A 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 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. 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 lowerbound and a succinctness result on the number of maximal motifs, which indicate the limit of a straightforward approach.

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 [9,13]. 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* and 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.

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

X. Deng and D. Du (Eds.): ISAAC 2005, LNCS 3827, pp. 724–737, 2005.

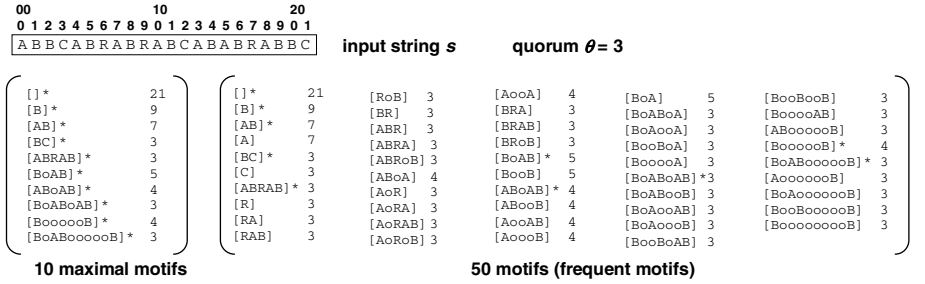

Fig. 1. Examples of maximal motifs (left) and motifs (right) for an input string s and a quorum θ, where * indicates a maximal motif and the number associated to each motif indicates its frequency. These 10 maximal motifs are representatives containing the whole information on the occurrences of all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

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 Fig. 1 all maximal motifs and all motifs on input sting $s = \texttt{ABBCABRABRABCABABRABBC}$ for quorum $\theta = 3$. Then, the pattern $x_1 = \texttt{R} \circ \texttt{B}$ 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 $x_2 = \texttt{ABRAB}$ 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 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.

We first show an exponential lowerbound and a succinctness result on the number of maximal motifs, which show the limit of a straightforward approach. 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 $O(n^2)$ 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. Section 3 gives lowerbounds 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 with polynomial space and polynomial delay from an input string. In Section 6, 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 $\Sigma(\Sigma \cup \{\circ\})^*\Sigma$ 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*.[1] 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 $x =$ 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. 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,

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

we define $\varepsilon \preceq x$. If $x \preceq y$ but $y \npreceq x$, 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 $\mathcal{P}$. 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 $x \prec z \prec y$.

Definition 3 (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 $\theta \geq 1$. 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 [9,13,12]. Let $\mathcal{L}$ be any location list and d be any integer. Then, we define the *shift of $\mathcal{L}$ with displacement d* by $\mathcal{L} + d = \{\, \ell + d \,|\, \ell \in \mathcal{L} \,\}$. We write $\mathcal{L} - x$ to represent the set $\mathcal{L} + y$ with $y = -x$.

Definition 5 (Parida *et al* [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.

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 does not holds 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 $z \prec y \prec z$ 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 6. *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 [1,3,12,13].

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 *varepsilon* otherwise. By definition, $\lceil \lfloor x \rfloor \rceil = x$ for any $x \in \mathcal{P}$. Let d be an integer called 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 $s = \texttt{EABCDABED}$, its infinite version is $\lfloor s \rfloor = \cdots \circ\circ \downarrow \texttt{EABCDABED} \circ\circ \cdots$, where $\downarrow$ indicates the origin $i = 0$. Then, $(\lfloor s \rfloor + 2) = \cdots \circ\circ \texttt{EA} \downarrow \texttt{BCDABED} \circ\circ \cdots$, and its finite version is $\lceil (\lfloor s \rfloor + 2) \rceil = \downarrow \texttt{EABCDABED} = \mathbf{s}$.

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 α and β, 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. For a location list $\mathcal{L} = \{d_1, \ldots, d_{|\mathcal{L}|}\}$, The *merge* of $\mathcal{L}$ ([3,12]) is the pattern $\bigoplus \mathcal{L} \in \mathcal{P}$ defined by

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

Lemma 2. *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.*

Lemma 3. *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.*

Definition 7 (closure operation [12,13]). *Given a pattern x and an input string s, the maximal 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].

Lemma 4. *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 $m = |\mathcal{L}(x)| \leq n$.*

Lemma 5 (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. *$Clo(x) = Clo(Clo(x))$.*
3. *If $x \preceq y$ then $Clo(x) \preceq Clo(y)$.*

Theorem 1 (characterization of maximal motifs [12]). *Let θ a quorum and x be a motif pattern 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) $x = Clo(x)$.

A maximal motif x is the unique maximal element with respect to $\preceq$ in the equivalence class $\{\, y \mid \mathcal{L}(x) = \mathcal{L}(y) + d$ for some integer $d \,\}$.

Lemma 6 ([12]). *Let* $\theta \geq 1$ *be a quorum and* $x, y \in \mathcal{M}$ *be maximal motifs.*

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

Proof. (1) The only-if direction is obvious from Lemma 1. The if direction follows from Theorem **??** and Property 1 of Lemma 5. □

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:

```
○○○○○○AB↓BCABRABRABCABABRABBC  = s − 2
○○○ABBCA↓BRABRABCABABRABBC○○○  = s − 5
ABBCABRA↓BRABCABABRABBC○○○○○○  = s − 8
```

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:

```
○○○○○○○○↓B○AB○AB○○○○○○○○○○○○  = ⨁S
```

Finally, we get the closure $Clo(x) =$ `B○AB○AB` by taking its infinite 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 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 $|\mathcal{M}|$ 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 $n = |s_i|$.*

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

Theorem 3 (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 the full paper [2] for the proofs of the above theorems. From Theorem 3, 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 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 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.

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 maximal motifs $y_1, \ldots, y_k \in \mathcal{M}$ and any integers $d_1, \ldots, d_k$ with $x \preceq y_i$,if

```
Algorithm MaxBasis( θ: quorum, s: input string, B: basis)
1  M^0 := B; i := 0
2  while (Δ ≠ ∅) do begin
3     Δ := ∅;
4     foreach y ∈ M^i and d ∈ {0, ..., n − 1} do
5        if y ⊕ (s + d) ∉ (⋃_{k=0}^{i} M^k ∪ Δ) then Δ = Δ ∪ {y ⊕ (s + d)}; output x;
6     M_{i+1} := Δ; 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.

$\mathcal{L}(x) = \bigcup_i \mathcal{L}(y_i)$ 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 9. *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 $x = y \oplus (s + d)$.*

Fig. 2 shows our algorithm MaxBasis that computes $\mathcal{M}$ from $\mathcal{B}_T$. 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 Fig. 2 enumerates all maximal motifs of $\mathcal{M}$ from $\mathcal{B}$ in $O(n^2)$ amortized time per 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 output-polynomial time algorithm for $\mathcal{B}_T$ and thus $\mathcal{M}$ in this approach. [2]

[2] 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.

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 maximal motifs x in s in $O(|\mathcal{L}(x)| \cdot n^2)$ delay and $O(|\mathcal{L}(x)| \cdot |x|)$ 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 patterns (Fig. 3). 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 called the *root motif*. If s contains at least two solid letters then $\perp = \varepsilon$, otherwise $\perp = a$ for the only letter a in s.

Lemma 10. *$\perp = Clo(\varepsilon)$ 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 6 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 $y = x[0..\ell]$. Then, we assign the unique parent to each non-root maximal motifs.

Definition 8 (parent of maximal motif). *Let y be a maximal motif such that $y \neq \perp$. If $\ell = core\_i(y)$ is the core index of y, then the* parent *of y, denoted by $\mathcal{P}(y)$, is the pattern $\mathcal{P}(y) = Clo(\lceil y[0..\ell-1] \rceil)$.* [3]

Lemma 11. *For every maximal motif y such that $y \neq \perp$, $\mathcal{P}(y)$ is always exists, unique, and maximal. Furthermore, $\mathcal{P}(y) \prec y$ 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. If ℓ is the core index of y, then $\mathcal{L}(p) \supset \mathcal{L}(y) \neq \emptyset$, and thus $\mathcal{P}(y)$ is defined and maximal. Furthermore, if x, y are maximal then Lemma 2 and Theorem 6 imply that $x = Clo(p) \prec Clo(y) = y$. □

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

Proof. From Lemma 11, 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 without using extra space. This is not an easy task since we have only reverse edges. We discuss this issue in the next subsection.

[3] 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.

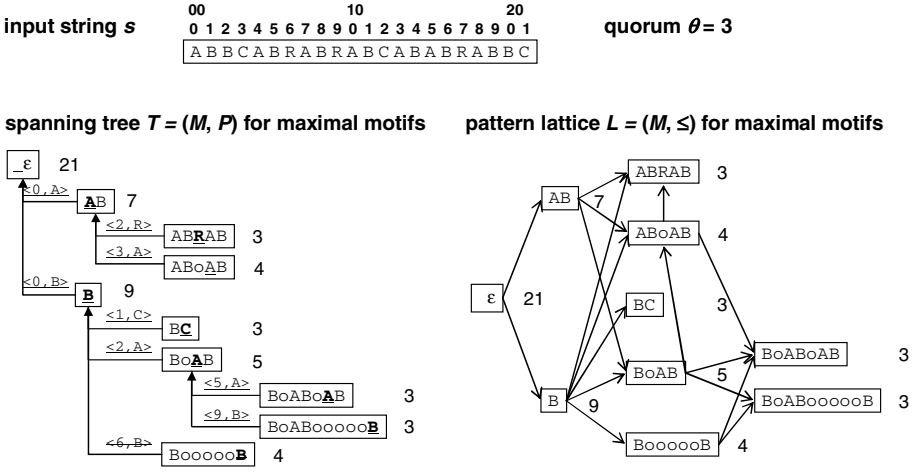

Fig. 3. The spanning tree $\mathcal{T} = (\mathcal{M}, \mathcal{P})$ (left) and the pattern lattice $\mathcal{L} = (\mathcal{M}, \preceq)$ (right) for maximal motifs of $\mathcal{M}$ on quorum $\theta = 3$ and input string s (top). Each box represents maximal motif x in $\mathcal{M}$ and the number right to the box indicates its frequency $|\mathcal{L}(x)|$. Each arrow indicates ordering, $\mathcal{P}$ or $\preceq$, of a tree/lattice. (Sec. 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. (Sec. 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* to 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, y such that $y \neq \perp$, a motif y is a* prefix-preserving closure expansion *(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 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 $k > core\_i(x)$, 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. 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 $\mathcal{L}(x) = \{0, 4, 7, 10, 15, 18\}$. 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 $y = \mathtt{ABRAB} = Clo(\mathtt{AB}\underline{\mathtt{R}}) = Clo(x \cdot \xi_1)$ with $\{4, 7, 15\}$, and $z = \mathtt{AB} \circ \mathtt{AB} = Clo(\mathtt{AB} \circ \underline{\mathtt{A}}) = Clo(x \cdot \xi_2)$ with $\{4, 7, 10, 15\}$.

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 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.*

The following theorem is the main result of this section.

Theorem 7 (correctness of ppc-extension). *For any maximal motifs x, y such that $y \neq \bot$. Then, (1) $x = \mathcal{P}(y)$ if and only if (2) $y = Clo(x\xi)$ 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 (2) for each y.*

Proof. We give a sketch of the proof. Please see the full paper [2] for the details. (1) to (2): Suppose $x = \mathcal{P}(y)$, and thus $x = Clo(y[0..\ell - 1])$ for the core index ℓ of y. First, we can show that $x[0..\ell - 1] = y[0..\ell - 1]$ holds. Otherwise, we can construct a more specific motif $y' = x[0..\ell - 1]y[\ell..|y| - 1]$ such that $y \prec y'$ but $\mathcal{L}(y) = \mathcal{L}(y')$ using Lemma 12. Next, if we take $\xi = \langle \ell \leftarrow y[\ell] \rangle$, then we can show that $y = Clo(x\xi)$ and ξ satisfies the condition of ppc-extension. This proves this direction. (2) to (1): This direction is easy. Suppose that $y = Clo(x\langle k \leftarrow c \rangle)$. Then, it follows from Lemma 13 that $\mathcal{P}(y) = Clo(y[0..k - 1])$. If $y[0..k - 1] = x[0..k - 1]$, then $Clo(y[0..k - 1]) = Clo(x[0..k - 1])$. Since x is maximal, we have $k - 1 \geq core\_i(x)$, and thus $Clo(x[0..k - 1]) = x$. Hence, we have $\mathcal{P}(y) = x$. 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 Fig. 4 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.

A straightforward implementation of the procedure Expand in Fig. 4 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 8. *Given a quorum $\theta \geq 1$ and an input string s of length n, the algorithm* MaxMotif *in Fig. 4 enumerates all maximal motifs x of $\mathcal{M}$ in $O(mn^2)$ amortized time per motif with $O(\ell m)$ space and $O(mn^2)$ delay, where $\ell = |x|$ and $m = |\mathcal{L}(x)|$.*

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 $\bot$. Since

```
Algorithm MaxMotif(θ: quorum, s: input string)
0  ⊥ = Clo(ε);                                  //the root motif ⊥.
1  call Expand(⊥, −1, θ, s);                    //core_i(⊥) = −1.

Procedure Expand(x: motif, ℓ: core index, θ: quorum, s: input string)
2  if |L(x)| < θ then return;
3  output x;
4  for k := ℓ + 1 to |s| do                     //core_i(x) = ℓ is ensured.
5    foreach c ∈ Σ do begin
6      y = Clo(x⟨k ← c⟩);                       //ppt-extension.
7      if ⌊x⌋[0..k − 1] = ⌊y⌋[0..k − 1] then
8        call Expand(y, k, s, θ);
9    end for
```

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

$\mathcal{T}$ is a tree and any maximal motif appears in $\mathcal{T}$, the algorithm enumerate $\mathcal{M}$ without duplicates. Let $W(x)$ be the work that Expand spends for each parent maximal motif x except the recursive call. Since the closure $Clo(x)$ at line 6 takes $O(|\mathcal{L}(x)| \cdot n)$ time, $W(x) = O(|\Sigma| \cdot |\mathcal{L}(x)| \cdot n^2)$ time and this gives the delay per maximal motif. Note that we have to charge to the parent x the work for all children since x may have no maximal children. To reduce computation time further, we replace line 5 to line 9 by the following procedure OccurrenceDeliver [4], which improves the work to $W(x) = O(|\mathcal{L}(x)| \cdot n^2)$, and thus gives the delay $D = W(x) \cdot |x|$ time since the depth of $\mathcal{T}$ is $\Theta(|x|)$.

```
OccurrenceDeliver(x, L(x), k) ≡
1  Initialize all L(c) := ∅ for all c ∈ Σ', and then Σ' := ∅;
2  foreach d ∈ L(x) do
3    L(c) := L(c) ∪ {d} and Σ' = Σ' ∪ {c}, where c := s[d + k];
5  foreach c ∈ Σ' do
6    Compute y = Clo(x⟨k ← c⟩) using L(c)
```

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

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

6 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 expansion, the algorithm enumerate all motifs without explicitly storing and checking the motifs enumerated so far. This drastically improves the space and the delay complexities compared with the previous algorithms with breadth-first search. As future research, we plan to empirical evaluation of MaxMotif algorithm on the real world datasets such as biological datasets.

In data mining, maximal motifs for *sets* are called *closed itemsets* [11]. There are a number of closed pattern discovery algorithms for itemsets [5,11], while only a few algorithms are known for sequences and trees [3,16,17]. Thus, it is an interesting research direction to extend the result of this paper for motif discovery in trees and graphs. Extension of the result for classes of unions of motifs [4] and basis of tiling motifs [13] is another research direction.

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 Series A, TCS-TR-A-05-6, Division of Computer Science, Hokkaido Univeristy, July 2005. `http://www-alg.ist.hokudai.ac.jp/tra.html`
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. Abdeddaïm, 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. X. Yan, J. Han, CloseGraph: Mining Closed Frequent Graph Patterns In *Proc. SIGKDD'03*, 2003.