An Efficient Polynomial Space and Polynomial Delay Algorithm for Enumeration of Maximal Motifs in a Sequence

Hiroki Arimura[1]* and Takeaki Uno[2]

[1] Hokkaido University, Kita 14-jo, Nishi 9-chome, Sapporo 060-0814, JAPAN
arim@ist.hokudai.ac.jp
[2] National Institute of Informatics, Tokyo 101–8430, JAPAN
uno@nii.jp

Abstract. In this paper, we consider the problem of enumerating all maximal motifs in an input string for the class of repeated motifs with wild cards. A maximal motif is such a representative motif that is not properly contained in any larger motifs with the same location lists. Although the enumeration problem for maximal motifs with wild cards has been studied in (Parida et al., CPM'01), (Pisanti et al.,MFCS'03) and (Pelfrene et al., CPM'03), its output-polynomial time computability has been still open. The main result of this paper is a polynomial space polynomial delay algorithm for the maximal motif enumeration problem for the repeated motifs with wild cards. This algorithm enumerates all maximal motifs in an input string of length n in $O(n^3)$ time per motif with $O(n)$ space, in particular $O(n^3)$ delay. The key of the algorithm is depth-first search on a tree-shaped search route over all maximal motifs based on a technique called prefix-preserving closure extension. We also show an exponential lower bound and a succinctness result on the number of maximal motifs, which indicate the limit of a straightforward approach. The results of the computational experiments show that our algorithm can be applicable to huge string data such as genome data in practice, and does not take large additional computational cost compared to usual frequent motif mining algorithms.

1 Introduction

Pattern discovery is to find all patterns within a class of combinatorial patterns that appear in an input data satisfying a specified constraint, and is a central task in computational biology, temporal sequence analysis, sequence and text mining [3]. We consider the pattern discovery problem for the class of patterns with wild cards, which are strings consisting of constant symbols (called *solid letters*) drawn from an alphabet and variables '$\circ$' (called *wildcards*) that matches any symbol [9, 13]. For instance, $\texttt{B} \circ \texttt{AB}$ and $\texttt{B} \circ \texttt{AB} \circ \circ \texttt{B}$ are examples of patterns.

* This work is done during the first author's visit in LIRIS, University Claude-Bernard Lyon 1, France.

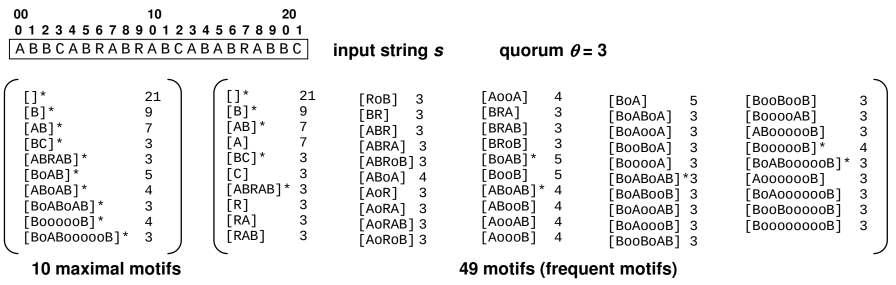

Fig. 1. Examples of maximal motifs (left) and motifs (right) for an input string s and a quorum θ, where * indicates a maximal motif and the number associated to each motif indicates its frequency. These 10 maximal motifs are representatives containing the whole information on the occurrences of all motifs in s.

A pattern p is said to appear at the jth position of a string s if any ith letter of p matches to $(j+i-1)$th letter of s, i.e., they are the same, or the ith letter is $\circ$. Given a positive integer θ called *quorum* and for an input string s, a frequent motif (or *motif*, for short) in s is a pattern that appears at least θ times in s.

Frequent motif discovery has a drawback that a huge number of motifs are often generated from an input string without conveying any useful information. To overcome this problem, we focus on discovery of *maximal motifs* [9, 12, 13]. The semantics of a pattern x is given by the location list $\mathcal{L}(x)$ consisting of the positions in an input string s at which the pattern occurs. A motif is said to be *maximal* if it is not properly contained by other motifs with the equivalent location lists allowing position shift. For example, we show in Figure 1 all maximal motifs and all motifs on input sting $s = \mathtt{ABBCABRABRABCABABRABBC}$ for quorum $\theta = 3$. Then, the pattern $x_1 = \mathtt{R} \circ \mathtt{B}$ is a motif having location list $\mathcal{L}(x_1) = \{6, 9, 17\}$ in s but not a maximal one since there is another motif $x_2 = \mathtt{ABRAB}$ which contains x_1 and whose location list $\mathcal{L}(x_2) = \{4, 7, 15\}$ is obtained from the location list $\mathcal{L}(x_1)$ by shifting leftward with two. In this example, we can also observe that there are only 10 maximal motifs among 50 motifs. In general, the number of maximal motifs can be exponentially smaller than the number of motifs, while the former can be exponential in the input size.

In this paper, we study the problem of enumerating all maximal motifs in an input string of length n. In particular, from a practical viewpoint, we are interested in those algorithms that have small space and delay complexities independent from the output size in addition to polynomial amortized time per maximal motif. Although an output-polynomial time algorithm for this problem is known[**?**], its time complexity is square of the output size, and the space complexity is not polynomial in the input size. The computability for maximal motifs discovery with polynomial space and delay has been still open.

We first show an exponential lower bound and a succinctness result on the number of maximal motifs, which show the limit of straightforward approaches. Next, by examining the previous approaches [9, 13, 12], we present a simple output-polynomial time algorithm for maximal motif enumeration by breadth-first search, which possibly requires exponential space and delay. Then, we present an efficient algorithm that enumerates all maximal motifs in an input string of length n with $O(n^3)$ time per motif with $O(n)$ space and $O(n^3)$ delay. A key of the algorithm is depth-first search on a *tree-shaped search route* for all maximal motifs build by *prefix-preserving closure extension*, which enable us to enumerate all maximal motifs without storing discovered motifs which are used for detecting duplications and maximality tests. To the best of our knowledge, this is the first result on a polynomial space polynomial delay algorithm for maximal pattern discovery for sequences.

The organization of this paper is as follows. In Section 2, we give definitions and basic results. Section 3 gives lower bounds on the number of maximal motifs. Section 2.2 prepares tools for studying maximal motifs and Section 4 reviews the previous results. In Section 5, we present our algorithm MaxMotif that enumerates all maximal motifs. Section 6 shows the result of computational experiments. In Section 7, we conclude this paper.

2 Preliminaries

2.1 Maximal Motifs

We briefly introduce basic definitions and results on maximal pattern enumeration according to [13, 12]. For definitions not found here, see text books on string algorithms, e.g., [6, 8]. Given an alphabet Δ, a *string* of length $n \geq 0$ is a consecutive sequence of letters $s = a[0] \cdots a[n-1] \in \Delta^*$, where $a[i] \in \Delta$, $0 \leq i \leq n-1$. For every $0 \leq i \leq j \leq n-1$, $s[i..j]$ denotes the substring $a_i a_{i+1} \cdots a_j$. Δ^* denotes the set of all possibly empty strings over Δ, and ε denotes the empty string. If $s = uvw$ for some $u, v, w \in \Delta^*$, then we say that u is a *prefix* and w is a *suffix* of s. For a set $S \subseteq \Delta^*$ of strings, we denote by $|S|$ the cardinality of S and by $||S|| = \sum_{s \in S} |s|$ the total length of S.

Let Σ be an alphabet of *solid characters* (or *constant letters*). Let $\circ \notin \Sigma$ be a distinguished letter not belonging to Σ, called the *wild card* (or *don't care*). A wild card $\circ$ matches any solid character $c \in \Sigma$ and also matches $\circ$ itself. An *input string* is a string $s = s[1] \cdots s[n] \in \Sigma^*$ consisting of solid characters of length $n \geq 0$.

Definition 1 (pattern [9, 13, 12]). *A* pattern *over Σ is a string x in $\Sigma \cup (\Sigma \cdot (\Sigma \cup \{\circ\})^* \cdot \Sigma)$ that starts and ends with a solid character, or an empty string ε.*

We denote the class of patterns by $\mathcal{P} = \{\varepsilon\} \cup \Sigma \cup (\Sigma \cdot (\Sigma \cup \{\circ\})^* \cdot \Sigma)$. For example, $\mathtt{ABC}$ and $\mathtt{B} \circ \mathtt{C}$ are patterns, but $\circ\mathtt{BC}$ and $\circ\circ\mathtt{B}\circ$ are not. Note that ε is a pattern in our definition. We define a binary relation $\preceq$ over letters and patterns, called the *specificity relation*.[3] For letters $a, b \in \Sigma \cup \{\circ\}$, we define $a \preceq b$ if either $a = b$ or $a = \circ$ holds. For patterns x and y, We say that x *occurs at position p in y* if there exists some index $0 \leq p \leq |y| - |x|$ such that for every index $0 \leq i \leq |x| - 1$, $x[i] \preceq y[p + i]$ holds. Then, we also say that p is an *occurrence* of x in y, and that x *matches* the substring $y[p..p + |x| - 1]$.

Example 1. Pattern $x = \mathtt{B} \circ \mathtt{D}$ occurs in pattern $y = \mathtt{AB} \circ \mathtt{DA}$ at position 1, and x occurs three times in string $s = \mathtt{EABCDABEDBCDE}$ at positions $2, 6$ and 9.

We extend the binary relation $\preceq$ from letters to patterns as follows. Let x and y be patterns in $\mathcal{P}$. If x occurs at some position p in y, then we define $x \preceq y$ and say that either x is *contained by* y or y is *more specific to* x. For any pattern x, we define $\varepsilon \preceq x$. If $x \preceq y$ but $y \not\preceq x$, then we define $x \prec y$ and say that either x is *properly contained by* y or y is *properly more specific to* x. We can see that if $x \preceq y$ and $y \preceq x$ hold, then x and y are identical each other. Furthermore, $\preceq$ is a partial order over $\mathcal{P}$.

Definition 2 (location list [9, 13, 12]). *For an input string $s \in \Sigma^*$ of length $n \geq 0$, the* location list *of pattern x is the set $\mathcal{L}(x) \subseteq \{0, \ldots, n-1\}$ of all the positions in s at which x occurs. The* frequency *of x on s is $|\mathcal{L}(x)|$.*

Example 2. The location list of pattern $x = \mathtt{B} \circ \mathtt{D}$ in the input string $s = \mathtt{EA}\underline{\mathtt{BCD}}\mathtt{A}\underline{\mathtt{BED}}\underline{\mathtt{BCD}}\mathtt{E}$ is $\mathcal{L}(x) = \{2, 6, 9\}$.

A *quorum* (or *minimum frequency threshold*) is any positive number $\theta \geq 1$.

Definition 3 (motif [9, 13, 12]). *Let $\theta \geq 1$ be a quorum. We say that pattern x is a θ-*motif *(or* motif*, for short) in s if $|\mathcal{L}(x)| \geq \theta$ holds.*

Let $\mathcal{L}$ be any location list and d be any integer. Then, we define the *shift of $\mathcal{L}$ with displacement d* by $\mathcal{L} + d = \{\, \ell + d \mid \ell \in \mathcal{L} \,\}$. We write $\mathcal{L} - x$ to represent the set $\mathcal{L} + y$ with $y = -x$.

Lemma 1 ([9, 12, 13]). *Let $\theta \geq 1$ and $x, y \in \mathcal{P}$ be any motifs. If $x \preceq y$ then $\mathcal{L}(x) \supseteq \mathcal{L}(y) + d$ for some integer $d \geq 0$.*

The converse of this lemma does not hold for motifs in general.

Example 3. Let $s = \mathtt{EABCDABEDABCDE}$ over $\Sigma = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}\}$ be an input string. Consider motifs $x = \mathtt{AB} \circ \mathtt{D}$ with location list $\mathcal{L}(x) = \{1, 5, 8\}$, $y = \mathtt{B} \circ \mathtt{D}$ with $\mathcal{L}(y) = \{2, 6, 9\}$, and $z = \mathtt{D}$ with $\mathcal{L}(z) = \{4, 8, 11\}$. We can see that $z \prec y \prec x$ holds and x, y, z are equivalent each other. For instance, $\mathcal{L}(z) = \mathcal{L}(x) + d$ with the displacement $d = 3$. Then, x is maximal in s, but y and z are not.

[3] The binary relation $\preceq$ is also called the *generalization relation* or the *subsumption relation* in artificial intelligence and data mining.

Definition 4 (maximal motif [9]). *Let $\theta \geq 1$ be a quorum. A motif x is* maximal *in s if for any motif y that properly contains x, there is no integer d such that $\mathcal{L}(y) = \mathcal{L}(x) + d$.*

In other words, θ-motif x is maximal in s iff there exists no θ-motif in s properly containing x that is equivalent to x under shift-invariance. Let θ be a quorum. We denote by $\mathcal{F}$ and $\mathcal{M}$ the sets of all (frequent) motifs and all maximal motifs, respectively. Clearly, $\mathcal{M} \subseteq \mathcal{F} \subseteq \mathcal{P}$ for any s and θ. A maximal motif y is a *successor* of maximal motif x (within $\mathcal{M}$) if $x \prec y$ and there is no maximal motif z such that $x \prec z \prec y$.

Now, we state our problem as follows.

Definition 5. *The* maximal motif enumeration problem *is, given an input string s of length n and a quorum $\theta \geq 1$, to enumerate all maximal motifs in s without repetition.*

2.2 Merge and Closure

In this subsection, we define merge and closure operations which are originally introduced in [2, 3, 12, 13].

An *infinite string* is a function from integers to symbols in $\Sigma \cup \{\circ\}$. For a finite string $x \in (\Sigma \cup \{\circ\})^*$, the *infinite or expanded version* of x is an infinite string $\lfloor x \rfloor$ defined by $\lfloor x \rfloor[i] = x[i]$ for $0 \leq i \leq |x| - 1$ and $\lfloor x \rfloor[i] = \circ$ otherwise. For an infinite string x, its *finite string version* (or *trimmed* version), denoted by $\lceil x \rceil$, is the longest substring of x that starts and ends with a solid character in Σ, i.e., $\lceil x \rceil \in \mathcal{P}$, if it exists and ε otherwise. By definition, if x is a pattern then $\lceil \lfloor x \rfloor \rceil = x$ for every finite x, but it is not the case for general string such as $x = \circ\mathtt{B}\circ\mathtt{C}\circ$.

Let d be a displacement. For an infinite string x, the infinite string $(x + d)$ is defined by $(x+d)[i] = x[i-d]$ for every i. For a finite string x, $(x+d) = \lfloor x \rfloor + d$. Then, $(x + d)$ is called the *shift of x by d*.

Example 4. Given a finite string $s = \mathtt{EABCDABED}$, its infinite version is $\lfloor s \rfloor = \cdots\circ\circ\downarrow\mathtt{EABCDABED}\circ\circ\cdots$, where $\downarrow$ indicates the origin $i = 0$. Then, $(\lfloor s \rfloor - 2) = \cdots\circ\circ\mathtt{EA}\downarrow\mathtt{BCDABED}\circ\circ\cdots$, and its finite version is $\lceil(\lfloor s \rfloor - 2)\rceil = \downarrow\mathtt{EABCDABED} = \mathtt{s}$.

Merge of infinite and finite strings. Next, we define the merge operator $\oplus$. For letters $a, b \in \Sigma$, we define $a \oplus a = a$ and $a \oplus \circ = \circ \oplus a = a \oplus b = \circ$ if $a \neq b$. For infinite strings α, β, the *merge* of α and β, denoted by $\alpha \oplus \beta$, is the infinite string such that $(\alpha \oplus \beta)[i] = \alpha[i] \oplus \beta[i]$ for every integer i. For finite strings $x, y \in \mathcal{P}$, the *merge* of x and y, denoted by $x \oplus y$, is the finite string $x \oplus y = \lceil \lfloor x \rfloor \oplus \lfloor y \rfloor \rceil \in \mathcal{P}$. Note that the operator $\oplus$ is associative and commutative.

Definition 6 (merge of location list [3, 12]). *For a location list $\mathcal{L} = \{d_1, \ldots, d_{|\mathcal{L}|}\}$ of a string s, the* merge *of $\mathcal{L} = \{d_1, \ldots, d_{|\mathcal{L}|}\} \subseteq \{0, \ldots, n-1\}$ is the pattern $\bigoplus \mathcal{L} \in \mathcal{P}$ defined by*

$$\bigoplus \mathcal{L} = \lceil (\lfloor s \rfloor + d_1) \oplus \cdots \oplus (\lfloor s \rfloor + d_{|\mathcal{L}|}) \rceil.$$

We can see the following properties hold.

Property 1. Let $\mathcal{L}, \mathcal{L}'$ be any location lists.

1. If $\mathcal{L} \supseteq \mathcal{L}'$ then $\bigoplus \mathcal{L} \preceq \bigoplus \mathcal{L}'$.
2. $\bigoplus \mathcal{L} = \bigoplus (\mathcal{L} + d)$ for any integer d.

Property 2. Let θ be a quorum and $\mathcal{L}$ be any location list such that $|\mathcal{L}| \geq \theta$. Then, $\bigoplus \mathcal{L}$ is a maximal motif.

Property 3. For any string $xy \in \Sigma(\Sigma \cup \{\circ\})^*$, $\mathcal{L}(xy) = \mathcal{L}(x) \cap (\mathcal{L}(y) + |x|)$.

Property 4. Let x and y be two strings in $\Sigma(\Sigma \cup \{\circ\})^*$ such that $x[i] = \circ$, $y[i] = c$, and $x[j] = y[j]$ for $j \neq i$. Then, we have $\mathcal{L}(y) = \mathcal{L}(x) \cap \{d | x[d+i] = c\}$.

Definition 7 (closure operation [12, 13] [4]). *Given a pattern x and an input string s, the motif $Clo(x) = \bigoplus \mathcal{L}(x)$ is called the* closure of x on s.

The above definition is a generalization of the closure operation [11, 16] from sets to motifs, and is introduced by [12, 13].

We can see the following lemma claiming the computability of the closure.

Lemma 2. *The closure $Clo(x)$ of a pattern x is unique and computable in $O(mn)$ time from x and $\mathcal{L}(x)$, where $n = |s|$ and $m = |\mathcal{L}(x)| \leq n$.*

Proof. The character of the jth position of $Clo(x)$ is $\lfloor x \rfloor[d_1] \oplus, \cdots, \lfloor x \rfloor[d_m]$ where $\mathcal{L}(x) = \{d_1, \ldots, d_m\}$. It can be computed in $O(m)$ time. Since the character of the jth position of $Clo(x)$ is '$\circ$' for any $j < 0$ and $j > n$, we can compute $Clo(x)$ in $O(mn)$ time. We can also observe that $Clo(x)$ is unique. □

Lemma 3 (properties of closure). *Let x, y be any patterns occurring in s.*

1. *$x \preceq Clo(x)$.*
2. *$Clo(x) = Clo(Clo(x))$.*
3. *If $x \preceq y$ then $Clo(x) \preceq Clo(y)$.*
4. *$Clo(x)$ is the unique maximal element w.r.t $\preceq$ in the equivalence class of patterns $[x] = \{\, y \,|\, \mathcal{L}(x) = \mathcal{L}(y) + d$ for some integer $d \,\}$ containing x.*

[4] The set-counterpart of the closure has been known in data mining and formal concept analysis [11, 16]. The closure $Clo(x)$ for motifs was introduced in [12] and called the *maximal extension* in [13].

Theorem 1 (characterization of maximal motifs [12]). *Let θ be a quorum and x be a motif in an input string s. Then, the following (i)–(iii) are equivalent:*

(i) *x is a maximal motif.*
(ii) *$x = \bigoplus \mathcal{L}$ and $|\mathcal{L}| \geq \theta$ for some $\mathcal{L} \subseteq \{0, \ldots, |s| - 1\}$.*
(iii) *$x = Clo(x)$.*

The next lemma says that the converse of Lemma 1 holds as follows for maximal motifs.

Lemma 4 ([12]). *Let $\theta \geq 1$ be a quorum and $x, y \in \mathcal{M}$ be maximal motifs. Then,*

1. *$x \preceq y$ iff $\mathcal{L}(x) \supseteq \mathcal{L}(y) + d$ for some integer d, and*
2. *if $x = y$ iff $\mathcal{L}(x) = \mathcal{L}(y) + d$ for some integer d.*

Proof. The only-if direction of the statement 1 is obvious from Lemma 1. Now, we show its if direction. Let $x, y \in M$ be maximal motifs. Suppose that $\mathcal{L}(x) \supseteq \mathcal{L}(y) + d$ holds for some d. From Property 1 for the merge operator $\bigoplus$, if $L(x) \supseteq L(y)$ then $\bigoplus L(x) \preceq \bigoplus L(y)$ holds. On the other hand, the characterization of Theorem 1 says that $x = Clo(x) = \bigoplus L(x)$ for every maximal motif x. Therefore, it follows that $x \preceq y$, and thus the result is proved. Then, the statement of 2 immediately follows from the above statement 1. □

In other words, ordered sets $(\mathcal{M}, \preceq)$ and $(\mathcal{L}_\theta, \supseteq)$ are isomorphic, where $\mathcal{L}_\theta = \{\, \mathcal{L}(x) \,|\, x \in \mathcal{F} \,\}$ is the class of the location lists for all (possibly non-maximal) motifs in s.

Example 5. Let $s =$ `ABBCABRABRABCABABRABBC` be an input string. Let $x =$ `B○○○○A` be a pattern with location list $\mathcal{L}(x) = \{2, 5, 8\}$. First, we compute the alignment of infinite strings $\mathcal{S} = \{(\lfloor s \rfloor - 2), (\lfloor s \rfloor - 5), (\lfloor s \rfloor - 8)\}$ as follows:

$$\begin{array}{ll}
\texttt{○○○○○○AB}{\downarrow}\underline{\texttt{B}}\texttt{C}\underline{\texttt{AB}}\texttt{R}\underline{\texttt{AB}}\texttt{RABCABABRABBC} & = s - 2 \\
\texttt{○○○ABBCA}{\downarrow}\underline{\texttt{B}}\texttt{R}\underline{\texttt{AB}}\texttt{R}\underline{\texttt{AB}}\texttt{CABABRABBC○○○} & = s - 5 \\
\texttt{ABBCABRA}{\downarrow}\underline{\texttt{B}}\texttt{R}\underline{\texttt{AB}}\texttt{C}\underline{\texttt{AB}}\texttt{ABRABBC○○○○○○} & = s - 8
\end{array}$$

where the underlines indicate the common letters. Then, we compute the merge $\bigoplus \mathcal{S} = (\lfloor s \rfloor - 2) \oplus (\lfloor s \rfloor - 5) \oplus (\lfloor s \rfloor - 8)$ of the infinite strings in $\mathcal{S}$ as follows:

$$\texttt{○○○○○○○○}{\downarrow}\texttt{B○AB○AB○○○○○○○○○○○○○} \quad = \bigoplus \mathcal{S}$$

Finally, we get the closure $Clo(x) =$ `B○AB○AB` by taking its finite version.

2.3 Enumeration algorithms

We introduce terminology for enumeration algorithms according to [7, 15]. An *enumeration algorithm* for an enumeration problem Π is an algorithm $\mathcal{A}$ that receives an *instance* I and outputs all *solutions* S in the answer set $\mathcal{S}(I)$ into a write-only output stream O without duplicates. Let $N = ||I||$, $M = |\mathcal{S}(I)|$ be the input and the output sizes on I, and $T_{\mathcal{A}}$ be the total running time of $\mathcal{A}$ for computing all solutions on I. Then, $\mathcal{A}$ is of *output-polynomial* (P-OUTPUT) if $T_{\mathcal{A}}$ is bounded by a polynomial $q(N, M)$. $\mathcal{A}$ is of *polynomial enumeration time* (P-ENUM) if the *amortized time* for each solution $x \in \mathcal{S}$ is bounded by a polynomial $p(N)$ in N, i.e., $T_{\mathcal{A}} = O(M \cdot p(N))$. $\mathcal{A}$ is of *polynomial delay* (P-DELAY) if the *delay*, which is the maximum computation time between two consecutive outputs, is bounded by a polynomial $p(N)$ in the input size N. $\mathcal{A}$ is of *polynomial space* (P-SPACE) if the maximum size of its working space, except the size of output stream O, is bounded by a polynomial $p(N)$. By definition, P-OUTPUT is the weakest and P-DELAY is the strongest among P-OUTPUT, P-ENUM, and P-DELAY, that is, any P-DELAY algorithm is P-ENUM, and any P-ENUM algorithm is P-OUTPUT.

The class of *tree-search algorithm* is an important class of P-ENUM algorithms, which traverses a spanning tree $\mathcal{T}$ over $\mathcal{S}(I)$, and outputs each solution at each node with work time $P(N)$. In general, if $P(N)$ is polynomial then a tree-search algorithm $\mathcal{A}$ is a P-ENUM with linear total complexity $M \cdot P(N)$, but not necessarily P-DELAY since the delay depends on the maximum depth of the tree. The following lemma by Uno [15] gives a useful technique to translate a tree-search enumeration algorithm in P-ENUM into one in P-DELAY.

Lemma 5 (Uno [15]). *Any tree-search algorithm for Π with polynomial work time $P(N)$ per node can be transformed into a polynomial delay algorithm for Π with delay $O(P(N))$.*

Proof. For completeness, we sketch the proof of [15]. The modified algorithm $\mathcal{A}'$ with the delay $3 \cdot P(N) + O(1)$ visits all nodes of $\mathcal{T}$ in a depth-first order as the original. At each node of depth $d \geq 0$ with solution x, $\mathcal{A}'$ outputs x in the first visit (preorder) if d is even, and in the last visit (postorder) if d is odd. At the leaf node, the algorithm always outputs a solution, immediately. Then, we can see that on at least one node among three consecutive visiting nodes, a solution is output. □

3 Lower bounds for the number of maximum motifs

We show the following lower bound of the number of maximal motifs in a given sequence, which justifies output-sensitive algorithms for the maximal motif enumeration problem. The upper bound of $|\mathcal{M}|$ is obviously $2^{O(n)}$.

Theorem 2 (exponential lowerbound of maximal motifs). *There is an infinite series of input strings $s_0, s_1, s_2, \ldots$, such that for every $i = 0, 1, 2, \ldots$, the number $|\mathcal{M}|$ of maximal motifs in s_i is bounded below by $2^{\Omega(n)}$, that is, exponential in $n = |s_i|$.*

Proof. Let $\Sigma = \{\texttt{#}, \texttt{0}, \texttt{1}\}$ and $n \geq 1$ be any nonnegative integer. We define the input string $s = \texttt{#}t_1\texttt{#} \cdots \texttt{#}t_n\texttt{#}$ over Σ so that s has a family of exponentially many motifs that have mutually distinct location lists. For every $i = 1, \ldots, n$, the i-th block $t_i = b_1 \cdots b_n \in \{0,1\}^n$ is defined as follows: for every $j = 1, \ldots, n$, b_j is $\texttt{1}$ if $i = j$ and $\texttt{0}$ otherwise. These n blocks $t_1, \ldots, t_n$ correspond to the boolean matrix of size $n \times n$ with 1's on the diagonal. Next, we define a family $\mathcal{X}$ of patterns as follows. For every bit string $b = b_1 \ldots b_n \in \{0,1\}^n$ of length n we define pattern $x(b) = \texttt{#}p_1 \ldots p_n\texttt{#}$ in $\mathcal{X}$ as follows: for every $i = 1, \ldots, n$, $p_i = \circ$ if $b_i = 1$, and $p_i = \texttt{0}$ if $b_i = 0$. For example, if $b = \texttt{0010}$ then $x(b) = \texttt{00} \circ \texttt{0}$. Then, we can show that pattern $x(b)$ matches the i-th block t_i for every $i = 1, \ldots, n$ iff $b_i = 1$. Let $\{d_1, \ldots, d_n\} \subseteq \{0, \ldots, |s| - 1\}$ be the set of the initial n positions of delimiters $\texttt{#}$ in s. Then, the location lists of $x(b)$ is $\mathcal{L}(x(b)) = \{\ d_i \,|\, 1 \leq i \leq n,\ b_i = \texttt{1}\ \}$. Furthermore, if $b \neq b'$ then $\mathcal{L}(x(b)) \neq \mathcal{L}(x(b'))$ for any $b, b' \in \{0,1\}^n$. Therefore, we know that all patterns in $\mathcal{X}$ have mutually distinct location lists, and thus they are maximal in s provided they satisfy a quorum. Let $\theta = \frac{1}{2}n$ be a quorum. Then, we can show that the number of maximum motifs within $\mathcal{X}$ is bounded below by $F = \sum_{k=\theta}^{n} \binom{n}{k} = 2^{\Omega(n)}$. This completes the proof. $\square$

The following lemma shows that frequent closed itemset mining is a special case of maximal motif mining. A *t*ransaction database D is a collection of *t*ransactions where a transaction is a subset of a given set E of items. D may includes many identical transactions. For a subset x of E, an *o*ccurrence of x is a transaction including x. The set of occurrences of x is called the *d*enotation of x, and the frequency of x is the cardinality of its denotation. x is called a *c*losed itemset[11, 16] if x is properly included in another subset of E which has the same denotation as x. See the details of the notion of closed itemsets and its enumeration algorithm in [11, 16]

It is known that the closed itemset enumeration (with frequency at least 1) is equivalent to the maximal bipartite clique enumeration in bipartite graphs [5]. Since counting maximal bipartite cliques in a bipartite graph is known to be #P-complete[17], we can see that counting maximal motifs is also #P-complete.

Lemma 6 (transformation from closed itemsets to maximal motifs). *For every transaction database $r = \{t_1, \ldots, t_m\}$ consisting of m transactions over n items of total size $N = mn$, there exists an input sequence s of length $O(nm)$ such that the set $\mathcal{M}_\theta^r$ of θ-frequent closed itemsets in r is equivalent to the set of $\mathcal{M}_{\theta+m}^s$ of maximal motifs in s with quorum $\theta + m$.*

Proof. Let $\Sigma = \{\texttt{\#}, \texttt{0}, \dots, \texttt{n+1}, \bar{\texttt{1}}, \dots, \bar{\texttt{n}}\}$. We define $s(t)$ for transaction t by the string of length n such that ith letter is i if $i \in t$, and is $\bar{i}$ otherwise. Then, we define $s = s(t_1)\texttt{\#}\cdots\texttt{\#}s(t_m)s(\{0,\dots,n+1\})\cdots s(\{0,\dots,n+1\})$ where $s(\{0,\dots,n+1\}) = 0123\cdots n+1$ and it is repeated m times. Note that there is no # between $s(\{1,\dots,n\})$'s. We can see that any pattern x of frequency at most $m+1$ includes only letters in $\texttt{0}, \dots, \texttt{n}, \circ$, and can not include two same letters. Thus, such x is obtained from a substring of $0123\cdots n+1$ by replacing some letters the substring by $\circ$. Since any such pattern matches $0123\cdots n+1$, the frequency of such pattern is always no less than m. Moreover, we can see that if such a pattern x does not begin with 0 or does not end with $n+1$, it never be a maximal motif.

For an itemset I, we define its corresponding pattern by the pattern obtained from $0123\cdots n+1$ by replacing items not in I by $\circ$. Then, we can see that x appears at the position of $s(t_i)$ if and only if t_i includes I. Thus, I is a closed pattern if and only if it is a maximal motif, and the frequency of I is equal to the frequency of the corresponding maximal motif minus m.

$\square$

From the following corollary obtained by the lemma, we know that the maximal motif enumeration problem is at least as hard as the closed itemset enumeration problem [5, 11, 16].

Corollary 1. *Any algorithm for enumerating maximal motifs in s of size n in time $T(n)$ per motif, space $S(n)$, and delay $D(n)$ can be transformed into an algorithm for enumerating closed itemsets with minimum support in $O(T(N^2))$ time per itemset, space $O(S(N^2))$, and delay $O(D(N^2))$ on r of size N.*

Theorem 3 (#P-completeness of maximal motifs). *The problem of counting the number of maximal motifs in a given string is #P-complete.*

The following theorem says that the number of motifs can be exponentially larger than the number of maximal motifs.

Theorem 4 (succinctness of maximal motifs). *There is an infinite series of input strings $s_0, s_1, s_2, \dots,$ such that for every $i \geq 0$ with quorum $\theta = \frac{1}{2n+2}|s_i|$, the number $F = |\mathcal{F}|$ of motifs in s_i is exponential (more precisely $2^{\Omega(\frac{1}{2n+2}|s_i|)}$), while the number $M = |\mathcal{M}|$ of maximal motifs in s_i is no more quadratic in $|s_i|$.*

Proof. Theorem 1 of Uno *et al.* [16] says that there is an infinite series of transaction databases such that the number F of frequent itemsets and the number M of frequent closed itemsets in a transaction database of size mn are $F = 2^{\Omega(n)}$ and $M = O(m^2)$, respectively. Combining Lemma 6 above and the constructions of Theorem 1 of [16], where $N = \Theta(m^2)$ for large m and n, we obtain the result.

$\square$

From Theorem 4, we know that a straightforward algorithm for enumerating $\mathcal{M}$ based on the enumeration of motifs does not work efficiently. This is also true for most real world datasets. Figure 1 shows an example, where there are only 10 maximal motifs among 50 motifs in a string of length 21.

4 Previous approaches for maximal motif enumeration

We give a brief review on possible approaches for output-sensitive computation of $\mathcal{M}$ and summarize the previous results.

4.1 Previous approaches

A most straightforward method of generating maximal motifs is to use frequent pattern generation. We enumerate all motifs in an input string s, classify them into equivalence classes according to their location lists, and find the maximal motifs for each equivalence class. This method requires $O(|\mathcal{F}|)$ time and $O(||\mathcal{F}||)$ memory. Since $O(|\mathcal{F}|)$ can be exponentially larger than $|\mathcal{M}|$, we cannot obtain any output-sensitive algorithm in time and memory in this way. The algorithm in [?] uses maximal motifs instead of solid characters to extend the pattern. In the worst case it combines any pairs of maximal motifs, thus the time complexity is output polynomial but square of the output size. Moreover, the space complexity is not polynomial in the input size, since we have to store maximal motifs previously discovered in memory.

Another possible method is to use the *basis* for maximal motifs [10, 12, 13]. Parida *et al* [9] introduced the use of the basis for maximal motif enumeration. A *basis* for $\mathcal{M}$ is a subset $\mathcal{B} \subseteq \mathcal{M}$ of motifs such that $\mathcal{M}$ can be generated by finite applications of an operation, e.g., $\oplus$, over $\mathcal{M}$. Presently, the basis $\mathcal{B}_I$ of *irredundant motifs* [9], the basis $\mathcal{B}_T$ of *tiling motifs* [13], and the basis $\mathcal{B}_P$ of *primitive motifs* [12] have been proposed. A maximal motif $x \in \mathcal{M}$ is *tiling* if for any set of maximal motifs $y_1, \ldots, y_k \in \mathcal{M}$ and any set of integers $d_1, \ldots, d_k$ with $x \preceq y_i$, if $\mathcal{L}(x) = \bigcup_i (\mathcal{L}(y_i) + d_i)$ then $x = y_i$ for some i. Pisanti *et al.* [14] describe a simple algorithm for computing $\mathcal{M}$ from $\mathcal{B}_T$ in $O(|\mathcal{M}|^2 \cdot n)$ total time and $O(||\mathcal{M}||)$ space. However, the total time is not linear in $|\mathcal{M}|$. Thus, it is of P-OUTPUT but not of P-ENUM.

4.2 An improved algorithm for generating $\mathcal{M}$ from a basis $\mathcal{B}_T$

We can improve Pisanti *et al.*'s method for $\mathcal{M}$ adopting an idea used in [12] for generation of $\mathcal{B}_T$ from s. The next lemma is essential for our algorithm.

Lemma 7. *Any maximal motif $x \in \mathcal{M}$ satisfies either (i) $x \in \mathcal{B}_T$, or (ii) there exist some $y \in \mathcal{M}$ and integer d such that $x \prec y$ and $x = y \oplus (s + d)$.*

```
Algorithm MaxBasis( $\theta$: quorum, $s$: input string, $\mathcal{B}$: basis)
1  $\mathcal{M}^0 := \mathcal{B}$; $i := 0$
2  while ($\Delta \neq \emptyset$) do begin
3    $\Delta := \emptyset$;
4    for each $y \in \mathcal{M}^i$ and $d \in \{0, \ldots, n-1\}$ do
5      if $y \oplus (s+d) \notin (\bigcup_{k=0}^{i} \mathcal{M}^k \cup \Delta)$ then $\Delta = \Delta \cup \{y \oplus (s+d)\}$; output $y \oplus (s+d)$;
6    $\mathcal{M}^{i+1} := \Delta$; $i := i+1$;
7  end
```

Fig. 2. An polynomial time enumeration algorithm for generating $\mathcal{M}$ from $\mathcal{B}$ based on breadth-first search. This algorithm does not have polynomial space or polynomial delay.

Figure 2 shows our algorithm MaxBasis that computes $\mathcal{M}$ from $\mathcal{B}_T$. In the algorithm, the set $\mathcal{M}^i$ is the set of maximal motifs which can be generated from $\mathcal{B}$ by applying the operation in line 5 i times. We use a trie to store $\mathcal{M}^i$ and Δ for $O(|x|)$ membership of pattern x. Thus, we can implement MaxBasis to run in $O(|\mathcal{M}| \cdot n^2)$ total computation time.

Theorem 5 (generation of maxmal motifs from the basis). *Given a quorum $\theta \geq 1$, an input string s of length n, and the basis $\mathcal{B}_T$ of tiling motifs, the algorithm* MaxBasis *in Figure 2 enumerates all maximal motifs of $\mathcal{M}$ from $\mathcal{B}$ in $O(n^2)$ amortized time per maximal motif with $O(||\mathcal{M}||)$ space.*

Since its space complexity and delay are $O(||\mathcal{M}||)$ and $O(|\mathcal{M}| \cdot n^2)$, respectively, MaxBasis is neither a polynomial space or polynomial delay even given a basis $\mathcal{B}_T$ as input. Note that it is still open whether the basis $\mathcal{B}_T$ (or $\mathcal{B}_P$) is output-polynomial time computable from s since the total running time of the algorithms in [12] and [14] are only bounded by $O(n^\theta \sum_{i=1}^{\theta} |\mathcal{B}_T^i|)$ or $n^{O(\theta)}$, where $\mathcal{B}_T^i$ is the basis for quorum $i \geq 1$. Hence, it seems difficult to obtain an output-polynomial time algorithm for $\mathcal{B}_T$ and thus $\mathcal{M}$ in this approach. [5]

5 A polynomial space polynomial delay algorithm using depth-first search

In this section, we present an efficient depth-first search algorithm MaxMotif that, given a quorum $\theta \geq 1$ and an input string s of length n, enumerates all

[5] Parida *et al.* [10] presented an output-polynomial time algorithm for the class of *flexible motifs*, and claimed that they also presented a similar algorithm for maximal motifs with wild cards in [9]. Since these algorithms seem to depend on an unproved conjecture in [9], however, we did not include them. At least, the algorithm in [10] requires the space and the delay proportional to the output size $|\mathcal{M}|$. Thus, it is not polynomial space and polynomial delay.

maximal motifs x in s in $O(|\mathcal{L}(x)| \cdot n^2)$ delay and $O(n)$ space. In what follows, we fix input string s of length $n \geq 1$ and $1 \leq \theta \leq n$. Unlike MaxBasis in the previous section, MaxMotif uses depth-first search over $\mathcal{M}$ to avoid the use of extra storage for keeping all discovered motifs. In the following sections, we explain the details of the algorithm.

5.1 Building tree-shaped search route for maximal motifs

We first build a tree-shaped search route $\mathcal{T} = (\mathcal{V}, \mathcal{P}, \perp)$ for traversing all maximal motifs (Figure 3). The node set $\mathcal{V} = \mathcal{M}$ consists in all maximal motifs in s, $\mathcal{P}$ is the set of reverse edges defined later, and $\perp = Clo(\varepsilon)$ is the root called the *root motif*. If s contains at least two distinct solid letters then $\perp = \varepsilon$, otherwise $\perp = a$ for the only letter a in s.

Lemma 8. *$\perp = Clo(\varepsilon)$ is the unique shortest maximal motif in s.*

Proof. Since $\mathcal{L}(\perp) = \{0, \ldots, n-1\}$ is the largest location list on s, it follows from Lemma 4 that $Clo(\varepsilon) \preceq x$ for any maximal motif x. □

Next, we define the set $\mathcal{P}$ of reverse edges from a child to its parent as follows. Given a maximal motif x, the *core index* of x, denoted by $core\_i(x)$, is the smallest index $0 \leq \ell \leq |x|-1$ such that $\mathcal{L}(x) = \mathcal{L}(y)$ for the prefix $y = x[0..\ell]$, if $x \neq \perp$. For $x = \perp$, $core\_i(x)$ is defined by -1. Then, we assign the unique parent to each non-root maximal motif.

Definition 8 (parent of maximal motif). *For a maximal motif y such that $y \neq \perp$, the* parent *of y, denoted by $\mathcal{P}(y)$, is the pattern $\mathcal{P}(y) = Clo(\lceil y[0..core\_i(y)-1]\rceil)$.* [6]

Lemma 9. *For every maximal motif y such that $y \neq \perp$, $\mathcal{P}(y)$ always exists, is unique, and is a maximal motif. Furthermore, $\mathcal{P}(y) \prec y$ holds.*

Proof. Since $y \neq \perp$, $\mathcal{L}(y) \neq \mathcal{L}(\perp)$, thereby $\mathcal{L}(\lceil y[0..\ell-1]\rceil) \neq \mathcal{L}(y)$ holds for some $l \geq 0$. This assures that $core\_i(y) - 1 \geq -1$ and $\mathcal{P}(y)$ is always defined. From the definition, $\mathcal{P}(y)$ is unique, and clearly a maximal motif. □

Theorem 6. *$\mathcal{T} = (\mathcal{V}, \mathcal{P}, \perp)$ is a spanning tree for all maximal motifs in $\mathcal{M}$.*

Proof. From Lemma 9, all maximal motifs y but $\perp$ have the unique parent $\mathcal{P}(y)$ such that $\mathcal{P}(y) \prec y$. Since the relation $\preceq$ is acyclic on $\mathcal{M}$, i.e., there is no infinite decreasing chain of maximal motifs of $\mathcal{M}$, the result follows. □

The remaining task is to show how to enumerate all children y of a given parent motif x within polynomial memory space of the input size. This is not an easy task since we have only reverse edges. We discuss this issue in the next subsection.

[6] In the definition, $y[0..core\_i(y)-1] \in \Sigma(\Sigma \cup \{\circ\})^* \cup \{\varepsilon\}$ may not be a proper pattern. Thus, we use $\lceil y[0..core\_i(y)-1]\rceil$ instead of $y[0..core\_i(y)-1]$ to remove the trailing $\circ$'s.

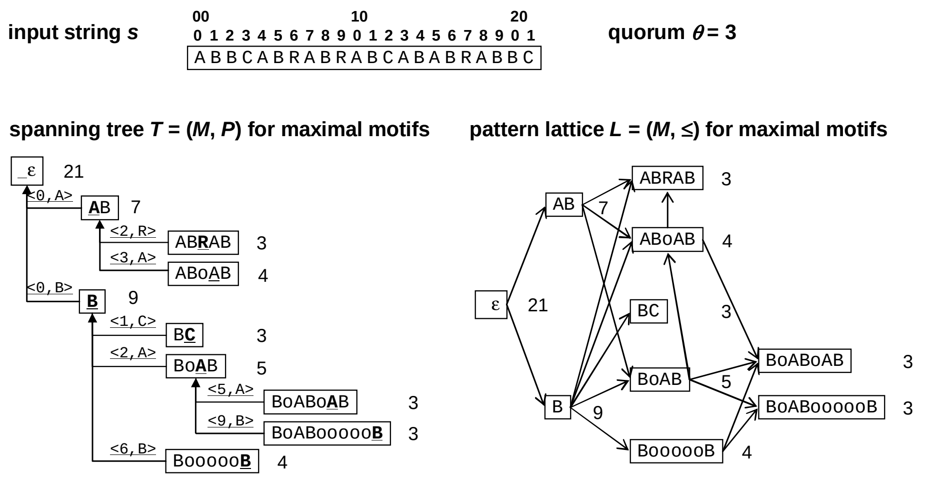

Fig. 3. The spanning tree $\mathcal{T} = (\mathcal{M}, \mathcal{P})$ (left) and the pattern lattice $\mathcal{L} = (\mathcal{M}, \preceq)$ (right) for maximal motifs of $\mathcal{M}$ on quorum $\theta = 3$ and input string s (top). Each box represents maximal motif x in $\mathcal{M}$ and the number right to the box indicates its frequency $|\mathcal{L}(x)|$. Each arrow indicates ordering, $\mathcal{P}$ or $\preceq$, of a tree/lattice. (Section 5.1). An arrow in the tree $\mathcal{T}$ indicates the ppc-extension with seed $\langle k, c\rangle$. The newly introduced letter c is written in bold face. (Section 5.2). There are 10 maximal motifs among 49 motifs in s.

5.2 Prefix-preserving closure extension

We introduce the prefix-preserving closure extension defined as follows. A *substitution* for motif x is a pair $\xi = \langle k \leftarrow c\rangle \in \mathsf{Z} \times \Sigma$ of integer k and solid letter c. If $\lfloor x \rfloor[k] = \circ$, ξ is *compatible* with x. The *application* of ξ to x, denoted by $x\xi = x\langle k \leftarrow c\rangle$, is the motif $\lceil y \rceil$, where y is the infinite string such that for every integer i, $y[i] = c$ if $i = k$ and $y[i] = \lfloor x \rfloor[i]$ otherwise. For example, if $x = \mathtt{BA} \circ \mathtt{B}$, then $x\langle -1 \leftarrow \mathtt{C}\rangle = \mathtt{CBA} \circ \mathtt{B}$, $x\langle 2 \leftarrow \mathtt{C}\rangle = \mathtt{BACB}$, and $x\langle 6 \leftarrow \mathtt{C}\rangle = \mathtt{BA} \circ \mathtt{B} \circ \circ \mathtt{C}$.

Definition 9 (ppc-extension). *For any maximal motifs x and y, y is a* prefix-preserving closure extension *(or a* ppc-extension*) of x if the following (i)–(iii) hold:*

(i) $y = Clo(x\langle k \leftarrow c\rangle)$ for some substitution, called its seed, *$\xi = \langle k \leftarrow c\rangle \in \mathsf{Z} \times \Sigma$ compatible with x, that is, y is obtained by first substituting c at index k and then taking its closure,*

(ii) the index k satisfies $k > core\_i(x)$, and

(iii) $x[0..k-1] = y[0..k-1]$, that is, the prefix of length $k-1$ is preserved, where $x[0..k-1]$ is obtained from x by padding trailing $\circ$'s if necessary.

Example 6. In Figure 3, we show an example of the spanning tree for $\mathcal{M}$ generated by the ppc-extension for an input string $s = \mathtt{ABBCABRABRABCABABRABBC}$ and quorum $\theta = 3$. Then, we have maximal motif $x = \mathtt{AB}$ with location list $\mathcal{L}(x) = \{0, 4, 7, 10, 13, 15, 18\}$. If we apply substitutions $\xi_1 = \langle 2, \mathtt{R}\rangle$ and $\xi_2 = \langle 3, \mathtt{A}\rangle$ to x, respectively, then we obtain the ppc-extension $y = \mathtt{ABRAB} = \mathit{Clo}(\mathtt{AB\underline{R}}) = \mathit{Clo}(x \cdot \xi_1)$ with $\{4, 7, 15\}$, and $z = \mathtt{AB} \circ \mathtt{AB} = \mathit{Clo}(\mathtt{AB} \circ \underline{\mathtt{A}}) = \mathit{Clo}(x \cdot \xi_2)$ with $\{4, 7, 10, 15\}$.

Lemma 10. *Let x be any maximal motif and $y = \mathit{Clo}(x\langle k \leftarrow c\rangle)$ be a ppc-extension of x. Then, k is the core index of y.*

Proof. Since k is larger than the core index of x, we have that $\mathcal{L}(\lceil y[0..k-1]\rceil) = \mathcal{L}(x)$. Thus, the core index of y is greater than $k - 1$. Since $y = \mathit{Clo}(\lceil y[0..k]\rceil)$, $\mathcal{L}(y) = \mathcal{L}(\lceil y[0..i]\rceil)$ holds for any $i \geq k$. Therefore, the core index of y is k.

The goal of this subsection is to prove the following theorem.

Theorem 7 (correctness of ppc-extension). *For any maximal motifs x, y such that $y \neq \bot$, (1) $x = \mathcal{P}(y)$ if and only if (2) $y = \mathit{Clo}(x\xi)$ is a prefix-preserving closure extension of x for some substitution $\xi = \langle k \leftarrow c\rangle \in \mathsf{Z} \times \Sigma$ compatible with x. Furthermore, there exists exactly one ξ satisfying condition (2) for each y.*

Proof. (1) to (2): Suppose $x = \mathcal{P}(y)$, and thus $x = \mathit{Clo}(\lceil y[0..\ell-1]\rceil)$ for the core index ℓ of y. First, we show that $x[0..\ell-1] = y[0..\ell-1]$ holds. Since $x = \mathit{Clo}(\lceil y[0..\ell-1]\rceil)$, we have $x[0..\ell-1] \preceq y[0..\ell-1]$. Thus, if $x[0..\ell-1] \neq y[0..\ell-1]$, a more specific motif $y' = x[0..\ell-1]y[\ell..|y|-1]$ satisfies $y \prec y'$ but $\mathcal{L}(y) = \mathcal{L}(y')$ using Lemma 3. Next, if we take $\xi = \langle \ell \leftarrow y[\ell]\rangle$, then the definition of core index assures that $y = \mathit{Clo}(x\xi)$. Thus, ξ satisfies the condition of ppc-extension. This proves this direction.

(2) to (1): Suppose that $y = \mathit{Clo}(x\langle k \leftarrow c\rangle)$. Then, it follows from Lemma 10 that k is the core index of y, thus $\mathcal{P}(y) = \mathit{Clo}(\lceil y[0..k-1]\rceil)$. Since y is a ppc extension, we have $y[0..k-1] = x[0..k-1]$, and then $\mathit{Clo}(\lceil y[0..k-1]\rceil) = \mathit{Clo}(\lceil x[0..k-1]\rceil)$. By condition (iii) of ppc-extension, we can show that choice of ξ is unique. □

5.3 A polynomial space polynomial delay algorithm

Based on Theorem 7, we present in Figure 4 our algorithm MaxMotif that enumerates all maximal motifs in a given input string by the depth-first search over $\mathcal{M}$ applying the ppc-extension to each maximal motif.

Although a straightforward implementation of the procedure Expand in Figure 4 requires $O(n^4)$ time for each maximal motif, we can reduce the computation time further.

Theorem 8. *Given a quorum $\theta \geq 1$ and an input string s of length n, the algorithm* MaxMotif *in Figure 4 enumerates all maximal motifs x of $\mathcal{M}$ in $O(\mathcal{L}(x)n^2)$ amortized time per motif with $O(n)$ space and $O(\mathcal{L}(x)n^2)$ delay.*

```
Algorithm MaxMotif($\theta$: quorum, $s$: input string)
input: a quorum $\theta$ and an input string $s$;
output: all maximal motifs in $\mathcal{M}$;
0  $\perp = Clo(\varepsilon)$;                                //the root motif $\perp$.
1  call Expand($\perp, \{0, ..., n-1\}, -1, \theta, s$);     //$core\_i(\perp) = -1$.

Procedure Expand($x$, $\mathcal{L}(x)$, $core\_i(x)$, $\theta$, $s$)
   input: maximal motif $x$, $\mathcal{L}(x)$, core_i(x), quorum $\theta$, and input string $s$;
   output: all descendants of $x$ in $\mathcal{T}$;
2  if $|\mathcal{L}(x)| < \theta$ then return;
3  output $x$;
4  for each $k, core\_i(x) + 1 \leq |s| - 1$ such that $x[k] = \circ$ do
5    for each  $c \in \Sigma$ do begin
6      $y = Clo(x\langle k \leftarrow c\rangle)$;                    //ppt-extension.
7      if $\lfloor x \rfloor[0..k-1] = \lfloor y \rfloor[0..k-1]$ then
8        call Expand($y, \mathcal{L}(y), k, \theta, s$);
9    end for
10 end for
```

Fig. 4. A polynomial space polynomial delay enumeration algorithm for $\mathcal{M}$.

Proof. By Theorem 6 and Theorem 7, we see that the algorithm MaxMotif visits all maximal motifs on the spanning tree $\mathcal{T}$ starting from the root $\perp$. Since $\mathcal{T}$ is a tree and any maximal motif appears in $\mathcal{T}$, the algorithm enumerates $\mathcal{M}$ without duplicates.

Let $W(x)$ be the work that Expand spends for each maximal motif x except the recursive call. For each $y = x\langle k \leftarrow c\rangle$, computing the closure $Clo(y)$ at line 6 takes $O(|\mathcal{L}(y)| \cdot n)$ time. Thus, $W(x) = O(|\Sigma| \cdot |\mathcal{L}(x)| \cdot n^2)$ time and this gives the delay per maximal motif.

To improve the computation from line 5 to line 9, we compute $\mathcal{L}(y)$ for all $y = x\langle k \leftarrow c\rangle$ at once, by using the technique called *Occurrence Deliver* [7]. Suppose that we have an empty bucket for each letter $c \in \Sigma$. Then, for each position $d \in \mathcal{L}(x)$, we insert d to the bucket of the letter $s[d+k]$. After this operation, the content of the bucket of c is $\mathcal{L}(x\langle k \leftarrow c\rangle)$. Now we can see that for each k line 5 to line 9 takes $O(|\mathcal{L}(x)|n)$ time. This does not increase the space complexity of $O(n)$.

By applying the technique by Uno [15] that transforms any tree-search enumeration algorithm with work time $W(x)$ at each node into a $W(x)$ delay algorithm, we finally obtain an algorithm with delay $O(|\mathcal{L}(x)| \cdot n^2)$. □

In summary, the time complexity of the algorithm is $O(n^3)$, the space complexity is $O(n)$, and the delay is $O(n^3)$.

[7] Occurrence deliver is introduced for closed itemsets enumeration in Uno *et al.* [16]

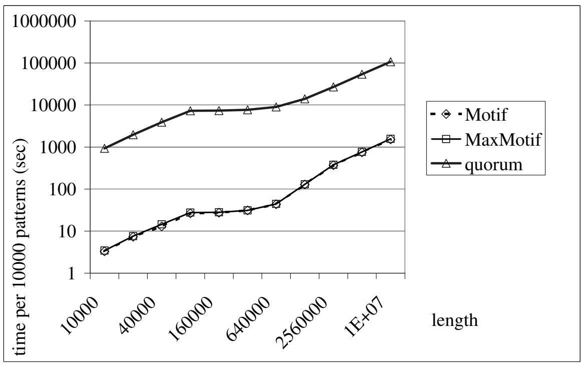

Fig. 5. Computation time with the increase of string length

Corollary 2. *The maximal motif enumeration problem is solvable in polynomial space and polynomial delay in the input size $n = |s|$.* □

6 Computational experiments

In this section, we show the results of computational experiments of our MaxMotif algorithm. The aims of the experiments are to see,

1. the computation time for the real world data,
2. the increase of the computation time against the increase of the length of the input string,
3. the increase of the computation time against the decrease of quorum, and
4. what kind of patterns we get.

We compared these with frequent motif mining. The algorithm for frequent motif mining is a usual depth-first backtracking type one, with occurrence deliver, so the main body of the implementations of both algorithms are the same.

We coded the algorithms with C language with standard library functions. The computer has CPU of Intel Pentium M 1.2GHz and 256MB memory, the OS was cygwin, a Linux emulator on Windows, and the compiler was gcc. We chose Y chromosome of the human genome[8] as a test instance, and some text files additionally. Y chromosome has repeated structures in it, thus it is a good instance for frequent pattern mining. By the experiments, we evaluated the change of computation time with the change of the input string length and the quorum.

First, we see the change of the computation time by the increase of the input string length. We made strings for the experiments by taking the first k letters of Y chromosome for several k's. For each string, we gave a quorum so that nearly 10,000 frequent patterns appear. The results are shown in Figure 5. The X axis

[8] We took the genome data at ftp://ftp.ncbi.nih.gov/genomes/H_sapiens/

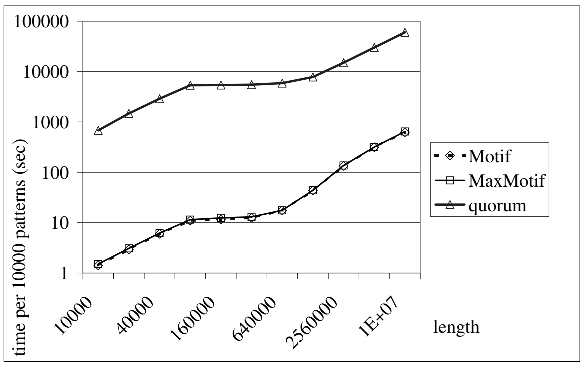

Fig. 6. Computation time with the increase of string length without any consecutive six ∘'s

is the length of the input strings, and Y axis is the computation time for 10,000 patterns. Both axes are drawn in log scales. We also draw lines for the quorum.

We can see that the computation time is not so long for genome sequence of 10 million length, if we want to find not so many patterns. Perhaps MaxMotif works for practical huge genome sequences. Both the number of patterns and the computation time are quite similar, between MaxMotif and frequent motif mining. This shows that the computational costs for generating ppc extensions, which is mainly for checking the equalities of prefixes, are not large. Thus, we can see that we need no additional computational cost to find only maximal patterns. The lines of the computation time and quorum is close to parallel. This means that the computation time depends deeply on quorum but not so much on the length of input string, when we discover thousands of patterns.

In the output, we saw many unnatural patterns, such as $\mathtt{A} \circ \cdots \circ \mathtt{T} \circ \cdots \circ \mathtt{A}$. They are maximal motifs, but seem to be unnecessary in practice. To avoid such patterns, we restricted consecutive ∘'s. Here we restrict six consecutive ∘'s in this experiments. The results are shown in Figure 6. The computation time per pattern does not increase, but the quorum to get 10,000 patterns is decreased. This implies that we avoided many such unnecessary patterns, and find more infrequent but necessary patterns.

Next, we see the result by the change of quorum. The test instance was a substring of 4,000 letters taken from Y chromosome, and we execute the algorithm for several quorums from 320 to 4. The results are shown in Figure 7. Figure 8 shows the results when we forbid any consecutive three ∘'s.

As we can see, there is no difference on the number of patterns, between frequent motifs and frequent maximal motifs when the quorum is large. However, when the quorum is small, the difference is quite huge. Especially, when the quorum was smaller then eight, frequent motif mining algorithm did not terminate in three hours. This implies that maximal motif is a good model for mining many sequence patterns with a small frequency.

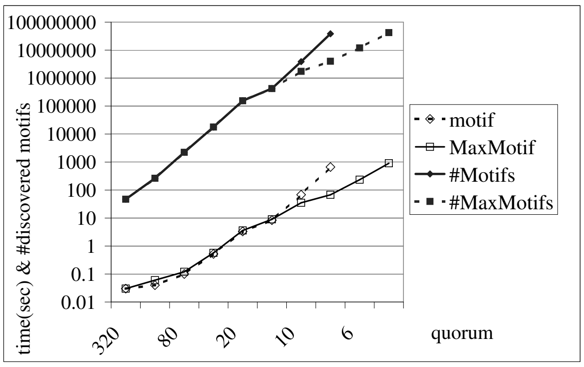

Fig. 7. The number of patterns and computation time with the decrease of quorum

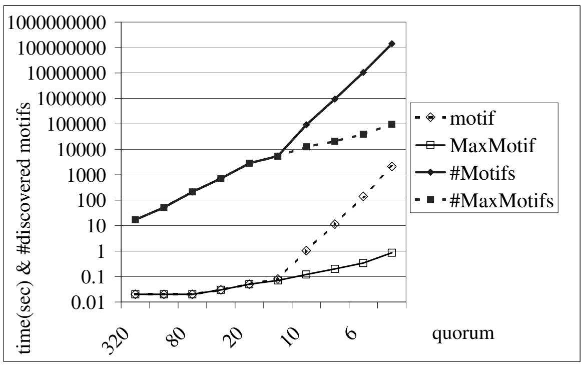

Fig. 8. The number of patterns and computation time with the decrease of quorum with forbidding any consecutive three ∘'s

Finally, we report the results of an execution for text data; We examined the source code of MaxMotif itself. The source code contains about 15000 letters, and we ignored any "new line" characters. The computation was done in two seconds, and we got about 20,000 maximal motifs. However, there were many patterns such as "******************∘∘*". Such patterns match the comment line, such as

```
/************************************************/
/***** main routine ****************************/
/************************************************/
```

One can see that many combinations of consecutive '*' and ∘ make maximal motifs for this string, and such patterns are almost part of the output. Even if we forbid some consecutive ∘'s, still they remain in the output. To avoid such patterns, we introduced "removing overlapping occurrences". For a pattern x, if two positions d and d' in $\mathcal{L}(x)$ satisfy that $d < d' < d + |x|$, that is, overlapping, we consider that d' is redundant, and remove it from $\mathcal{L}(x)$ when we take the closure. Then, we could avoid almost all such redundant patterns efficiently.

As a conclusion, we could see that maximal motifs can be a good model in practice, and we can solve the problem in short time even if the input string is so huge. However, we did not see any result when the size of alphabet is large; in our experiments, it was 4 for genome sequences, and at most 100 for text data. It is a remaining task in this research.

7 Conclusion

In this paper, we presented a polynomial space polynomial delay algorithm for enumerating all maximal motifs in an input string for the class of motifs with wild cards. By the use of depth-first search based on the prefix-preserving extension, the algorithm enumerates all maximal motifs without explicitly storing and checking the maximal motifs enumerated so far. This drastically improves the space and the delay complexities compared with the previous algorithms with breadth-first search. The experimental results show the efficiency of our algorithm in practice, even if the input string is so huge, such as genome sequences of over 10 million length.

In the context of data mining, maximal motifs for *sets* of this type are called *closed itemsets*, and since the end of 90's, related enumeration problems have been extensively studied for various classes of combinatorial objects, such as sets, sequences, trees, and graphs [3, 5, 11, 12, 16, 18]. On the other hand, however, most of them are heuristic algorithms based on branch-and-bound, and there are only a few results on output-polynomial time enumeration for these classes [3, 5, 16]. Hence, the result of this paper will be a first step towards efficient enumeration of maximal patterns for complex combinatorial objects such as sequences, trees, and graphs [3].

Pattern discovery for classes of *unions* of patterns is one of the difficult problems in machine learning. Since the class of maximal motifs is a class of unions generated from tiling motifs [13], it will be a future problem to extend the proposed enumeration method to pattern discovery for unions of sequence patterns, e.g. [4]. Implementation of the proposed algorithm with practical improvement and evaluation of the algorithm on the real world datasets, e.g., biological datasets, are also interesting future problems.

References

1. A. Apostolico, M. Comin 2, L. Parida, Conservative Extraction of Over-Represented Extensible Motifs," *ISMB (Supplement of Bioinformatics)* 21, 9-18, 2005.
2. A. Apostolico and L. Parida, Compression and the wheel of fortune, In *Proc. the 2003 Data Compression Conference (DCC'03)*, IEEE, 2003.

3. H. Arimura, T. Uno, An output-polynomial time algorithm for mining frequent closed attribute trees, In *Proc. ILP'05*, LNAI 3625, 1–19, August 2005.
4. H. Arimura, T. Shinohara, S. Otsuki, Finding minimal generalizations for unions of pattern languages and its application to inductive inference from positive data, In *STACS'94*, LNCS 775, Springer-Verlag, 649–660, 1994.
5. E. Boros V. Gurvich, L. Khachiyan, K. Makino, The complexity of generating maximal frequent and minimal infrequent sets, In *Proc. STACS '02*, LNCS, 133-141, 2002.
6. M. Crochemore and W. Rytter, *Jewels of Stringology*, World Scientific, 2002.
7. L. A. Goldberg, Polynomial space polynomial delay algorithms for listing families of graphs, In *Proc. the 25th STOC*, ACM, 218–225, 1993.
8. D. Gusfield, Algorithms on strings, trees, and sequences, Cambridge, 1997.
9. L. Parida, I. Rigoutsos, A. Floratos, D. Platt, and Y. Gao, Pattern discovery on character sets and real-valued data: linear bound on irredundant motifs and effcient polynomial time algorithm, In *Proc. the 11th SIAM Symposium on Discrete Algorithms (SODA'00)*, 297–308, 2000.
10. L. Parida, I. Rigoutsos, D. E. Platt, An Output-Sensitive Flexible Pattern Discovery Algorithm. In *Proc. CPM'01*, LNCS 2089, 131–142, 2001.
11. N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal, Discovering Frequent Closed Itemsets for Association Rules, In *Proc. ICDT'99*, 398–416, 1999.
12. J. Pelfrêne, S. Abdeddaim, and J. Alexandre, Extending Approximate Patterns, In *Proc. CPM'03*, LNCS 2676, 328–347, 2003.
13. N. Pisanti, M. Crochemore, R. Grossi, and M.-F. Sagot, A basis of tiling motifs for generating repeated patterns and its complexity for higher quorum, In *Proc. MFCS'03*, LNCS 2747, 622–631, 2003.
14. N. Pisanti, M. Crochemore, R. Grossi, and M.-F. Sagot, A comparative study of bases for motif inference, In *String Algorithmics*, KCL publications, 2004.
15. T. Uno, Two general methods to reduce delay and change of enumeration algorithms, NII Technical Report, NII-2003-004E, April 2003.
16. T. Uno, T. Asai, Y. Uchida, H. Arimura, An efficient algorithm for enumerating closed patterns in transaction databases, In *Proc. DS'04*, LNAI 3245, 16-30, 2004.
17. L. G. Valiant, The complexity of computing the permanent, *Theoretical Computer Science* 8, 189-201, 1979.
18. X. Yan, J. Han, CloseGraph: mining closed frequent graph patterns, In *Proc. SIGKDD'03*, 2003.