Faster Subsequence and Don't-Care Pattern Matching on Compressed Texts

Yamamoto, Takanori
九州大学

Bannai, Hideo
九州大学

Inenaga, Shunsuke
九州大学

Takeda, Masayuki
九州大学

出版情報：Lecture Notes in Computer Science. 6661/2011, pp.309-322, 2011-06. Springer
バージョン：

Faster Subsequence and Don't-Care Pattern Matching on Compressed Texts

Takanori Yamamoto[1], Hideo Bannai[1], Shunsuke Inenaga[2], and Masayuki Takeda[1]

[1]Department of Informatics, Kyushu University
[2]Graduate School of Information Science and Electrical Engineering, Kyushu University
744 Motooka, Nishiku, Fukuoka 819–0395, Japan
takanori.yamamoto@i.kyushu-u.ac.jp
{bannai,takeda}@inf.kyushu-u.ac.jp
inenaga@c.csce.kyushu-u.ac.jp

Abstract. Subsequence pattern matching problems on compressed text were first considered by Cégielski *et al.* (Window Subsequence Problems for Compressed Texts, Proc. CSR 2006, LNCS 3967, pp. 127–136), where the principal problem is: given a string T represented as a straight line program (SLP) $\mathcal{T}$ of size n, a string P of size m, compute the number of minimal subsequence occurrences of P in T. We present an $O(nm)$ time algorithm for solving all variations of the problem introduced by Cégielski *et al.*. This improves the previous best known algorithm of Tiskin (Towards approximate matching in compressed strings: Local subsequence recognition, Proc. CSR 2011), which runs in $O(nm \log m)$ time. We further show that our algorithms can be modified to solve a wider range of problems in the same $O(nm)$ time complexity, and present the first matching algorithms for patterns containing VLDC (variable length don't care) symbols, as well as for patterns containing FLDC (fixed length don't care) symbols, on SLP compressed texts.

1 Introduction

A *straight-line program* (*SLP*) [6] is a context free grammar in the Chomsky normal form that derives a single string. SLPs are a widely accepted abstract model of various text compression schemes, since texts compressed by any grammar-based compression algorithm (e.g. [12, 8]) can be represented as SLPs, and those compressed by the LZ-family (e.g., [16, 17]) can be quickly transformed to SLPs. An SLP of a string of size N can be as small as $O(\log N)$. SLPs are a promising representation of a given string, not only for reducing the storage size of the data, but for efficiently conducting various string processing operations [13, 5]. Recently, *self indices* based on SLPs have also appeared [4].

Subsequence pattern matching [1] and its related problems have extensively been studied. Window subsequences are also known as serial episodes in data mining applications [10]. Now our interest is: Can we efficiently solve subsequence matching problems on compressed strings? When both text and pattern are given as SLPs, subsequence matching is NP-hard [9]. Therefore, in the sequel we only consider the case where the text is given as an SLP, while the pattern is given as an uncompressed string.

Subsequence problems on SLP-compressed texts were first considered in [3]. The principal problem considered is to compute the number of minimal subsequence occurrences of P in T. They presented $O(nm^2 \log m)$ time algorithms for solving the problems for an SLP of size n and subsequence pattern of length m. Later, an improved algorithm running in time $O(nm^{1.5})$ was presented by Tiskin [14]. Later, Tiskin improved the running time to $O(nm \log m)$ [15]. In this paper, we further reduce the time complexities to $O(nm)$.

The contribution of this paper is twofold. Firstly, we improve the algorithm for building the L and R arrays of [3], from $O(nm^2 \log m)$ to $O(nm)$, therefore reducing the overall time complexity of the algorithms for the subsequence pattern matching problems to $O(nm)$. Following the ideas of [3], we give a simpler presentation of these algorithms.

Secondly, we show that the algorithm can be extended to cope with patterns that contain *don't care* symbols, and give $O(nm)$-time matching algorithms for patterns containing VLDC (variable length don't care) symbols, as well as an $O(nm)$-time matching algorithm for patterns containing FLDC (fixed length don't care) symbols. There has been work on pattern matching for patterns containing FLDC symbols on a compressed representation of Sturmian words [2]. On the other hand, our algorithms can search *arbitrary* SLPs for patterns containing don't cares, and hence are applicable to more practical compressed texts.

2 Preliminaries

2.1 Strings

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 empty string ε is a string of length 0, namely, $|\varepsilon| = 0$. For a string $T = XYZ$, X, Y and Z are called a *prefix*, *substring*, and *suffix* of T, respectively. The i-th character of a string T is denoted by $T[i]$ for $0 \leq i \leq |T| - 1$, and the substring of a string T that begins at position i and ends at position j is denoted by $T[i : j]$ for $0 \leq i \leq j \leq |T| - 1$. For convenience, let $T[i : j] = \varepsilon$ if $j < i$.

A string P of length m is a *subsequence* of string T, if there exist indices $0 \leq i_0 < \cdots < i_{m-1} \leq |T| - 1$ such that $P[0] = T[i_0], \ldots, P[m-1] = T[i_{m-1}]$. The pair (i_0, i_{m-1}) is called an *occurrence* of subsequence P in T. Let $Occ(T, P)$ denote the set of all occurrences of subsequence P in T. An occurrence $(u, v) \in Occ(T, P)$ is *minimal* if P is *not* a subsequence of $T[u + 1 : v]$ nor $T[u : v - 1]$. For strings X, Y, if an occurrence $(u, v) \in Occ(XY, P)$ satisfies $0 \leq u < |X|$ and $|X| \leq v < |XY|$, we say that this occurrence *crosses* X and Y.

2.2 Straight Line Programs

In this paper, we treat strings described in terms of *straight line programs* (*SLPs*). A straight line program $\mathcal{T}$ is a sequence of assignments such that $X_1 = expr_1, X_2 = expr_2, \ldots, X_n = expr_n$, where each X_i is a variable and each $expr_i$ is an expression, where $expr_i = a$ ($a \in \Sigma$), or $expr_i = X_\ell X_r$ ($\ell, r < i$).

Denote by T the string derived from the last variable X_n of the program $\mathcal{T}$. The *size* of the program $\mathcal{T}$ is the number n of assignments in $\mathcal{T}$. Note that $|T| = O(2^n)$.

Let $val(X_i)$ represent the string derived from X_i. When it is not confusing, we identify a variable X_i with $val(X_i)$. Then, $|X_i|$ denotes the length of the string X_i derives. For assignment $X_i = X_\ell X_r$, if an occurrence (u, v) of subsequence P in $val(X_i)$ crosses $val(X_\ell)$ and $val(X_r)$, we say that (u, v) is a crossing subsequence occurrence of P in X_i.

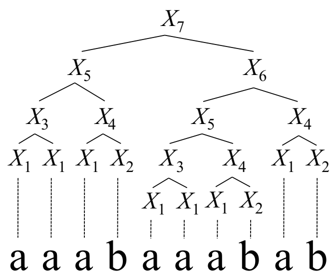

Fig. 1. An example of an SLP $X_1 = \texttt{a}$, $X_2 = \texttt{b}$, $X_3 = X_1X_1$, $X_4 = X_1X_2$, $X_5 = X_3X_4$, $X_6 = X_5X_4$, $X_7 = X_5X_6$, that derives string `aaabaaabab`.

3 Subsequence Matching Problems on Compressed Texts

This section is organized as follows: We first review an $O(nm)$ time algorithm for calculating tables Q^L and Q^R, which can determine whether a string P of length m is a subsequence of the string derived from an SLP $\mathcal{T}$ of size n (Subsequence Recognition). A brief description of the algorithm appears in [14], where it is noted that the algorithm "*has been known in folklore*", which was pointed out by Y. Lifshits. We then describe how to efficiently compute auxiliary tables L and R using Q^L and Q^R. Following the ideas in [3], we use L and R to give straightforward descriptions of $O(nm)$ time algorithms for solving the problem of finding all minimal subsequence occurrences of a pattern in a SLP-compressed text (Subsequence Matching), and its window-accumulated version (Window Subsequence Matching).

3.1 Subsequence Recognition

For $i = 1, \ldots, n, j = 0, \ldots, m$, let $Q^L(i, j)$ denote the length of the longest prefix of $P[j : m-1]$ which is a subsequence of X_i. We have that P is a subsequence of T, if and only if $Q^L(n, 0) = m$.

Lemma 1 ([14]). *Given a pattern P of length m and an SLP $\mathcal{T}$ of size n representing text T, $Q^L(i, j)$ for $i = 1, \ldots, n, j = 0, \ldots, m$ can be calculated in $O(nm)$ time and space.*

Proof. $Q^L(i,j)$ can be defined recursively, as follows. For the base case, if $X_i = a$ for some $a \in \Sigma$, then

$$Q^L(i,j) = \begin{cases} 1 & \text{if } j < m \text{ and } P[j] = a, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

If $X_i = X_\ell X_r$, then

$$Q^L(i,j) = Q^L(\ell,j) + Q^L(r,j') \tag{2}$$

where $j' = j + Q^L(\ell,j)$, because $P[j : j + Q^L(\ell,j) - 1]$ is the longest prefix of $P[j : m-1]$ that is a subsequence of X_ℓ, and the rest is the longest prefix of $P[j + Q^L(\ell,j) : m-1]$ that is a subsequence of X_r. Since each $Q^L(i,j)$ can be calculated in constant time, $Q^L(i,j)$ for $i = 1,\ldots,n, j = 0,\ldots,m$ can be calculated in $O(nm)$ time and space. □

Thus we can test whether a pattern P is a subsequence of an SLP $\mathcal{T}$ in $O(nm)$ time.

We similarly define $Q^R(i,j)$ as the length of the longest suffix of $P[0 : m-j-1]$ that is a subsequence of X_i, which can also be calculated in $O(nm)$ time and space.

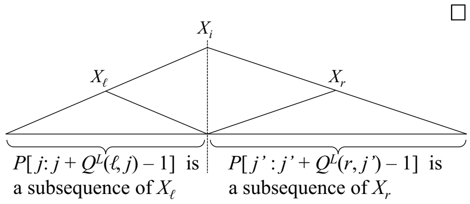

Fig. 2. Lemma 1. $Q^L(i,j) = Q^L(\ell,j) + Q^L(r,j')$ where $j' = j + Q^L(\ell,j)$. $Q^L(\ell,j)$ is the length of the prefix of $P[j : m-1]$ which is a subsequence of X_ℓ, and $Q^L(r,j')$ is the length of the prefix of the rest of it.

3.2 Subsequence Matching

Auxiliary Tables. We next define $L(i,j)$ and $R(i,j)$ that are central to the algorithm presented in [3]. We define $L(i,j)$ as the length of the shortest prefix of X_i, for which $P[j : m-1]$ is a subsequence. When there is no such prefix of X_i, $L(i,j)$ is defined as ∞. We similarly define $R(i,j)$ as the length of the shortest suffix of X_i, for which $P[0 : m-j-1]$ is a subsequence. When there is no such suffix of X_i, $R(i,j)$ is defined as ∞. Only these values for L (resp. R) corresponding to suffixes (resp. prefixes) of P are required in the algorithms which follow. However, the algorithm presented in [3] required the values for L and R corresponding to *all substrings* of P to compute these values, therefore making the running time of the algorithm $O(nm^2 \log m)$. We improve their algorithm by showing that we can calculate $L(i,j)$ (resp. $R(i,j)$) using only values corresponding to suffixes (resp. prefixes) of P with support from $Q^L(i,j)$ (resp. $Q^R(i,j)$), and reduce the running time to $O(nm)$.

Lemma 2. *Given a pattern P of length m, an SLP $\mathcal{T}$ of size n representing text T, and $Q^L(i,j)$ (resp. $Q^R(i,j)$) for $i = 1,\ldots,n, j = 0,\ldots,m$, $L(i,j)$ (resp. $R(i,j)$) for all $i = 1,\ldots,n, j = 0,\ldots,m$ can be calculated in $O(nm)$ time and space.*

Proof. We shall only describe how to calculate $L(i,j)$ using $Q^L(i,j)$, since the case for $R(i,j)$ and $Q^R(i,j)$ is essentially the same. $L(i,j)$ can be defined recursively as follows: For the base case, if $X_i = a$ for some $a \in \Sigma$, then

$$L(i,j) = \begin{cases} 0 & \text{if } j = m, \\ 1 & \text{if } j = m-1 \text{ and } P[j : m-1] = a, \\ \infty & \text{otherwise.} \end{cases}$$

If $X_i = X_\ell X_r$, then

$$L(i,j) = \begin{cases} L(\ell,j) & \text{if } L(\ell,j) \neq \infty, \\ |X_\ell| + L(r,j') & \text{if } L(\ell,j) = \infty, \end{cases}$$

where $j' = j + Q^L(\ell,j)$. This is because: When $L(\ell,j) \neq \infty$, $P[j : m-1]$ is a subsequence of X_ℓ, and $L(\ell,j)$ is the length of the shortest prefix of X_ℓ for which $P[j : m-1]$ is a subsequence. Since X_ℓ is a prefix of X_i, the length of the shortest prefix of $X_i = X_\ell X_r$ for which $P[j : m-1]$ is a subsequence is clearly equal to $L(\ell,j)$. When $L(\ell,j) = \infty$, $P[j : m-1]$ is not a subsequence of X_ℓ. This implies that the value of $L(i,j)$ is at least $|X_\ell|$. The exact value of $L(i,j)$ can be efficiently computed from $Q^L(\ell,j)$, as follows. Since $L(i,j) - |X_\ell|$ equals to the shortest prefix of X_r for which $P[j' : m-1]$ is a subsequence, we have $L(i,j) - |X_\ell| = L(r,j')$ where $j' = j + Q^L(\ell,j)$.

Therefore, given the Q^L table, each $L(i,j)$ can be computed in constant time. Hence $L(i,j)$ for all $i = 1,\ldots,n, j = 0,\ldots,m$ can be computed in $O(nm)$ time and space. ☐

Counting Minimal Occurrences. For text T represented by an SLP $\mathcal{T}$ of size n, we show how to calculate the number of minimal occurrences of subsequence P of length m in T in time $O(nm)$, using $L(i,j)$ and $R(i,j)$. Let M_i denote the number of minimal occurrences of P in $val(X_i)$. Since $val(X_n) = T$, the desired output is the value of M_n.

Our algorithm is based essentially on the same ideas as described in [3]. However, we note that they did not provide a rigorous proof of correctness, and the pseudo-code shown in their paper seems to contain some errors. Below, we give a simple presentation of the algorithm and a proof of correctness.

For any variable $X_i = X_\ell X_r$, let $C(\ell,r)$ denote the number of minimal occurrences of P in X_i that cross X_ℓ and X_r.

Lemma 3. *Given a pattern P of length m, an SLP $\mathcal{T}$ of size n, and $C(\ell,r)$ for all variables of form $X_i = X_\ell X_r$, the values M_i for $i = 1,\ldots,n$ can be calculated in $O(n)$ time.*

Proof. M_i is recursively computable as follows. For the base case, if $X_i = a$ for some $a \in \Sigma$, then $M_i = 0$ if $P \neq a$ and $M_i = 1$ if $P = a$. If $X_i = X_\ell X_r$, then $M_i = M_\ell + M_r + C(\ell,r)$. Hence we can compute M_i for all $i = 1,\ldots,n$ recursively, in total of $O(n)$ time. ☐

What remains is how to calculate $C(\ell, r)$ for all variables of type $X_i = X_\ell X_r$. For $k = 0, \ldots, m$, consider the following pairs (u_k, v_k) where u_k is the beginning position in X_i, of the shortest suffix of X_l for which $P[0 : m-1-k]$ is a subsequence (or $-\infty$ if such a suffix does not exist), and v_k is the ending position in X_i, of the shortest prefix of X_r for which $P[m-k : m-1]$ is a subsequence (or ∞ if such a prefix does not exist), i.e., $u_k = |X_\ell| - R(\ell, k)$ and $v_k = |X_\ell| + L(r, m-k) - 1$ (see also Fig. 3 (Left)). Clearly u_k and v_k are monotonically non-decreasing, that is, $u_{k-1} \leq u_k < |X_\ell| = u_m$, and $v_0 = |X_\ell| - 1 < v_k \leq v_{k+1}$ for $k = 1, \ldots, m-1$. When both $0 \leq u_k < |X_\ell|$ and $|X_\ell| \leq v_k < |X_i|$ hold, then (u_k, v_k) is a crossing subsequence occurrence of P in X_i. Note that neither (u_0, v_0) nor (u_m, v_m) are crossing occurrences. Let $Occ^{SS}(\ell, r) = \{(u_k, v_k) \mid k = 1, \ldots, m-1\}$. It is easy to see that every minimal crossing subsequence occurrence of P in X_i must be an element of $Occ^{SS}(\ell, r)$, and it remains to identify them.

Lemma 4. *$(u_k, v_k) \in Occ^{SS}(\ell, r)$ is a minimal occurrence if and only if $\nexists k' \in \{0, \ldots, m\}$ s.t. $(u_k, v_k) \neq (u_{k'}, v_{k'})$, and $u_k \leq u_{k'}$ and $v_{k'} \leq v_k$.*

Proof. ($\Longrightarrow$) If for some $k' \in \{0, \ldots, m\}$ s.t. $(u_{k'}, v_{k'}) \neq (u_k, v_k)$ we have $u_k \leq u_{k'}$ and $v_{k'} \leq v_k$, then (u_k, v_k) cannot be a minimal occurrence by definition.

($\Longleftarrow$) We show the contraposition. Assume (u_k, v_k) is not a minimal occurrence. If $u_k = -\infty$ (or resp. $v_k = \infty$), then $u_k \leq u_0 = -\infty$ (resp. $v_m \leq v_k = \infty$) and from the monotonicity of u_ks and v_ks, we can choose $k' = 0$ (resp. $k' = m$). If $u_k \neq -\infty$ and $v_k \neq \infty$, there exist some occurrence $(u, v) \neq (u_k, v_k)$ s.t. $u_k \leq u$ and $v \leq v_k$. If (u, v) is a crossing occurrence, then a minimal occurrence $(u_{k'}, v_{k'})$ can be chosen from $Occ^{SS}(\ell, r)$ s.t. $u \leq u_{k'}$ and $v_{k'} \leq v$. If it is not, then $v \leq |X_l| - 1$ or $u \geq |X_l|$, and we can choose (u_0, v_0) or (u_m, v_m). $\square$

Lemma 5. *Consider $(u_k, v_k) \in Occ^{SS}(\ell, r)$, and let $K = \{k' \mid (u_k, v_k) = (u_{k'}, v_{k'}),\ k' = 1, \ldots, m-1\}$, $k_s = \min K$ and $k_e = \max K$. Then, (u_k, v_k) is minimal if and only if $u_{k_s - 1} < u_k$ and $v_k < v_{k_e + 1}$.*

Proof. From the monotonicity of u_k and v_k, and from Lemma 4, we have that (u_k, v_k) is minimal if and only if

$$\begin{aligned}
&\nexists k' \in \{0, \ldots, m\} \text{ s.t. } (u_k, v_k) \neq (u_{k'}, v_{k'}),\quad (u_k \leq u_{k'}) \wedge (v_{k'} \leq v_k)\\
&\iff \forall k' \in \{0, \ldots, m\} \text{ s.t. } (u_{k'}, v_{k'}) \neq (u_k, v_k),\quad (u_{k'} < u_k) \vee (v_k < v_{k'})\\
&\iff ((u_{k_s-1} < u_k) \vee (v_k < v_{k_s-1})) \wedge ((u_{k_e+1} < u_k) \vee (v_k < v_{k_e+1}))\\
&\iff (u_{k_s-1} < u_k) \wedge (v_k < v_{k_e+1}).
\end{aligned}$$

$\square$

Lemma 6. *Given a pattern P of length m, an SLP $\mathcal{T}$ of size n, and $L(i, j)$, $R(i, j)$ for $i = 1, \ldots, n$, $j = 0, \ldots, m$, $C(\ell, r)$ for all variables of form $X_i = X_\ell X_r$, can be computed in total of $O(nm)$ time.*

Proof. A pseudo-code of our algorithm which computes $C(\ell, r)$ is shown in Algorithm 1 (see also Fig. 3 (Right)). The time complexity is clearly $O(m)$ for each $X_i = X_\ell X_r$, and hence $O(nm)$ in total. The correctness is due to Lemma 5. $\square$

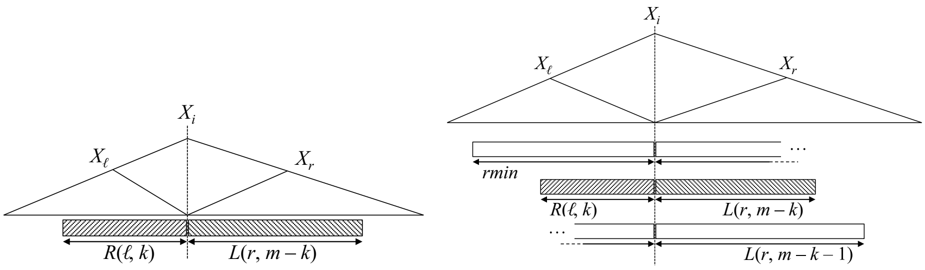

Fig. 3. (Left) If $R(\ell, k) \neq \infty$ and $L(r, m-k) \neq \infty$, there is a crossing subsequence occurrence of P. $P[0 : k-1]$ is a subsequence of $X_\ell[|X_\ell| - R(\ell, k) : |X_\ell| - 1]$, and $P[k : m-1]$ is a subsequence of $X_r[0 : L(r, m-k) - 1]$. (Right) Illustration of Algorithm 1. When $rmin > R(\ell, k)$ and $L(r, m-k) < L(r, m-k-1)$, then $(|X_\ell| - R(\ell, k), |X_\ell| + L(r, m-k) - 1)$ is a crossing minimal occurrence. We then update $rmin \leftarrow R(\ell, k)$ to find the next crossing minimal occurrence.

Algorithm 1: Counting Minimal Crossing Subsequence Occurrences.

Input: SLP variable $X_i = X_\ell X_r$, pattern P, auxiliary tables L, R.
Output: The number of minimal crossing subsequence occurrences $C(\ell, r)$.

1 $C \leftarrow 0$; $rmin \leftarrow R(\ell, 0)$;
2 **for** $k \leftarrow 1$ **to** $m-1$ **do**
3   **if** $rmin > R(\ell, k)$ **and** $L(r, m-k) < L(r, m-k-1)$ **then**
4         $C \leftarrow C + 1$; $rmin \leftarrow R(\ell, k)$;
5 **return** C ;

Finally, we obtain the main result of this section.

Theorem 1. *Given a pattern P of length m and an SLP $\mathcal{T}$ of size n representing text T, the number of minimal subsequence occurrences of P in T can be calculated in $O(nm)$ time.*

Window Subsequence Matching. Cégielski *et al.* [3] introduced several window-accumulated variants of subsequence pattern matching on compressed texts. The principal problem is: Given an SLP $\mathcal{T}$ generating text T, a pattern P, and non-negative integer w, count the number of minimal subsequence occurrences (u, v) of P in T such that $v - u + 1 \leq w$.

Our algorithm for counting minimal occurrences can readily be extended to this window-accumulated variant. See Algorithm 1. By simply adding "$R(\ell, k) + L(r, m-k) \leq w$" in the **if**-condition of line 3, we can solve the problem in the same complexity $O(nm)$. We remark that the other variants considered in [3] can also be solved in the same complexity. Details are omitted due to lack of space.

4 Don't-Care Pattern Matching Problems on Compressed Texts

In this section we show that the ideas of Section 3 can be extended to solve pattern matching problems for patterns with fixed length don't care (FLDC) and variable length don't care (VLDC) symbols, in the same complexity $O(nm)$.

4.1 FLDC Pattern Matching on Compressed Texts

We can find *substrings* of X_i matching P, the same way as counting minimal subsequence occurrences. If a subsequence P of T occurs in (i_0, i_{m-1}) and $i_{m-1} - i_0 + 1 = m$, obviously the substring $T[i_0 : i_{m-1}]$, is equal to P.

The above idea can be extended to a pattern matching problem where the pattern includes *fixed length don't care* (*FLDC*) symbols. Let the symbol '$\circ$' denote a don't care character that can match an arbitrary character in Σ. We call $P \in (\Sigma \cup \{\circ\})^*$ an *FLDC pattern.* An FLDC pattern P of length m occurs in string T at position i_0, if $T[i_0 + i] = P[i]$ or $P[i] = \circ$ for all $0 \leq i \leq m - 1$.

To count the occurrences of an FLDC pattern P using our window subsequence matching algorithms, we only need to count minimal subsequence occurrences of P that fit in a window of size $|P|$ with the exception that $\circ$ can match any single character. We can do this by simply modifying the base cases of $Q^L(i, j)$ and $L(i, j)$ as follows: If $X_i = a$ for some $a \in \Sigma$, then

$$Q^L(i,j) = \begin{cases} 1 & \text{if } j < m \text{ and } (P[j] = a \underline{\text{ or } P[j] = \circ}), \\ 0 & \text{otherwise.} \end{cases}$$

$$L(i,j) = \begin{cases} 0 & \text{if } j = m, \\ 1 & \text{if } j = m-1 \text{ and } (P[j : m-1] = a \underline{\text{ or } P[j : m-1] = \circ}), \\ \infty & \text{otherwise.} \end{cases}$$

The base cases of $Q^R(i, j)$ and $R(i, j)$ should be modified similarly as well.

4.2 VLDC Pattern Matching on Compressed Texts

Let the symbol '$\star$' denote a variable-length don't care character that can match an arbitrary string in Σ^*. We call $P \in (\Sigma \cup \{\star\})^*$ a *variable-length don't care (VLDC)* pattern. In the sequel, we only consider VLDC patterns that start and end with $\star$, and the $\star$'s do not occur consecutively. Consider any VLDC pattern $P = \star s_1 \star s_2 \star \cdots \star s_{m'} \star$, where each $s_j \in \Sigma^+$. The length of P is $m = \sum_{j=1}^{m'} |s_j|$. Each s_j is called the j-th *segment* of P. VLDC pattern P is said to *match* a string $T \in \Sigma^*$ if there exist indices $0 \leq i_0 < i_0 + |s_1| \leq i_1 < i_1 + |s_2| \leq \cdots < i_{m'-1} + |s_{m'}| \leq |T| - 1$ such that $s_1 = T[i_0 : i_0 + |s_1| - 1], \ldots, s_{m'} = T[i_{m'-1} : i_{m'-1} + |s_{m'}| - 1]$. The pair $(i_0, i_{m'-1} + |s_{m'}| - 1)$ is called an *occurrence* of VLDC pattern P in T. An occurrence (u, v) of VLDC pattern P in T is *minimal* if neither $(u + 1, v)$ nor $(u, v - 1)$ is an occurrence of P in T. Note that if each segment is a single character, then the above notion is equivalent to that of subsequences.

In what follows, we present how to compute minimal occurrences of a VLDC pattern in an SLP-compressed text. We will extend the notion of the auxiliary tables L, R, Q^L, and Q^R to cope with VLDC pattern matching. In so doing, we firstly introduce some new notion.

For any $X_i = X_\ell X_r$ and s_j, let

$$Occ^{\ddagger}(X_i, s_j) = \left\{ k \;\middle|\; \begin{array}{l} X_\ell[|X_\ell| - k : |X_\ell| - 1] = s_j[0 : k-1], \\ X_r[0 : |s_j| - k - 1] = s_j[k : |s_j| - 1], k = 1, \ldots, \min\{|s_j| - 1, |X_\ell|\} \end{array} \right\}.$$

Namely, values in $Occ^{\ddagger}(X_i, s_j)$ correspond to lengths of overlap with X_ℓ, for all crossing *substring* occurrences of s_j in X_i. We can compute $Occ^{\ddagger}(X_i, s_j)$ for all $i = 1, \ldots, n$, $j = 1, \ldots, m'$ in total of $O(nm)$ time and space, as follows: Let h be the length of the longest segment of P. We decompress the prefix and suffix of length h of each variable X_i, i.e., we compute strings $A_i = X_i[|X_i| - \min\{h, |X_i|\} : |X_i| - 1]$ and $B_i = X_i[0 : \min\{h, |X_i|\} - 1]$. This can be done in total of $O(nm)$ time and space. Let $X_i = X_\ell X_r$. We can then compute $Occ^{\ddagger}(X_i, s_j)$ in $O(|s_j|)$-time by using any standard linear-time pattern matching algorithm (e.g. [7]) for text $A_\ell B_r$ and pattern s_j. Moreover, $Occ^{\ddagger}(X_i, s_j)$ forms a single arithmetic progression [11], and can thus be represented as the first element, the last element, and the number of elements, which require only $O(1)$ space. Overall it takes $O(nm)$ time and space to compute $Occ^{\ddagger}(X_i, s_j)$ for all $i = 1, \ldots, n$, $j = 1, \ldots, m'$.

Let $LCP(X_i, s_j, k)$ denote the length of the longest common prefix of X_i and $s_j[k : |s_j| - 1]$. We can also compute $LCP(X_i, s_j, k)$ in $O(nm)$ time and space for all $i = 1, \ldots, n, j = 1, \ldots, m', k = 0, \ldots, |s_j|$, by the following recursion: For the base case, if $X_i = a$ for some $a \in \Sigma$, then $LCP(X_i, s_j, k) = 0$ if $X_i \neq s_j[k]$, and $LCP(X_i, s_j, k) = 1$ if $X_i = s_j[k]$. If $X_i = X_\ell X_r$, then

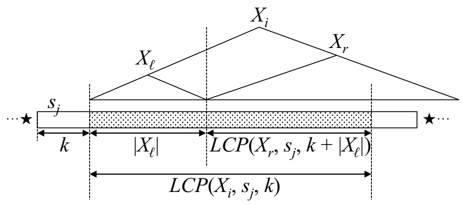

Fig. 4. Illustration of the recursion for $LCP(X_i, s_j, k)$. If $LCP(\ell, j, k) = |X_\ell|$, then $LCP(i, j, k) = |X_\ell| + LCP(r, j, k + |X_\ell|)$.

$$LCP(X_i, s_j, k) = \begin{cases} |X_\ell| + LCP(X_r, s_j, k + |X_\ell|) & \text{if } LCP(X_\ell, s_j, k) = |X_\ell|, \\ LCP(X_\ell, s_j, k) & \text{otherwise.} \end{cases}$$

Let $LCS(X_i, s_j, k)$ denote the length of the longest common suffix of X_i and $s_j[0 : |s_j| - k - 1]$. $LCS(X_i, s_j, k)$ can also be computed similarly in $O(nm)$ time and space.

For any VLDC pattern $P = \star s_1 \star s_2 \star \cdots \star s_{m'} \star$, we define a sub-pattern $segsub^L(P, j, k, q)$ of P, for $j = 1, \ldots, m' + 1$, $k = 0, \ldots, |s_j| - 1$, $q = 0, \ldots, m' -$

$j+1$, as follows:

$$segsub^L(P,j,k,q) = \begin{cases} \varepsilon & \text{if } q=0 \text{ or } j>m', \\ \star s_j \star \cdots \star s_{j+q-1}\star & \text{if } q>0, j\le m', k=0, \\ s_j[k:|s_j|-1]\star\cdots\star s_{j+q-1}\star & \text{if } q>0, j\le m', k>0. \end{cases}$$

Let $Q^L(i,j,k)$ denote the maximum number of segments in the sub-patterns $segsub^L(P,j,k,q)$ that match $val(X_i)$, i.e.,

$$Q^L(i,j,k) = \max\{q \mid segsub^L(P,j,k,q) \text{ matches } val(X_i)\}.$$

Also, we define $L(i,j,k)$ as the length of the shortest prefix of $val(X_i)$ that matches the sub-pattern giving $Q^L(i,j,k)$, i.e.,

$$L(i,j,k) = \min\{p \mid segsub^L(P,j,k,Q^L(i,j,k)) \text{ matches } X_i[0:p-1]\}$$

We define $Q^R(i,j,k)$ and $R(i,j,k)$ similarly, but be careful that $segsub^R(P,j,k,q)$ for $j=0,\ldots,m', k=0,\ldots,|s_j|-1, q=0,\ldots,j$ is defined as follows:

$$segsub^R(P,j,k,q) = \begin{cases} \varepsilon & \text{if } q=0 \text{ or } j=0, \\ \star s_{j-q+1}\star\cdots\star s_j\star & \text{if } q>0, j>0, k=0, \\ \star s_{j-q+1}\star\cdots\star s_j[0:|s_j|-k-1] & \text{if } q>0, j>0, k>0. \end{cases}$$

Lemma 7. *Given an SLP $\mathcal{T}$ and VLDC pattern $P=\star s_1\star\cdots\star s_{m'}\star$, $Q^L(i,j,k)$ (resp. $Q^R(i,j,k)$) and $L(i,j,k)$ (resp. $R(i,j,k)$) can be also computed in $O(nm)$ time and space for all $i=1,\ldots,n$, $j=1,\ldots,m'+1$ (resp. $j=0,\ldots,m'$) and $k=0,\ldots,|s_j|-1$.*

Proof. $Q^L(i,j,k)$ and $L(i,j,k)$ can be defined recursively as follows. For the base case, $X_i=a$, $(a\in\Sigma)$, then

$$Q^L(i,j,k) = \begin{cases} 1 & \text{if } 1\le j\le m' \text{ and } k=|s_j|-1 \text{ and } s_j[|s_j|-1]=a, \\ 0 & \text{otherwise.} \end{cases}$$

$$L(i,j,k) = \begin{cases} 1 & \text{if } Q^L(i,j,k)>0, \\ 0 & \text{otherwise.} \end{cases}$$

If $X_i = X_\ell X_r$, $|s_j|-k>|X_\ell|$ and $k>0$, then

$$Q^L(i,j,k) = \begin{cases} Q^L(r,j,k+|X_\ell|) & \text{if } LCP(X_\ell,s_j,k)=|X_\ell|, \\ 0 & \text{if } LCP(X_\ell,s_j,k)<|X_\ell|. \end{cases} \quad (3)$$

$$L(i,j,k) = \begin{cases} |X_\ell|+L(r,j,k+|X_\ell|) & \text{if } Q^L(i,j,k)>0, \\ 0 & \text{if } Q^L(i,j,k)=0. \end{cases} \quad (4)$$

(See also Fig 5 (Left).)

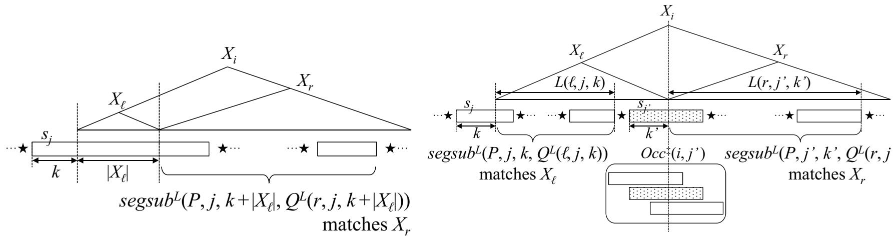

Fig. 5. (Left) Illustration of Equations (3) and (4) of Lemma 7. If $|s_j| - k > |X_\ell|$ and $LCP(i, j, k) = |X_\ell|$, then $Q^L(i, j, k) = Q^L(r, j, k + |X_\ell|)$ and $L(i, j, k) = |X_\ell| + L(r, j, k + |X_\ell|)$. (Right) Illustration of Equation (5) of Lemma 7. j' and k' can be computed in $O(1)$ time. Then, $Q^L(i, j, k)$ and $L(i, j, k)$ can be also computed in $O(1)$ time. Since $s_{j'}$ and $s_{j'-1}$ cannot overlap, k' must satisfy $k' + L(\ell, j, k) \leq |X_\ell|$.

If $X_i = X_\ell X_r$ and, $|s_j| - k \leq |X_\ell|$ or $k = 0$, then let $j' = j + Q^L(\ell, j, k)$ and $k' = \max\{x \mid x \in Occ^{\ddagger}(X_i, s_{j'}) \cup \{0\}, x + L(\ell, j, k) \leq |X_\ell|\}$. $Q^L(i, j, k)$ and $L(i, j, k)$ can be computed as follows: If $k = 0$ or $Q^L(\ell, j, k) > 0$, then $Q^L(i, j, k) = Q^L(\ell, j, k) + Q^L(r, j', k')$ and

$$L(i, j, k) = \begin{cases} |X_\ell| + L(r, j', k') & \text{if } Q^L(r, j', k') > 0, \\ L(\ell, j, k) & \text{if } Q^L(r, j', k') = 0. \end{cases} \tag{5}$$

(See also Fig 5 (Right).)
Otherwise ($k > 0$ and $Q^L(\ell, j, k) = 0$), $Q^L(i, j, k) = 0$ and $L(i, j, k) = 0$.

j' and k' can be computed in $O(1)$ time if $Q^L(\ell, j, k)$, $L(\ell, j, k)$ and $Occ^{\ddagger}(X_i, s_{j'})$ are already computed, and $Occ^{\ddagger}$ is represented as an arithmetic progression. Hence $Q^L(i, j, k)$ and $L(i, j, k)$ for all $i = 1, \ldots, n$, $j = 1, \ldots, m'$, and $k = 0, \ldots, |s_j|-1$ can be computed in $O(nm)$ time and space. $Q^R(i, j, k)$ and $R(i, j, k)$ can be computed similarly using $LCS(X_i, s_j, k)$. $\square$

An occurrence (u, v) of VLDC pattern P is a crossing occurrence in $X_i = X_\ell X_r$ if $0 \leq u < |X_\ell|$ and $|X_\ell| \leq v < |X_i|$. Let M_i and $C(\ell, r)$ denote the number of minimal occurrences and the number of minimal crossing occurrences of VLDC pattern P in $X_i = X_\ell X_r$, respectively.

Lemma 8. *Given a VLDC pattern P of length m, an SLP $\mathcal{T}$ of size n, and $C(\ell, r)$ for all variables of form $X_i = X_\ell X_r$, the values M_i for $i = 1, \ldots, n$ can be calculated in $O(n)$ time.*

Proof. M_i can be defined recursively as follows. For the base case ($X_i = a \in \Sigma$), if $P = \star a \star$ then $M_i = 1$, otherwise $M_i = 0$. For the case $X_i = X_\ell X_r$, $M_i = M_\ell + M_r + C(\ell, r)$. Thus M_i can be computed for all $i = 1, \ldots, n$, in $O(n)$ total time and space, if $C(\ell, r)$ for all variables of form $X_i = X_\ell X_r$ are already computed.

In what follows we describe how to compute $C(\ell, r)$ for each $X_i = X_\ell X_r$ in $O(m)$ time. Algorithm 2 shows a pseudo-code of our algorithm to compute

$C(\ell, r)$. For convenience, for $i = 1, \ldots, n, j = 0, \ldots, m'$ and $k \in Occ^{\ddagger}(X_i, s_j) \cup \{0\}$, let

$$\mathbf{L}(i,j,k) = \begin{cases} 0 & \text{if } j = 0, \\ L(i,j,k) & \text{if } j > 0 \text{ and } Q^L(i,j,k) = m' - j + 1, \\ \infty & \text{otherwise.} \end{cases}$$

$$\mathbf{R}(i,j,k) = \begin{cases} 0 & \text{if } j = 0, \\ R(i,j,k) & \text{if } j > 0 \text{ and } Q^R(i,j,k) = j, \\ \infty & \text{otherwise.} \end{cases}$$

Note that conceptually, the tables L and R for subsequences correspond to $\mathbf{L}$ and $\mathbf{R}$ defined above, and when $segsub^L(P, j, k, m' - j + 1)$ does not match X_i, then $\mathbf{L}(i,j,k) = \infty$, and when $segsub^R(P, j, k, j)$ does not match X_i, then $\mathbf{R}(i,j,k) = \infty$. Hence we can compute the number of crossing VLDC pattern occurrences in a similar way to the case of subsequence patterns.

Care is taken for possible crossing occurrences when a segment is crossing X_i. For any j and $k > 0$, only occurrences $(|X_\ell| - \mathbf{R}(\ell, j, |s_j| - k), |X_\ell| + \mathbf{L}(r, j, k) - 1)$ for which $k \in Occ^{\ddagger}(X_i, s_j)$ can be crossing occurrences of P in X_i (see also Fig. 6 (Left)). For $j = 2, \ldots, m'$ and $k = 0$, occurrences $(|X_\ell| - \mathbf{R}(\ell, j-1, 0), |X_\ell| + \mathbf{L}(r, j, 0) - 1)$ can be crossing occurrences of P in X_i (see also Fig. 6 (Right)). By checking these possible crossing occurrences in decreasing order of j and k, we can compute the number of crossing occurrences as described in Algorithm 2. Since the number of candidates is $d = \Sigma_{j=1}^{m'} |Occ^{\ddagger}(X_i, s_j)| + m' + 1 = O(m)$, we can compute all the crossing occurrences in a total of $O(nm)$ time and space. $\square$

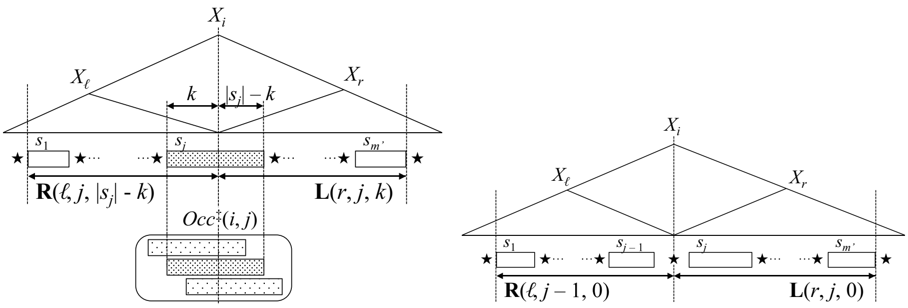

Fig. 6. Illustration of Algorithm 2. (Left) If $k \in Occ^{\ddagger}(X_i, s_j)$, $\mathbf{R}(\ell, j, |s_j| - k) \neq \infty$ and $\mathbf{L}(r, j, k) \neq \infty$, then $(|X_\ell| - \mathbf{R}(\ell, j, |s_j| - k), |X_\ell| + \mathbf{L}(r, j, k) - 1)$ is a candidate of a crossing occurrence. (Right) If $k = 0$, $\mathbf{R}(\ell, j-1, 0) \neq \infty$ and $\mathbf{L}(r, j, 0) \neq \infty$, then $(|X_\ell - \mathbf{R}(\ell, j-1, 0), |X_\ell| + \mathbf{L}(r, j, 0) - 1)$ is a candidate of a crossing occurrence.

Consequently, we obtain the main result of this section:

Algorithm 2: Counting Minimal Crossing VLDC Occurrences.

Input: SLP variable $X_i = X_\ell X_r$, pattern P, auxiliary tables $L(i,j,k)$, $R(i,j,k)$.
Output: The number of minimal crossing VLDC occurrences $C(\ell, r)$.

$d \leftarrow 0$; $(R[0], L[0]) \leftarrow (\mathbf{R}(\ell, m', 0), 0)$;
for $j \leftarrow m'$ **to** 1 **do**
  forall the $k \in Occ^{\ddagger}(X_i, s_j)$ *in descending order* **do**
    $d \leftarrow d + 1$; $(R[d], L[d]) \leftarrow (\mathbf{R}(\ell, j, |s_j| - k), \mathbf{L}(r, j, k))$;
  $d \leftarrow d + 1$; $(R[d], L[d]) \leftarrow (\mathbf{R}(\ell, j-1, 0), \mathbf{L}(r, j, 0))$;
$C \leftarrow 0$; $rmin \leftarrow R[0]$;
for $d' \leftarrow 1$ **to** $d - 1$ **do**
  if $rmin > R[d']$ *and* $L[d'] < L[d'+1]$ **then**
    $C \leftarrow C + 1$; $rmin \leftarrow R[d']$;
return C ;

Theorem 2. *Given a VLDC pattern P of length m and an SLP $\mathcal{T}$ of size n representing text T, the number of minimal occurrences of P in T can be calculated in $O(nm)$ time.*

Window VLDC Pattern Matching. This algorithm for VLDC patterns can be also extended to window-accumulated problems by adding the condition "$R[d'] + L[d'] \leq w$".

5 Conclusion

All algorithms we presented in this paper run in $O(nm)$ time and space. A natural open problem is if this can be reduced further. Other open problems are mixing variable and fixed length don't care symbols, and constraining the minimum and maximum lengths of strings that variable-length don't care symbols can match.

Acknowledgments

This work was supported by KAKENHI 22680014.

References

1. Baeza-Yates, R.A.: Searching subsequences. Theoretical Computer Science 78(2), 363–376 (1991)
2. Baturo, P., Rytter, W.: Compressed string-matching in standard sturmian words. Theoretical Computer Science 410(30–32), 2804–2810 (2009)
3. Cégielski, P., Guessarian, I., Lifshits, Y., Matiyasevich, Y.: Window subsequence problems for compressed texts. In: Proc. CSR 2006. LNCS, vol. 3967, pp. 127–136 (2006)

4. Claude, F., Navarro, G.: Self-indexed text compression using straight-line programs. In: Proc. MFCS 2009. LNCS, vol. 5734, pp. 235–246 (2009)
5. Hermelin, D., Landau, G.M., Landau, S., Weimann, O.: A unified algorithm for accelerating edit-distance computation via text-compression. In: Proc. STACS 2009. pp. 529–540 (2009)
6. Karpinski, M., Rytter, W., Shinohara, A.: An efficient pattern-matching algorithm for strings with short descriptions. Nordic Journal of Computing 4, 172–186 (1997)
7. Knuth, D.E., Morris, J.H., Pratt, V.R.: Fast pattern matching in strings. SIAM J. Comput. 6(2), 323–350 (1977)
8. Larsson, N.J., Moffat, A.: Offline dictionary-based compression. In: Proc. Data Compression Conference 1999. pp. 296–305. IEEE Computer Society (1999)
9. Lifshits, Y., Lohrey, M.: Querying and embedding compressed texts. In: MFCS 2006. LNCS, vol. 4162, pp. 681–692 (2006)
10. Mannila, H., Toivonen, H., Verkamo, A.I.: Discovery of frequent episodes in event sequences. Data Mining and Knowledge Discovery 1(3), 259–289 (1997)
11. Miyazaki, M., Shinohara, A., Takeda, M.: An improved pattern matching algorithm for strings in terms of straight-line programs. In: Proc. CPM 1997. LNCS, vol. 1264, pp. 1–11. Springer-Verlag (1999)
12. Nevill-Manning, C.G., Witten, I.H., Maulsby, D.L.: Compression by induction of hierarchical grammars. In: Data Compression Conference 1994. pp. 244–253. IEEE Computer Society (1994)
13. Rytter, W.: Grammar compression, LZ-encodings, and string algorithms with implicit input. In: Proc. ICALP 2004. LNCS, vol. 3142, pp. 15–27 (2004)
14. Tiskin, A.: Faster subsequence recognition in compressed strings. J. Math. Sci. 158(5), 759–769 (2009)
15. Tiskin, A.: Towards approximate matching in compressed strings: Local subsequence recognition. In: Proc. CSR 2011 (2011), to appear
16. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Transactions on Information Theory IT-23(3), 337–349 (1977)
17. Ziv, J., Lempel, A.: Compression of individual sequences via variable-length coding. IEEE Transactions on Information Theory 24(5), 530–536 (1978)