Occ}_{\Delta}^{3}(X, Y)$ can be computed in $O(\log s)$ time with extra $O(h^2 + mn)$ work time and space. Thus, the whole time complexity is $O(h^2 + mn + O(m n \log s) = O(h^2 + mn \log s)$, 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 colage 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 colage systems. In this paper we developed an FCPM algorithm for simple colage systems which runs in $O(||D||^2 + mn \log |S|)$ time using $O(||D||^2 + mn)$ space. Since $n = ||D|| + ||S||$, 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 colage 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