An Efficient Pattern Matching Algorithm on a Subclass of Context Free Grammars

Shunsuke Inenaga[1], Ayumi Shinohara[2,3], and Masayuki Takeda[2,4]

[1] Department of Computer Science, P.O. Box 68 (Gustaf Hällströmin katu 2b) FIN-00014, University of Helsinki, Finland
inenaga@cs.helsinki.fi

[2] Department of Informatics, Kyushu University 33, Fukuoka 812-8581, Japan
{ayumi, takeda}@i.kyushu-u.ac.jp

[3] PRESTO, Japan Science and Technology Agency (JST)

[4] SORST, Japan Science and Technology Agency (JST)

Abstract. There is a close relationship between formal language theory and data compression. Since 1990's various types of *grammar-based text compression* algorithms have been introduced. Given an input string, a grammar-based text compression algorithm constructs a context-free grammar that only generates the string. An interesting and challenging problem is pattern matching on context-free grammars $\mathcal{P}$ of size m and $\mathcal{T}$ of size n, which are the descriptions of pattern string P of length M and text string T of length N, respectively. The goal is to solve the problem in time proportional *only* to m and n, *not* to M nor N. Kieffer et al. introduced a very practical grammar-based compression method called *multilevel pattern matching code* (*MPM code*). In this paper, we propose an efficient pattern matching algorithm which, given two MPM grammars $\mathcal{P}$ and $\mathcal{T}$, performs in $O(mn^2)$ time with $O(mn)$ space. Our algorithm outperforms the previous best one by Miyazaki et al. which requires $O(m^2n^2)$ time and $O(mn)$ space.

1 Introduction

In 1990's formal language theory found text data compression to be a very promising application area; data compression is the discipline which aims to reduce space consumption of the data by removing its redundancy, and this is achievable by constructing a *context-free grammar* $\mathcal{G}$ which only generates the input text string w. Namely, the grammar $\mathcal{G}$ is such that its language $L(\mathcal{G})$ is $\{w\}$. Such a context-free grammar adroitly extracts, and succinctly represents, repeated segments of the input string, and thus gives a superbly compact representation of the string. According to this observation, many types of ingenious *grammar-based text compression* algorithms have been introduced so far. Examples of grammar-based text compressions are SEQUITUR [15, 17], Re-Pair [12], byte pair encoding (BPE) [5], grammar transform [9, 10], and straight-line programs (SLPs) [8].

C.S. Calude, E. Calude and M.J. Dinneen (Eds.): DLT 2004, LNCS 3340, pp. 225–236, 2004.
© Springer-Verlag Berlin Heidelberg 2004

As strings are the most basic type for data storage, *pattern matching* has been an omnipresent problem in Computer Science [3]. Due to rapid spread and increase of compressed data, we naturally face the pattern matching problem with compressed strings. Namely, we are here required to do pattern matching on two compressed strings (text and pattern) that are described in the form of a context-free grammar. This problem is also called the *fully compressed pattern matching problem* [18]. The problem is formalized as follows:

Input: context-free grammars $\mathcal{P}$ and $\mathcal{T}$ generating only pattern P and text T, respectively.
Output: all occurrences of P in T.

Let m and n be the sizes of the grammars $\mathcal{P}$ and $\mathcal{T}$ respectively, and M and N be the lengths of the strings P and T, respectively. What should be emphasized here is that the goal is to solve this problem in time proportional *only* to m and n, *not* to M nor N. Although there exist a number of $O(M+N)$-time algorithms that solve the pattern matching problem for uncompressed strings P and T [6, 4], none of them supplies us with a polynomial time solution to the compressed version of the problem since M (resp. N) can be *exponentially large* with respect to m (resp. n). Therefore, in order for us to develop a polynomial time solution, quite a limited amount of computational space is avaliable, and this makes the problem by far harder to solve.

The first polynomial-time solution to the problem was given by Karpinski et al. for straight-line programs (SLPs) [8]. SLPs are a grammar-based compression method which constructs a context-free grammar in the Chomsky normal form. They proposed an algorithm which runs in $O((m+n)^4 \log(m+n))$ time using $O((m+n)^3)$ space. Later on, Miyazaki et al. [13] gave an improved algorithm running in $O(m^2n^2)$ time using $O(mn)$ space.

Since computing a minimal SLP that generates a given string is known to be NP-complete, it is of great significance to develop approximative algorithms for generating small grammars [19, 2]. One of those algorithms is the *multilevel pattern matching code* (*MPM code*) introduced by Kieffer et al. [11]. MPM code is attractive in that it performs in linear time with respect to the input string size, and is capable of exponential compression - the generated grammar size can be exponentially small with respect to the input string size. It is also noteworthy that MPM grammars have a hierarchical structure, which suggests that MPM code has a potential for recognizing lexical and grammatical structures in strings similarly to SEQUITUR [14, 16].

In this paper, we consider the pattern matching problem on MPM grammars. Although the algorithm by Miyazaki et al. [13] for general SLPs requires $O(m^2n^2)$ time and $O(mn)$ space, our algorithm specialized for MPM grammars performs in $O(mn^2)$ time within $O(mn)$ space.

2 Preliminaries

Let $\mathcal{N}$ be the set of natural numbers, and $\mathcal{N}^+$ be positive integers. Let Σ be a finite *alphabet*. An element of Σ^* is called a *string*. The length of a string T is

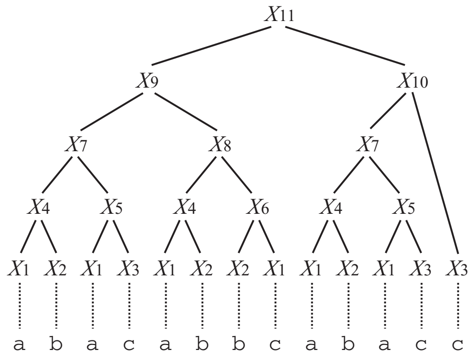

Fig. 1. Derivation tree of the MPM for string `abacabbcabacc`

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 $i = 1, 2, \ldots, |T| - p$.

Let X be any variable of a context-free grammar. We define the *length* of X to be the length of the string X produces, and denote it by $|X|$.

A *multilevel pattern matching grammar* (*MPM grammar*) $\mathcal{T}$ 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 the form either:

- $expr_i = a$ $(a \in \Sigma)$, or
- $expr_i = X_\ell X_r$ $(\ell, r < i)$ where $|X_\ell| \geq |X_r|$ and $|X_\ell|$ is a power of 2,

and $\mathcal{T} = X_n$. MPM grammar $\mathcal{T}$ is a context-free grammar in the Chomsky normal form such that its language $L(\mathcal{T})$ is $\{T\}$. The size of $\mathcal{T}$ is n and is denoted by $\|\mathcal{T}\|$. For example, MPM grammar $\mathcal{T}$ for $T =$ `abacabbcabacc` is:

$$X_1 = \mathtt{a}, X_2 = \mathtt{b}, X_3 = \mathtt{c}, X_4 = X_1X_2, X_5 = X_1X_3, X_6 = X_2X_3, X_7 = X_4X_5,$$
$$X_8 = X_4X_6, X_9 = X_7X_8, X_{10} = X_7X_3, X_{11} = X_9X_{10},$$

and $\mathcal{T} = X_{11}$. Note $\|\mathcal{T}\| = 11$. Fig. 1 illustrates the derivation tree of $\mathcal{T}$.

The *height* of variable X, denoted by $height(X)$, is defined as follows:

$$height(X) = \begin{cases} 1 & \text{if } X = a \ (a \in \Sigma), \\ \max(height(X_\ell), height(X_r)) + 1 & \text{if } X = X_\ell X_r. \end{cases}$$

That is, $height(X)$ is the length of the longest path from X to a leaf. In the running example, $height(X_{10}) = 4$, $height(X_{11}) = height(\mathcal{T}) = 5$, and so on (see Fig. 1). It is easy to see $height(\mathcal{T}) \leq n$.

The *pattern matching problem for strings in terms of MPM grammars* is, given two MPM grammars $\mathcal{T}$ and $\mathcal{P}$ that are the descriptions of text T and pattern P, to find all occurrences of P in T. Namely, we compute the following set:

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

In the sequel, we use X and X_i for variables of $\mathcal{T}$, and Y and Y_j for variables of $\mathcal{P}$. Let $\|\mathcal{T}\| = n$ and $\|\mathcal{P}\| = m$.

3 Overview of Algorithm

In this section, we show an overview of our algorithm that outputs a compact representation of $Occ(T, P)$ for given MPM grammars $\mathcal{T}$ and $\mathcal{P}$.

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.

Observation 1 ([7]). *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 ([7]). *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 $a, d, t \in \mathcal{N}^+$, we define $\langle a, d, t\rangle = \{a+(i-1)d \mid i \in [1, t]\}$. Assume that for $t = 0$, $\langle a, d, t\rangle = \emptyset$. Note that t denotes the cardinality of the set $\langle a, d, t\rangle$. By Lemma 1, $Occ^{\uparrow}(X, Y, k)$ can be represented as the triple $\langle a, d, t\rangle$ with the minimum element a, the common difference d, and the length t of the progression. By *'computing $Occ^{\uparrow}(X, Y, k)$'*, we mean to calculate the triple $\langle a, d, t\rangle$ such that $\langle a, d, t\rangle = Occ^{\uparrow}(X, Y, k)$.

For a set U of integers and an integer k, we denote $U \oplus k = \{i + k \mid i \in U\}$ and $U \ominus k = \{i - k \mid i \in U\}$. For MPM variables $X = X_\ell X_r$ and Y, we denote $Occ^{\triangle}(X, Y) = Occ^{\uparrow}(X, Y, |X_\ell| + 1)$.

Lemma 2 ([13]). *For any MPM variables $X = X_\ell X_r$ and Y,*

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

(See Fig. 2.)

Lemma 2 implies that $Occ(X_n, Y)$ can be represented by a combination of

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

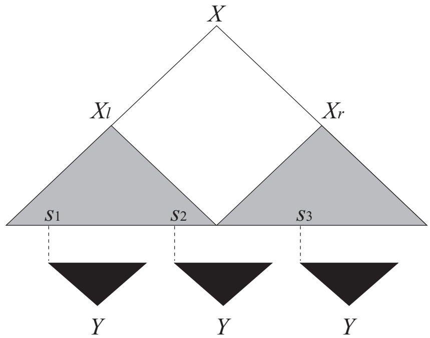

Fig. 2. $s_1, s_2, s_3 \in Occ(X, Y)$, where $s_1 \in Occ(X_\ell, Y)$, $s_2 \in Occ^{\triangle}(X, Y)$ and $s_3 \in Occ(X_r, Y)$

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 Section 4, we will show each $App[i, j]$ is computable in $O(height(X_i))$ time. Since $height(X_i) \leq n$, we can construct the whole table App in $O(mn^2)$ time. The size of the whole table is $O(mn)$, since each triple occupies $O(1)$ space. We therefore have the main result of the paper, as follows:

Theorem 1. *Given two MPM grammars $\mathcal{T}$ and $\mathcal{P}$, $Occ(T, P)$ can be computed in $O(mn^2)$ time with $O(mn)$ space.*

4 Details of Algorithm

In this section, we show that $Occ^{\triangle}(X_i, Y_j)$ is computable in $O(height(X_i))$ time for each variable X_i in $\mathcal{T}$ and Y_j in $\mathcal{P}$.

The following two lemmas and one observation are necessary to prove Lemma 5 which is one of the key lemmas for our algorithm.

Lemma 3 ([7]). *For strings $X, Y \in \Sigma^*$ and integer $k \in \mathcal{N}$, let $\langle a, d, t\rangle = Occ^{\uparrow}(X, Y, k)$. If $t \geq 1$, then d is the shortest period of $X[s : b + |Y| - 1]$ for any $s \in \langle a, d, t-1\rangle$ and $b = a + (t-1)d$.*

Proof. First we see that d is a period of $X[a : b + |Y| - 1]$ as follows. Since $\langle a, d, t\rangle = Occ^{\uparrow}(X, Y, k)$, we know

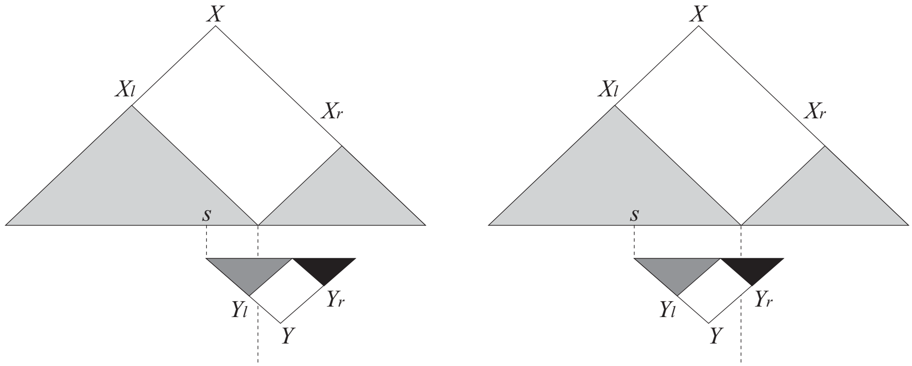

Fig. 3. $s \in Occ^{\triangle}(X,Y)$ if and only if either $s \in Occ^{\triangle}(X,Y_\ell)$ and $s+|Y_\ell| \in Occ(X,Y_r)$ (left case), or $s \in Occ(X,Y_\ell)$ and $s+|Y_\ell| \in Occ^{\triangle}(X,Y_r)$ (right case)

$$\begin{aligned} Y &= X[a : a+|Y|-1], \\ Y &= X[a+d : a+d+|Y|-1], \\ &\vdots \\ Y &= X[b : b+|Y|-1]. \end{aligned}$$

By these equations, we have

$$X[i] = X[i+d] \text{ for all } i \in [a, b+|Y|-1-d],$$

which shows that d is a period of $X[s : b+|Y|-1]$ for any $s \in \langle a,d,t-1\rangle$.

We now suppose that $X[s : b+|Y|-1]$ has a smaller period $d' < d$ for the contrary. That is, $X[i] = X[i+d']$ for all $i \in [s, b+|Y|-1-d']$. Then we have $Y[i] = X[s+i-1] = X[s+d'+i-1]$ for all $i \in [1,|Y|]$. Since $b-s \geq b-(a+(t-2)\cdot d) = b-(b-d) = d > d'$, we have $s+d' \in Occ^{\uparrow}(X,Y,k)$. However, this contradicts with $\langle a,d,t\rangle = Occ^{\uparrow}(X,Y,k)$, since $s+d' \notin \langle a,d,t\rangle$. Thus d is the shortest period of $X[s : b+|Y|-1]$ for any $s \in \langle a,d,t-1\rangle$. $\square$

Observation 2 ([13]). *For any MPM variables X, $Y = Y_\ell Y_r$, and integer $k \in \mathcal{N}$,*

$$\begin{aligned} Occ^{\triangle}(X,Y) = &\left(Occ^{\triangle}(X,Y_\ell) \cap (Occ(X,Y_r) \ominus |Y_\ell|)\right) \\ &\cup \left(Occ(X,Y_\ell) \cap (Occ^{\triangle}(X,Y_r) \ominus |Y_\ell|)\right). \end{aligned}$$

(See Fig. 3.)

Lemma 4 ([7]). *For any strings $X, Y_1, Y_2 \in \Sigma^*$ and integers $k_1, k_2 \in \mathcal{N}$, $Occ^{\uparrow}(X,Y_1,k_1) \cap (Occ^{\uparrow}(X,Y_2,k_2) \ominus |Y_1|)$ 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.*

For strings $X, Y \in \Sigma^*$ we consider the two following queries:

Single-Match Query: Given integer $s \in \mathcal{N}$, return if $s \in Occ(X,Y)$ or not.
Covering-Match Query: Given integer $k \in \mathcal{N}$, return triple $\langle a,d,t\rangle$ which represents $Occ^{\uparrow}(X,Y,k)$.

Lemma 5. *For any MPM variables X and $Y = Y_\ell Y_r$ and integer $k \in \mathcal{N}$, computing $Occ^{\triangle}(X,Y)$ is reducible in constant time to the following queries:*

(1) covering-match query $Occ^{\uparrow}(X, Y_\ell, |X_\ell|+1) = Occ^{\triangle}(X, Y_\ell)$,
(2) covering-match query $Occ^{\uparrow}(X, Y_r, |X_\ell|+1) = Occ^{\triangle}(X, Y_r)$,
(3) at most two covering-match queries $Occ^{\uparrow}(X, Y', k_1)$ and $Occ^{\uparrow}(X, Y', k_2)$ for some integers k_1, k_2, where Y' is either Y_ℓ or Y_r, and
(4) at most two single-match queries $s_1, s_2 \in Occ(X, Y')$ for some integers s_1, s_2, where Y' is either Y_ℓ or Y_r.

Proof. We perform two covering-match queries $Occ^{\triangle}(X, Y_\ell)$ and $Occ^{\triangle}(X, Y_r)$, and let $\langle a_1, d_1, t_1\rangle$ and $\langle a_2, d_2, t_2\rangle$ be answers of them, respectively. Depending on the cardinalities of triples, we have the four following cases:

(a) when $t_1 \le 1$ and $t_2 \le 1$.
At most two single-match queries are necessary for the following reasons. If $t_1 = 0$, we know $Occ^{\triangle}(X, Y_\ell) = \emptyset$. If $t_1 = 1$, we perform a single-match query $a_1 + |Y_\ell| \in Occ(X, Y_r)$, and we have

$$\begin{aligned} Occ^{\triangle}(X, Y_\ell) \cap (Occ(X, Y_r) \ominus |Y_\ell|) &= \{a_1\} \cap (Occ(X, Y_r) \ominus |Y_\ell|) \\ &= \begin{cases} \{a_1\} & \text{if } a_1 + |Y_\ell| \in Occ(X, Y_r), \\ \emptyset & \text{otherwise.} \end{cases} \end{aligned}$$

Similarly, if $t_2 = 0$ we know $Occ^{\triangle}(X, Y_r) = \emptyset$. If $t_2 = 1$, we have

$$\begin{aligned} Occ(X, Y_\ell) \cap (Occ^{\triangle}(X, Y_r) \ominus |Y_\ell|) &= Occ(X, Y_\ell) \cap (\{a_2\} \ominus |Y_\ell|) \\ &= \begin{cases} \{a_2 - |Y_\ell|\} & \text{if } a_2 - |Y_\ell| \in Occ(X, Y_\ell), \\ \emptyset & \text{otherwise.} \end{cases} \end{aligned}$$

By Observation 2, $Occ^{\triangle}(X,Y)$ is a union of these two sets. Trivially, the union operation can be done in constant time since each of these two sets is either singleton or empty.

(b) when $t_1 \ge 2$ and $t_2 \le 1$.
First we compute $A = Occ^{\triangle}(X, Y_\ell) \cap (Occ(X, Y_r) \ominus |Y_\ell|) = \langle a_1, d_1, t_1\rangle \cap (Occ(X, Y_r) \ominus |Y_\ell|)$, by using one covering-match query and at most one single-match query. Let $b_1 = a_1 + (t_1 - 1)d_1$. We consider two sub-cases depending on the length of Y_r with respect to $b_1 - a_1 = (t_1 - 1)d_1 \ge d_1$, as follows.

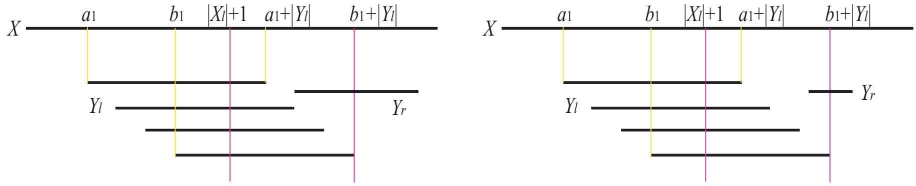

Fig. 4. Long case (left) and short case (right)

– the case $|Y_r| \geq b_1 - a_1$ (see the left of Fig. 4). By this assumption, we have $b_1 - |Y_r| \leq a_1$, which implies $[a_1, b_1] \subseteq [b_1 - |Y_r|, b_1]$. Thus

$$\begin{aligned} A &= \langle a_1, d_1, t_1 \rangle \cap (Occ(X, Y_r) \ominus |Y_\ell|) \\ &= (\langle a_1, d_1, t_1 \rangle \cap [a_1, b_1]) \cap (Occ(X, Y_r) \ominus |Y_\ell|) \\ &= (\langle a_1, d_1, t_1 \rangle \cap [b_1 - |Y_r|, b_1]) \cap (Occ(X, Y_r) \ominus |Y_\ell|) \\ &= \langle a_1, d_1, t_1 \rangle \cap ([b_1 - |Y_r|, b_1] \cap (Occ(X, Y_r) \ominus |Y_\ell|)) \\ &= \langle a_1, d_1, t_1 \rangle \cap (([b_1 - |Y_r| + |Y_\ell|, b_1 + |Y_\ell|] \cap Occ(X, Y_r)) \ominus |Y_\ell|) \\ &= \langle a_1, d_1, t_1 \rangle \cap (Occ^{\uparrow}(X, Y_r, b_1 + |Y_\ell|) \ominus |Y_\ell|), \end{aligned}$$

where the last equality is due to Observation 1. Here, we perform covering-match query $Occ^{\uparrow}(X, Y_r, b_1 + |Y_\ell|)$. According to Lemma 4, $\langle a_1, d_1, t_1 \rangle \cap (Occ^{\uparrow}(X, Y_r, b_1 + |Y_\ell|) \ominus |Y_\ell|)$ can be computed in constant time.

– the case $|Y_r| < b_1 - a_1$ (see the right of Fig. 4). The basic idea is the same as in the previous case, but covering-match query $Occ^{\uparrow}(X, Y_r, b_1 + |Y_\ell|)$ is not enough, since $|Y_r|$ is 'too short'. However, additional single-match query $a_1 + |Y_\ell| \in Occ(X, Y_r)$ fills up the gap, as follows.

$$\begin{aligned} A &= \langle a_1, d_1, t_1 \rangle \cap (Occ(X, Y_r) \ominus |Y_\ell|) \\ &= (\langle a_1, d_1, t_1 \rangle \cap [a_1, b_1]) \cap (Occ(X, Y_r) \ominus |Y_\ell|) \\ &= (\langle a_1, d_1, t_1 \rangle \cap ([a_1, b_1 - |Y_r| - 1] \cup [b_1 - |Y_r|, b_1])) \cap (Occ(X, Y_r) \ominus |Y_\ell|) \\ &= \langle a_1, d_1, t_1 \rangle \cap (S \cup Occ^{\uparrow}(X, Y_r, b_1 + |Y_\ell|)) \ominus |Y_\ell|), \\ &\qquad \text{where } S = [a_1 + |Y_\ell|, b_1 + |Y_\ell| - |Y_r| - 1] \cap Occ(X, Y_r). \end{aligned}$$

By Lemma 3, d_1 is the shortest period of $X[a_1 : b_1 + |Y| - 1]$. Therefore, we have $X[a_1 + |Y_\ell| : b_1 + |Y_\ell| - 1] = u^{t_1}$ where u is the suffix of Y_ℓ of length d_1. Thus, if $a_1 + |Y_\ell| \in Occ(X, Y_r)$, $S = \langle a_1 + |Y_\ell|, d_1, t' \rangle$, where t' is the maximum integer satisfying $a_1 + |Y_\ell| + (t' - 1)d_1 \leq b_1 + |Y_\ell| - |Y_r| - 1$. Since $Occ^{\uparrow}(X, Y_r, b_1 + |Y_\ell|)$ forms a single arithmetic progression by Lemma 1, the union operation can be done in constant time. Otherwise (if $a_1 + |Y_\ell| \notin Occ(X, Y_r)$), we have $S = \emptyset$ for the same reason, and thus the union operation can be done in constant time.

We now consider set $B = Occ(X, Y_\ell) \cap (Occ^{\triangle}(X, Y_r) \ominus |Y_\ell|)$. Since $t_2 \leq 1$, $Occ^{\triangle}(X, Y_r)$ is either singleton or empty. If it is empty, $B = \emptyset$. If it is singleton $\{a_2\}$, we just perform single-match query $a_2 - |Y_\ell| \in Occ(X, Y_\ell)$. If the answer is 'yes', then $B = \{a_2 - |Y_\ell|\}$, and otherwise $B = \emptyset$.

The union operation for $Occ^{\triangle}(X,Y) = A \cup B$ can be done in constant time since B is at most singleton.

In total, a covering-match query and at most two single-match queries are enough to compute $Occ^{\triangle}(X,Y)$ in this case.

(c) when $t_1 \leq 1$ and $t_2 \geq 2$.
Symmetric to Case (b).

(d) when $t_1 \geq 2$ and $t_2 \geq 2$.
We can compute $A = Occ^{\triangle}(X, Y_\ell) \cap (Occ(X, Y_r) \ominus |Y_\ell|)$ in the same way as Case (b), since the proof for Case (b) does not depend on the cardinality of $Occ(X, Y_r)$. Also, computing $B = Occ(X, Y_\ell) \cap (Occ^{\triangle}(X, Y_r) \ominus |Y_\ell|)$ is symmetric to computing A. Recall that each of A and B is an intersection of two sets both form a single arithmetic progression. This implies that A and B also form a single arithmetic progression (it can be proven in a similar manner to Lemma 4). Hence the union operation for $Occ^{\triangle}(X,Y) = A \cup B$ can be done in constant time. Thus, two covering-match queries and at most two single-match queries are enough in this case.

□

The time complexity of a single-match query is the following:

Lemma 6 ([13]). *For any MPM variables X, Y and integer $s \in \mathcal{N}$, single-match query $s \in Occ(X,Y)$ can be done in $O(height(X))$ time.*

Now the only remaining thing is how to efficiently perform covering-match query $Occ^{\uparrow}(X,Y,k)$. We will show it in Lemma 7.

For any MPM variable $X = X_\ell X_r$, we recursively define the *leftmost descendant* $lmd(X,h)$ and the *rightmost descendant* $rmd(X,h)$ of X with respect to height h ($\leq height(X)$), as follows:

$$lmd(X,h) = \begin{cases} lmd(X_\ell, h) & \text{if } height(X) > h, \\ X & \text{if } height(X) = h, \end{cases}$$

$$rmd(X,h) = \begin{cases} rmd(X_r, h) & \text{if } height(X) > h, \\ X & \text{if } height(X) = h. \end{cases}$$

In the example of Fig. 1, $lmd(X_{10}, 3) = X_7$, $rmd(X_9, 2) = X_6$, $rmd(X_7, 1) = X_3$, and so on. For variable X_i ($1 \leq i \leq n$) and height h ($< height(Y)$), we precompute two tables storing $lmd(X_i, h)$ and $rmd(X_i, h)$ respectively. By using these tables, we can refer to any $lmd(X_i, h)$ and $rmd(X_i, h)$ in constant time. These tables can be constructed in $O(mn)$ time in a bottom-up manner.

Lemma 7. *For any MPM variables X, Y and integer $k \in \mathcal{N}$, covering-match query $Occ^{\uparrow}(X,Y,k)$ is reducible in $O(height(X))$ time to at most three covering-match queries $Occ^{\triangle}(L,Y)$, $Occ^{\triangle}(C,Y)$, and $Occ^{\triangle}(R,Y)$ where L, C, R are a descendant of X or X itself.*

Proof. Let $X = X_\ell X_r$ and $Y = Y_\ell Y_r$. If $k = |X_\ell| + 1$, then only one covering-match query $Occ^{\triangle}(X,Y)$ is enough. Now we assume $k \neq |X_\ell| + 1$.

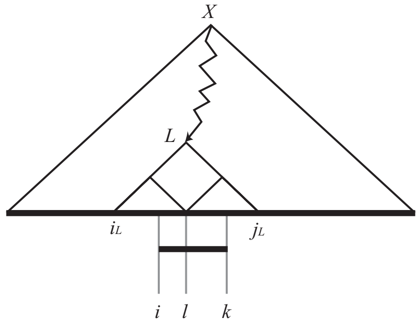

Fig. 5. Given integer k, the left (right, resp.) illustrates how to find L (R, resp.)

Let $i = \max(k-|Y|, 1)$ and $j = \min(k+|Y|-1, |X|)$. We consider the possibly shortest descendant L of X which covers the range $[i, k]$. (see the left of Fig. 5.) Let i_L, j_L be the integers such that $X[i_L : j_L] = L$. Let $l = i_L+|L_\ell|$. Similarly, we consider the possibly shortest descendant R of X which covers the range $[k, j]$. (see the right of Fig. 5.) Let i_R, j_R be the integers such that $X[i_R : j_R] = R$. Let $r = i_R + |R_\ell|$.

Assume $l = r$, that is, $L = R$. In this case only one covering-match query $Occ^{\triangle}(L, Y)$ is enough, since $k = l = i_L + |L_\ell|$ and thus

$$\begin{aligned} Occ^{\uparrow}(X, Y, k) &= Occ^{\uparrow}(L, Y, |L_\ell| + 1) \oplus (i_L - 1) \\ &= Occ^{\triangle}(L, Y) \oplus (i_L - 1). \end{aligned}$$

In case $l < r$, we have the following sub-cases.

(1) when L is a descendant of R.
Depending on the shapes of $R = R_\ell R_r$ and $Y = Y_\ell Y_r$, we have the four following sub-cases:
  (a) when $|R_\ell| = |R_r|$ and $|Y_\ell| = |Y_r|$. In this case, $L = rmd(R_\ell, height(Y) + 1)$. Then,

$$\begin{aligned} Occ^{\uparrow}(X, Y, k) = ((&Occ^{\triangle}(L, Y) \cap [k-|Y|-i_L+1 : k-i_L+1]) \oplus (i_L-1)) \\ &\cup ((Occ^{\triangle}(R, Y) \cap [k-|Y|-i_R+1 : k-i_R+1]) \oplus (i_R-1)). \end{aligned}$$

  Since $Occ^{\triangle}(L, Y)$ and $Occ^{\triangle}(R, Y)$ form a single arithmetic progression by Lemma 1, the intersection and union operations take $O(1)$ time.
  (b) when $|R_\ell| > |R_r|$ and $|Y_\ell| = |Y_r|$. Since $|R_\ell|$ and $|Y|$ are a power of 2, we have $L = rmd(R_\ell, height(Y) + 1)$. Thus we have the same equation as in Case (1)-(a).
  (c) when $|R_\ell| = |R_r|$ and $|Y_\ell| > |Y_r|$. We have the two following sub-cases:
    (i) when $r-k+|Y| \leq 2\times|Y_\ell|$. In this case, $L = rmd(R_\ell, height(Y_\ell)+1)$. Thus we have the same equation as in Case (1)-(a).

(ii) when $r-k+|Y| > 2\times|Y_\ell|$. In this case, $L = rmd(R_\ell, height(Y_\ell)+2)$. Let $C = L_r$. Then,

$$
\begin{aligned}
&Occ^{\uparrow}(X,Y,k)\\
&= ((Occ^{\triangle}(L,Y) \cap [k-|Y|-i_L+1 : k-i_L+1]) \oplus (i_L-1))\\
&\quad \cup ((Occ^{\triangle}(C,Y) \cap [k-|Y|-p+1 : k-p+1]) \oplus (p-1))\\
&\quad \cup ((Occ^{\triangle}(R,Y) \cap [k-|Y|-i_R+1 : k-i_R+1]) \oplus (i_R-1)),
\end{aligned}
$$

where $p = i_L + |L_\ell|$. By Lemma 1, the intersection and union operations can be done in $O(1)$ time.

(d) when $|R_\ell| > |R_r|$ and $|Y_\ell| > |Y_r|$. Since $|R_\ell|$ is a power of 2, we can use the same equations as in Case (1)-(c).

(2) when L is an ancestor of R.
Depending on the shapes of $L = L_\ell L_r$ and $Y = Y_\ell Y_r$, we have the four following sub-cases:

(a) when $|L_\ell| = |L_r|$ and $|Y_\ell| = |Y_r|$. This is symmetric to Case (1)-(a).
(b) when $|L_\ell| > |L_r|$ and $|Y_\ell| = |Y_r|$. Let $L_r = L_{\ell(r)}L_{r(r)}$. Since $|L_{\ell(r)}|$ is a power of 2, we can use the same strategy as in Case (2)-(a).
(c) when $|L_\ell| = |L_r|$ and $|Y_\ell| > |Y_r|$. This is a symmetric to Case (1)-(c).
(d) when $|L_\ell| > |X_r|$ and $|Y_\ell| > |Y_r|$. Let $L_r = L_{\ell(r)}L_{r(r)}$. Since $|L_{\ell(r)}|$ is a power of 2, we can use the same strategy as in Case (2)-(c).

Since each of R, L is a descendant of X or X itself, we can find them in $O(height(X))$ time by a top-down traversal on X. Moreover, C can be found in constant time from L or R.

□

By Lemmas 5, 6 and 7, we conclude that each entry $App[i,j]$ representing $Occ^{\triangle}(X_i, Y_j)$ can be computed in $O(height(X_i))$ time. Since $height(X_i) \leq n$, given two MPM grammars $\mathcal{T}$ and $\mathcal{P}$, we can compute $Occ(T,P)$ in $O(mn^2)$ time.

5 Conclusions and Further Discussions

This paper considered the pattern matching problem on a subclass of context-free grammars called *multilevel pattern matching grammars* (*MPM grammars*). MPM code was developed by Kieffer et al. [11] for efficient grammar-based text compression. Since MPM grammar sizes can be exponentially small with respect to the original string sizes, it is a rather hard task to solve the pattern matching problem in time proportional only to the grammar sizes. In this paper, we developed an efficient pattern matching algorithm which, given two MPM grammars $\mathcal{P}$ and $\mathcal{T}$, runs in $O(mn^2)$ time with $O(mn)$ space, where $m = \|\mathcal{P}\|$ and $n = \|\mathcal{T}\|$. Our algorithm outperforms the previous best algorithm of [13] running in $O(m^2n^2)$ time using $O(mn)$ space. An interesting open problem is whether an $O(mn)$-time solution is achievable or not.

As a final remark we mention that MPM grammars can be seen as text compression by *ordered binary decision diagrams* (*OBDDs*) [1]. OBDDs were originally developed to represent a Boolean function as a directed acyclic graph. OBDDs are also used for *symbolic* or *implicit* graph algorithms [20]. MPM code turns out to reveal yet another application of OBDDs to text compression.

References

1. R. E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24:293–318, 1992.
2. M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Rasala, A. Sahai, and A. Shelat. Approximating the smallest grammar: Kolmogorov complexity in natural models. In *Proc. STOC'02*, pages 792–801, 2002.
3. M. Crochemore and W. Rytter. *Text Algorithms.* Oxford University Press, New York, 1994.
4. M. Crochemore and W. Rytter. *Jewels of Stringology.* World Scientific, 2002.
5. P. Gage. A new algorithm for data compression. *The C Users Journal*, 12(2), 1994.
6. D. Gusfield. *Algorithms on Strings, Trees, and Sequences.* Cambridge University Press, New York, 1997.
7. 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.
8. M. Karpinski, W. Rytter, and A. Shinohara. An efficient pattern-matching algorithm for strings with short descriptions. *Nordic J. Comput.*, 4(2):172–186, 1997.
9. J. Kieffer and E. Yang. Grammar-based codes: a new class of universal lossless source codes. *IEEE Transactions on Information Theory*, 46(3):737–754, 2000.
10. J. Kieffer and E. Yang. Grammar-based codes for universal lossless data compression. *Communications in Information and Systems*, 2(2):29–52, 2002.
11. J. Kieffer, E. Yang, G. Nelson, and P. Cosman. Universal lossless compression via multilevel pattern matching. *IEEE Transactions on Information Theory*, 46(4):1227–1245, 2000.
12. J. Larsson and A. Moffat. Offline dictionary-based compression. In *Proc. DCC'99*, pages 296–305. IEEE Computer Society, 1999.
13. M. Miyazaki, A. Shinohara, and M. Takeda. An improved pattern matching algorithm for strings in terms of straight line programs. *Journal of Discrete Algorithms*, 1(1):187–204, 2000.
14. C. Nevill-Manning and I. Witten. Compression and explanation using hierarchical grammars. *Computer Journal*, 40(2/3):103–116, 1997.
15. C. Nevill-Manning and I. Witten. Identifying hierarchical structure in sequences: a linear-time algorithm. *J. Artificial Intelligence Research*, 7:67–82, 1997.
16. C. Nevill-Manning and I. Witten. Inferring lexical and grammatical structure from sequences. In *Proc. DCC'97*, pages 265–274. IEEE Computer Society, 1997.
17. C. Nevill-Manning and I. Witten. Phrase hierarchy inference and compression in bounded space. In *Proc. DCC'98*, pages 179–188. IEEE Computer Society, 1998.
18. W. Rytter. Algorithms on compressed strings and arrays. In *Proc. SOFSEM'99*, volume 1725 of *LNCS*, pages 48–65. Springer-Verlag, 1999.
19. W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theoretical Comput. Sci.*, 302(1–3):211–222, 2003.
20. P. Woelfel. Symbolic topological sorting with OBDDs. In *Proc. MFCS'03*, volume 2747 of *LNCS*, pages 671–680. Springer-Verlag, 2003.