Mining Maximal Flexible Patterns in a Sequence[⋆]

Hiroki Arimura[1], Takeaki Uno[2]

[1] Graduate School of Information Science and Technology, Hokkaido University
Kita 14 Nishi 9, Sapporo 060-0814, Japan
arim@ist.hokudai.ac.jp
[2] National Institute of Informatics
2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
uno@nii.jp

Abstract. We consider the problem of enumerating all maximal flexible patterns in an input sequence database for the class of flexible patterns, where a *maximal pattern* (also called a closed pattern) is the most specific pattern among the equivalence class of patterns having the same list of occurrences in the input. Since our notion of maximal patterns is based on position occurrences, it is weaker than the traditional notion of maximal patterns based on document occurrences. Based on the framework of reverse search, we present an efficient depth-first search algorithm MaxFlex for enumerating all maximal flexible patterns in a given sequence database without duplicates in $O(||\mathcal{T}|| \times |\Sigma|)$ time per pattern and $O(||\mathcal{T}||)$ space, where $||\mathcal{T}||$ is the size of the input sequence database $\mathcal{T}$ and $|\Sigma|$ is the size of the alphabet on which the sequences are defined. This means that the enumeration problem for maximal flexible patterns is shown to be solvable in polynomial delay and polynomial space.

1 Introduction

The rapid growth of fast networks and large-scale storage technologies has led to the emergence of a new kind of massive data called *semi-structured data* emerged, which is a collection of weakly structured electronic data modeled by combinatorial structures, such as sequences, trees, and graphs. Hence, demand has arisen for efficient knowledge discovery algorithms for such semi-structured data.

In this paper, we consider the *maximal pattern discovery problem* for the class of flexible patterns in a sequence database [6, 7, 11, 12], which is also called the *closed sequence mining problem* [11, 12]. A *flexible pattern* is a sequence of

⋆ This research was partly supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Specially Promoted Research, 17002008, 2007 on "semi-structured data mining", and 18017015, 2007 on "developing high-speed high-quality algorithms for analyzing huge genome database".

constant strings separated by special gap symbols '$*$' such as `AB*B*ABC`, which means that the substring `AB` appears first in the input sequence, followed by `B` and then `ABC`. A pattern is *maximal w.r.t. position occurrences* in a sequence database if there is no properly more specific pattern that has the same set of occurrences in the input sequences. Thus, the *maximal flexible pattern discovery problem* is to enumerate all maximal flexible patterns in a given sequence database without duplicates.

For any minimum frequency threshold parameter $\sigma \geq 0$, the set of all frequent maximal patterns $\mathcal{M}_\sigma$ in input sequences contains complete information on the set of all frequent patterns $\mathcal{F}_\sigma$, and furthermore, $\mathcal{M}_\sigma$ is typically much smaller than $\mathcal{F}_\sigma$ if σ is small. Thus, the solution for maximal pattern discovery has the merit of increasing both efficiency and comprehensiveness of frequent pattern mining. On the other hand, the (frequent) maximal pattern discovery problem has high computational complexity compared with frequent pattern discovery. Thus, we need a lightweight and fast mining algorithm to solve the maximal pattern problem. In terms of algorithm theory, the efficiency of such enumeration algorithms is evaluated according to the worst-case computation time per solution. Particularly, if the algorithm works in polynomial space in terms of the input size, and the maximum time required to output the next pattern after outputting the previous one, called the delay, is of polynomial order of the input size, the algorithm said to be good.

As related works, Wang and Han [12] gave an efficient maximal pattern discovery algorithm BIDE for the class $\mathcal{SP}$ of sequential episodes [5] or subsequence patterns [12], where an episode is a pattern of the form $a_1 * \cdots * a_n$ ($a_i \in \Sigma, 1 \leq i \leq n$). Arimura and Uno [4, 6] gave a polynomial delay and polynomial time algorithm MaxMotif for maximal pattern discovery for the class $\mathcal{RP}$ of rigid patterns (or motifs with wildcards) [7], where a rigid pattern is of the form $w_1 \circ \cdots \circ w_n$ ($w_i \in \Sigma^*, 1 \leq i \leq n$) for a single symbol wildcard $\circ$. However, no polynomial space and polynomial delay algorithm, or even no output polynomial time algorithm, has been known for the maximal pattern discovery problem for the class $\mathcal{FP}$ of flexible patterns. Here maximal patterns in $\mathcal{FP}$ are the patterns which are maximal among the patterns appearing at the same locations (or positions) of the given sequence database.

As a main result of this paper, for the class $\mathcal{FP}$ of flexible patterns, we present an efficient depth-first search algorithm MaxFlex that enumerates all maximal patterns $P \in \mathcal{FP}$ in a given sequence database $\mathcal{T}$ without duplicates in $O(|\Sigma| \times ||\mathcal{T}||)$ time per maximal pattern using $O(||\mathcal{T}||)$ space, where $||\mathcal{T}||$ is the size of $\mathcal{T}$, and $|\Sigma|$ is the size of the alphabet. A key to the algorithm is a depth-first search tree built on maximal patterns based on the reverse search framework [1]. Besides this, we discuss how to implement an efficient location list computation and maximality test. As a corollary, we show that the maximal pattern discovery problem for the class $\mathcal{FP}$ of flexible patterns is polynomial space and polynomial delay solvable. This result properly generalizes the output-polynomial complexity of the class $\mathcal{SP}$ of subsequence patterns [12].

The organization of this paper is as follows. In Section 2, we introduce the class $\mathcal{FP}$ of flexible patterns and define our data mining problem. We show an adjacency relation between maximal flexible patterns that implicitly induces a tree-shaped search route in Section 3, and describe algorithm MaxFlex that performs a depth-first search on the search route in Section 4. We conclude this paper in Section 5.

2 Preliminaries

Let Σ be an alphabet of symbols. We denote the set of all possibly empty strings and the set of all non-empty finite strings over Σ by Σ^* and $\Sigma^+ = \Sigma^* - \{\varepsilon\}$, respectively. Let $s = a_1 \cdots a_n \in \Sigma^*$ be a string over Σ of length n. We denote the length of s by $|s|$, i.e., $|s| = n$. The string with no symbol is called an *empty string*, and it is denoted by ε. For any $1 \leq i \leq j \leq n$, we denote the *i-th symbol* of s by $s[i] = a_i$. For i and j such that $1 \leq i \leq j \leq |s|$, the string $a_i \cdots a_j$ is called a *substring* of s, and denoted by $s[i..j]$. We say that a substring v *occurs in s at position i* iff $v = s[i..j]$. For two strings $s = a_1 \cdots a_n$ and $t = b_1 \cdots b_n$, the *concatenation* of t to s is the string $s = a_1 \cdots a_n b_1 \cdots b_n$, and denoted by $s \bullet t$ or simply st. A string u is called a *prefix* of s if $s = uv$ holds for some string v, and is called a *suffix* of s if $s = vu$ holds for some v. For a set $\mathcal{S}$ of strings, we denote the *cardinality* of S by $|S|$. The sum of the string lengths in $\mathcal{S}$ is called the *total size* of $\mathcal{S}$ and denoted by $||\mathcal{S}||$.

2.1 Patterns and their location lists

We introduce the class $\mathcal{FP}$ of flexible patterns [6], also known as erasing regular patterns. Let $\Sigma = \{a, b, c, \ldots\}$ be a finite alphabet of *constant symbols*. A *gap* or a *variable-length don't care*, (*VLDC*) is a special symbol $* \notin \Sigma$, which represents an arbitrarily long possibly-empty finite string in Σ^*. A *constant string* is a string composed only of constant symbols. A *flexible pattern* (*pattern*, for short) over Σ is a sequence $P = w_0 * w_1 * \cdots * w_d$ of non-empty constant strings separated by gap symbols, where each constant string $w_i \in \Sigma^+$, $(0 \leq i \leq d, d \geq 0)$ is called a *segment* of P. w_0 is called the *first segment* of P and denoted by $seg_0(P)$. $w_1 * \cdots * w_d$ is called the *segment suffix* of P, and denoted by $sfx_0(P)$. A flexible pattern is called an *erasing regular pattern* in the field of machine learning (Shinohara [10]) and a *VLDC pattern* in the field of pattern matching. The size of P is $|P| = \sum_i^d |w_i|$. For every $d \geq 0$, a d-pattern is a pattern with $d+1$ segments. We denote the class of flexible patterns over Σ by $\mathcal{FP}$. Clearly, $\Sigma^+ \subseteq \mathcal{FP}$.

Example 1. Let $\Sigma = \{A, B, C\}$. Then, $P_0 = AB$, $P_1 = A{*}B$, $P_2 = AB{*}A{*}ABC$ are flexible patterns.

Let $P = w_0 {*} \cdots {*} w_d \in \mathcal{FP}, w_i \in \Sigma^+$ be a d-pattern ($d \geq 0$). A *substitution* for P is a d-tuple $\theta = (u_1, \ldots, u_d) \in \mathcal{FP}^d$ of non-empty constant strings. We define the application of θ to P, denoted by $P\theta$, as the string $P\theta = w_0 u_1 w_1 u_2 \cdots u_d w_d \in \mathcal{FP}$, where the i-th occurrence of the variable $*$ is replaced with the i-th string u_i

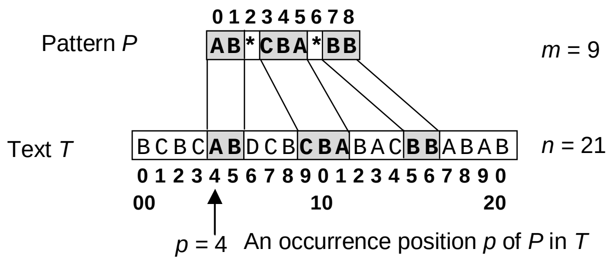

Fig. 1. A flexible pattern P and its position in a constant string T.

for every $i = 1, \ldots, d$. The string $P\theta$ is said to be an *instance* of P by substitution θ.

We define a binary relation $\sqsubseteq$ over $\mathcal{P}$, called the *specifity relation*, as follows [2, 6, 10]. A *position* in a flexible pattern P is a number $p \in \{1, \ldots, |P|\}$. Here each position i means the position of ith constant symbol.

For flexible patterns $P, Q \in \mathcal{FP}$, we say P *occurs in* Q if there exists a substitution θ such that $\alpha(P\theta)\beta = Q$ holds for some $\alpha, \beta \in \mathcal{FP}$. We say P occurs in Q at position $|\alpha| + 1$. Here $|\alpha|$ also means the number of constant symbols in α. The *location list* for P in Q, denoted by $LO(P, Q)$ is the set of all possible positions of pattern P in Q.

If P occurs in Q we say that either Q is *more specific than* P or P is *more general than* Q, and write $P \sqsubseteq Q$. If both $P \sqsubseteq Q$ and $Q \not\sqsubseteq P$ hold, we say Q is *properly more specific than* P and write $P \sqsubset Q$. Note that $P = Q$ holds if and only if $P \sqsubseteq Q$ and $Q \sqsubseteq P$ hold.

Lemma 1. *$(\mathcal{FP}, \sqsubseteq)$ is a partial ordering with the smallest element ε.*

Example 2. For patterns $P_3 = A * BA * B$ and $P_4 = BAB * BA * AB * B$, we can see that $P_3 \sqsubseteq P_4$ by the embedding $B\underline{A}B * \underline{BA} * A\underline{B} * B$, where the image of P_4 is indicated by underlines. Formally, $P_3 \sqsubseteq P_4$ holds because P_3 has an instance $P_3\theta = AB * BA * AB$ as a substring of P_4 for the substitution $\theta = (B*, *A)$. Clearly, $P_3 \sqsubset P_4$ since $P_3 \sqsubseteq P_4$ but $P_4 \not\sqsubseteq P_3$.

A *sequence database* $\mathcal{T} = \{T_1, \ldots, T_m\}$ is a set of constant strings $T_i \in \Sigma^*$. We denote the number of strings by m. The sum of the sizes of $T_1, \ldots, T_m$ is called the *size* of $\mathcal{T}$ and denoted by $||\mathcal{T}||$, i.e., $||\mathcal{T}|| = \sum_{i=1}^{m} |T_i|$. In what follows, we fix the sequence database unless stated otherwise. For a flexible pattern P and a set of flexible patterns $\mathcal{T}$, which can be a sequence database, the *location set* for P in $\mathcal{T}$, denoted by $LO(P, \mathcal{T})$ is the set of location lists $LO(P, T_i)$ in all $T_i \in \mathcal{T}$.

For flexible patterns P and Q, the largest position in $LO(P, Q)$ is called the *rightmost position* of P in Q, and denoted by $p_{max}(P, Q)$. If $LO(P, Q)$ is empty, we define $p_{max}(P, Q)$ as $-\infty$. If P is an empty sequence, we define $p_{max}(P, Q)$ as $|Q| + 1$.

Lemma 2. *For any $P, T \in \mathcal{FP}$, $LO(P, T) = \{p \mid p \in LO(seg_0(P), T), p + |seg_0(P)| \leq p_{max}(sfx_0(P), T)\}$.*

The *frequency $frq(P, \mathcal{T})$* of a flexible pattern P in a sequence database $\mathcal{T}$ is $|LO(P, \mathcal{T})| = |\{LO(P, T_i) \mid T_i \in \mathcal{T}, LO(P, T_i) \neq \emptyset\}|$. A *minimum support threshold* is a non-negative integer $0 < \sigma \leq n$. A flexible pattern P is *σ-frequent* in $\mathcal{T}$ if its frequency in $\mathcal{T}$ is no less than σ, i.e., $frq(P, \mathcal{T}) \geq \sigma$.

2.2 Maximal pattern enumeration problem

Definition. 1 A flexible pattern P is *maximal* in $\mathcal{T}$ if there is no proper specialization Q of P that has the same location list, that is, there is no $Q \in \mathcal{FP}$ such that $P \sqsubset Q$ and $LO(P, \mathcal{T}) = LO(Q, \mathcal{T})$.

We see that a pattern $P \in \mathcal{FP}$ is maximal iff P is a maximal element w.r.t. $\sqsubseteq$ in the equivalence class $[P]_{\mathcal{T}} = \{Q \in \mathcal{FP} | P \equiv_{\mathcal{T}} Q\}$ under the equivalence relation $\equiv_{\mathcal{T}}$ over $\mathcal{FP}$ defined by $P \equiv_{\mathcal{T}} Q \Leftrightarrow LO(P, \mathcal{T}) = LO(Q, \mathcal{T})$.

Lemma 3. *The maximal patterns in each equivalence class $[P]_{\mathcal{T}}$ are not unique in general.*

We denote the family of the maximal patterns in $\mathcal{T}$ by $\mathcal{M}$, and the family of σ-frequent flexible patterns by $\mathcal{F}_\sigma$. Let $\mathcal{M}_\sigma = \mathcal{F}_\sigma \cap \mathcal{M}$ be the family of σ-frequent maximal patterns. It is easy to see that the number of frequent flexible patterns in $\mathcal{T}$ can be exponential in $||\mathcal{T}||$ and the same is true for $\mathcal{M}_\sigma$.

Lemma 4. *There is an infinite sequence $\mathcal{T}_1, \mathcal{T}_2, \ldots$ of sequence databases such that the number of maximal flexible patterns in $\mathcal{T}_i$ is exponential in $||\mathcal{T}_i||$.*

Now, we state our data mining problem as follows.

Position Maximal Flexible Pattern Enumeration Problem:
Input: sequence database $\mathcal{T}$ over an alphabet Σ, a minimum support threshold σ
Output: all maximal σ-frequent flexible patterns in $\mathcal{M}_\sigma$ in $\mathcal{T}$ without duplicates

Our goal is to develop an efficient enumeration algorithm for this problem.

3 Tree-shaped Search Route for Maximal Flexible Patterns

In this section, we introduce a tree-shaped search route $\mathcal{R}$ spanning all elements in $\mathcal{M}$. In section 4, we give a memory efficient algorithm for enumerating all maximal flexible patterns based on the depth-first search over $\mathcal{R}$. Our strategy is as follows: First we define a binary relation between maximal patterns, called the *parent function*, which indicates a directed edge from a child to its parent. The parent function induces an adjacency relation on $\mathcal{M}$, whose form is a spanning tree.

We start with several technical lemmas.

Definition. 2 A flexible pattern Q is said to be a *prefix specialization* of another flexible pattern P if $1 \in LO(P, Q)$. If Q is a specialization of P but not a prefix specialization, Q is said to be a *non-prefix specialization* of P.

The following two lemmas are essential for flexible patterns. The first one says that a limited kind of monotonicity holds for flexible patterns. The proof is obvious from the transitivity of the specialization relation.

Lemma 5. *For any $P, Q, T \in \mathcal{FP}$ if P is a prefix specialization of Q, then $LO(P, T) \supseteq LO(Q, T)$.*

The second lemma gives us a technical property that a non-prefix specialization of P always has a location list different from that of P. Thus, attaching a new symbol to the left of a flexible pattern never preserves its location list.

Lemma 6. *Let $T, P, Q \in \mathcal{FP}$ such that $P \sqsubset Q \sqsubseteq T$. Then, if Q is a non-prefix specialization of P then $LO(P, T) \neq LO(Q, T)$.*

Proof. Since Q is a specialization of P but not a prefix specialization, $LO(P, Q)$ includes a position $p > 1$. This means that P occurs in $T[p_{max}(Q, T)+p-1..|T|]$, thus $p_{max}(P, T) \neq p_{max}(Q, T)$. This implies $LO(P, T) \neq LO(Q, T)$. □

Corollary 1. *A flexible pattern $Q \in \mathcal{FP}$ is maximal if and only if none of its prefix specializations has a location list equal to Q.*

Definition. 3 The *parent* $\mathcal{P}(P)$ of a flexible pattern P is the flexible pattern obtained from P by removing its first symbol and $*$ if the following operator is $*$.

Lemma 7. *For any non-empty flexible pattern $Q \in \mathcal{FP}$, its parent is always defined and unique.*

Lemma 8. *For any non-empty flexible pattern P and constant symbol a, $LO(a \bullet P, T) = \{p \in LO(a, T) \mid p{+}1 \in LO(P, T)\}$, and $LO(a{*}P, T) = \{p \in LO(a, T) \mid p < p_{max}(P, T)\}$.*

Corollary 2. *For any flexible pattern P, $frq(P, \mathcal{T}) \leq frq(\mathcal{P}(P), \mathcal{T})$.*

The proof of the above lemma is omitted, but it is not difficult. A *root pattern* in T is a maximal pattern P such that $LO(P, \mathcal{T}) = LO(\varepsilon, \mathcal{T})$. The root pattern is either ε or a symbol a if all symbols in any sequence $T \in \mathcal{T}$ are a. Now, we are ready for the main result of this section.

Theorem 1 (reverse search property of $\mathcal{M}$). *Let $Q \in \mathcal{M}$ be a maximal flexible pattern in $\mathcal{T}$ that is not a root pattern. Then, $\mathcal{P}(Q)$ is also a maximal flexible pattern in $\mathcal{T}$, that is, if $Q \in \mathcal{M}$ then $\mathcal{P}(Q) \in \mathcal{M}$ holds.*

Proof. Let Q be a maximal flexible pattern that is not a root pattern. To prove the theorem by contradiction, we suppose that there is a proper specialization P' of $\mathcal{P}(Q)$ such that $LO(\mathcal{P}(Q), \mathcal{T}) = LO(P', \mathcal{T})$. If P' is not a prefix specialization of $\mathcal{P}(Q)$, then from Lemma 6, we see that $LO(P, \mathcal{T}) \neq LO(P', \mathcal{T})$. Thus, we consider the case that P' is a prefix specialization of $\mathcal{P}(Q)$.

From the definition of the parent, $Q = a \circledcirc \mathcal{P}(Q)$ for some $a \in \Sigma$ and $\circledcirc \in \{\bullet, *\}$. Let $Q' = a \circledcirc P' \in \mathcal{FP}$. Since P' is a proper prefix specialization of $\mathcal{P}(Q)$, Q' is also a proper prefix specialization of Q. Let T be a sequence in $\mathcal{T}$ and p be a position $p \in LO(Q, T)$. If $\circledcirc = \bullet$, then $p + 1 \in LO(\mathcal{P}(Q), T)$ and thus also $p + 1 \in LO(P', T)$. Thus we have $p \in LO(Q', T)$. If $\circledcirc = *$, then $p < p_{max}(\mathcal{P}(Q), T)$ thus also $p < p_{max}(P', T)$. Thus, we have $p \in LO(Q', T)$. In both cases, we have $LO(Q, T) \subseteq LO(Q', T)$, thus $LO(Q, T) = LO(Q', T)$. This immediately implies that $LO(Q, \mathcal{T}) = LO(Q', \mathcal{T})$, contradiction. □

Definition. 4 A *search route* for $\mathcal{M}$ w.r.t. $\mathcal{P}$ is a directed graph $\mathcal{R} = (\mathcal{M}, \mathcal{P}, \perp)$ with root, where $\mathcal{M}$ is the set of nodes, i.e., the set of all maximal flexible patterns in $\mathcal{T}$, $\mathcal{P}$ is the set of *reverse edge* such that $(P, Q) \in \mathcal{P}$ iff $P = \mathcal{P}(Q)$ holds, and $\perp \in \mathcal{M}$ is the root pattern in $\mathcal{T}$.

Since each non-root node has its parent in $\mathcal{M}$ by Theorem 1 and $|\mathcal{P}(P)| < |P|$, the search route $\mathcal{R}$ is actually a directed tree with reverse edges. Therefore, we have the following corollary.

Corollary 3. *For any sequence database $\mathcal{T}$, $\mathcal{R} = (\mathcal{M}, \mathcal{E}, \perp)$ forms a rooted spanning tree with the root $\perp$.*

We have the following lemma on the shape of $\mathcal{T}$.

Lemma 9. *Let $P \in \mathcal{M}$ be any maximal pattern in $\mathcal{T}$ and $m = |P|$. Then,*

(i) the depth of P in $\mathcal{T}$ (the length of the unique path from the root to P) is at most m.
(ii) the branching of P in $\mathcal{T}$ (the number of the children for P) is at most $2|\Sigma|$.

4 A Polynomial Time and Polynomial Delay Algorithm

In Figure 2 shows a polynomial space and polynomial delay enumeration algorithm MaxFlex for maximal flexible patterns. This algorithm starts from the bottom pattern $\perp$ and searches from smaller to larger all maximal patterns in a depth-first search manner over the search route $\mathcal{R}$.

However, since the search route $\mathcal{R}$ is defined by the reverse edges from children to their parents, it is not an easy task to traverse the edges. We firstly explain how to compute all children of given parent pattern $P \in \mathcal{M}$. The following lemma ensures that any child can be obtained by attaching a new symbol with an operator to the left of P.

Lemma 10. *For any maximal flexible patterns $P, Q \in \mathcal{M}$, $P = \mathcal{P}(Q)$ if and only if Q is maximal, and $Q = a \circledcirc P$ holds for some constant symbol $a \in \Sigma$ and $\circledcirc \in \{\bullet, *\}$.*

```
Algorithm MaxFlex($\Sigma$, $\mathcal{T}$, $\sigma$):
input: sequence database $\mathcal{T}$ on an alphabet $\Sigma$ s.t. any $T \in \mathcal{T}$ is in $\Sigma^*$, minimum support threshold $\sigma$
output: All maximal patterns in $\mathcal{M}_\sigma$
1  compute the root pattern $\bot$ //the maximal pattern in $\mathcal{T}$ equivalent to $\varepsilon$
2  ExpandMaxFlex($\bot, LO(\bot, \mathcal{T}), \mathcal{T}, \sigma$);

Procedure ExpandMaxFlex($P, LO(P, \mathcal{T}), \mathcal{T}, \sigma$):
input: maximal pattern $P$, location list $LO(P, \mathcal{T})$, sequence database $\mathcal{T}$, minimum support threshold $\sigma$
output: all maximal patterns in $\mathcal{M}_\sigma$ that are descendants of $P$
1  if $frq(P, \mathcal{T}) < \sigma$ then return
2  if $P$ is not maximal in $\mathcal{T}$ then return
3  output $P$
4  foreach pair of $a \in \Sigma$ and $\circledcirc \in \{\bullet, *\}$ do begin
5      ExpandMaxFlex($a \circledcirc P, LO(a \circledcirc P, \mathcal{T}), \mathcal{T}$)
6  end
```

Fig. 2. An algorithm MaxFlex for enumerating all maximal flexible patterns in a sequence database.

Since $\mathcal{P}(Q)$ is defined for any non-empty pattern Q, we know from Lemma 10 that any flexible pattern can be obtained from $\bot$ by a finite number of applications of the operator $\bullet$ or $*$. Therefore, Theorem 1 ensures that we can correctly prune all descendants of the current pattern P if it is no longer maximal in depth-first search of $\mathcal{M}$. Moreover, if $frq(P, \mathcal{T}) < \sigma$, no descendant of P is frequent. Thus, we can also prune the descendants.

Secondly, we discuss the computation time of the algorithm. The bottleneck of the computation is the check of the maximality of the current pattern P, thus we need an efficient way to test it.

The notion of the refinement operator was introduced by Shapiro [8]. The refinement operator for flexible patterns in $\mathcal{FP}$, under the name of erasing regular patterns, was introduced by Shinohara [10]. The following version is due to [2, 3].

Definition. 5 Let $P \in \mathcal{FP}, P = w_0 * \cdots * w_d$ be any flexible pattern. Then, we define the set $\rho(P) \subseteq \mathcal{FP}$ as the set of all patterns $Q \in \mathcal{FP}$, called *basic refinements* of P if Q is obtained from P by one of the following operations:

(1) replace w_i by $w_i * a$ for some $a \in \Sigma$ and $0 \leq i \leq d$.
(2) replace $w_i * w_{i+1}$ by $w_i w_{i+1}$ for some $0 \leq i \leq d-1$.

Lemma 11. *A flexible pattern P is maximal in $\mathcal{T}$ if and only if there is no basic refinement $Q \in \rho(P)$ such that $LO(P, \mathcal{T}) = LO(Q, \mathcal{T})$.*

Suppose that P and T are flexible patterns and the segments of a flexible pattern P are $w_0, \ldots, w_d$. Let $left(P, T, i), 0 \leq i \leq d$ be the minimum position p

such that $T[1..p]$ is a specification of $w_0 * \cdots * w_i$. If T is not a specification of P, $left(P, T, i)$ is defined by $+\infty$. If P is an empty sequence, we define $left(P, T, i) = 0$.

For any segment w and position p in T, let $succ(w, T, p)$ be the smallest position q such that $q \geq p$ and $q \in LO(w, T)$. For any string w, computing $succ(w, T, p)$ for all pairs of $T \in \mathcal{T}$ and $p \in LO(w, T)$ can be done in $O(||\mathcal{T}||)$ time.

Lemma 12. *There is a basic refinement obtained by replacing w_i by $w_i * a$ having the same location list as P if and only if there is a common symbol in $T[left(P, T, i) + 1..p_{max}(w_{i+1} * \cdots * w_d, T) - 1]$ for all $T \in \mathcal{T}$.*

Proof. The if part of the statement is obvious. We check the "only if" part, by showing that there is a common symbol a. Suppose that a basic refinement obtained by replacing w_i by $w_i * a$ has the same location list as P. Then for any $T \in \mathcal{T}$, there is a position q such that $T[q] = a$, $T[1..q-1]$ is a specification of $w_0 * \cdots * w_i$, and $T[q+1..|T|]$ is a specification of $w_{i+1} * \cdots * w_d$. Then, from the definition of p_{max} and $left$, $left(P, T, i) < q < p_{max}(w_{i+1} * \cdots * w_d, T)$. This states the statement of the lemma. □

Similarly, we obtain the following lemma.

Lemma 13. *There is a basic refinement obtained by replacing $w_i * w_{i+1}$ by $w_i w_{i+1}$ having the same location list as P if and only if $succ(w_i w_{i+1}, T, left(P, T, i-1)) + |w_i w_{i+1}| \leq p_{max}(w_{i+2} * \cdots * w_d, T)$ for any $T \in \mathcal{T}$.*

Lemma 14. *Using $left$, $succ$, and p_{max}, we can determine whether there is a basic refinement having the same location list as P in $O(||\mathcal{T}||)$ time.*

Proof. The statement is clear for basic refinements obtained by replacing $w_i * w_{i+1}$ by $w_i w_{i+1}$. Thus, we consider basic refinements obtained by replacing w_i by $w_i * a$.

Let $Count(a, i), a \in \Sigma, 0 \leq i \leq d$ be the number of sequences $T \in \mathcal{T}$ such that $T[left(P, T, i) + 1..p_{max}(w_{i+1} * \cdots * w_d, T) - 1]$ includes a. What we have to do is to check whether $Count(a, i) = m$ holds for some $a \in \Sigma$ or not. For any i, $Count(a, i)$ for all $a \in \Sigma$ can be computed in $O(||\mathcal{T}||)$ time.

For $i > 0$, let $diff(P, T, i)$ be the set of symbols with signs, such as $+a, -a, +b, -b$, such that $+a$ is included in $diff(P, T, i)$ iff $T[left(P, T, i-1) + 1..p_{max}(w_i * \cdots * w_d, T) - 1]$ does not include a but $T[left(P, T, i) + 1..p_{max}(w_{i+1} * \cdots * w_d, T) - 1]$ includes a, and $-a$ is included in $diff(P, T, i)$ iff $T[left(P, T, i-1) + 1..p_{max}(w_i * \cdots * w_d, T) - 1]$ includes a but $T[left(P, T, i) + 1..p_{max}(w_{i+1} * \cdots * w_d, T) - 1]$ does not include a. Using $diff(P, T, i)$, $Count(a, i+1)$ for all $a \in \Sigma$ can be obtained from $Count(a, i-1)$ of all $a \in \Sigma$ in $O(\sum_{T \in \mathcal{T}} |diff(P, T, i)|)$ time. Since each $diff(P, i, T)$ can be computed in $O((left(P, T, i) - left(P, T, i-1)) + (p_{max}(w_{i+1} * \cdots * w_d, T) - p_{max}(w_i * \cdots * w_d, T)))$ time, computing $diff(P, i, T)$ for all pairs of $i, 1 \leq i \leq d$ and $T \in \mathcal{T}$ takes $O(||\mathcal{T}||)$ time. This concludes the lemma. □

Lemma 15. *Using succ, $left(P, T, i)$ for all $T \in \mathcal{T}$ can be computed in $O(||\mathcal{T}||)$ time.*

By combining the above lemmas, we get the main result of this paper, which says that MaxFlex is a memory and time efficient algorithm.

Theorem 2. *Let Σ be an alphabet, $\mathcal{T}$ a sequence database, and σ a minimum support threshold. Then, the algorithm* MaxFlex *in Fig. 2 enumerates all maximal flexible patterns $P \in \mathcal{M}_\sigma$ of $\mathcal{T}$ without duplicates in $O(|\Sigma| \times ||\mathcal{T}||)$ time per maximal flexible pattern within $O(||\mathcal{T}||d)$ space, where d is the maximum number of gaps in a flexible pattern $P \in \mathcal{M}_\sigma$.*

Proof. The correctness of the algorithm is clear, thus we discuss the complexity. From Lemmas 14 and 15, the computation time for checking the maximality of a flexible pattern can be done in $O(||\mathcal{T}||)$ time, by using p_{max} and *succ*. For the task, *succ* is needed only for segments w and consecutive segments ww' in the current pattern P. Thus, the memory complexity is $O(||\mathcal{T}||d)$.

We next show how much time we need to compute those for a flexible pattern $a \circledcirc P$ by using those for P. Actually, for any $i > 0$ and position p, $succ(w_i, T, p)$, $succ(w_i w_j, T, p)$, and $p_{max}(w_i * \cdots * w_d, T)$ are common to P and P', hence we have to compute those only for $i = 0$. Thus, it can be done in $O(||\mathcal{T}||)$ time. We have at most $2|\Sigma|$ candidates for the children of a maximal flexible pattern, thus the statement holds. □

The *delay* of an enumeration algorithm is the maximum computation time between a pair of consecutive outputs.

Corollary 4. *The maximal pattern enumeration problem for the class $\mathcal{FP}$ of flexible patterns w.r.t. position maximality is solvable in polynomial delay and polynomial space in the input size.*

5 Conclusion

In this paper, we considered the maximal pattern discovery problem for the class $\mathcal{FP}$ of flexible patterns [6], which are also called erasing regular patterns in machine learning. The motivation of this study is the potential application to the optimal pattern discovery problem in machine learning and knowledge discovery. Our main result was a polynomial space and polynomial delay algorithm for enumerating all maximal patterns appearing in a given string without duplicates in terms of position-maximality defined through the equivalence relation between the location sets. Extending this work to document-based maximal patterns and to more complex classes of flexible patterns are interesting future problems.

Acknowledgments

The authors would like to thank Tetsuji Kuboyama, Akira Ishino, Kimihito Ito, Shinichi Shimozono, Takuya Kida, Shin-ichi Minato, Ayumi Shinohara, Masayuki Takeda, Kouichi Hirata, Akihiro Yamamoto, Thomas Zeugmann, and Ken Satoh, for their valuable discussions and comments. The authors also thank the anonymous referees for their valuable comments that greatly improved the quality of this paper.

References

1. D. Avis and K. Fukuda, Reverse Search for Enumeration, *Discrete Appl. Math.*, 65, 21–46, 1996.
2. H. Arimura, R. Fujino, T. Shinohara, Protein motif discovery from positive examples by minimal multiple generalization over regular patterns, In *Proc. GIW'94*, 39-48, 1994.
3. H. Arimura, T. Shinohara, S. Otsuki, Finding minimal generalizations for unions of pattern languages and its application to inductive inference from positive data, In *Proc. STACS'94*, Springer, LNCS 775, 649–660, 1994.
4. H. Arimura, T. Uno, A polynomial space and polynomial delay algorithm for enumeration of maximal motifs in a sequence, In *Proc. ISAAC'05*, LNCS 3827, Springer, Dec. 2005.
5. H. Mannila, H. Toivonen, A. I. Verkamo, Discovery of frequent episodes in event sequences, *Data Min. Knowl. Discov.*, 1(3), 259–289, 1997.
6. L. Parida, I. Rigoutsos, *et al.*, Pattern discovery on character sets and real-valued data: linear-bound on irredandant motifs and efficient polynomial time algorithms, In *Proc. SODA'00*, SIAM-ACM, 2000.
7. N. Pisanti, M. Crochemore, R. Gross, M.-F. Sagot, A basis of tiling motifs for generating repeated patterns and its complexity of higher quorum, In *Proc. MFCS'03*, Springer, 2003.
8. E. Y. Shapiro, *Algorithmic Program Debugging*, MIT Press, 1982.
9. S. Shimozono, H. Arimura, S. Arikawa, Efficient discovery of optimal word-association patterns in large text databases, *New Generation Comput.*, 18(1), 49–60, 2000.
10. T. Shinohara, Polynomial time inference of extended regular pattern Languages. *Proc. RIMS Symp. on Software Sci. & Eng.*, 115–127, 1982.
11. X. Yan and J. Han, R. Afshar, CloSpan: mining closed sequential patterns in large databases, In Proc. SDM 2003, SIAM, 2003.
12. J. Wang and J. Han, BIDE: efficient mining of frequent closed sequences, In Proc. ICDE'04, 2004.