Efficient Computation of Maximal Anti-Exponent in Palindrome-Free Strings

Golnaz Badkobeh[1] ⋆, Maxime Crochemore[2,3], Manal Mohamed[2], and Chalita Toopsuwan[2]

[1] University of Warwick, UK
[2] King's College London, UK
[3] Université Paris-Est, France

Abstract. A palindrome is a string $x = a_1 \cdots a_n$ which is equal to its reversal $\widetilde{x} = a_n \cdots a_1$. We consider gapped palindromes which are strings of the form $uv\widetilde{u}$, where u, v are strings, $|v| \geq 2$, and $\widetilde{u}$ is the reversal of u. Replicating the standard notion of string exponent, we define the anti-exponent of a gapped palindrome $uv\widetilde{u}$ as the quotient of $|uv\widetilde{u}|$ by $|uv|$. To get an efficient computation of maximal anti-exponent of factors in a palindrome-free string, we apply techniques based on the suffix automaton and the reversed Lempel-Ziv factorisation. Our algorithm runs in $O(n)$ time on a fixed-size alphabet or $O(n \log \sigma)$ on a large alphabet, which dramatically outperforms the naive cubic-time solution.

1 Introduction

A ***palindrome*** is a string $x = a_1 \cdots a_n$ which is equal to its reversal $\widetilde{x} = a_n \cdots a_1$. For example, $x_1 = \texttt{abba} = \widetilde{x_1}$ and $x_2 = \texttt{abaaba} = \widetilde{x_2}$ are palindromes.

The understanding of palindromic structures is one of the fundamental problems in language theory and algorithm design. Early studies by Manacher [16] and Galil [10] contributed to the construction of linear-time algorithms to find palindromes in a string. Crochemore and Rytter [6] presented a parallel algorithm to compute even-length palindromes in $O(\log n)$ time using n processors. Knuth, Morris and Pratt gave a linear-time algorithm to compute palstars (concatenations of even-length palindromes) in a given string [13].

The palindromic structure plays an important role in molecular biology and it is significant to both DNA and RNA sequences [18, 20]; for example, many restriction enzymes recognize specific palindromic sequences and cut them. However, the definition of a biological palindrome is slightly different from the definition above, as it needs to take into account Watson-Crick base pair rules. A nucleotide sequence is a palindrome, if it is equal to its reversed complement (C complements G and A complements T). For example, the DNA sequence `ACCTAGGT` is a palindrome because it is equal to the reversal of its complement `TGGATCCA`.

⋆ Corresponding author: `golnaz.badkobeh@gmail.com`

In this work, we study ***gapped palindromes***, which are strings of the form $uv\widetilde{u}$, where u, v are strings, $|v| \geq 2$, and $\widetilde{u}$ is the reversal of u. The strings u and $\widetilde{u}$ are called the anti-borders of the gapped palindrome. For example, `desserts make me stressed` has anti-borders '`desserts `' and '` stressed`' (example from [14]). This palindrome-like structure is also important in molecular biology; for example, gapped palindromes form stem-loop intra molecular base pairing structures known as hairpins or hairpin loops. Hairpins can be found in single-stranded DNA but more frequently in RNA, where the structure of the molecule influences its biological function; see [15, 21] for more examples of related genome research.

Gusfield [12] presented a linear-time algorithm for computing fixed-length gapped palindromes. Kolpakov and Kucherov [14] studied maximal gapped palindromes, i.e gapped palindromes with anti-borders that cannot be extended outward or inward while preserving the palindromic structure. They proposed two linear-time algorithms for computing two classes of gapped palindromes: The first algorithm computes maximal long-armed palindromes, where a long-armed palindrome is a gapped palindrome $uv\widetilde{u}$, such that $|v| \leq |u|$. The second algorithm computes maximal length-constrained palindromes, where a length-constrained palindrome is a gapped palindromes $uv\widetilde{u}$, such that $MinGap \leq |v| \leq MaxGap$ and $MinLen \leq |u|$, for some constants $MinGap$, $MaxGap$ and $MinLen$.

A closely related problem was presented in [3], in which a linear-time algorithm to find the longest previous reverse factor occurring at each position of a string is proposed. Such a factor is a principal notion used for the optimal detection of various types of palindromes. The ability to compute the longest previous reverse factor found many applications especially for RNA secondary structure prediction and text compression when reverse factors are accounted for [11]. This development led to the reversed Lempel-Ziv factorisation used in [14].

In this article, we consider a fixed palindrome-free string, that is, a string containing no palindrome of length greater than 1. For such string, we present a linear-time algorithm to compute the maximal anti-exponent of the gapped palindromes (a preliminary version was presented in [2]). This notion encompasses the detection of the most significant gapped palindromes occurring in a string and can be extended easily to biological palindromes.

The solution proposed in this article is a special type of divide-and-conquer technique. The technique we use is unbalanced contrary to what is traditional to impose for improving the running time or the memory space of resulting recursive algorithms. In fact, the balanced divide-and-conquer approach is unlikely to improve the running time of our solution as it would lead to a $O(n \log n)$-time algorithm. Our technique is essentially based on the reversed Ziv-Lempel factorisation of the input string, in which factors have various lengths. Despite the unbalanced feature, the solution provides an algorithm running in linear time, at least on a fixed-sized alphabet. This strategy has been initiated in [4] and applied since then to a variety of problems related to repeats occurring in strings, like in [1].

2 Prelimnaries

Let $x = x[1]x[2]\cdots x[n]$ be a *string* of length $|x| = n$ over an ordered alphabet Σ of size $\sigma = |\Sigma|$. Let $x[i]$ be the *letter* of x at position i, $1 \leq i \leq n$. The *empty* string is denoted by ϵ. A *factor* of x is a string of the form $x[i\mathinner{.\,.}j] = x[i]x[i+1]\ldots x[j]$, $1 \leq i \leq j \leq n$. A factor $x[i\mathinner{.\,.}j]$ is a *prefix* of x if $i = 1$, and a *suffix* of x if $j = n$. The *reversal* of x is the string $\widetilde{x} = x[n]x[n-1]\cdots x[1]$. If $x = \widetilde{x}$, then x is a *palindrome.*

The string x has *period* p, if $x[i] = x[i+p]$, whenever both sides of the equality are defined. *The period* of x, denoted by $\mathrm{period}(x)$, is the smallest period of x. *The exponent* of x, denoted by $\exp(x)$, is defined as $\exp(x) = n/\mathrm{period}(x)$. For example, $\exp(\texttt{restore}) = 7/5$, $\exp(\texttt{mama}) = 2$ and $\exp(\texttt{alfalfa}) = 7/3$.

A factor $w = uv\widetilde{u}$ is a *gapped palindrome*, if u, v are strings, $|v| \geq 2$, and v is not a palindrome. Here, u and $\tilde{u}$ are called the *anti-borders* of w if and only if u is the longest prefix of w for which $\widetilde{u}$ is a suffix. Note that a gapped palindrome is not a palindrome because the gap $|v| \geq 2$ is not allowed to be a palindrome.

A gapped palindrome is said to be *maximal* if its anti-borders cannot be extended outward or inward preserving the palindromic structure as in Fig 1. The *anti-exponent* of w is defined as $|w|/|uv|$. Further, the *maximal anti-exponent* of x is defined as the maximum value among the anti-exponents of all gapped palindromes occurring in x.

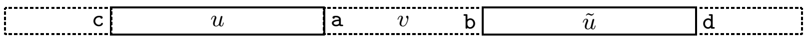

Fig. 1. Both anti-borders cannot be extended inward ($\texttt{a} \neq \texttt{b}$) or outward ($\texttt{c} \neq \texttt{d}$) preserving the palindromic structure whenever letters `a, b, c, d` exist.

In this paper, we consider a fixed palindrome-free string x of length n (containing no palindrome of length greater than 1). Note that a palindrome-free string contains no gapped palindrome of anti-exponent greater than 2. For such string, we compute the maximal anti-exponent of its factors.

3 Algorithm Scheme

The core result of this paper is algorithm MaxAntiExpGP, that computes the maximal anti-exponent of a fixed palindrome-free string x. The algorithm detects and processes potential gapped palindromes of the form $uv\tilde{u}$, where u and v are strings and $|v| \geq 2$. This is realised with the help of procedure MaxAntiExp, explained in the next section, which detects those gapped palindromes in the concatenation of two strings and whose anti-exponents are not less than the current maximal anti-exponent.

Algorithm MaxAntiExpGP relies on the reversed Lempel-Ziv factorisation; see [14] for more details. The reversed Lempel-Ziv factorisation of a string x is defined as a sequence of non-empty strings, z_1, z_2, $\ldots$, z_k satisfying the following properties:

- $x = z_1 z_2 \cdots z_k$,
- z_i is the longest prefix of $z_i z_{i+1} \cdots z_k$ occurring in $\widetilde{z_1 z_2 \cdots z_{i-1}}$,
- when this prefix is empty, z_i is the first letter of $z_i z_{i+1} \cdots z_k$, this letter does not occur previously in $z_1 z_2 \cdots z_{i-1}$.

For example, the reversed Lempel-Ziv factorisation of string *aababaabab* is *a.a.b.a.baa.bab*. The reversed factorisation of a given string of length n can be computed in $O(n)$ in both time and space by exploiting the suffix array and the LCP array (see [7]).

In the following, we modify the reversed factorisation for the purpose of our algorithm by defining z_1 as the longest prefix of x in which no letter occurs more than once.

Algorithm MaxAntiExpGP analyses strings z_2 to z_k sequentially. At step i, the algorithm assumes that z_1, z_2, $\ldots$, z_{i-1} have been processed and $\tilde{e}$ is equal to the maximal anti-exponent of the prefix $z_1.z_2 \cdots z_{i-1}$ of x. The gapped palindromes that need to be considered at this step are those involving string z_i. These gapped palindromes $uv\tilde{u}$ are either internal to z_i or occur partially in z_i. Note that $\tilde{u}$ can only occur within $z_{i-1}z_i$ and none of $z_1, z_2, \cdots, z_{i-1}$ can be a factor of $\tilde{u}$. This follows directly from the definition of the reversed factorisation.

We further distinguish four possible cases according to the location of the gapped palindrome $uv\tilde{u}$ as follows (see Fig. 2):

(i) Both occurrences of u and $\tilde{u}$ are inside z_i.
(ii) The occurrence of u is inside z_{i-1}, while $\tilde{u}$ ends in z_i.
(iii) The occurrence of u starts in z_{i-1}, while $\tilde{u}$ is inside z_i.
(iv) The occurrence of u starts in $z_1 \cdots z_{i-2}$, while $\tilde{u}$ is inside $z_{i-1}z_i$.

In Case (i), the gapped palindrome $uv\tilde{u}$, which is inside z_i, occurs previously in $z_1.z_2 \cdots z_{i-1}$ as $\tilde{u}\tilde{v}u$. Although, theses two gapped palindromes are different, they have the same anti-exponent. Therefore this case needs no further action. The other cases are handled by calls to MaxAntiExp procedure described in the following section. For any two strings z, w and a positive rational number $\tilde{e}$, MaxAntiExp$(z, w, \tilde{e})$ returns the maximal anti-exponent of zw, if such value is greater than $\tilde{e}$, and returns $\tilde{e}$ otherwise.

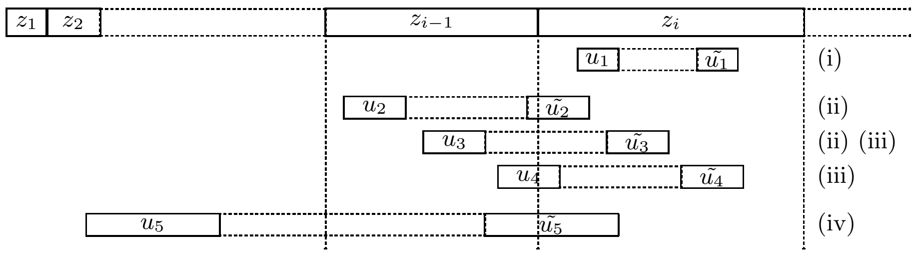

Fig. 2. All possible locations of a gapped palindrome $uv\tilde{u}$ involving strings z_i of the reversed factorisation of the string: (i) both u and $\tilde{u}$ are inside z_i; (ii) occurrence of u is inside z_{i-1}; (iii) occurrence of $\tilde{u}$ is inside z_i; (iv) occurrence of $\tilde{u}$ is inside $z_{i-1}z_i$.

MaxAntiExpGP(x)
1 $(z_1, z_2, \ldots, z_k) \leftarrow$ reversed-factorisation of x
2 $\triangleright$ z_1 is the longest prefix of x in which no letter repeats
3 $\tilde{e} \leftarrow 1$
4 **for** $i \leftarrow 2$ **to** k **do**
5 $\quad \tilde{e} \leftarrow$ MaxAntiExp$(z_{i-1}, z_i, \tilde{e})$
6 $\quad \tilde{e} \leftarrow$ MaxAntiExp$(\widetilde{z_i}, \widetilde{z_{i-1}}, \tilde{e})$
7 $\quad$ **if** $i > 2$ **then**
8 $\quad\quad \tilde{e} \leftarrow$ MaxAntiExp$(z_1 \cdots z_{i-2}, z_{i-1}z_i, \tilde{e})$
9 **return** $\tilde{e}$

Theorem 1. *For any given palindrome-free string x, Algorithm* MaxAntiExpGP *computes the maximal anti-exponent of x.*

Proof.

Procedure MaxAntiExp$(z, w, \tilde{e})$ is designed to check for gapped palindromes of anti-exponents greater than $\tilde{e}$. These gapped palindromes are of the form $uv\tilde{u}$ such that u occurs in z and $\tilde{u}$ is inside w.

Recall that the maximal anti-exponent of any fixed palindrome-free string is at least 1, thus $\tilde{e}$ is correctly initialised to 1 (Line 3).

At the beginning of each iteration i, $2 \leq i \leq k$, the algorithm assumes that $\tilde{e}$ is the maximal anti-exponent of $z_1 z_2 \cdots z_{i-1}$. Recall that $\tilde{u}$ cannot start in $z_1 z_2 \cdots z_{i-2}$, otherwise z_{i-1} would not satisfy the the definition of the reversed factorisation. Additionally, any gapped palindrome within z_i does not need to be considered. This is because the anti-exponent of such gapped palindrome and its reversal are equal; the reversal of such gapped palindrome must occur in $z_1 z_2 \cdots z_{i-1}$ by definition of z_i

As discussed earlier, there are three cases to be considered: Line 5 deals with gapped palindromes satisfying case (ii), Line 6 deals with gapped palindromes satisfying case (iii), and Line 8 deals with gapped palindromes satisfying case (iv). Thus, all relevant gapped palindromes are considered. This implies that the maximal anti-exponent $\tilde{e}$ returned by the algorithm is that of $z_1 z_2 \cdots z_k = x$, which completes the proof.

$\blacksquare$

Note that variable $\tilde{e}$ can be initialised by $(\sigma+1)/\sigma$, if x is long enough; see the following remark.

Remark 1. Let x be a given string such that $|x| > \sigma$. Then, for long enough x, let y be a factor of x which is composed of one appearance of all letters from Σ, hence $|y| = \sigma$. If a is a letter from Σ such that a is adjacent to y in x, and a is the first letter of y, then the factor ya is a gapped palindrome. The anti-exponent of this gapped palindrome is $(|y|+1)/|y|$. Then variable $\tilde{e}$ can be initialised to $(\sigma+1)/\sigma$.

4 Computing the Maximal Anti-Exponent

Procedure $\textsc{MaxAntiExp}(z, w, \tilde{e})$ is designed to compute the maximal anti-exponent of zw by considering gapped palindromes $uv\widetilde{u}$ such that u occurs in z, $\widetilde{u}$ is inside w, and whose anti-exponent is at least $\tilde{e}$. In particular, at each position of z, the procedure finds the factors of zw beginning in this position that have the form $uv\tilde{u}$ with $\tilde{u}$ inside w and updates the current maximal anti-exponent with the value of $|uv\widetilde{u}|/|uv|$. Before detailing the procedure, we present the suffix automaton data structure which is the fundamental algorithmic tool used by $\textsc{MaxAntiExp}$.

The *suffix automaton* of string w, denoted $\mathcal{S}(w)$, is the minimal partial deterministic finite automaton whose language is the set of suffixes of w (see [5, Section 6.6] for more description and for efficient construction); an example is given in Fig. 3. This data structure has an *initial* state, denoted s_0, and a *transition function* represented by the edges in the figure.

Let goto denotes the transition function, then the *suffix-link*, $\mathcal{SL}_w$, and the *length* function, $\mathcal{L}_w$, are defined as follows: For a given non empty string x such that $s_i = \text{goto}(s_0, x)$, then $\mathcal{SL}_w[s_i] = s_j = \text{goto}(s_0, x')$, where x' is the longest suffix of x such that $s_i \neq s_j$. As for the length function, $\mathcal{L}_w[s]$ is length of the longest factor x of w such that $s = \text{goto}(s_0, x)$. Additionally, we define the *shortest-path* function, denoted $\mathcal{SP}_w$, as follows: $\mathcal{SP}_w[s]$ is the length of the shortest-path from s_0 to s; see Table 1 for complete example.

s_j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$\mathcal{SL}_w[s_j]$		s_0	s_{11}	s_{12}	s_{13}	s_{14}	s_{12}	s_1	s_{13}	s_{14}	s_1	s_0	s_0	s_0	s_{11}
$\mathcal{L}_w[s_j]$	0	1	2	3	4	5	6	7	8	9	10	1	2	1	2
$\mathcal{SP}_w[s_j]$	0	1	2	3	4	5	6	7	8	9	10	2	3	4	4

Table 1. Suffix-links $\mathcal{SL}_w[s_j]$, the lengths function $\mathcal{L}_w[s_j]$ and shortest-paths $\mathcal{SP}_w[s_j]$ for each state s_j, $0 \le j \le 14$, of the suffix automaton in Fig. 3.

Observe that each state s of $\mathcal{S}(w)$ is associated with a set of factors $\mathcal{F}_w(s)$ such that $\mathcal{F}_w(s) = \{x \mid x = w[i,j], s = \text{goto}(s_0, x), 1 \le i \le j \le n\}$. Furthermore,

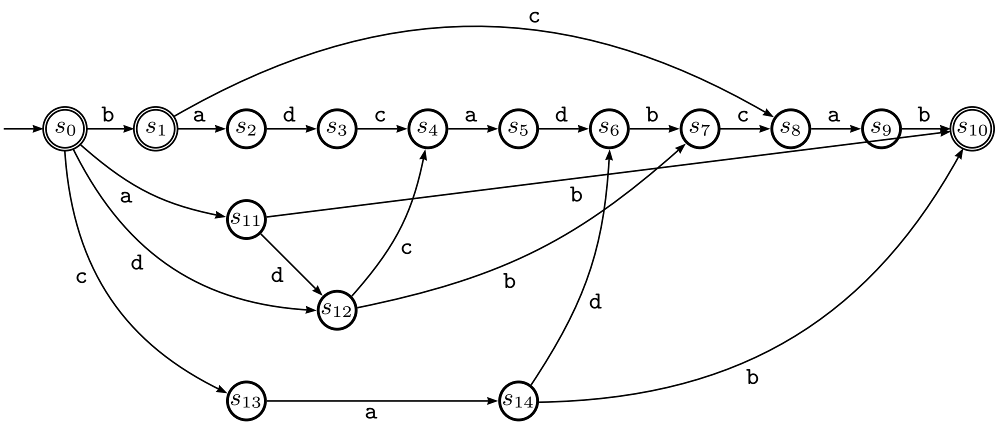

Fig. 3. Suffix automaton of string $w = \texttt{badcadbcab}$.

the Length of the longest factor $x' \in \mathcal{F}_w(s)$ is denoted by $\mathcal{L}_w[s]$, and $\mathcal{SP}_w[s]$ denotes position j in w such that there exists a factor $x'' \in \mathcal{F}_w(s)$, $x'' = w[i\,..\,j]$ and j is as small as possible.

For example, in Fig. 3, $\mathcal{F}_w(s_5) = \{badca, adca, dca\}$, while $\mathcal{F}_w(s_{12}) = \{ad, d\}$. Thus, $\mathcal{L}_w[s_5] = 3$, $\mathcal{L}_w[s_{12}] = 5$, $\mathcal{SP}_w[s_5] = 5$ and $\mathcal{SP}_w[s_{12}] = 3$.

The construction of the suffix automaton $\mathcal{S}(w)$ together with arrays $\mathcal{SL}_w$, $\mathcal{L}_w$ and $\mathcal{SP}_w$ can be done in linear time [5, Section 6.6]. It is well-known that the suffix automaton $\mathcal{S}(w)$ has no more than $2|w|$-2 states and $3|w| - 4$ edges independently of the alphabet size [5]. The transition function can be implemented in $O(1)$ time for a fixed size alphabet, or $O(\log \sigma)$ for a large alphabet; transition function may be implemented by lists of successors.

Moreover, the suffix automaton $\mathcal{S}(w)$ can be used to compute the factor r such that r is the longest prefix of w whose reversal occurs in w, Note that w here is a fixed palindrome-free string, thus, r and $\widetilde{r}$ do not overlap (see Fig. 4). Such factor can be computed by spelling $\tilde{w}$ from the initial state s_0 of $\mathcal{S}(w)$; this is only valid if w is not a palindrome. Procedure MaxAntiExp aims to extend r to the left and $\widetilde{r}$ to the right; this is achieved by spelling $\tilde{z}$ and exploiting the suffix automaton $\mathcal{S}(w)$.

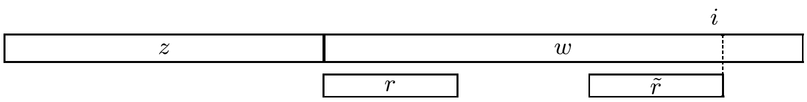

Fig. 4. Factor r is the longest prefix of w whose reversal occurs in w, where r and $\widetilde{r}$ do not overlap and position i is the end position of $\widetilde{r}$.

Recall that procedure MaxAntiExp considers for each position in z, the factors of the form $uv\tilde{u}$ starting at this position such that $\tilde{u}$ is inside w. The procedure tries to update the current maximal anti-exponent with the value of $|uv\tilde{u}|/|uv|$. If $\tilde{u}$ occurs more than once inside w, the procedure considers the left-most occurrence as this is associated with the factor of the greatest anti-exponent. The following lemmas allow MaxAntiExp to discard some of these factors and hence compute the maximal anti-exponent of zw efficiently.

Lemma 1. *Let u' be a prefix of u such that $\tilde{u'}$ and $\tilde{u}$ are associated with the same state of $\mathcal{S}(w)$. And let $uv\tilde{u}$ and $u'v'\tilde{u'}$ be two gapped palindromes in zw occurring at same the position. Then the anti-exponent of $uv\tilde{u}$ is grater than that of $u'v'\tilde{u'}$.*

Proof.

The hypothesis implies that both u and u' occur at the same position in z (see Fig. 5). Thus, the gapped palindromes $uv\tilde{u}$ and $u'v'\tilde{u'}$ are of the same length, $|uv\tilde{u}| = |u'v'\tilde{u'}|$. If $|u'| \leq |u|$, then the anti-exponent of $u'v'\tilde{u'}$ is not greater than that of $uv\tilde{u}$. ■

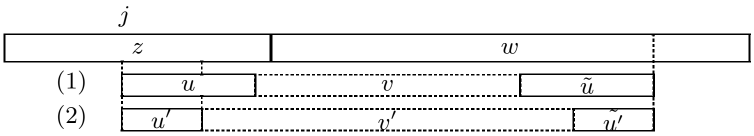

Fig. 5. Gapped palindrome (1) has a greater anti-exponent than that of gapped palindrome (2).

Note that the anti-border $\tilde{u'}$ may have an internal occurrence in $uv\tilde{u}$, which would lead to a gapped palindrome having a greater anti-exponent. For example, let $z =$ `abcad` and $w =$ `badcba`. Then the gapped palindrome `abcadbadcba` has anti-exponent 11/8 while the anti-border `ba` infers gapped palindrome `abcadba` of greater anti-exponent 7/5.

Lemma 2. *Let $uv'\tilde{u}$ and $uv\tilde{u}$ be two gapped palindromes occurring at positions j and k of zw, respectively, such that $j < k$. Then the anti-exponent of the gapped palindrome $uv'\tilde{u}$ is not greater than that $uv\tilde{u}$.*

Proof. Let $\tilde{u}$ be the the anti-border of gapped palindromes $uv\tilde{u}$ and $uv'\tilde{u}$. The procedure considers the left-most occurrence of $\tilde{u}$ in w. Thus, both gapped palindromes end at the same position and $|v| < |v'|$ (see Fig. 6). Therefore, $1 + |u|/|uv| > 1 + |u|/|uv'|$, which completes the proof. ■

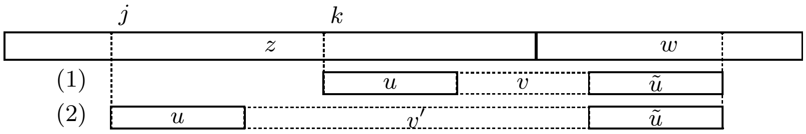

Fig. 6. Gapped palindrome (1) occurring at position k has a greater anti-exponent than gapped palindrome (2) occurring at position $j < k$.

MaxAntiExp$(z, w, \tilde{e})$

1 $\mathcal{S} \leftarrow$ *suffix automaton of* w
2 $(s, \ell) \leftarrow$ ReadR$(\mathcal{S}, w)$
3 $\triangleright$ ReadR() computes factor r as in Fig.4 then spells $\tilde{r}$ exploiting $\mathcal{S}(w)$.
4 Mark(s_0)
5 **for** $j \leftarrow 1$ **to** $\min\{\lfloor |w|/(\tilde{e}-1)\rfloor, |z|\}$ **do**
6   **while** goto$(s, z[|z|-j+1]) =$ NIL **and** $s \neq s_0$ **do**
7         $(s, \ell) \leftarrow (\mathcal{SL}_w[s], \mathcal{L}_w[\mathcal{SL}_w[s]])$
8   **if** goto$(s, z[|z|-j+1]) \neq$ NIL **then**
9         $(s, \ell) \leftarrow (\text{goto}(s, z[|z|-j+1]), \ell+1)$
10         $(s', \ell') \leftarrow (s, \ell)$
11   **while** s' unmarked **do**
12             $\tilde{e} \leftarrow \max\{\tilde{e}, (j + \mathcal{SP}_w[s])/(j - \ell' + \mathcal{SP}_w[s])\}$
13   **if** $\ell' = \mathcal{L}_w[s']$ **then**
14   Mark(s')
15             $(s', \ell') \leftarrow (\mathcal{SL}_w[s'], \mathcal{L}[\mathcal{SL}_w[s']])$
16 **return** $\tilde{e}$

Now we are ready to give the details of the procedure MaMaxAntiExp. In Line 1, the suffix automaton $\mathcal{S}(w)$ is built. Then in Line 2, the suffix automaton is used by function ReadR to first compute factor r; which is the longest prefix of w whose reversal occurs in w as explained earlier (see Fig. 4). Next, function ReadR proceeds by spelling $\tilde{r}$ and exploiting $\mathcal{S}(w)$. The current state of the automaton together with the length of r are finally returned.

Procedure MaxAntiExp proceeds by spelling $\tilde{z}$ and exploiting $\mathcal{S}(w)$ (Lines 5 to 15). At iteration j, let $uv\tilde{u}$ be the gapped palindrome of zw beginning in position $|z| - j + 1$, and s be is the current state of $\mathcal{S}(w)$ such that $\tilde{u}$ is the longest factor in $\mathcal{F}_w(s)$. Let u' be a prefix of u, then any gapped palindromes of the forms $uv_1\tilde{u}$ or $u'v_2\tilde{u'}$ that begin in position $k < |z| - j + 1$ cannot have anti-exponent greater than that of $uv\tilde{u}$. Therefore, the current state s is marked to inform the next steps of the procedure. The value of the anti-exponent of $uv\tilde{u}$ can be easily determined as demonstrated in Fig. 7.

Example 1. Let $z =$ `bcadbacbdac` and $w =$ `badcadbcab`. The maximal anti-exponent of zw is 17/10. The computations performed at each step of procedure MaxAntiExp are as follows:

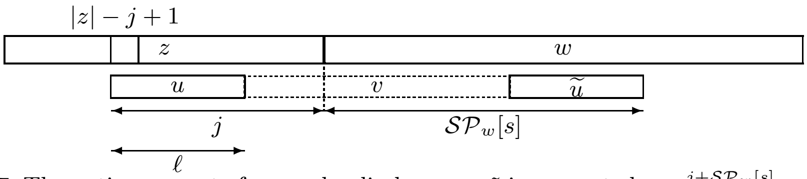

Fig. 7. The anti-exponent of gapped palindrome $uv\tilde{u}$ is computed as $\frac{j+\mathcal{SP}_w[s]}{j+\mathcal{SP}_w[s]-\ell}$, where s is the current state of $\mathcal{S}(w)$ and $\ell = |u|$.

j		1	2	3	4	5	6	7	8	9	10	11
$z[12-j]$		`c`	`a`	`d`	`b`	`c`	`a`	`b`	`d`	`a`	`c`	`b`
s	s_{10}	s_8	s_9	s_6	s_7	s_8	s_9	s_{10}	s_{12}	s_{11}	s_{13}	s_1
ℓ	2	2	3	3	4	5	6	7	1	1	1	1
anti-exp		9/7 5/4	11/8 7/5	9/6 6/4	11/7 7/6	13/8	15/9	**17/10**	11/10	11/10	14/13	12/11

Theorem 2. *Given strings z, w over an ordered alphabet Σ, and a rational number $\tilde{e} \geq 1 + 1/\sigma$. Let $\mathcal{G}$ be the set of all gapped palindromes $uv\tilde{u}$ in zw such that u begins in z, $\tilde{u}$ is inside w, and the anti-exponent of $uv\tilde{u}$ is greater than $\tilde{e}$. Then procedure* MaxAntiExp *returns the maximum anti-exponent of a gapped palindrome from $\mathcal{G}$ if $\mathcal{G}$ is not empty, and returns $\tilde{e}$ otherwise.*

Proof.

The correctness of procedure MaxAntiExp relies on Lemmas 1 & 2 and exploiting the properties of the suffix automaton.

Firstly, we show that the procedure does not require to investigate more positions than those specified in Line 5. This is because all gapped palindromes from $\mathcal{G}$, which begin earlier in z, have anti-exponents less than $\tilde{e}$.

Secondly, let $\mathcal{G}_j$ be the subset of $\mathcal{G}$ whose elements are gapped palindromes beginning in position $|z| - j + 1$ in z. Then for all possible j, we show that the procedure identifies correctly the subset of $\mathcal{G}_j$ that needs to be considered.

The following properties related to state s of $\mathcal{S}(w)$ and ℓ, are known from [5, Section 6.6]: Let u be the longest prefix of $z[|z|-j+1\,..\,|z|]w$ whose reversal is inside w, then (1) s is the state reached by spelling $\tilde{r}z[|z|]\,..\,z[|z|-j+1]$, where r is the longest prefix of w whose reversal $\tilde{r}$ appears in w, and (2) $\ell = |u| = |\tilde{u}|$. These properties hold after executing Line 2 where variables s and ℓ are initialized from the benefit of spelling $\tilde{r}$ by function ReadR. At Line 9, $\tilde{u}$ is the longest anti-border of a gapped palindrome in $\mathcal{G}_j$. Lines 11 to 15 check out the anti-exponents of $u_1v_1\tilde{u_1}, u_2v_2\tilde{u_2}, u_3v_3\tilde{u_3}, \cdots$, such that $\tilde{u_1}$ is a suffix of $\tilde{u}$, $\tilde{u_1} \in \mathcal{F}_w(s'_1)$, $s'_1 = \mathcal{SL}_w[s]$ and s'_1 is unmarked state. Similarly, for $i = 2, 3, \ldots$, $\tilde{u_i}$ is a suffix of $\tilde{u_{i-1}}$, $\tilde{u_i} \in \mathcal{F}_w(s'_i)$, $s'_i = \mathcal{SL}_w[s'_{i-1}]$ and s'_i is unmarked state. The procedure tries to update $\tilde{e}$ with the anti-exponent of each $u_iv_i\tilde{u_i}$ (Line 9). At Line 12, the procedure checks if state s' needs to be marked. This is done to avoid checking gapped palindromes $u_iv'\tilde{u_i}$ belong to sets $\mathcal{G}_k, k > j$ (Lemma 1).

Finally note the initial state of $\mathcal{S}(w)$ is marked in Line 4 because it corresponds to an empty string u, that is a gapped palindrome of exponent 1, which is not greater than the values of $\tilde{e}$. This proves that the algorithm runs through all relevant gapped palindromes in $\mathcal{G}$. ∎

5 Complexity Analysis

Proposition 1. *Applied for strings z, w and a rational number $\tilde{e} \geq 1 + 1/\sigma$, procedure* MaxAntiExp *requires $O(|w|\sigma)$ space and $O(|w|+|z|)$ time, or $O(|w|)$ space and $O((|w| + |z|) \log \sigma)$ time for a large alphabet.*

Proof.
The space required for the algorithm is exclusively used to store the suffix automaton $\mathcal{S}(w)$ and arrays $\mathcal{SL}_w$, $\mathcal{L}_w$ and $\mathcal{SP}_w$. Note that the suffix automaton $\mathcal{S}(w)$ has no more than $2|w| - 2$ states and $3|w| - 4$ edges independently of the alphabet size [5]. According to the implementation of the transition function of the automaton, the space complexity of procedure MaxAntiExp is either $O(|w|\sigma)$ or $O(|w|)$ for a large alphabet.

As for the time complexity, the construction of the automaton together with the arrays $\mathcal{SL}_w$, $\mathcal{L}_w$ and $\mathcal{SP}_w$, are known from [5, Section 6.6] to require $O(|w|)$ time (Line 1). The time required by Line 2 is either $O(|w|)$ time or $O(|w| \log \sigma)$ for a large alphabet, according to the implementation of the transition function. Recall that the transition function can be implemented in $O(1)$ time, or $O(\log \sigma)$ for a large alphabet.

Each iteration of the loop (excluding Lines 11 to 15) costs in $O(1)$ time for a fixed-size alphabet or $O(\log \sigma)$ time for a large alphabet; this is mainly the cost of goto. Therefore the total running time of the for loop is either $O(\min\{\lfloor |w|/(\tilde{e} - 1)\rfloor, |z|\})$ for a fixed size alphabet or $O(\min\{\lfloor |w|/(\tilde{e} - 1)\rfloor, |z|\} \log \sigma)$ for a large alphabet.

Next, let us consider the number of times Line 12 is executed, this is done once for each u_i associated with an unmarked state. If it is done more than once for a given position, then the second value of s' comes from the suffix-link. A crucial observation is that condition at Line 13 holds for such a state. Therefore, since $\mathcal{S}(w)$ has no more than $2|w| - 2$ states, the total number of extra executions of Line 12 is at most $2|w| - 2$. Which gives a total of $O(|w|)$ time for a fixed size alphabet or $O(|w| \log \sigma)$ time for large alphabet. Summing the above contributions to time and space completes the proof.

∎

Theorem 3. *Applied to any palindrome-free string of length n, Algorithm* MaxAntiExpGP *requires $O(n)$ time and $O(n\sigma)$ space, or $O(n \log \sigma)$ time and $O(n)$ space for a large alphabet.*

Proof. Computing the reversed factorisation $(z_1, z_2, \ldots, z_k)$ of a string of length n takes $O(n)$ time independently of alphabet size and $O(n)$ space.

Next instructions execute in linear space; this follows directly from Proposition 1. Note that the space bound is independent of the alphabet size.

Line 5 takes $O(|z_{i-1}| + |z_i|)$ time for a fixed size alphabet or $O(|z_{i-1}| + |z_i|)$ time for large alphabet, $i = 2, \ldots, k$. This sums up, for large enough input, to either $O(n)$ time for a fixed size alphabet or $O(n \log \sigma)$ time for a large alphabet. The same argument applies for Line 6 & 8 which completes the prof.

■

6 Conclusion

In this paper, algorithm MaxAntiExpGP calculates the maximal anti-exponent of a fixed palindrome-free string. The algorithm first computes the the LZ reversed factorisation of the input string. Then, for each pair of adjacent reversed factors, the algorithm calls procedure MaxAntiExp to calculate the associated maximal anti-exponent. Algorithm MaxAntiExpGP runs in $O(n)$ time for a fixed-size alphabet or $(O(n \log \sigma)$ time for a large alphabet, where n is the size of the input string and σ is the size of the alphabet Σ.

However, as far as we know, the number of distinct gapped palindromes in a string x whose anti-exponents equals to the maximal anti-exponent of x is currently unknown and constitutes an interesting combinatoric problem.

Another interesting question is the notion of a smallest unavoidable anti-exponent that we call the *anti-repetitive threshod* of the alphabet. If $\tilde{e}$ is this anti-exponent, then it is the smallest rational number for which there exists an infinite string whose the anti-exponents of its finite gapped palindromes are at most $\tilde{e}$. Dejean [9] introduced similar notion for factor exponents and called it the repetitive threshold $\mathrm{RT}(\sigma)$ of an alphabet of size σ. It is the smallest rational number for which there exists an infinite string whose finite factors have exponent at most $\mathrm{RT}(\sigma)$. It is known from Thue [19] that $RT(2) = 2$, Dejean [9] proved that $RT(3) = 7/4$ and stated the exact values of $\mathrm{RT}(\sigma)$ for every alphabet size $\sigma > 3$. Her conjecture was eventually proved in 2009 after partial proofs given by several authors (see [17, 8] and ref. therein).

Beyond the algorithmic aspect of the study of gapped palindromes, our paper opens a new research subject in Combinatorics on Words.

References

1. G. Badkobeh and M. Crochemore. Computing maximal-exponent factors in an overlap-free string. *Journal of Computer and System Sciences*, 82(477–487), 2016.
2. G. Badkobeh, M. Crochemore, and C. Toopsuwan. Maximal anti-exponent of gapped palindromes. In *Proceedings of the International Conference on Digital Information and Communication Technology and its Applicationsm DICTAP*, pages 205–210, 2014.
3. S. Chairungsee and M. Crochemore. Efficient computing of longest previous reverse factors. In Y. Shoukourian, editor, *Procedddings of the International Conference on Computer Science and Information Technologies CSIT*, pages 27–30, Yerevan, 2009. The National Academy of Sciences of Armenia Publishers.

4. M. Crochemore. Recherche linéaire d'un carré dans un mot. *C. R. Acad. Sc. Paris Sér. I Math.*, 296(18):781–784, 1983.
5. M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on Strings*, chapter 6.6. Cambridge University Press, 2007. 392 pages.
6. M. Crochemore and W. Rytter. Usefulness of the Karp-Miller-Rosenberg algorithm in parallel computations on strings and arrays. *Theoretical Computer Science*, 88(1):59 – 82, 1991.
7. M. Crochemore and W. Rytter. *Text Algorithms*, chapter 8.1 Searching for symmetric words, pages 111–114. Oxford University Press, 1994.
8. J. D. Currie and N. Rampersad. A proof of Dejean's conjecture. *Mathematics of Computation*, 80(274):1063–1070, 2011.
9. F. Dejean. Sur un théorème de Thue. *Journal of Combinatorial Theory, Series A*, 13(1):90–99, 1972.
10. Z. Galil. Real-time algorithms for string-matching and palindrome recognition. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 161–173, 1976.
11. S. Grumbach and F. Tahi. Compression of DNA sequences. In J. A. Storer and M. Cohn, editors, *Proceedings of the IEEE Data Compression Conference DCC*, pages 340–350. IEEE Computer Society, 1993.
12. D. Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology.* Cambridge University Press, Cambridge, 1997.
13. D. E. Knuth, J. H. M. Jr., and V. R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977.
14. R. Kolpakov and G. Kucherov. Searching for gapped palindromes. In *Proceedings of the Combinatorial Pattern Matching CPM*, pages 18–30, 2008.
15. L. Lu, H. Jia, P. Dröge, and J. Li. The human genome-wide distribution of DNA palindromes. *Functional & integrative genomics*, 7(3):221–227, 2007.
16. G. Manacher. A new linear-time "on-line" algorithm for finding the smallest initial palindrome of a string. *J. ACM*, 22(3):346–351, 1975.
17. M. Rao. Last cases of Dejean's conjecture. *Theoretical Computer Science*, 412(27):3010–3018, 2011.
18. S. Rozen, H. Skaletsky, J. Marszalek, P. Minx, H. Cordum, R. Waterston, R. Wilson, and D. Page. Abundant gene conversion between arms of palindromes in human and ape Y chromosomes. *Nature*, 423(6942):873–876, 6 2003.
19. A. Thue. Über unendliche Zeichenreihen. *Norske Vid. Selsk. Skr. I Math-Nat. Kl.*, 7:1–22, 1906.
20. T. Tsunoda, M. Fukagawa, and T. Takagi. Time and memory efficient algorithm for extracting palindromic and repetitive subsequences in nucleic acid sequences. In *Proceedings of the 4th Pacific Symposium on Biocomputing PSB*, pages 202–213, 1999.
21. P. E. Warburton, J. Giordano, F. Cheung, Y. Gelfand, and G. Benson. Inverted repeat structure of the human genome: the X-chromosome contains a preponderance of large, highly homologous inverted repeats that contain testes genes. *Genome research*, 14(10a):1861–1869, 2004.