

A New Family of String Classifiers Based on Local Relatedness

Yasuto Higa¹, Shunsuke Inenaga^{2,1}, Hideo Bannai¹, and Masayuki Takeda^{1,3}

¹ Department of Informatics, Kyushu University, Japan
{y-higa, shunsuke.inenaga, bannai, takeda}@i.kyushu-u.ac.jp

² Japan Society for the Promotion of Science

³ SORST, Japan Science and Technology Agency (JST)

Abstract. This paper introduces a new family of *string classifiers* based on local relatedness. We use three types of local relatedness measurements, namely, *longest common substrings (LCStr's)*, *longest common subsequences (LCSeq's)*, and *window-accumulated longest common subsequences (wLCSeq's)*. We show that finding the optimal classifier for given two sets of strings (the positive set and the negative set), is NP-hard for all of the above measurements. In order to achieve practically efficient algorithms for finding the best classifier, we investigate pruning heuristics and fast string matching techniques based on the properties of the local relatedness measurements.

1 Introduction

In recent years, we have witnessed a massive increase in the amount of *string* data available in many different domains, such as text data on the Internet, and biological sequence data. Finding meaningful knowledge in the form of string *patterns* from such data sets has become a very important research topic. In this light, we and others have been studying the problem of finding the optimal pattern which distinguishes between a positive set of strings and a negative set of strings [1,2,3,4,5,6,7,8,9,10,11]. The optimal pattern discovery problem aims to find the highest scoring pattern with respect to a certain scoring function, usually preferring patterns which are contained in all or most strings in the positive data set, but not contained in all or most strings of the negative data set.

In previous work, the classification models are mainly based on *distance*, where a string is classified to be consistent with a pattern when the string contains a substring that is within a certain fixed distance of the pattern, and not consistent otherwise. In this paper, we consider a classification model based on *relatedness*, where a string is classified to be consistent with the string classifier when it is locally more similar to the classifier than a certain fixed relatedness, and not consistent otherwise. Although the two seem to be equivalent concepts that are simply worded differently, the latter gives rise to a new, richer family of string classifiers.

We study and show the subtle differences between them from an algorithmic perspective. We consider three types of relatedness measurements, namely,

longest common substrings (*LCStr's*), longest common subsequences (*LCSeq's*), and window-accumulated longest common subsequences (*wLCSeq's*). We show that finding the optimal classifier for given two sets of strings is NP-hard for all of the above measurements. In order to achieve practically efficient algorithms for finding the best classifier, we investigate pruning heuristics and fast string matching techniques based on the properties of the local relatedness measurements. Our preliminary experiments on DNA sequence data showed that the algorithm for the *wLCSeq* measurement runs in acceptable amount of time.

2 Preliminaries

2.1 Notations

Let \mathcal{N} be the set of non-negative integers. Let Σ be a finite *alphabet*, and let Σ^* be the set of all *strings* over Σ . The *length* of string s is denoted by $|w|$. The empty string is denoted by ε , that is, $|\varepsilon| = 0$. For set $S \subseteq \Sigma^*$ of strings, we denote by $|S|$ the number of strings in S , and by $\|S\|$ the total length of strings in S .

Strings x , y and z are said to be a *prefix*, *substring*, *suffix* of string $s = xyz$, respectively. Let $Substr(s)$ be the set of the substrings of string s . When string x is a substring of string s , then s is said to be a *superstring* of x . The i -th character of string s is denoted by $s[i]$ for $1 \leq i \leq |s|$, and the substring that begins at position i and ends at position j is denoted by $s[i..j]$ for $1 \leq i \leq j \leq |s|$. We say that string q is a *subsequence* of string s if $q = s[i_1] \cdots s[i_{|q|}]$ for some $1 \leq i_1 < \cdots < i_{|q|} \leq |s|$. Let $Subseq(s)$ be the set of the subsequences of string s . When string x is a subsequence of string s , then s is said to be a *supersequence* of x .

For two strings $p, s \in \Sigma^*$, if $q \in Substr(p) \cap Substr(s)$, then q is called a *common substring* of p and s . When no common substrings of p and s are longer than q , q is called a *longest common substring (LCStr for short)* of p and s . Then we denote $lcstr(p, s) = |q|$. It is easy to see that $lcstr(p, s) \leq \min\{|p|, |s|\}$.

Similarly, if $q \in Subseq(p) \cap Subseq(s)$, then q is called a *common subsequence* of p and s . When no common subsequences of p and s are longer than q , q is called a *longest common subsequence (LCSeq for short)* of p and s . Then we denote $lcseq(p, s) = |q|$. It is easy to see that $lcseq(p, s) \leq \min\{|p|, |s|\}$.

2.2 Relatives of a String

The functions *lcstr* and *lcseq* are relatedness measures that quantify the affinities between two strings. Further, we consider an extended version of *lcseq*, *window-accumulated LCSeq measure*, where we compute the *lcseq* values for p against all substrings of s of length $\leq d$ (d is a positive integer). The new measure *wlcseq^d* is defined as follows.

$$wlcseq^d(p, s) = \max\{lcseq(p, t) \mid t \in Substr(s) \text{ and } |t| \leq d\}.$$

We note that when d is long enough then $wlcseq^d(p, s) = lcseq(p, s)$.

Let δ be one of the measures $lcstr$, $lcseq$, and $wlcseq^d$. A k -relative of a string p under δ is any string s with $\delta(s, p) \geq k$. The set of k -relatives of p is denoted by $Re^\delta(p; k)$. That is,

$$Re^\delta(p; k) = \{s \in \Sigma^* \mid \delta(p, s) \geq k\}.$$

Definition 1. *The language class w.r.t. $Re^\delta(p; k)$ for each δ is as follows.*

$$\begin{aligned} \text{LCSTRL} &= \{Re^{lcstr}(p; k) \mid p \in \Sigma^* \text{ and } k \in \mathcal{N}\} \\ \text{LCSEQL} &= \{Re^{lcseq}(p; k) \mid p \in \Sigma^* \text{ and } k \in \mathcal{N}\} \\ \text{wLCSEQL}^d &= \{Re^{wlcseq^d}(p; k) \mid p \in \Sigma^* \text{ and } k \in \mathcal{N}\} \end{aligned}$$

Remark 1. In [3] the subsequence pattern discovery problem was discussed, where a pattern $p \in \Sigma^*$ matches any supersequence of p . We note that LCSEQL subsumes the languages of the subsequence patterns since $Re^{lcseq}(p; |p|)$ is exactly the set of supersequences of p .

Remark 2. In [2] the pattern discovery problem for the window-accumulated subsequence patterns (episode patterns) was addressed, where a window-accumulated subsequence pattern is formally an ordered pair $\langle p, d \rangle \in \Sigma^* \times \mathcal{N}$ and matches a string s if there is a substring t of s such that $|t| \leq d$ and $p \in \text{Subseq}(t)$. Then, one can see that wLCSEQL^d generalizes the languages of window-accumulated subsequence patterns since we have $Re^{wlcseq^d}(p; |p|)$ is identical to the language of the window-accumulated subsequence pattern $\langle p, d \rangle$.

3 Finding Best String Classifiers

3.1 Score Function

Suppose Π is a set of descriptions over some finite alphabet, and each $\pi \in \Pi$ defines a language $L(\pi) \subseteq \Sigma^*$. Let $good$ be a function from $\Pi \times 2^{\Sigma^*} \times 2^{\Sigma^*}$ to the real numbers. The problem we consider in this paper is: *Given two sets $S, T \subseteq \Sigma^*$ of strings, find a description $\pi \in \Pi$ that maximizes score $good(\pi, S, T)$.* Intuitively, score $good(\pi, S, T)$ expresses the ‘goodness’ of π as a classifier for S and T . The definition of $good$ varies with applications. For example, the χ^2 values, entropy information gain, and Gini index are often used. Essentially, these statistical measures are defined by the number of strings that satisfy the rule specified by π . Any of the above-mentioned measures can be expressed by the following form:

$$\begin{aligned} good(\pi, S, T) &= f(x_\pi, y_\pi, |S|, |T|), \\ \text{where } x_\pi &= |S \cap L(\pi)| \text{ and } y_\pi = |T \cap L(\pi)|. \end{aligned}$$

When S and T are fixed, $x_{\max} = |S|$ and $y_{\max} = |T|$ are regarded as constants. On this assumption, we abbreviate the function to $f(x_\pi, y_\pi)$. In the sequel, we assume that f is *pseudo-convex* (also called *conic* in the previous work [3]) and can

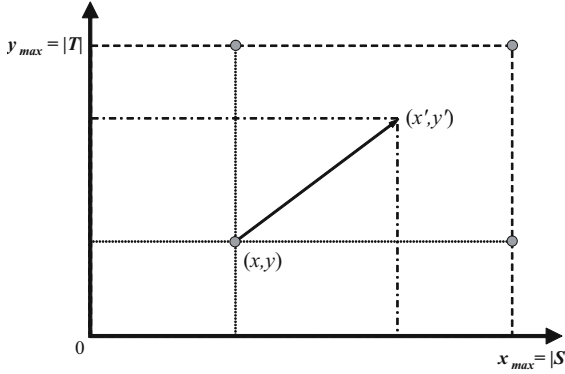


Fig. 1. Illustration of the domain of score function f . For pseudo-convex score functions, the highest score of an arbitrary point in the rectangle defined by the four highlighted points is the maximum score of the four points (Lemma 1).

be evaluated in constant time. We say that a function f from $[0, x_{max}] \times [0, y_{max}]$ to real numbers is pseudo-convex if

- for any $0 \leq y \leq y_{max}$, there exists an x_1 such that
 - $f(x, y) \geq f(x', y)$ for any $0 \leq x < x' \leq x_1$, and
 - $f(x, y) \leq f(x', y)$ for any $x_1 \leq x < x' \leq x_{max}$.
- for any $0 \leq x \leq x_{max}$, there exists a y_1 such that.
 - $f(x, y) \geq f(x, y')$ for any $0 \leq y < y' \leq y_1$, and
 - $f(x, y) \leq f(x, y')$ for any $y_1 \leq y < y' \leq y_{max}$.

The following is an important property of pseudo-convex functions.

Lemma 1 ([6]). For any $0 \leq x < x' \leq x_{max}$ and $0 \leq y < y' \leq y_{max}$,

$$f(x', y') \leq \max\{f(x, y), f(x, y_{max}), f(x_{max}, y), f(x_{max}, y_{max})\}.$$

3.2 Problem and Complexities

In this paper we consider the following problem for each relatedness measure δ .

Definition 2 (Finding best string classifier under δ according to f)

Input: Two finite sets $S, T \subseteq \Sigma^*$ of strings, and a positive integer k .

Output: A string $p \in \Sigma^*$ that maximizes score $f(x_\pi, y_\pi)$, where $\pi = \langle p, k \rangle$, and $x_\pi = |S \cap Re^\delta(p; k)|$ and $y_\pi = |T \cap Re^\delta(p; k)|$.

Theorem 1. The optimization problem of Definition 2 under $lcstr$ is NP-hard.

Proof. Reduction from MINIMUM SET COVER. (Given graph $G = (V, E)$, a set $C = \{C_1, \dots, C_m\}$ where each $C_i \subseteq V$ for each $1 \leq i \leq m$, and integer c ,

does there exist a subset $C' \subseteq C$ such that $|C'| \leq c$ and any vertex in V is contained in at least one member of C' ?) Consider the function

$$f(x, y) = \begin{cases} 0 & \text{if } x < x_{\max} \\ y_{\max} - y & \text{otherwise } (x = x_{\max}) \end{cases}$$

and create an instance of finding the best string classifier under *lcstr* as follows. Let $k = \lceil \log_2 m \rceil$, and let \bar{i} denote the k -bit binary representation of integer $i \leq m$. S will consist of $|V|$ strings, each string corresponding to a node in V . For each $v \in V$, define $s_v = \sum_{i:v \in C_i} \bar{i} \Phi$. Let $x = \sum_{s \in X} \#s$, where $X = \{s_v[\bar{i}..(i+k-1)] \mid v \in V, 1 \leq i \leq |s_v| - k + 1, \exists j (i \leq j \leq i+k-1), \text{ s.t. } s_v[j] = \Phi\}$, that is, x is the concatenation of all length k substrings of strings in S that contain the character Φ , each preceded by $\#$. We define $T = \{\bar{i}x \mid 1 \leq i \leq m\}$, where each string corresponds to the member $C_i \in C$. Then, the existence of a set cover $C' = \{C_{i_1}, \dots, C_{i_c}\}$ of size c implies the existence of a string p giving $f(x_\pi, y_\pi) = y_{\max} - c$ and vice versa: Suppose C' is a cover of V , and consider $p = \bar{i}_1 \Phi \bar{i}_2 \dots \Phi \bar{i}_c$. Since each $v \in V$ is contained in some member $C_{i_j} \in C'$, each $s_v \in S$ will share the length k substring \bar{i}_j with p . Noting that the character ‘ Φ ’ is not contained in any strings of S or T , p can only share length k substrings via each \bar{i}_j ($1 \leq j \leq c$). Therefore, p will only be a k -relative to the c strings $\{\bar{i}_j x \mid C_{i_j} \in C'\} \subseteq T$. On the other hand, suppose p is a string that is a k -relative of all strings in S but for only c strings in T . Notice that p can only be a k -relative of strings in S or T by sharing length k substrings that correspond to some \bar{i} ($1 \leq i \leq m$), since otherwise: (1) if the length k substring contained Φ , then p would also be a k -relative of x and hence all strings in T , and (2) if the length k substring contained $\#$, then p would not be a k -relative of any string in S . For each different \bar{i} that p contains, a unique string $\bar{i}x \in T$ also becomes a k -relative of p . Therefore, p will contain exactly c substrings $\bar{i}_1, \dots, \bar{i}_c$, where each $s_v \in S$ will contain at least one \bar{i}_j ($1 \leq j \leq c$). This implies a set cover of size c consisting of C_{i_j} ($1 \leq j \leq c$). \square

Theorem 2. *The optimization problem of Definition 2 under *lcseq* is NP-hard.*

Proof. Reduction from MINIMUM SET COVER (See Theorem 1). Consider f as in Theorem 1. Consider the alphabet $\Sigma = \{\sigma_1, \dots, \sigma_m\}$. Let $k = 1$, $S = \sum_{\sigma_i:v \in C_i} \sigma_i$, $T = \{\sigma_1, \dots, \sigma_m\}$ ($\sigma_i \neq \sigma_j$ for $i \neq j$). Since $k = 1$, a string containing character σ_i will be a relative of a string s if and only if s also contains the character σ_i . Therefore, the existence of a string p giving $f(x_\pi, y_\pi) = y_{\max} - c$ implies the existence of a set cover $C' = \{C_{i_1}, \dots, C_{i_c}\}$ of size c , and vice versa. \square

Recall that when d is long enough, we have $wlseq^d(p, d) = lcseq(p, s)$. Thus, the optimization problem for the *lcseq* measure is a special case of that for the $wlseq^d$ measure. Hence we get the following result.

Theorem 3. *The optimization problem of Definition 2 under *wlseq*^d is NP-hard.*

Since the optimization problem of finding the best π is NP-hard for all types of the similarity measures, we inherently face exponentially many candidates for the best string classifier. The two keys to a practically efficient computation of the best π are: *reducing the number of candidates (pruning)*, and *quickly counting x_π and y_π for each candidate π (fast string matching)*.

3.3 Branch-and-Bound Algorithms

Let δ be one of the measures *lcstr*, *lcseq*, and *wlcseq*^d. Consider given k . Note that, for any string p of length less than k , we always have $Re^\delta(p; k) = \emptyset$. On the other hand, we need to care about all the $|\Sigma|^k$ strings of length k , because all of those strings have a possibility of being a ‘seed’ of the best string with the highest score. Then, we examine longer candidates by appending new characters to the right of the $|\Sigma|^k$ strings. The next lemma states a useful property on $Re^\delta(p; k)$.

Lemma 2. *Let k be any positive integer. For any two strings p, p' in Σ^* , if p is a prefix of p' , then $Re^\delta(p'; k) \supseteq Re^\delta(p; k)$.*

The following pruning lemma are derived from Lemmas 1 and 2.

Lemma 3. *Let k be any positive integer. For any two strings p, p' such that p is a prefix of p' , let $\pi = \langle p, k \rangle$ and $\pi' = \langle p', k \rangle$. Then,*

$$f(x_{\pi'}, y_{\pi'}) \leq \max\{f(x_\pi, y_\pi), f(x_\pi, y_{\max}), f(x_{\max}, y_\pi), f(x_{\max}, y_{\max})\},$$

where $x_\pi = |S \cap Re^\delta(p; k)|$ and $y_\pi = |T \cap Re^\delta(p; k)|$.

What remains is how to compute the numbers x_π and y_π of strings s in S and T , respectively, such that $\delta(p, s) \geq k$, where $\pi = \langle p, k \rangle$.

Firstly, we consider the case of $\delta = \textit{lcstr}$.

Definition 3 (Computing local relatedness under measure *lcstr*)

Given: *A finite set $S \subseteq \Sigma^*$ of strings, and a positive integer k .*

Query: *A string $p \in \Sigma^*$.*

Answer: *The cardinality of $Re^{\textit{lcstr}}(p; k) \cap S$.*

Theorem 4. *The problem of computing local relatedness under the measure *lcstr* can be solved in $O(|p|)$ time using $O(|S|)$ extra space after $O(\|S\|)$ time and space preprocessing.*

Proof. We build the directed acyclic word graph (DAWG) [12] from the strings in S . We note that the nodes of DAWG represent the equivalence classes under some equivalence relation defined on $Substr(S) = \bigcup_{s \in S} Substr(s)$. Each equivalence class can be written as $\{x'y \mid x' \text{ is a suffix of } x\}$ for some strings x, y with $xy \in Substr(S)$, and therefore it can contain at most one string t of length k . For every node v containing such string t , we associate v with the list of ID’s of strings in S that are superstrings of t . Such a data structure can be built only in $O(\|S\|)$ time

and space. To compute the cardinality of $Re^{lcstr}(p; k)$ for every candidate p , we use this data structure as a finite-state sequential machine. The machine makes state-transitions scanning the characters of p one by one. Whenever the machine is in a state (node) such that the corresponding equivalence class contains no string of length $\leq k$, it makes “failure transition” navigated by the suffix links. If the current state has outputs, then it implies that all the strings s listed in the outputs satisfy $lcstr(p, s) \geq k$. The number of failure transitions executed is bounded by the number of ordinary state transitions executed, and is therefore at most $|p|$. The cardinality is thus computed in $O(|p|)$ time. \square

We can therefore compute x_π and y_π in $O(|p|)$ time using $O(|S| + |T|)$ extra space after $O(\|S\| + \|T\|)$ time and space preprocessing.

Secondly, we consider $\delta = lcseq$. We do not preprocess the input S and T , and simply compute $lcseq(p, s)$ against all strings s in $S \cup T$. Each $lcseq(p, s)$ is computable in $O(|p| \cdot |s|)$ time by a standard dynamic programming (DP) method. Section 4 explains the DP method. We can compute x_π and y_π in $O(|p| \cdot (\|S\| + \|T\|))$ time.

Lastly, we will devote the next full section to the case of $\delta = wlcseq^d$, as this case needs to be explained in details.

4 Computing Local Relatedness Under $wlcseq^d$

Given two strings p, s , a standard technique for computing $lcseq(p, s)$ is the *dynamic programming* method, where we compute the DP matrix of size $(|p| + 1) \times (|s| + 1)$ for which $DP[i, j] = lcseq(p[1..i], s[1..j])$ for $1 \leq i \leq |p|$ and $1 \leq j \leq |s|$. The recurrence for computing the DP matrix is the following:

$$DP[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ \max(DP[i - 1, j], DP[i, j - 1]) & \text{if } i, j > 0 \text{ and } p[i] \neq s[j], \\ DP[i - 1, j - 1] + 1 & \text{if } i, j > 0 \text{ and } p[i] = s[j]. \end{cases}$$

Therefore, to compute $lcseq(p, s) = DP[|p|, |s|]$, we need $O(|p| \cdot |s|)$ time and space. Pair (i, j) is said to be a *partition point* of DP , if $DP[i, j] = DP[i - 1, j] + 1$. P denotes the set of the partition points of DP . Remark that P is a compressed form of DP , and the size of P is $O(L|s|)$, where $L = lcseq(p, s)$. See Fig. 2 for an example of a DP matrix and its partition points. The partition point set P is implemented by double-linked lists as shown in Fig. 3, where the cells are vertically sorted by the $lcseq$ values, and the values in the cells represent the corresponding row indices.

The problem considered in this section is the following.

Definition 4 (Computing local relatedness under measure $wlcseq^d$)

Given: A finite set $S \subseteq \Sigma^*$ of strings, and a positive integer k .

Query: A string $p \in \Sigma^*$.

Answer: The cardinality of $Re^{wlcseq^d}(p; k) \cap S$.

		c	b	a	c	b	a	a	b	a
	0	0	0	0	0	0	0	0	0	0
b	0	0	1	1	1	1	1	1	1	1
c	0	1	1	1	2	2	2	2	2	2
d	0	1	1	1	2	2	2	2	2	2
a	0	1	1	2	2	2	3	3	3	3
b	0	1	2	2	2	3	3	3	4	4
a	0	1	2	3	3	3	4	4	4	5

Fig. 2. The DP matrix for $lcseq(p, s)$, where $p = bcdaba$ and $s = cbacbaaba$. Note $lcseq(p, s) = DP[|p|, |s|] = DP[6, 9] = 5$. The colored entries represent the partition points of the DP matrix.

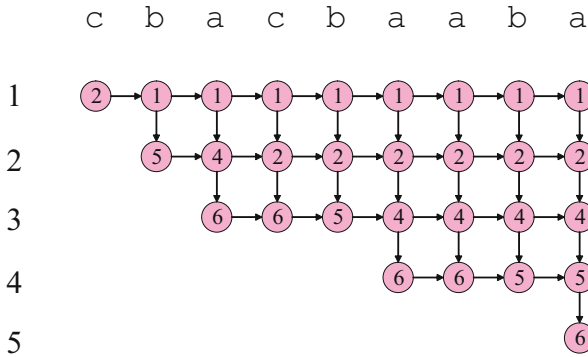


Fig. 3. Double-linked list implementation of the partition point set P for the DP matrix of Fig. 2. The cells are sorted vertically by the $lcseq$ values, and the values in the cells represent the corresponding row indices.

The above problem can be solved by computing the length of the LCSeq of p and every d -gram of $s \in S$, and by counting the number of strings s in S for which the maximum LCSeq length is not less than k . Here, we need to compute $lcseq(p, s[j..j + d - 1])$ for every position $1 \leq j \leq |s| - d + 1$ of each string $s \in S$. If naively computing the DP matrices for all pairs $p, s[j..j + d - 1]$, it takes total of $O(d|p| \cdot \|S\|)$ time and $O(d|p|)$ space. However, we can establish the following lemma that leads us to a more efficient solution.

Theorem 5. *The problem of computing local relatedness under the measure $wlcseq^d$ can be solved in $O(d\|S\|)$ time using $O(\ell d)$ extra space, where*

$$\ell = \max\{lcseq(p, s[j..j + d - 1]) \mid s \in S \text{ and } 1 \leq j \leq |s| - d + 1\}.$$

Proof. Let us concentrate on one string s in S . For any $1 \leq h \leq h' \leq |s|$, let $DP_{h'}^h$ and $P_{h'}^h$ denote the DP matrix and the corresponding partition point set for $lcseq(p, s[h..h'])$, respectively. For any $1 \leq j \leq |s| - d + 1$, let $r = j + d - 1$.

Now we are computing P_r^j for $j = |s| - d + 1, \dots, 2, 1$, in the decreasing order of j . Namely, we compute the partition point set for a *sliding window* of width d (See Fig. 4). Computing P_r^j from P_{r+1}^{j+1} is done in two rounds. Firstly, compute P_{r+1}^j from P_{r+1}^{j+1} , then compute P_r^j from P_{r+1}^j .

Let $\ell_s = \max\{lcseq(p, s[j..r]) \mid 1 \leq j \leq |s| - d + 1\}$. The second step of computing P_r^j from P_{r+1}^j can be done in $O(\ell_s)$ time based on the following observations. It follows from the property of DP matrices that for any $1 \leq i \leq |p|$ and $j \leq h \leq r$, we have $DP^j[i, h] = DP^{j+1}[i, h]$. Therefore, we can compute P_r^j from P_{r+1}^j by simply deleting the $(r + 1)$ -th column of P_{r+1}^j . The number of entries in that column is at most $lcseq(p, s[j..r + 1]) = lcseq(p, s[j..j + d]) \leq lcseq(p, s[j..j + d - 1]) + 1 \leq \ell_s + 1$. Thus it can be done in $O(\ell_s)$ time.

The first step of computing P_{r+1}^j from P_{r+1}^{j+1} , can be done efficiently based on the algorithm of Landau et al. [13]. They presented an algorithm which, given two strings p and s , computes $P_{|s|}^j$ representing $lcseq(p, s[j..|s|])$ for every $j = |s|, \dots, 2, 1$. It runs in total of $O(L|s|)$ time and space, where $L = lcseq(p, s)$, by implementing the partition point set with a double linked list (see Fig. 3). Combining this algorithm with the method mentioned in the above paragraph, we are able to compute P_{r+1}^j from P_{r+1}^{j+1} .

Now let us clarify the complexities for computing $lcseq(p, s[j..r])$ for all $j = |s| - d + 1, \dots, 1$. The space consumption is clearly $O(\ell_s d)$, since the size of P_r^j is bounded by $O(\ell_s d)$. Regarding the time complexity, as mentioned above, deleting the last column takes $O(\ell_s)$ time. When adding a new column, in the worst case, a new partition point is inserted into each of the d columns of P_{r+1}^{j+1} . Insertion of these new partition points can be done in $O(d)$ time in aggregate [13], by the double-linked list implementation. Thus, the time cost is $O(\ell_s |s| + d|s|) = O(d|s|)$, since $\ell_s \leq \min\{|p|, d\}$. In conclusion, the whole space requirement is $O(\max\{\ell_s \mid s \in S\} \cdot d) = O(\ell d)$, and the total time requirement is $O(\sum_{s \in S} (d|s|)) = O(d\|S\|)$. \square

Therefore, we can compute x_π and y_π for $wlcseq^d$ measure in $O(d(\|S\| + \|T\|))$ time using $O(\ell' d)$ space, where $\ell' = \max\{lcseq(p, s[j..j + d - 1]) \mid s \in S \cup T \text{ and } 1 \leq j \leq |s| - d + 1\}$. We remark that $\ell' \leq \min\{|p|, d\}$.

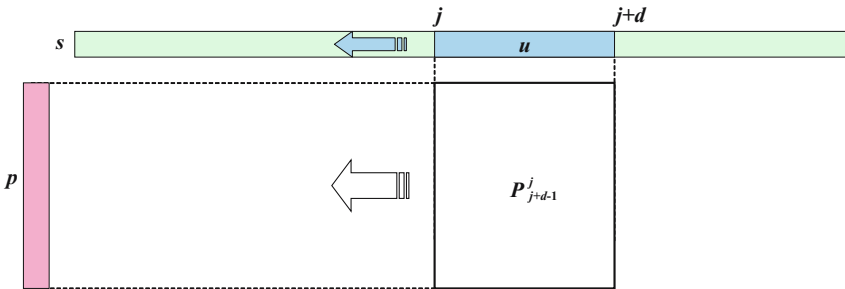


Fig. 4. Illustration for computing P_{j+d-1}^j representing $lcseq(p, s[j..j + d - 1])$ for every $1 \leq j \leq n - d + 1$, in a sliding window manner

5 Discussions

The problem of finding the optimal string classifier for two given sets S, T of strings, has been extensively studied in the recent years. The pursuit of better classification had been done mostly by enriching the ‘pattern class’ in which the best pattern is searched for. This paper suggested a new family of string classifiers based on local relatedness measures, *lcstr*, *lcseq*, and *wlcseq*^{*d*}. A big difference between the previous ones and our new family is that our string classifier does not necessarily appear as a pattern in the strings in S . Namely, the classifier can be composed of a superstring or supersequence of the strings in S . Therefore, when there do not exist good patterns which classify S and T , our new approach is expected to give us a good classifier that may lead to a more meaningful knowledge. Preliminary experiments conducted on DNA sequence data showed that the optimal string classifier discovery algorithm for the *wlcseq*^{*d*} measure runs in acceptable amount of time for modest settings of k and d . Experiments supporting practical effectiveness of our method, remain as our future work.

References

1. Arimura, H., Wataki, A., Fujino, R., Arikawa, S.: A fast algorithm for discovering optimal string patterns in large text databases. In: International Workshop on Algorithmic Learning Theory (ALT’98). Volume 1501 of LNAI., Springer-Verlag (1998) 247–261
2. Hirao, M., Inenaga, S., Shinohara, A., Takeda, M., Arikawa, S.: A practical algorithm to find the best episode patterns. In: Proc. 4th International Conference on Discovery Science (DS2001). Volume 2226 of LNAI., Springer-Verlag (2001) 435–440
3. Hirao, M., Hoshino, H., Shinohara, A., Takeda, M., Arikawa, S.: A practical algorithm to find the best subsequence patterns. Theoretical Computer Science **292**(2) (2002) 465–479
4. Shinohara, A., Takeda, M., Arikawa, S., Hirao, M., Hoshino, H., Inenaga, S.: Finding best patterns practically. In: Progress in Discovery Science. Volume 2281 of LNAI., Springer-Verlag (2002) 307–317
5. Inenaga, S., Bannai, H., Shinohara, A., Takeda, M., Arikawa, S.: Discovering best variable-length-don’t-care patterns. In: Proceedings of the 5th International Conference on Discovery Science. Volume 2534 of LNAI., Springer-Verlag (2002) 86–97
6. Shinozaki, D., Akutsu, T., Maruyama, O.: Finding optimal degenerate patterns in DNA sequences. Bioinformatics **19** (2003) ii206–ii214
7. Lanctot, J.K., Li, M., Ma, B., Wang, S., Zhang, L.: Distinguishing string selection problems. Information and Computation **185** (2003) 41–55
8. Takeda, M., Inenaga, S., Bannai, H., Shinohara, A., Arikawa, S.: Discovering most classificatory patterns for very expressive pattern classes. In: 6th International Conference on Discovery Science (DS 2003). Volume 2843 of LNCS., Springer-Verlag (2003) 486–493
9. Bannai, H., Hyvrö, H., Shinohara, A., Takeda, M., Nakai, K., Miyano, S.: Finding optimal pairs of patterns. In: 4th International Workshop on Algorithms in Bioinformatics (WABI 2004). Volume 3240 of LNBI., Springer-Verlag (2004) 450–462

10. Bannai, H., Hyyrö, H., Shinohara, A., Takeda, M., Nakai, K., Miyano, S.: An $O(N^2)$ algorithm for discovering optimal Boolean pattern pairs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* **1**(4) (2004) 159–170 (special issue for selected papers of WABI 2004).
11. Inenaga, S., Bannai, H., Hyyrö, H., Shinohara, A., Takeda, M., Nakai, K., Miyano, S.: Finding optimal pairs of cooperative and competing patterns with bounded distance. In: *The 7th International Conference on Discovery Science (DS 2004)*. Volume 3245 of *LNAI*, Springer-Verlag (2004) 32–46
12. Crochemore, M., Rytter, W.: *Text Algorithms*. Oxford University Press (1994)
13. Landau, G.M., Myers, E., Ziv-Ukelson, M.: Two algorithms for LCS consecutive suffix alignment. In: *Proc. Fifteenth Annual Combinatorial Pattern Matching Symposium (CPM2004)*. Volume 3109 of *LNCS*, Springer-Verlag (2004) 173–193