Polynomial Space and Delay Algorithms 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 finding all maximal motifs in an input string for the class of repeated motifs with wild cards. A maximal motif is such a representative motifs that is not properly contained in larger motifs with the same location lists. The enumeration problem for maximal motifs with wild cards has been introduced in (Parida et al., SODA'00, CPM'01), and has been studied in (Parida et al., CPM'01), (Pisanti et al.,MFCS'03) and (Pelfrene et al., CPM'03). However, its output-polynomial time computability, in particular, with polynomial space and delay is 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 with $O(n^3)$ time per motif with $O(n^2)$ space and $O(n^3)$ delay by depth-first search. The key of the algorithm is a tree-shaped search route over all maximal motifs based on a technique called prefix-preserving closure extension, which enable us to enumerate all maximal motifs without storing all discovered motifs. We also show exponential lower bound and succinctness results on the number of maximal motifs, which indicate the limit of a straightforward approach.

Keywords: enumeration problems, sequence pattern discovery, basis of tiling motifs, bioinformatics, output-polynomial time algorithms

Correspondence:
Hiroki Arimura
Graduate School of Information Science and Technology
Hokkaido University
N14 W9, 060-0814 Sapporo, Japan
e-mail: arim@ist.hokudai.ac.jp
phone: +81-11-706-7678, fax: +81-11-706-7890

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

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 it 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 *wild cards*) that matches any symbol [8, 12]. For instance, $\mathtt{B}\circ\mathtt{AB}$ and $\mathtt{B}\circ\mathtt{AB}\circ\circ\mathtt{B}$ are examples of patterns. Given a positive integer θ called *quorum*, a frequent motif (or *motif*, for short) in an input string 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 [8]. The semantics of a pattern x is given by the location list $\mathcal{L}(x)$ consisting of the positions in 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. In Fig. 1, we show all maximal motifs and all motifs with $\theta = 3$ on input sting $s = \mathtt{ABBCABRABRABCABABRABBC}$ and quorum $\theta = 3$. For example, the pattern $\mathtt{R}\circ\mathtt{B}$ with location list $\{6, 9, 17\}$ is a motif but not maximal since there is another motif $\mathtt{ABRAB}$ with location list $\{4, 7, 15\}$ obtained from $\{6, 9, 17\}$ by shifting leftward with 2. 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 is 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 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 motif. However, the output-polynomial time computability for maximal motif discovery with polynomial space and delay is still open.

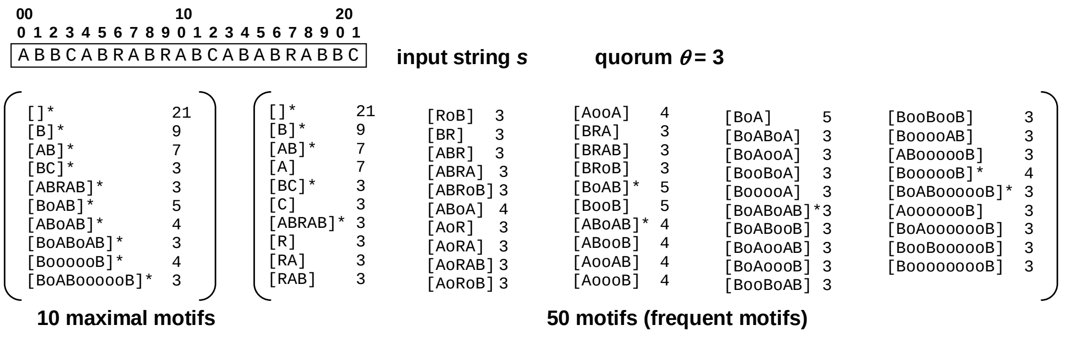
First we show exponential lower bound and succinctness results on the number of maximal motifs, which show the limit of a straightforward approach. By examining the previous approaches [8, 12, 11], we present a simple output-polynomial time algorithm for maximal motif enumeration by breadth-first search, 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 $O(n^2)$ 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 for duplication 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. In Section 3 gives lower bound results on the number of maximal motifs. In Section 4, we prepare tools for studying maximal motifs. In Section 5, we review the previous results on maximal motif enumeration. In Section 6, we present our algorithm MaxMotif that enumerates all maximal motifs with polynomial space and polynomial delay from an input string. 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 [12, 11]. We denote the set of all integers by Z. Given an alphabet Δ, a *string* over Δ is a consecutive sequence of letters $s = a[0] \cdots a[n-1] \in \Delta^*$, where $n \geq 0$ and $a[i] \in \Delta$ for every $0 \leq i \leq n-1$. The *length* of s is $|s| = n$ and the *empty string* is the string ε of length 0. If $s = uvw$ for some $u, v, w \in \Delta^*$, then we say that u is a *prefix*, v is a *substring*, and w is a *suffix* of s. For every $0 \leq i \leq j \leq n-1$, $s[i..j]$ denotes the substring


00 10 20
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
A B B C A B R A B R A B C A B A B R A B B C
input string *s*
quorum *θ* = 3
10 maximal motifs
50 motifs (frequent motifs)

[]*	21
[B]*	9
[AB]*	7
[BC]*	3
[ABRAB]*	3
[BoAB]*	5
[ABoAB]*	4
[BoABoAB]*	3
[BooooB]*	4
[BoABooooB]*	3

[]*	21	[RoB]	3	[AooA]	4	[BoA]	5	[BooBooB]	3		
[B]*	9	[BR]	3	[BRA]	3	[BoABoA]	3	[BooooAB]	3		
[AB]*	7	[ABR]	3	[BRAB]	3	[BoAooA]	3	[ABooooB]	3		
[A]	7	[ABRA]	3	[BRoB]	3	[BooBoA]	3	[BooooB]*	4		
[BC]*	3	[ABRoB]	3	[BoAB]*	5	[BooooA]	3	[BoABooooB]*	3		
[C]	3	[ABoA]	4	[BooB]	5	[BoABoAB]*	3	[AooooooB]	3		
[ABRAB]*	3	[AoR]	3	[ABoAB]*	4	[BoABooB]	3	[BoAooooooB]	3		
[R]	3	[AoRA]	3	[ABooB]	4	[BoAooAB]	3	[BooBooooB]	3		
[RA]	3	[AoRAB]	3	[AooAB]	4	[BoAoooB]	3	[BooooooooB]	3		
[RAB]	3	[AoRoB]	3	[AoooB]	4	[BooBoAB]	3				

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.

$a_i a_{i+1} \cdots a_j$. 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 $s \in \Sigma^*$ of solid characters. In what follows, we fix an arbitrary input string $s = s[1] \cdots s[n] \in \Sigma^*$ of length $n \geq 1$.

Definition 1 (pattern [8, 12, 11]). A *pattern* over Σ is a string x in $\Sigma(\Sigma \cup \{\circ\})^*\Sigma$ 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, ABC and B $\circ$ C are patterns, but $\circ$BC and $\circ$ $\circ$ 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.

Definition 3 (occurrence [8, 12, 11]). 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. The empty pattern ε occurs at any $0 \leq p \leq n - 1$. 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 $x =$ B $\circ$ D occurs in pattern $y =$ AB $\circ$ DA at position 1, and x occurs three times in string $s =$ EABCDABEDBCDE at positions $2, 6$ and 9.

We extend binary relation $\preceq$ from letters to patterns as follows. For patterns x and y in $\Sigma(\Sigma \cup \{\circ\})^*\Sigma$, 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 patterns in $\Sigma(\Sigma \cup \{\circ\})^*\Sigma$. 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 $x \prec z \prec y$. A *location list* is any subset $\mathcal{L} \subseteq \{0, \ldots, n - 1\}$.

Definition 4 (location list [8, 12, 11]). 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 on s at which x occurs. The cardinality $|\mathcal{L}(x)|$ is called the *frequency* of x on s.

Example 2. Consider the input string $s =$ EABCDABEDBCDE over $\Sigma = \{$A, B, C, D, E$\}$. Then, the location list of pattern $x =$ B $\circ$ D in s is $\mathcal{L}(x) = \{2, 6, 9\}$.

Lemma 1. *For any motifs x and y, the following properties hold.*

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

1. If $x \preceq y$ then $\mathcal{L}(y) \subseteq \mathcal{L}(x) + d$ for some nonnegative integer d.
2. If $x \prec y$ then $|\mathcal{L}(x)| > |\mathcal{L}(y)|$ and $|x|_\Sigma < |y|_\Sigma$ hold, where $|x|_\Sigma$ is the number of solid letters in x.

The converse of properties 1 and 2 of Lemma 1 does not hold in general. A *minimum frequency threshold*, or *quorum*, is any positive number $\theta \geq 1$. 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 [8, 12, 11]. 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 $y = -x$.

Definition 6 (Parida *et al* [8]). 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.

Example 3. Let $s = \texttt{EABCDABEDABCDE}$ over $\Sigma = \{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}, \texttt{E}\}$ be an input string. Consider motifs $x = \texttt{AB} \circ \texttt{D}$ with location list $\mathcal{L}(x) = \{1, 5, 8\}$, $y = \texttt{B} \circ \texttt{D}$ with $\mathcal{L}(y) = \{2, 6, 9\}$, and $z = \texttt{D}$ with $\mathcal{L}(z) = \{4, 8, 11\}$. We can see that $z \prec y \prec z$ 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.

Now, we state our problem as follows.

Definition 7. 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 Enumeration algorithms

We introduce terminology for enumeration algorithms according to [7, 14]. An *enumeration algorithm* for an enumeration problem Π is an algorithm $\mathcal{A}$ that receives an *instance* I of Π and produces all *solutions* S in the *answer set* $\mathcal{S}(I)$ of Π, e.g., maximal motifs in our problem, into a write-only output stream O in which each solution appears exactly once.

Let $\mathcal{A}$ be an enumeration algorithm, $N = ||I||$ and $M = |\mathcal{S}(I)|$ be the input and the output sizes on I, $p(N), q(N, M)$ be polynomials. $\mathcal{A}$ is of *output-polynomial* (P-OUTPUT) if the total running time of $\mathcal{A}$ for all solutions 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 $p(N)$. That is, the total running time is linear, $M \cdot p(N)$, in the output size M. $\mathcal{A}$ is of *polynomial delay* (P-DELAY) or *exact polynomial enumeration time* 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, without the size of output stream O, is bounded by a polynomial $p(N)$. We often call $\mathcal{A}$ in P-ENUM and P-DELAY, respectively, a *polynomial time enumeration* algorithm and an *exact polynomial time enumeration* algorithm. By definition, P-OUTPUT is weakest and P-DELAY is strongest among P-OUTPUT, P-ENUM, and P-DELAY.

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 $|\mathcal{M}_\theta|$ is obviously $2^{O(N)}$.

Theorem 1 (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: We show a sketch of the proof. Let $\Sigma = \{\texttt{\#}, \texttt{0}, \texttt{1}\}$ and $n \geq 1$ be any nonnegative integer. Let $s = \texttt{\#}t_1\texttt{\#}\cdots\texttt{\#}t_n\texttt{\#}$ over Σ be the input string corresponding a boolean matrix with 1's on the diagonal, where each block $t_i = b_1 \cdots b_n \in \{0,1\}^n$ $(1 \leq i \leq n)$ is defined as: $b_j = \texttt{1}$ if $i = j$ and $b_j = \texttt{0}$ otherwise for every $j = 1, \ldots, n$. Next, we define the set $\mathcal{X}$ of patterns consisting of all patterns of the form $x = \texttt{\#}p_1 \ldots p_n\texttt{\#} \in \{\texttt{\#}\}\cdot\{\texttt{0}, \circ\}^n\cdot\{\texttt{\#}\}$, which represents a bit vector of length n with $\circ$ as on and 0 as off bits. Then, we can show that $\mathcal{L}_x$ contains the position of i-th block t_i iff p_i is $\circ$ (on bit). This means that all of 2^n patterns have mutually distinct location lists, and thus, they are maximal in s if $|\mathcal{L}_x| \geq \theta$. If $\theta = \frac{1}{2}n$ then the size $|\mathcal{X} \cap \mathcal{M}|$ is bounded below by $\sum_{k=\theta}^{n} \binom{n}{k} = 2^{\Omega(n)}$. □

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

Theorem 2 (succinctness of maximal motifs). *There is an infinite series of input strings $s_0, s_1, s_2, \ldots$, such that for every $i \geq 0$ with quorum $\theta = \frac{1}{2}N$, the number $F = |\mathcal{F}|$ of motifs in s_i is exponential (more precisely $2^{\Omega(N)}$) in the input size N, while the number $M = |\mathcal{M}|$ of maximal motifs in s_i is linear in N, where $N = |s_i|$.*

See Section A of Appendix for the proof of the above theorem. From Theorem 2, we know that a straightforward algorithm for $\mathcal{M}$ based on enumeration of motifs does not work efficiently. This is also true for most real world datasets. Fig. 1 shows an example, where there are only 10 maximal motifs among 50 motifs in a string of length 21.

4 Merge and Closure

In this section, we introduce two operations, called merge and closure, for studying maximal motifs [1, 3, 12, 11]. We start with technical definitions.

For alphabet Δ, we denote by Δ^∞ the class of infinite strings, which are functions of the form $\alpha : \mathsf{Z} \to \Delta$. For a string x of length $m \geq 0$, we define its *infinite version* (or *expanded* version) $\lfloor x \rfloor$ as an infinite string $\lfloor x \rfloor \in (\Sigma \cup \{\circ\})^\infty$ such that for every integer i, $\lfloor x \rfloor[i] = x[i]$ if index i is defined, i.e., $0 \leq i \leq n-1$, and $\lfloor x \rfloor[i] = \circ$ otherwise. Conversely, for infinite string $x \in (\Sigma \cup \{\circ\})^\infty$, its *finite string version* (or *trimmed* version) is the finite string $\lceil x \rceil \in \{\varepsilon\} \cup \Sigma \cup (\Sigma\cdot(\Sigma \cup \{\circ\})^*\cdot\Sigma)$ obtained from x by taking the longest substring that starts and ends with a solid character. If x contains no solid character, $\lceil x \rceil$ is not defined. 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 $x = \circ\texttt{B}\circ\texttt{C}\circ$.

For infinite string x and integer d, we define the *shift of* x with displacement d by the infinite string $(x + d)$ such that $(x + d)[i] = x[i - d]$ for every integer i.

Definition 8 (shift). For finite string x and integer d, the *shift* of x by displacement d is an infinite string $(x + d) = \lfloor x \rfloor + d$.

Example 4. Given a finite string $s = \texttt{EABCDABED}$, its infinite version is $\lfloor s \rfloor = \cdots\circ\circ\downarrow\texttt{EABCDABED}\circ\circ\cdots$, where $\downarrow$ indicates the position of origin $i = 0$. With displacement $d = 2$, the shift of $\lfloor s \rfloor$ is $(\lfloor s \rfloor + 2) = \cdots\circ\circ\texttt{EA}\downarrow\texttt{BCDABED}\circ\circ\cdots$. Again, its finite version is $\lceil(\lfloor s \rfloor + 2)\rceil = \texttt{EABCDABED} = \texttt{s}$, where $\downarrow$ is eliminated.

4.1 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, the *merge* of α and β, denoted by $x \oplus y$, as the finite string or pattern $x \oplus y = \lceil\, \lfloor x \rfloor \oplus \lfloor y \rfloor\, \rceil$.

Let $X = \{\alpha_1, \ldots, \alpha_k\}$ be any set of possibly infinite strings. Since $\oplus$ is associative and commutative, we denote by $\alpha_1 \oplus \cdots \oplus \alpha_k = \bigoplus_{i=1}^{k} \alpha_i = \bigoplus X$ pairwise applications of $\oplus$ to all elements of X from left to right.

○○○○○○AB↓BCABRABRABCABABRABBC	$s-2$
○○○ABBCA↓BRABRABCABABRABBC○○○	$s-5$
ABBCABRA↓BRABCABABRABBC○○○○○○	$s-8$
○○○○○○○○↓B○AB○AB○○○○○○○○○○○○○	$(s-2)\oplus(s-5)\oplus(s-8)$
B○AB○AB	$\lceil(s-2)\oplus(s-5)\oplus(s-8)\rceil = Clo(x)$

Fig. 2. Computing the closure $Clo(x)$ of pattern $x = \mathtt{B} \circ \mathtt{AB}$ in input string $s =$ ABBCABRABRABCABABRABBC.

Definition 10 (merge of location list [3, 11]). Let s be an input string s and $\mathcal{L} = \{d_1, \ldots, d_k\} \subseteq \{0, \ldots, n-1\}$ be any location list on s. Then, the *merge* of $\mathcal{L}$, denoted by $\bigoplus \mathcal{L}$, is defined by the pattern $\bigoplus \mathcal{L} = \bigoplus_{i=1}^{k}(s + d_i)$ if $\bigoplus_{i=1}^{k}(s + d_i)$ is finite, i.e., it starts and ends with a solid character. Otherwise, $\bigoplus \mathcal{L}$ is the empty string ε.

Lemma 4. *Let $\mathcal{L}, \mathcal{L}'$ be any location lists. Then, the following* (1) *and* (2) *hold:* (1) *If $\mathcal{L} \subseteq \mathcal{L}'$ then $\bigoplus \mathcal{L}' \preceq \bigoplus \mathcal{L}$.* (2) *$\bigoplus \mathcal{L} = \bigoplus(\mathcal{L} + d)$ for any integer d.*

Definition 11 (closure operation [11, 12]). Given a pattern x, the *closure* [4] of x on s is the pattern $Clo(x) = \bigoplus \mathcal{L}(x)$.

Example 5. In Fig. 2, we show an example computation of the closure of pattern $x = \mathtt{B} \circ \mathtt{A}$ with $\mathcal{L}(x) = \{2, 5, 8\}$ on input string $s =$ ABBCABRABRABCABABRABBC, where indices start from zero. We first shift s leftward for each position d of x. This operation sets the origin $\downarrow$ of s to each occurrence d. Then, we take the merge $t = (s-2) \oplus (s-5) \oplus (s-8)$ by finding a common solid letter in each column one by one. Finally, we obtain the closure $Clo(x) = \lceil t \rceil = \mathtt{B} \circ \mathtt{AB} \circ \mathtt{AB}$ by trimming the resulting string.

Lemma 6. *$Clo(x)$ is always defined, unique, and computable in $O(n|\mathcal{L}(x)|)$ time given motif x, its location list $\mathcal{L}(x)$, and input string s of length n.*

Lemma 7 (properties of closure). *Let x, y be any patterns occurring in s and X, Y be any location lists.*

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

Theorem 4 (characterization of maximal motifs [11]). *For any motif x in s, x is maximal in s iff $Clo(x) = x$ holds.*

Theorem 5. *Let $x, y \in \mathcal{M}$ be maximal motifs. Then, $x \preceq y$ iff $\mathcal{L}(x) + d \supseteq L_y$ for some integer d. Furthermore, $x = y$ iff $\mathcal{L}(x) + d = \mathcal{L}(y)$ for some integer d.*

Proof: The theorem immediately follows from Theorem 4. □

5 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.

5.1 Frequent pattern generation.

A most straightforward method of generating maximal motifs is to enumerate all motifs in 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.

[4] The closure operation $\bigoplus \mathcal{L}(x)$ for motifs was introduced in [11] and called the *maximal extension* in [12]. The set-counterpart of the closure has been known in data mining [10, 15].

```
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   foreach $y \in \mathcal{M}^i$ and $d \in \{0, \dots, n-1\}$ 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 $x$;
6   $\mathcal{M}_{i+1} := \Delta$; $i := i+1$;
7 end
```

Fig. 3. 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.

5.2 Using the basis of motifs.

Parida *et al* [8] 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 $\mathcal{M}$. Parida *et al.* [8] defined the basis $\mathcal{B}_I$ of *irredundant motifs*. Pisanti *et al.* [12] introduced the basis $\mathcal{B}_T$ of *tiling motifs*. A maximal motif is *tiling* if for any maximal motifs $y_1, \dots, y_k$ and any integers $d_1, \dots, d_k$ with $x \preceq y_i$, if $\mathcal{L}(x) = \bigcup_i \mathcal{L}(y_i)$ then $x = y_i$ for some i. Pelfrêne *et al* [11] introduced the basis $\mathcal{B}_P$ of *primitive motifs*. Pisanti *et al.* [13] propose a simple method for generating $\mathcal{M}$ from $\mathcal{B}_T$ by iteratively computing the merge $x = y \oplus (z+d)$ for all possible maximal motifs $y, z \in \mathcal{M}$ generated so far and all displacements d with explicit check of duplication. This method requires $O(|\mathcal{M}|^2 \cdot n)$ total time and $O(||\mathcal{M}||)$ space given $\mathcal{B}_T$. Since the total time is not linear in $|\mathcal{M}|$, it does not have amortized polynomial enumeration time per motif while it has output-polynomial time.

5.3 An improved method with using the basis.

We can improve Pisanti *et al.*'s method for $\mathcal{M}$ adopting an idea used in [11] for generation of $\mathcal{B}_P$ (and $\mathcal{B}_T$) from s. Fig. 3 shows our algorithm MaxBasis that computes $\mathcal{M}$ from $\mathcal{B}_T$ based on the next lemma:

Lemma 8 (generation of maxmal motifs from the basis). *Let $\theta \geq 1$ be a quorum. 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 some integer d such that $x \prec y$ and $x = y \oplus (s+d)$.*

Theorem 6. *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 Fig. 3 enumerates all maximal motifs of $\mathcal{M}$ from $\mathcal{B}$ in $O(n^2)$ amortized time per motif with $O(||\mathcal{M}||)$ space.*

Here, we are given $\mathcal{B}_T$ and $|\mathcal{B}_T| \leq |\mathcal{M}|$. We use a trie to store $\mathcal{M}^i$ and Δ for $O(|x|)$ membership of pattern x. Since the total time is $O(|\mathcal{M}| \cdot n^2)$, linear in the output size, MaxBasis has polynomial amortized time per motif. However, its space complexity and delay are $O(||\mathcal{M}||)$ and $O(|\mathcal{M}_T| \cdot n^2)$, respectively, in the worst case. Thus, MaxBasis is not a polynomial space polynomial delay algorithm even given the 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 [12, 13, 11]. For every integer $i \geq 1$, let us denote the basis for quorum i by $\mathcal{B}_T^i$. The total running time of the algorithms in [11] and [13] are only bounded by $O(n^\theta \sum_{i=1}^{\theta} |\mathcal{B}_T^i|)$ or $n^{O(\theta)}$, which may not be $O(|\mathcal{B}_T|) = O(|\mathcal{B}_T^\theta|)$. Hence, it seems difficult to obtain output-polynomial time algorithm for $\mathcal{B}_T$ and thus $\mathcal{M}$ in this approach. [5]

[5] Parida *et al.* [9] 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 [8]. Since these algorithms seem to depend on an unproved conjecture in [8], however, we did not include them. At least, the algorithm in [9] requires the space and the delay proportional to the output size $|\mathcal{M}|$. Thus, it is not polynomial space and polynomial delay.

6 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 maximal motifs x in s in $O(|\mathcal{L}(x)| \cdot n^2)$ delay and $O(|\mathcal{L}(x)| \cdot |x|)$ space in the input size n. In what follows, we fix input string s of length $n \geq 1$ and $1 \leq \theta \leq n$.

6.1 Building tree-shaped search route for maximal motifs

Our algorithm MAXMOTIF in Fig. 5 computes all maximal motifs in exactly $O(n^3)$ time per motif, thus with $O(n^3)$ delay, and with $O(n^2)$ space using depth-first search over $\mathcal{M}$, which enables us to avoid the use of extra storage for keeping all discovered motifs. We explain the details of the algorithm in the following sections.

We first build a tree-shaped search route $\mathcal{T} = (\mathcal{V}, \mathcal{P}, \perp)$ for traversing all maximal patterns (Fig. 4). The node set $\mathcal{V} = \mathcal{M}$ consists of all maximal motifs of $\mathcal{M}$, $\mathcal{P}$ is the set of reverse edges defined later, and $\perp = Clo(\varepsilon)$ is the root motif called the *bottom motif*.

Lemma 9. *$\perp = Clo(\varepsilon)$ is the unique shortest maximal motif in s.*

Proof: Clearly, $\mathcal{L}(Clo(\varepsilon)) = \mathcal{L}(\varepsilon)$ is the largest location list $\{0, \ldots, n-1\}$. Thus it follows from Lemma 5 that $Clo(\varepsilon) \preceq x$ for any maximal motif x. □

If s contains at least two solid letters then $\perp = \varepsilon$, otherwise $\perp = a$ for the only letter a in s. 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 $y = x[0..\ell]$.

Then, we assign the unique parent to each non-bottom maximal motifs.

Definition 12 (parent of maximal motif). Let y be a maximal motif such that $y \neq \perp$ and $\ell =$ *core_i*(y) be the core index of x. Then, the *parent* of y, denoted by $\mathcal{P}(y)$, is the pattern $\mathcal{P}(y) = Clo(\lceil y[0..\ell-1] \rceil)$. [6]

Lemma 10. *For every maximal motif x such that $x \neq \perp$, $\mathcal{P}(x)$ is always exists, unique, and maximal. Furthermore, $\mathcal{P}(x) \prec x$ holds.*

Proof: Let $p = \lceil y[0..\ell-1] \rceil$. If $y \neq \perp$, then $\ell - 1 =$ *core_i*$(y) - 1 \geq -1$ and p is always defined. Since $p \preceq y$ and $\mathcal{L}(y) \neq \emptyset$, $\mathcal{L}(p) \supseteq \mathcal{L}(y) \neq \emptyset$. From Theorem 4, $\mathcal{P} = Clo(p)$ is uniquely defined and maximal. if ℓ is the core index of y, $\mathcal{L}(p) \supset \mathcal{L}(y)$. Since both of x, y are maximal, it follows from Lemma 4 and Theorem 5 that $x = Clo(p) \prec Clo(y) = y$. □

Lemma 11. *The relation $\preceq$ is* acyclic *on $\mathcal{M}$, i.e., there is no infinite decreasing chain $x_0 \succ x_1 \succ \cdots \succ x_i \succ \cdots$, $i \geq 0$, of maximal motifs of $\mathcal{M}$.*

Theorem 7. *$\mathcal{T} = (\mathcal{V}, \mathcal{P}, \perp)$ is a spanning tree for all maximal motifs in $\mathcal{M}$.*

The remaining task is to show how to enumerate all children y of a given parent motif x without using extra space, which is not an easy task since we have only reverse edges. We discuss this issue in the next subsection.

6.2 Prefix-preserving closure extension

We introduce the prefix-preserving closure extension defined as follows. A *substitution* for motif x is a pair $\xi = \langle k \leftarrow c \rangle \in \mathrm{Z} \times \Sigma$ of integer k and solid letter c. If $x[k] = \circ$, ξ is *compatible* to x. The *application* of ξ to x is the motif $x\xi = x\langle k \leftarrow c \rangle$ defined as follows. For finite string x with index k *inside* the string, i.e., $0 \leq i \leq |x| - 1$, we define $x\xi$ as

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

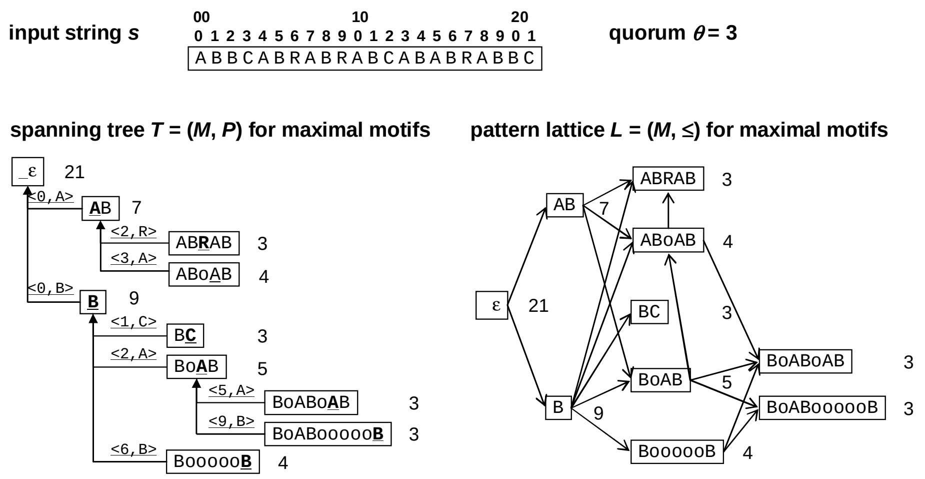


Fig. 4. The spanning tree $\mathcal{T} = (\mathcal{M}, \mathcal{P})$ (left) and the pattern lattice $\mathcal{L} = (\mathcal{M}, \preceq)$ (right) for maximal motifs of $\mathcal{M}$ on quorum $\theta = 3$ and input string s (top). Each box represents maximal motif x in $\mathcal{M}$ and the number right to the box indicates its frequency $|\mathcal{L}(x)|$. Each arrow indicates ordering, $\mathcal{P}$ or $\preceq$, of a tree/lattice. (Sec. 6.1). 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. (Sec. 6.2). There are 10 maximal motifs among 49 motifs in s.

follows (ξ is called *replacement*): for every index i, $x\xi[i] = c$ if $i = k$ and $x\xi[i] = x[i]$ otherwise. For infinite string x, we define $x\xi$ in the same way. For finite string x with the index k *outside* the string, i.e., $k > |x| - 1$ or $k < 0$ (ξ is called *right extension* or *left extension*), we define $x\xi = \lceil \lfloor x \rfloor \xi \rceil$. 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 13 (ppc-extension). For any maximal motifs x, y such that $y \neq \bot$, a motif y is a *prefix-preserving closure expansion* (or a *ppc-extension*) of x if the following (i)–(iii) hold:

(i) $y = \mathit{Clo}(x\langle k \leftarrow c\rangle)$ for some substitution, called the *seed*, $\xi = \langle k \leftarrow c\rangle \in \mathsf{Z} \times \Sigma$ compatible to x, that is, y is obtained by first substituting c at index i and then taking its closure,
(ii) the index k satisfies $k > \mathit{core\_i}(x)$, and
(iii) $x[0..k-1] = y[0..k-1]$, that is, the prefix of length $i-1$ is preserved, where $x[0..k-1]$ is the string $\lfloor x \rfloor [0..k-1]$ obtained from x by padding trailing $\circ$'s if necessary.

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

The following theorem is the main result of this section. For the proof, please see Section B of Appendix.

Theorem 8 (correctness of ppc-extension). *For any maximal motifs x, y such that $y \neq \bot$. Then, (i) $x = \mathcal{P}(y)$ if and only if (ii) $y = \mathit{Clo}(x\xi)$ is a prefix-preserving closure expansion of x for some substitution $\xi = \langle k \leftarrow c\rangle \in \mathsf{Z} \times \Sigma$ compatible to x. Furthermore, there exists exactly one ξ satisfying condition (ii) for each y.*

6.3 A polynomial space polynomial delay algorithm

Based on Theorem 8, we present in Fig. 5 our algorithm MaxMotif that enumerates all maximal motif in a given input string by the depth-first search over $\mathcal{M}$ applying the ppc-extension to each maximal motifs.

```
Algorithm MaxMotif($\theta$: quorum, $s$: input string)
1 Expand($\bot, \theta, s$);

Procedure Expand($x$: motif, $\theta$: quorum, $s$: input string)
2 if $|\mathcal{L}(x)| < \theta$ then return;
3 output $x$;
4 for $k := core\_i(x) + 1$ to $|s|$ do
5   foreach $c \in \Sigma$ do begin
6     $y = Clo(x\langle k \leftarrow c\rangle)$;
7     if $x[0..k-1] = y[0..k-1]$ then
8       call Expand($y, s, \theta$);
9   end for
```

Fig. 5. A polynomial space polynomial delay enumeration algorithm for $\mathcal{M}$.

A straightforward implementation of the procedure Expand in Fig. 5 requires $O(|\mathcal{L}(x)| \cdot n)$ time for each of $n \cdot |\Sigma|$ possible children at line 4 to line 9 even when none of them satisfies the quorum θ. This only yields an algorithm with $O(|\Sigma| \cdot |\mathcal{L}(x)| \cdot n^2) = O(|\Sigma| n^3)$ time and delay. Then, we have the following theorem.

Theorem 9. *Given a quorum $\theta \geq 1$ and an input string s of length n, the algorithm* MaxMotif *in Fig. 5 enumerates all maximal motifs x of $\mathcal{M}$ in $O(|\mathcal{L}(x)| \cdot n^2)$ amortized time per motif with $O(|\mathcal{L}(x)| \cdot |x|)$ space and $O(|\mathcal{L}(x)| \cdot n^2)$ delay.*

Proof: By Theorem 7 and Theorem 8, we see that the algorithm MaxMotif visits all maximal motifs on the spanning tree $\mathcal{T}$ starting from the root $\bot$. Since $\mathcal{T}$ is a tree and any maximal motif appears in $\mathcal{T}$, the algorithm enumerate $\mathcal{M}$ without duplicates.

Now, we estimate the work time $W(x)$ at line 2 and at line 4 to line 8 for each parent maximal motif x as follows. For each x, its location list $\mathcal{L}(x)$ at line 2 and its its closure $Clo(x)$ at line 6 can be computed in $O(|\mathcal{L}(x)| \cdot n)$ time. By Lemma 15, if x is obtained from its parent p by ppc-extension with seed $\langle k, c\rangle$, then $core\_i(x) = k$, and this can be computed in $O(1)$ time at line 4 by keeping k. Thus, a straightforward implementation, requires the total computation time $O(|\Sigma| \cdot |\mathcal{L}_x| \cdot n)$ at line 5 to line 8.

To reduce computation time further, we replace line 5 to line 8 by the following procedure OccurrenceDeliver [7] that computes $y = Clo(x\langle k \leftarrow c\rangle)$ and $\mathcal{L}(y)$ for all solid letter c such that $\mathcal{L}(y) \neq \emptyset$ during a single scanning on $\mathcal{L}(x)$. Its total running time is $O(\sum_{c \in \Sigma'} |\mathcal{L}(c)| \cdot n) = O(|\mathcal{L}(x)| \cdot n)$ time.

```
OccurrenceDeliver ≡
1 Initialize all $\mathcal{L}(c) := \emptyset$ for all $c \in \Sigma'$, and then $\Sigma' := \emptyset$;
2 foreach $d \in \mathcal{L}(x)$ do
3   $\mathcal{L}(c) := \mathcal{L}(c) \cup \{d\}$ and $\Sigma' = \Sigma' \cup \{c\}$, where $c := s[d+k]$;
5 foreach $c \in \Sigma'$ do
6   Compute $y = Clo(x\langle k \leftarrow c\rangle)$ using $\mathcal{L}(c)$
```

This yields the work $W(x) = O(|\mathcal{L}(x)| \cdot n^2)$ instead of $O(|\Sigma| \cdot |\mathcal{L}(x)| \cdot n^2)$. From this, the delay is $W(x) \cdot |x|$ time since the depth of $\mathcal{T}$ is $\Theta(|x|)$. Next, we reduce the delay $W(x) \cdot |x|$ to $W(x)$ using a technique by Uno [14] that transforms any tree-search enumeration algorithm with work time $W(x)$ at each node into a $W(x)$ delay enumeration algorithm by changing the timing of output alternatively according to the parity of the depth of x. For details, please consult [14]. The space complexity is obviously $O(|\mathcal{L}(x)| \cdot |x|)$ since each recursive call has $|x|$ ancestors. □

[7] Occurrence deliver is introduced for closed itemsets enumeration in Uno *et al.* [15]

In summary, MaxMotif computes all maximal motifs in $O(n^3)$ time per motif with $O(n^2)$ space and $O(n^3)$ delay in the input size n.

Corollary 10 *The maximal motif enumeration problem is solvable in polynomial space and polynomial delay in the input size $n = |s|$.*

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 realized by the prefix-preserving expansion technique we introduced, the algorithm enumerate all motifs in the unique way without explicitly storing and checking the motifs enumerated so far. This method drastically improves the space and the delay complexities of the previous output-polynomial time algorithms based on breadth-first search.

In data mining, maximal motifs for *sets* are called *closed itemsets* [10]. There are a number of closed itemset discovery algorithms for closed itemsets [5, 10], while only a few algorithms are known for closed sequences and closed trees [3, 15, 16]. We extend the ppc-extension, which is originally introduced for closed itemsets, for maixmal sequences. Thus, 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.

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 [12], 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 and L. Parida, Compression and the wheel of fortune, In *Proc. the 2003 Data Compression Conference (DCC'03)*, IEEE, 2003.
2. H. Arimura, T. Uno, A Polynomial Space Polynomial Delay Algorithm for Enumeration of Maximal Motifs in a Sequence, Technical Report TCS-TR-A-05-6, Division of Computer Science , Hokkaido Univeristy, July 2005.
3. H. Arimura, T. Uno, An Output-Polynomial Time Algorithm for Mining Frequent Closed Attribute Trees, In *Proc. Inductive Logic Programming*, LNAI 3625, Springer, 1–19, August 2005. (to appear)
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. 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.
9. L. Parida *et al.*, An output-sensitive algorithm for ..., In *Proc. the 12th International Conference on Combinatorial Pattern Matching (CPM'01)*, LNCS 2089, 131–142, 2001.
10. N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal, Discovering Frequent Closed Itemsets for Association Rules, In *Proc. ICDT'99*, 398–416, 1999.
11. J. Pelfrêne, S. Abdedda¨m, and J. Alexandre, Extending Approximate Patterns, In *Proc. CPM'03*, LNCS 2676, 328–347, 2003.
12. 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.
13. N. Pisanti, M. Crochemore, R. Grossi, and M.-F. Sagot, A comparative study of bases for motif inference, In *String Algorithmics*, C. Iliopoulos and T. Lecroq editors, KCL publications, 2004.
14. T. Uno, Two General Methods to Reduce Delay and Change of Enumeration Algorithms, NII Technical Report, NII-2003-004E, April 2003.

15. T. Uno, T. Asai, Y. Uchida, H. Arimura, An efficient algorithm for enumerating closed patterns in transaction databases, In *Proc. DS'04*, LNAI 3245, Springer-Verlag, 16-30, 2004.
16. X. Yan, J. Han, CloseGraph: Mining Closed Frequent Graph Patterns In *Proc. SIGKDD'03*, 2003.

Appendix

This appendix is not included in the submission and can be ignored

A Proof of Theorem 2

Lemma 3 (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 $O(N + m^2)$ such that the number M^r_θ of θ-frequent closed itemsets in r equals to the number M^s_θ of maximal motifs in s.*

Proof: (Sketch) Given items $\mathcal{I} = \{1, \ldots, n\}$ and a transaction database $r = \{t_1, \ldots, t_m\}$, $\Sigma = \{\ \mathtt{a}_j, \mathtt{b}^i_j, \#^i_j \mid \mathtt{a} \in \mathcal{I}, 1 \le i \le m, 1 \le j \le n\ \} \cup \{\#\}$ is an alphabet for s. The letters $\mathtt{a}_j, \mathtt{b}^i_j$, and $\#^i_j$, resp., stand for items, blanks, and delimiters. We define the input string $s = s_1 w_1 \cdots s_{n-1} w_{n-1} s_n w^n$ over Σ of length $N = mn + \frac{1}{2}m^2$. For every $i = 1, \ldots, n$, let t_i be the i-th transaction of r. Then, the block s_i is defined by $s_i = x_1 \cdots x_n$ such that for every $j = 1, \ldots, n$, $x_j = \mathtt{a}_j$ if $j \in t_i$ and $x_j = \mathtt{b}^i_j$ if $j \notin t_i$. The delimiter w_i is defined by $w_i = y_1 \cdots y_i$ such that $y_j = \#^i_j$ for every $j = 1, \ldots, i$. It is important here that $|w_i| = i$ for $i = 1, \ldots, m$. Then, we can see that given quorum $\theta \ge 2$, any motif occurs only inside blocks and consists only of $\circ$ and $\mathtt{a}_j$'s. Furthermore, there is a one-to-one correspondence between all θ-frequent itemsets in r and all θ-motifs in s for $\theta \ge 2$. We see that the lattice $(\mathcal{FI}_\theta, \subseteq)$ of all frequent itemsets in r is isomorphic to the lattice $(\mathcal{F}_\theta, \preceq)$ of all motifs in s (see full paper for details). This induces a one-to-one correspondence between all closed itemsets in r and all maximal motifs in s. Thus, the result immediately follows. □

Theorem 2. *There is an infinite series of input strings $s_0, s_1, s_2, \ldots$, such that for every $i \ge 0$ with quorum $\theta = \frac{1}{2}N$, the number $F = |\mathcal{F}|$ of motifs in s_i is exponential (more precisely $2^{\Omega(N)}$) in the input size N, while the number $M = |\mathcal{M}|$ of maximal motifs in s_i is linear in N, where $N = |s_i|$.*

Proof: Theorem 1 of Uno *et al.* [15] 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 3 above and the constructions of Theorem 1 of [15], where $N = \Theta(m^2)$ for large m and n, we obtain the result. □

B Proof of Theorem 8

In this section, we prove Theorem 8 of Section. We start with the following technical lemmas.

Lemma 12. *For any string $xy \in \Sigma(\Sigma \cup \{\circ\})^*$, $\mathcal{L}(xy) = \mathcal{L}(x) \cap (\mathcal{L}(y) + |x|)$.*

Lemma 13. *Let x, y be any motifs. If $\mathcal{L}(x) = \mathcal{L}(y)$ then $\mathcal{L}(x\xi) = \mathcal{L}(y\xi)$ for any substitution $\xi = \langle k \leftarrow c \rangle$ compatible to both x and y.*

For any string $x \in (\Sigma \cup \{\circ\})^*$, $x[0..k-1]$ denotes the prefix of x with length k if $|x| \ge k$. If $|x| < k$, then $x[0..k-1]$ denotes the string of length k obtained from x by padding trailing $\circ$'s, i.e., $x[0..k-1] = \lfloor x \rfloor [0..k-1]$.

Lemma 14. *Let y be any maximal motif such that $y \ne \bot$ and ℓ be the core index of y. For the parent $x = \mathcal{P}(y)$ of y, $x[0..\ell-1] = y[0..\ell-1]$ holds.*

Proof: Let $x = Clo(y[0..\ell-1]) = \mathcal{P}(y)$. By construction, $y[0..\ell-1] \preceq x[0..\ell-1]$. Suppose to contradict that $x[0..\ell-1] \ne y[0..\ell-1]$, that is, the closure operation filled some position $i < \ell$ in $y[0..\ell-1]$ with a solid letter. Then, we have $y[0..\ell-1] \prec x[0..\ell-1]$ (*1). Now, let $y' = x[0..\ell-1]y[\ell..|y|-1]$ be a new string obtained from y by replacing the prefix $y[0..\ell-1]$ with $x[0..\ell-1]$. Clearly, $y \prec y'$. On the other hand, from (2) of Lemma 7, we have $\mathcal{L}(x[0..\ell-1]) = \mathcal{L}(y[0..\ell-1]) + d$ for some d since $x = Clo(y[0..\ell-1])$ and $\mathcal{L}(Clo(y[0..\ell-1])) = \mathcal{L}(y[0..\ell-1]) + d$ for some d. By Lemma 12, we can show that for any strings $x_1, x_2, y_1, y_2 \in \Sigma(\Sigma \cup \{\circ\})^*$, if $\mathcal{L}(x_1) = \mathcal{L}(x_2)$ and $\mathcal{L}(y_1) = \mathcal{L}(y_2)$ then $\mathcal{L}(x_1y_1) = \mathcal{L}(x_2y_2)$. Thus, if $\mathcal{L}(x[0..\ell-1]) = \mathcal{L}(y[0..\ell-1])$ then we have $\mathcal{L}(y') = \mathcal{L}(y)$ (*2). It follows from (*1) and (*2) that y is not maximal. This is the contradiction, and thus we conclude that $x[0..\ell-1] = y[0..\ell-1]$. □

Lemma 15. *Let x be any maximal motif and $y = Clo(x\langle k \leftarrow c\rangle)$ be a ppc-extension of x. Then, k is the core index of y.*

Proof: Let $\xi = \langle k \leftarrow c\rangle$ be the substitution generating $y = Clo(x\xi)$. (a) By definition of ppc-expansion, $k - 1 \geq core\_i(x)$ and $x[0..k-1] = y[0..k-1]$ hold. Thus, we have $\mathcal{L}(x) = \mathcal{L}(x[0..k-1]) = \mathcal{L}(y[0..k-1])$. Since x is maximal, any substitution ξ results $|\mathcal{L}(x)| \neq |\mathcal{L}(x\xi)| = |\mathcal{L}(y)|$. Therefore, we have $\mathcal{L}(y[0..k-1]) \neq \mathcal{L}(y)$. (b) Next, we show that $\mathcal{L}(y[0..k]) = \mathcal{L}(y)$. Firstly, from assumptions, $x[k] = \circ$, $y[k] = c$, and $x[0..k-1] = y[0..k-1]$ hold. This implies that $y[0..k] = x\xi[0..k]$, and thus that $\mathcal{L}(y[0..k]) = \mathcal{L}(x\xi[0..k])$ (*1). Next, if $k > core\_i(x)$ then we have $\mathcal{L}(x[0..k]) = \mathcal{L}(x)$. By applying Lemma 13, it follows that $\mathcal{L}(y[0..k]) = \mathcal{L}(x[0..k]\xi) = \mathcal{L}(x\xi)$ (*2). Finally, from (2) of Lemma 7, if $y = Clo(x\xi)$ then $\mathcal{L}(x\xi) = \mathcal{L}(Clo(x\xi)) + d = \mathcal{L}(y) + d$ for some d (*3). Combining (*1)–(*3), we have $\mathcal{L}(y[0..k]) = \mathcal{L}(y) + d$ for some d. Since $y[0..k]$ is a prefix of y, we can show that $d = 0$ must hold. From (a) and (b), we know that k is the core index of y. □

Lemma 16. *For any maximal motif y such that $y \neq \perp$, if $x = P(y)$ then y is a prefix-preserving expansion $Clo(x\xi)$ of x for some $\xi = \langle k \leftarrow c\rangle \in \mathsf{Z} \times \Sigma$.*

Proof: Suppose that $x = \mathcal{P}(y)$. Then, $x = Clo(y[0..\ell-1])$ for the core prefix $\ell = core\_(y)$ of y. Let $k = \ell$ and $c = x[\ell]$. Then, we can show that $x[\ell] = \circ$ because $x = Clo(y[0..\ell-1]) \prec Clo(y[0..\ell]) = y$. Thus, the substitution $\xi = \langle \ell \leftarrow c\rangle$ is compatible to x. Now, we show that $y = Clo(x\xi)$ for $\xi = \langle \ell \leftarrow c\rangle$. First, we show $\mathcal{L}(x) = \mathcal{L}(x[0..\ell-1])$. By (2) of Lemma 7, $x = Clo(y[0..\ell-1])$ implies $Clo(y[0..\ell-1]) \preceq x$. By taking the prefix of length k, we have $y[0..\ell-1] \preceq x[0..\ell-1]$. Since $x[0..\ell-1]$ is a prefix of x, we have $x[0..\ell-1] \preceq x$. Therefore, if $y[0..\ell-1] \preceq x[0..\ell-1] \preceq$ then we have $\mathcal{L}(y[0..\ell-1]) \supseteq \mathcal{L}(x[0..\ell-1]) \supseteq \mathcal{L}(x)$ (*1). On the other hand, $x = Clo(y[0..\ell-1])$ since x is the parent of y. Thus, $\mathcal{L}(y[0..\ell-1]) = \mathcal{L}(x)$. From this, the inclusion in the above inclusion formula is actually equality, that is, we can conclude that $\mathcal{L}(x[0..\ell-1]) = \mathcal{L}(x)$. By Lemma 13, this implies $\mathcal{L}(x\xi) = \mathcal{L}(x[0..\ell-1]\xi) = \mathcal{L}(x\xi[0..\ell])$ (*1). On the other hand, by Lemma 14, we know that $y[0..\ell-1] = x[0..\ell-1]$. From this, we can show that $x\xi[0..\ell] = x[0..\ell-1]y[\ell] = y[0..\ell-1][\ell] = y[0..\ell]$, where if the length of $x[0..\ell-1]$ is shorter than k then we appropriately add to $x[0..\ell-1]$ by trailing $\circ$'s. Therefore, we have $\mathcal{L}(x\xi[0..\ell]) = \mathcal{L}(y[0..\ell])$ (*2). Since ℓ is the core index of y, we have $\mathcal{L}(y[0..\ell]) = \mathcal{L}(y)$ (*3). From (*1)–(*3), we have $\mathcal{L}(x\xi) = \mathcal{L}(y)$. By taking the merge operation of the both side, we have $Clo(x\xi) = Clo(y) = y$ since y is a maximal motif and Theorem 4 holds. This completes the proof. □

Lemma 17. *Let x be any maximal motif. If $y = Clo(x\langle k \leftarrow c\rangle)$ is a prefix-preserving expansion of x for some $\langle k, c\rangle \in \mathsf{Z} \times \Sigma$, then $x = \mathcal{P}(y)$ holds.*

Proof: Suppose that $y = Clo(x\langle k \leftarrow c\rangle)$. Then, it follows from Lemma 15 that k is the core index of y. Thus, we have $\mathcal{P}(y) = Clo(y[0..k-1])$. On the other hand, we have $y[0..k-1] = x[0..k-1]$ and thus $Clo(y[0..k-1]) = Clo(x[0..k-1])$ since y is a ppc-extension of x. Since x is maximal, $k - 1 \geq core\_i(x)$, we have $Clo(x[0..k-1]) = x$. Combining the above observations, we have $\mathcal{P}(y) = x$. □

Lemma 18. *For all $i = 1, 2$, let $\xi_i = \langle k_i \leftarrow c_i\rangle \in \mathsf{Z} \times \Sigma$ be substitutions and let $y_i = Clo(x\xi_i)$ be the ppc-extension of x generated by ξ_i. If $\langle k_1, c_1\rangle \neq \langle k_2, c_2\rangle$ then y_1 and y_2 are distinct.*

Proof: Assume that $\xi_1 \neq \xi_2$ and $y_i = Clo(x\xi_i)$ for every $i = 1, 2$. Suppose to contradict that $y_1 = y_2 = y$ for some maximal motif y. Since y is the ppc-expansion of x with $\xi = \langle k_i \leftarrow c_i \rangle$, $x[0..k_i - 1] = y[0..k_i - 1]$ by definition for every $i = 1, 2$. Thus, each $x[0..k_i - 1]$ is a prefix of y. There are two cases below for k_1 and k_2. (a) Suppose first that $k_1 = k_2 = k$. Then, $\xi_1 \neq \xi_2$ implies $c_1 \neq c_2$. However, this is impossible since $c_1 = y[k] = c_2$. (b) Suppose next that $k_1 \neq k_2$. Assume without loss of generality that $k_1 < k_2$. This is again impossible since Since $\xi_1 = \langle k_1 \leftarrow c_1 \rangle$ is compatible to x, we know that $x[k_1] = \circ$ and $y[k_1] = c_1$. On the other hand, $x[0..k_2 - 1] = y[0..k_2 - 1]$ must hold for $k_2 > k_1$ since $y = Clo(x\xi_2)$ is the ppc-extension of x. This contradicts that $\circ = x[k_1] \neq y[k_1] = c_1$. Hence, by contradiction, we conclude that $y_1 \neq y_2$. □

Proof of Theorem 8: From Lemma 16, Lemma 17, and Lemma 18, the theorem immediately follows. □