Counting and Verifying Maximal Palindromes

Tomohiro I[1], Shunsuke Inenaga[2], Hideo Bannai[1], and Masayuki Takeda[1]

[1]Department of Informatics, Kyushu University
[2]Graduate School of Information Science and Electrical Engineering, Kyushu University
744 Motooka, Nishiku, Fukuoka, 819–0395 Japan.
tomohiro.i@i.kyushu-u.ac.jp
inenaga@c.csce.kyushu-u.ac.jp
{bannai,takeda}@inf.kyushu-u.ac.jp

Abstract. A palindrome is a symmetric string that reads the same forward and backward. Let $\mathit{Pals}(w)$ denote the set of maximal palindromes of a string w in which each palindrome is represented by a pair (c, r), where c is the center and r is the radius of the palindrome. We say that two strings w and z are pal-distinct if $\mathit{Pals}(w) \neq \mathit{Pals}(z)$. Firstly, we describe the number of pal-distinct strings, and show that we can enumerate all pal-distinct strings in time linear in the output size, for alphabets of size at most 3. These results follow from a close relationship between maximal palindromes and parameterized matching. Secondly, we present a linear time algorithm which finds a string w such that $\mathit{Pals}(w)$ is identical to a given set of maximal palindromes.

1 Introduction

1.1 Palindromes in Strings

A palindrome is a symmetric string that reads the same forward and backward. Namely, a string w is a palindrome if $w = xax^R$ where x is a string, x^R is a reversal of x, and a is either a single character or the empty string. Studying palindromic structures in strings have gathered much attention in theoretical computer science and in its applications.

In word combinatorics, palindromic structures of interesting family of words have been extensively studied. For example, palindromic factors of Fibonacci words and Sturmian words were investigated in [11, 12, 19, 26]. A concept called palindrome complexity of infinite words was introduced in [1] and its extension to finite words was proposed in [2]. Palindromic occurrences in ternary square-free words were studied in [10].

In algorithmics, several efficient algorithms to compute palindromes in a string have been proposed. Manacher [27] showed a linear-time algorithm to compute all prefix palindromes of an input string, which can immediately be extended to computing maximal palindromes of all positions of the string within the linear complexity. Another linear-time algorithm for prefix palindromes detection was proposed in the KMP pattern matching algorithm paper [24]. There exist efficient parallel algorithms to find all prefix palindromes or all maximal palindromes of a string [3, 4, 7]. A polynomial-time algorithm to compute all maximal palindromes from a given compressed string was proposed in [28].

In bioinformatics, some extended concepts of palindromes are known to be important in DNA and RNA sequence analysis [25]. An approximate palindrome, where the first half of the palindrome can be transformed into the reversal of the second half within a predefined edit distance, was introduced in [30]. Gusfield showed a linear-time algorithm to compute maximal palindromes with a fixed gap [20]. Kolpakov and Kucherov proposed linear-time solutions allowing more flexible gaps [25]. In [21] an efficient algorithm to compute all maximal approximate gapped palindromes was developed.

1.2 Our Contribution

It is natural and convenient to represent each maximal palindrome p of a string w by a pair (c, r) such that c is the center of p and r is the radius of p. This way the set of all maximal palindromes can be represented with $O(n)$ space, where n is the length of w. In what follows, we assume that the set of maximal palindromes of a string is represented in this way.

The contribution of this paper is twofold: Firstly, we show new properties of palindromes which are closely related to parameterized matching [5]. That is, if two strings are drawn from an alphabet of size at most 3, then they have the same set of maximal palindromes if and only if they parameterized match. Based on the above result and the results from [29], the number of distinct sets of palindromes for alphabets of size at most 3 can immediately be obtained. Besides, we show that there exists an efficient algorithm to compute a representative string for all distinct sets of maximal palindromes for alphabets of size at most 3.

Secondly, we study the problem of inferring a string from a given set of palindromic structures. Namely, given a set P of pairs (c, r), find a string whose maximal palindromes coincide with P. We propose a linear time solution to this problem, which outputs the lexicographically smallest string over a minimum alphabet.

1.3 Related Work

Inferring a string from other string data structures has been widely studied. An algorithm to find a string having a given border array was presented in [16], which runs in linear time for an unbounded alphabet. A simpler linear-time solution for the same problem for a bounded alphabet was shown in [14]. Linear-time and $O(n^{1.5})$-time inferring algorithms for parameterized versions of border arrays, on a binary alphabet and an unbounded alphabet, respectively, were recently proposed [22, 23]. Linear-time inferring algorithms for suffix arrays [15, 6], KMP failure tables [13, 18], prefix tables [8], cover arrays [9], directed acyclic word graphs [6] and directed acyclic subsequence graphs [6] have been proposed, which provide us with further insight concerning the data structures.

Counting and enumerating some of the above-mentioned data structures have also been studied in the literature [29, 31, 22, 23].

2 Preliminaries

Let Σ be a finite *alphabet*. An element of Σ^* is called a *string*. The length of a string w is denoted by $|w|$. The empty string ε is a string of length 0, that is, $|\varepsilon| = 0$. Let $\Sigma^+ = \Sigma^* - \{\varepsilon\}$. For a string $w = xyz$, x, y and z are called a *prefix*, *substring*, and *suffix* of w, respectively. The i-th character of a string w is denoted by $w[i]$ for $1 \leq i \leq |w|$, and the substring of a string w that begins at position i and ends at position j is denoted by $w[i:j]$ for $1 \leq i \leq j \leq |w|$. The empty substring ε of w is denoted by $w[i : i-1]$ for $1 \leq i \leq n$. For any string w, let w^R denote the reversed string of w, that is, $w^R = w[|w|] \cdots w[2]w[1]$.

A string w is called a *palindrome* if $w = w^R$. If $|w|$ is even, then w is called an *even palindrome*, that is, $w = xx^R$ for some $x \in \Sigma^+$. If $|w|$ is odd, then w is called an *odd palindrome*, that is, $w = xax^R$ for some $x \in \Sigma^*$ and $a \in \Sigma$. The *radius* of a palindrome w is $\frac{|w|}{2}$.

The *center* of a palindromic substring $w[i : j]$ of a string w is $\frac{i+j}{2}$. A palindromic substring $w[i : j]$ is called the *maximal palindrome* at the center $\frac{i+j}{2}$ if no other palindromes at the center $\frac{i+j}{2}$ have a larger radius than $w[i : j]$, i.e., if $w[i-1] \neq w[j+1]$, $i = 1$, or $j = |w|$. In particular, $w[1 : j]$ is called a *prefix palindrome* of w, and $w[i : |w|]$ is called a *suffix palindrome* of w.

We denote by $(c, r)_w$ the maximal palindrome of a string w whose center is c and radius is r. We simply write (c, r) when the string w is clear from the context. The set of all maximal even and odd palindromes of a string w is denoted by $\mathit{Pals}(w)$. It is clear that for any string w $\mathit{Pals}(w)$ has exactly $2|w| + 1$ elements. Let $\mathit{SPals}(w)$ denote the set of all suffix palindromes of w, that is, $\mathit{SPals}(w) = \{(c, r) \mid (c, r) \in \mathit{Pals}(w), c + r - 0.5 = n\}$.

For example, let $w = \texttt{abbacabbba}$. Then

$$\begin{aligned}\mathit{Pals}(w) = \{&(0.5, 0), (1, 0.5), (1.5, 0), (2, 0.5), (2.5, 2), (3, 0.5), (3.5, 0),\\ &(4, 0.5), (4.5, 0), (5, 3.5), (5.5, 0), (6, 0.5), (6.5, 0), (7, 0.5),\\ &(7.5, 1), (8, 2.5), (8.5, 1), (9, 0.5), (9.5, 0), (10, 0.5), (10.5, 0)\} \text{ and}\\ \mathit{SPals}(w) = \{&(8, 2.5), (10, 0.5), (10.5, 0)\}.\end{aligned}$$

3 Palindromes and Parameterized Matching

In this section we present new properties of palindromic structures in strings, with a tight relationship with parameterized matching which was originally introduced by Baker [5].

For any string w, let σ_w denote the number of distinct characters that appear in w. Any two strings w and z over the alphabet Σ of the same length are said to *parameterized match* (*p-match* in short) if there exists a renaming

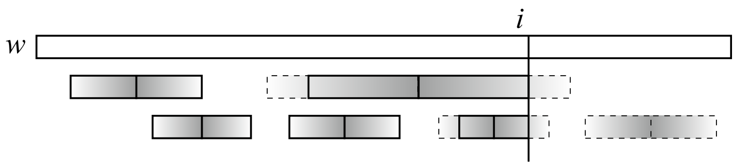

Fig. 1. Illustration for Observation 1.

bijection $f : \Sigma \to \Sigma$ which transforms one string into the other [5], that is, $w = f(z[1])f(z[2]) \cdots f(z[|z|])$. For instance, strings $w = \mathtt{abab}$ and $z = \mathtt{baba}$ p-match, since w can be transformed to z by applying a renaming function $f : \Sigma \to \Sigma$ such that $f(\mathtt{a}) = \mathtt{b}$ and $f(\mathtt{b}) = \mathtt{a}$.

The following intuitive property indeed holds.

Lemma 1. *If two strings w and z p-match, then $\mathit{Pals}(w) = \mathit{Pals}(z)$.*

Proof. Assume for contrary that $\mathit{Pals}(w) \neq \mathit{Pals}(z)$. Then there exists at least one center c such that $(c, r) \in \mathit{Pals}(w)$, $(c, r') \in \mathit{Pals}(z)$ and $r \neq r'$. Assume w.l.o.g. that $r > r'$. Then it holds that $w[c - r' - 0.5] = w[c + r' + 0.5]$ and $z[c - r' - 0.5] \neq z[c + r' + 0.5]$. Let $\mathtt{a} = w[c - r' - 0.5] = w[c + r' + 0.5]$, $\mathtt{b} = z[c - r' - 0.5]$, and $\overline{\mathtt{b}} = z[c + r' + 0.5]$, where $\overline{\mathtt{b}}$ denotes any character in $\Sigma - \{\mathtt{b}\}$. Then clearly there exists no bijection on the alphabet Σ that can transform w into z, since $\mathtt{a}$ at position $c - r' - 0.5$ needs to be mapped to $\mathtt{b}$ but $\mathtt{a}$ at position $c + r' + 0.5$ needs to be mapped to $\overline{\mathtt{b}}$. This contradicts that w and z p-match. $\square$

The reverse of Lemma 1 is also true if the strings are unary, binary or ternary. To show it, the following observation is useful.

Observation 1 *For any string w of length $n \geq 1$ and for any $i \leq n$,*

$$\begin{aligned}\mathit{Pals}(w[1:i]) = &\{(c, i + 0.5 - c) \mid (c, r) \in \mathit{Pals}(w), c \leq i + 0.5, c + r - 0.5 > i\} \\ &\cup \{(c, r) \mid (c, r) \in \mathit{Pals}(w), c + r - 0.5 \leq i\}.\end{aligned}$$

(See also Fig. 1)

Lemma 2. *If $\mathit{Pals}(w) = \mathit{Pals}(z)$ and $\sigma_w = \sigma_z \leq 3$, then w and z p-match.*

Proof. **Unary case $\sigma_w = \sigma_z = 1$.** This case is trivial.

Binary case $\sigma_w = \sigma_z = 2$. We prove it by induction on the length i of the strings. When $i = 2$, clearly two strings w and z of length 2 p-match, if $\mathit{Pals}(w) = \mathit{Pals}(z)$ and $\sigma_w = \sigma_z = 2$.

Suppose that the lemma holds for $i = n - 1 \geq 2$. Let w and z be any strings of length n over Σ, such that $\mathit{Pals}(w) = \mathit{Pals}(z)$ and $\sigma_w = \sigma_z = 2$. By Observation 1 $\mathit{Pals}(w[1 : n - 1]) = \mathit{Pals}(z[1 : n - 1])$, and by the induction hypothesis $w[1 : n - 1]$ and $z[1 : n - 1]$ p-match. Let $f : \Sigma \to \Sigma$ be the bijection which transforms $w[1 : n - 1]$ into $z[1 : n - 1]$.

1. When $w[n-1] = w[n]$. If $z[n-1] \neq z[n]$, then $(n-0.5, 1) \in \mathit{Pals}(w)$ and $(n-0.5, 0) \in \mathit{Pals}(z)$. However, this contradicts that $\mathit{Pals}(w) = \mathit{Pals}(z)$, and hence $z[n-1] = z[n]$. Then $f(w[n]) = f(w[n-1]) = z[n-1] = z[n]$, and therefore w and z p-match.
2. When $w[n-1] \neq w[n]$. If $z[n-1] = z[n]$, then $(n-0.5, 0) \in \mathit{Pals}(w)$ and $(n-0.5, 1) \in \mathit{Pals}(z)$. However, this contradicts that $\mathit{Pals}(w) = \mathit{Pals}(z)$, and hence $z[n-1] \neq z[n]$. Let $f(w[n-1]) = z[n-1] = \mathtt{a}$ and $f(w[n]) = \mathtt{b}$. Since $z[n] \neq \mathtt{a}$, $z[n] = \mathtt{b}$. Hence $f(w[n]) = z[n]$ and therefore w and z p-match.

Ternary case $\sigma_w = \sigma_z = 3$. We prove it by induction on the length i of the strings. When $i = 3$, clearly two strings w and z of length 3 p-match, if $\mathit{Pals}(w) = \mathit{Pals}(z)$ and $\sigma_w = \sigma_z = 3$.
Suppose that the lemma holds for $i = n - 1 \geq 3$. Let w and z be any strings of length n over Σ, such that $\mathit{Pals}(w) = \mathit{Pals}(z)$ and and $\sigma_w = \sigma_z = 3$. By similar arguments to the binary case, $w[1 : n-1]$ and $z[1 : n-1]$ p-match. Let $g : \Sigma \to \Sigma$ be the bijection which transforms $w[1 : n-1]$ into $z[1 : n-1]$.
1. When $w[n-1] = w[n]$. This case can be shown similarly to the binary case.
2. When $w[n-1] \neq w[n]$. By similar arguments to the binary case, we get $z[n-1] \neq z[n]$. Let m be the rightmost position of $w[1 : n-1]$ such that $w[m] \neq w[n-1]$. Let $g(w[n-1]) = z[n-1] = \mathtt{a}$ and $g(w[n]) = \mathtt{b}$.
   (a) When $w[m] = w[n]$. Since $w[m+1 : n-1]$ is unary, $w[m : n]$ is a maximal palindrome of w. Since $z[m : n]$ is a maximal palindrome of z, $z[n] = z[m] = g(w[m]) = \mathtt{b} = g(w[n])$. Hence w and z p-match.
   (b) When $w[m] \neq w[n]$. Let $g(w[m]) = \mathtt{c}$. Since $w[m+1 : n-1]$ is unary, $w[m+1 : n-1]$ is a maximal palindrome of w. Since $z[m+1 : n-1]$ is a maximal palindrome of z, $z[n] \neq z[m] = g(w[m]) = \mathtt{c}$. Additionally, since $z[n] \neq z[n-1] = g(w[n-1]) = \mathtt{a}$, $z[n] = \mathtt{b} = g(w[n])$. Consequently w and z p-match.

□

The next proposition follows from Lemma 2.

Proposition 1. *For any string w with $\sigma_w \leq 2$, there exist no string z such that $\mathit{Pals}(w) = \mathit{Pals}(z)$ and $\sigma_z > \sigma_w$.*

It is interesting to see that a similar argument to Lemma 2 does *not* hold if strings contain 4 or more distinct characters. For instance, two strings $w = \mathtt{abbcdaa}$ and $z = \mathtt{accdbcc}$ have the same set of maximal palindromes and $\sigma_w = \sigma_z = 4$, but w and z do not p-match. Also, Proposition 1 does not hold if $\sigma_w \geq 3$. For instance, $w = \mathtt{abcabb}$ and $z = \mathtt{abcdaa}$ have the same set of maximal palindromes, while $\sigma_w = 3$ and $\sigma_z = 4$. It is easy to extend the above examples to infinite sequences of strings with an alphabet of size 4 or more.

Let us define equivalence relations $\equiv_{\mathrm{pal}}$ and $\equiv_{\mathrm{pm}}$ on Σ^* by

$$w \equiv_{\mathrm{pal}} z \iff \mathit{Pals}(w) = \mathit{Pals}(z)$$
$$w \equiv_{\mathrm{pm}} z \iff w \text{ and } z \text{ p-match.}$$

By Lemma 1 and Lemma 2, the two equivalence relations are equivalent for $|\Sigma| \leq 3$. Also, if $|\Sigma| \geq 4$, then $\equiv_{\mathrm{pm}}$ is a refinement of $\equiv_{\mathrm{pal}}$.

Denote by $[w]_{\mathrm{pal}}$ and $[w]_{\mathrm{pm}}$ the equivalence classes with respect to $\equiv_{\mathrm{pal}}$ and $\equiv_{\mathrm{pm}}$, respectively. Define the representative of equivalence class $[w]_{\mathrm{pal}}$ to be the lexicographically smallest member of $[w]_{\mathrm{pal}}$, and call each representative a *pal-canonical* string. Moore et al. [29] counted the number of *p-canonical* strings, each of which is the representative (i.e., the lexicographically smallest member) of an equivalence class $[\cdot]_{\mathrm{pm}}$. They also presented an algorithm to enumerate all p-canonical strings. From Lemma 1, Lemma 2 and the results of [29], we immediately get the following theorems.

Theorem 1. *Let $p[k,n]$ be the number of distinct sets of maximal palindromes for strings of length n containing exactly k characters. Then $p[k,n] = S(n,k)$ for $1 \leq k \leq 3$ and $p[k,n] < S(n,k)$ for $k \geq 4$, where $S(n,k)$ is the Stirling number of the second kind.*

Theorem 2. *For every pair of integers $k \leq 3$ and $n \geq k$, all pal-canonical strings of length n consisting of exactly k characters can be computed in $O(p[k,n])$ time and space.*

Although the above theorems provide us with new insights and an efficient algorithm, yet they do not immediately help us solve the problem of inferring a string from a given set of maximal palindromes. In the next section, we will provide a linear-time algorithm to solve it.

4 Inferring a String from Maximal Palindromes

4.1 Problem

Let $\mathcal{N}$ be the set of non-negative integers, and let $\mathcal{Q} = \{i \mid i = \frac{j}{2}, j \in \mathcal{N}\}$. In this section we present a linear-time algorithm to solve the following problem.

Problem 1. Given a finite set $P \subset \mathcal{Q} \times \mathcal{Q}$, find a string w such that $P = \mathit{Pals}(w)$ if such exists.

Concerning Lemma 2 and Proposition 1 of the previous section, we try to find the *lexicographically smallest* string over a *minimum* alphabet in solving Problem 1.

4.2 Linear-Time Algorithm to Compute Maximal Palindromes from a String

Let us recall a linear-time algorithm to compute all maximal palindromes in a given string, which is an extended version of Manacher's algorithm that computes all prefix palindromes [27]. A pseudo-code of the algorithm is shown in Algorithm 1. The algorithm is based on the following lemma.

Algorithm 1: Manacher's algorithm to compute all maximal palindromes in a given string [27].

```
Input: string $w$ of length $n$.
Output: compute all maximal palindromes in $s$.
/* Let $w[0]$ and $w[n+1]$ be special symbols that do not match other
   symbols for convenience.                                              */
add $(0.5, 0)$ and $(n + 0.5, 0)$ to $P$;
$i \leftarrow 2$; $c \leftarrow 1$; $r \leftarrow 0.5$;
while $c \leq n$ do
    $j \leftarrow 2c - i$;        /* Set $j$ to be the mirrored position w.r.t. $c$. */
    while $w[i] = w[j]$ do
        $i++$; $j--$; $r++$;
    add $(c, r)$ to $P$;
    $d \leftarrow 0.5$;
    while $d \leq r$ do
        let $(c - d, r_\ell) \in P$;
        if $r_\ell = r - d$ then break;
        $r_r \leftarrow \min\{r - d, r_\ell\}$;
        add $(c + d, r_r)$ to $P$;
        $d \leftarrow d + 0.5$;
    if $d > r$ then $i++$; $r \leftarrow 0.5$;
    else $r \leftarrow r - d$;
    $c \leftarrow c + d$;                        /* Shift the value of $c$ by $d$. */
return $P$;
```

Lemma 3 ([27]). *For any string w, let $(c, r) \in Pals(w)$ and $(c - d, r_\ell) \in Pals(w)$ with some $0 < d \leq r$. Then $(c + d, r_r) \in Pals(w)$, where*

$$r_r = r_\ell \quad \text{if } r_\ell < r - d, \tag{1}$$

$$r_r \geq r - d \quad \text{if } r_\ell = r - d, \tag{2}$$

$$r_r = r - d \quad \text{if } r_\ell > r - d. \tag{3}$$

Recall Observation 1. This observation suggests to compute maximal palindromes from left to right in the input string, and therefore Algorithm 1 computes the radius of each center in increasing order of the centers. Let c be the currently focused center, that is, for every center less than c, the corresponding radius has already been computed. Then we compute the radius for c, comparing leftward and rightward substrings of c until a mismatch occurs. Let $(c, r) \in Pals(w)$. The key of the algorithm is that, if Condition 1 or 3 of Lemma 3 holds for a center between c and $c + r$, then the radius of the center can be determined in constant time. If there exists a center $c + d$ with which Condition 2 holds, then we shift the currently focused center to the center $c + d$, and compute the radius for $c + d$. Since in this case the radius for $c + d$ is at least $r - d$, the overall time complexity of Algorithm 1 is linear in the length of the input string.

Algorithm 2: Algorithm to compute the lexicographically smallest string over a minimum alphabet which has a given set of maximal palindromes.

```
Input: $P \subset \mathcal{Q} \times \mathcal{Q}$.
Output: The lexicographically smallest string $w$ over a minimum alphabet with
        $Pals(w) = P$, if such exists.
1  $w[1] \leftarrow$ a;
2  $i \leftarrow 2$; $c \leftarrow 1$; $\hat{r} \leftarrow 0.5$;
3  while $c \leq n$ do
4      $j \leftarrow 2c - i$;          /* Set $j$ to be the mirrored position w.r.t. $c$. */
5      let $(c, r) \in P$;
6      if $r < \hat{r}$ then return invalid;
7      while $\hat{r} < r$ do
8          $w[i] \leftarrow w[j]$;
9          $i++$; $j--$; $\hat{r}++$;
10         clear the list of forbidden characters;
11     add $w[j]$ to the list of forbidden characters for $w[i]$;     /* $w[i] \neq w[j]$. */
12     $d \leftarrow 0.5$;
13     while $d \leq r$ do
14         let $(c - d, r_\ell), (c + d, r_r) \in P$;
15         if $r_\ell = r - d$ then break;
16         if $r_r \neq \min\{r - d, r_\ell\}$ then return invalid;
17         $d \leftarrow d + 0.5$;
18     if $d > r$ then
19         let $x$ be the lexicographically smallest character not in the list of
           forbidden characters for $w[i]$;
20         $w[i] \leftarrow x$;
21         $i++$; $\hat{r} \leftarrow 0.5$;
22         clear the list of forbidden characters;
23     else $\hat{r} \leftarrow r - d$;
24     $c \leftarrow c + d$;                  /* Shift the value of $c$ by $d$. */
25 return $w[1 : n]$;
```

4.3 Our Algorithm to Compute a String from Maximal Palindromes

Now we consider Problem 1. Any $P \subset \mathcal{Q} \times \mathcal{Q}$ is said to be valid if there exists a string w such that $Pals(w) = P$, and is said to be invalid otherwise. For $P \subset \mathcal{Q} \times \mathcal{Q}$ to be valid, clearly P has to satisfy the following: For each $c = 0.5, 1, 1.5, \ldots, n, n{+}0.5$, there exists $(c, r) \in P$ with some $r \in \{0, 0.5, 1, 1.5, \ldots, k\}$, where $n = \lfloor |P|/2 \rfloor$ and $k = \min\{c - 0.5, n + 0.5 - c\}$. Hence in what follows we only consider as input a set P satisfying the above property.

Algorithm 2 shows our algorithm for Problem 1.

Theorem 3. *Given a valid set $P \subset \mathcal{Q} \times \mathcal{Q}$, Algorithm 2 computes the lexicographically smallest string w over a minimum alphabet such that $Pals(w) = P$, in linear time and space.*

Proof. Let $(c, r) \in P$. If Condition 1 or Condition 3 of Lemma 3 holds for the center c, then P is never rejected w.r.t. c in line 16 of Algorithm 2. Also, if Condition 2 of Lemma 3 holds for the center c, then P is never rejected w.r.t. c in line 6 of Algorithm 2. Therefore, if P is a valid set, then it is verified as valid by Algorithm 2.

In line 8 of Algorithm 2, $w[i]$ is set to $w[j]$ where $j = 2c - i$ is the mirrored position of i w.r.t. c. When i goes "outside" of the radius r of the maximal palindrome (c, r), then $w[j]$ is recorded as a forbidden character for $w[i]$ in line 11, that is, $w[i]$ cannot be equal to $w[j]$. Then in line 19, $w[i]$ is set to be the lexicographically smallest character that is not in the list of forbidden characters. Therefore, for a given valid set P Algorithm 2 computes the lexicographically smallest string w such that $P = \mathit{Pals}(w)$.

The key for time complexity analysis is how to choose the lexicographically smallest character in line 19. We use a bit vector F of length $n = \lfloor |P|/2 \rfloor$ where each $F[h]$ corresponds to the h-th lexicographically smallest character. We initialize F with $F[h] = 0$ for every $1 \leq h \leq n$. If a character $w[j]$ is forbidden for $w[i]$ in line 11 and if $w[j]$ is the h-th lexicographically smallest character, then we set $F[h] = 1$. After finding all forbidden characters for $w[i]$, we find the lexicographically smallest character for $w[i]$ by scanning F from left to right until reaching the smallest index k with $F[k] = 0$. After setting $w[i]$ to be the k-th lexicographically smallest character, we initialize every entry $F[h] = 1$ to $F[h] = 0$.

Since $\sigma_w \leq n$ for any string w of length n, the bit vector F is sufficiently large. For each position i, let $\mathit{fc}(i)$ and $\mathit{lc}(i)$ be the first (leftmost) center and the last (rightmost) center w.r.t. i, respectively. Namely, $w[i]$ is determined right after $\mathit{lc}(i)$ is obtained by shifting $\mathit{fc}(i)$ several times in line 24, Then, the number of forbidden characters for $w[i]$ does not exceed $2(\mathit{lc}(i) - \mathit{fc}(i))$. Hence the total number of forbidden characters for all i is bounded by $|P|$. Consequently we can maintain the bit vector F in a total of linear time and space. □

We remark that Algorithm 2 verifies some invalid sets to be valid. For instance, the following invalid set P is verified to be valid by Algorithm 2:

$$P = \{(0.5, 0), (1, 0.5), (1.5, 1), (2, 0.5), (2.5, 0), (3, 1.5), (3.5, 0), (4, 0.5), (4.5, 1), (5, 0.5), (5.5, 0)\}.$$

Therefore, we firstly use Algorithm 2 as a filter. Consider the case where Algorithm 2 verifies an input set P to be valid. Let w be the output string of Algorithm 2 w.r.t. P. We then run Algorithm 1 over w to compute $\mathit{Pals}(w)$. Finally, we check whether $\mathit{Pals}(w) = P$ or not. Note that P is valid if and only if $\mathit{Pals}(w) = P$. Hence we obtain:

Theorem 4. *Problem 1 can be solved in linear time.*

Reducing Extra Space. Here we consider to reduce extra working space of Algorithm 2. The next lemma is useful to estimate it.

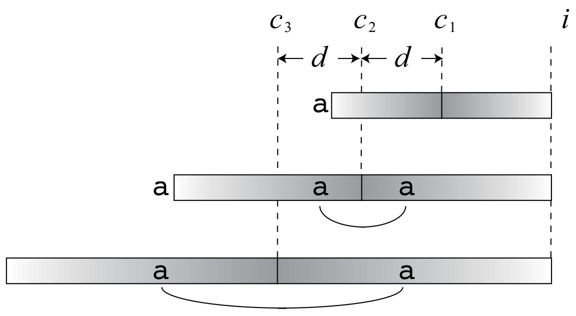

Fig. 2. Illustration for Proposition 2.

For any string w, let $SPC(w)$ be the set of centers of suffix palindromes of w, that is,

$$SPC(w) = \{c \mid (c, r) \in SPals(w)\}.$$

Lemma 4 ([17, 28]). *For any string w of length n, $SPC(w)$ can be represented by $O(\log n)$ arithmetic progressions.*

For any w of length n and $1 \leq i \leq n$, let

$$MC(i) = \{w[2c - i] \mid (c, r) \in Pals(w), i = c + r + 0.5\}.$$

Proposition 2. *Let P be the set of maximal palindromes of some string of length n. Then, there exists a string w over an alphabet of size $O(\log n)$ such that $Pals(w) = P$.*

Proof. Take any three elements from $SPals(w[1 : i-1])$ whose centers belong to the same arithmetic progression, i.e., $(c_1, r_1), (c_2, r_2), (c_3, r_3) \in SPals(w[1 : i-1])$ such that $c_1 = c_2 + d = c_3 + 2d$ for some $d > 0$ (See also Fig. 2). By mirroring $w[2c_2 - i]$ w.r.t. c_3, we get $w[2c_2 - i] = w[2c_3 - (2c_2 - i)] = w[2(c_3 - c_2) + i] = w[i-2d]$. Similarly by mirroring $w[2c_1-i]$ w.r.t. c_2, $w[2c_1-i] = w[2c_2-(2c_1-i)] = w[2(c_2 - c_1) + i] = w[i - 2d]$. Then we get $w[2c_1 - i] = w[i - 2d] = w[2c_2 - i]$. By Lemma 4, $|MC(i)| = O(\log i)$. Since the number of forbidden characters for each position i is at most $|MC(i)|$, we conclude that $O(\log n)$ distinct characters are sufficient in total. $\square$

Since Algorithm 2 always computes a string over a minimum alphabet, a bit vector of size $O(\log n)$ is actually enough for maintaining the forbidden characters for each position i of the output string.

5 Conclusions and Future Work

In this paper we studied the problem of counting the number of distinct sets of maximal palindromes, and that of finding a string from a given set of maximal palindromes. For the first problem, we showed the exact number for an alphabet of size at most 3. We also showed that there exists an algorithm that enumerates all pal-canonical strings for an alphabet of size at most 3, which runs in linear time in the output size. These results follow from the close relationship between maximal palindromes and parameterized pattern matching for alphabets of size at most 3. For the second problem, we presented a linear time algorithm that finds the lexicographically smallest string over a minimum alphabet.

Our future work includes the followings.

1. Counting the number of pal-canonical strings for an alphabet of arbitrary size.
2. Enumerating all distinct sets of maximal palindromes for an alphabet of arbitrary size.
3. Finding a string that has a given set of maximal palindromes and contains exactly k characters, where k is a predefined parameter.

References

1. Allouche, J.P., Baake, M., Cassaigne, J., Damanik, D.: Palindrome complexity. Theoretical Computer Science 292(1), 9–31 (2003)
2. Anisiu, M.C., Anisiu, V., Kása, Z.: Total palindrome complexity of finite words. Discrete Mathematics 310(1), 109–114 (2010)
3. Apostolico, A., Breslauer, D., Galil, Z.: Optimal parallel algorithms for periods, palindromes and squares. In: Proc. ICALP 1992. LNCS, vol. 623, pp. 296–307 (1992)
4. Apostolico, A., Breslauer, D., Galil, Z.: Parallel detection of all palindromes in a string. Theoretical Computer Science 141, 163–173 (1995)
5. Baker, B.S.: Parameterized pattern matching: Algorithms and applications. Journal of Computer and System Sciences 52(1), 28–42 (1996)
6. Bannai, H., Inenaga, S., Shinohara, A., Takeda, M.: Inferring strings from graphs and arrays. In: Proc. MFCS 2003. LNCS, vol. 2747, pp. 208–217 (2003)
7. Breslauer, D., Galil, Z.: Finding all periods and initial palindromes of a string in parallel. Algorithmica 14(4), 355–366 (1995)
8. Clément, J., Crochemore, M., Rindone, G.: Reverse engineering prefix tables. In: Proc. STACS 2009. pp. 289–300 (2009)
9. Crochemore, M., Iliopoulos, C., Pissis, S., Tischler, G.: Cover array string reconstruction. In: Proc. CPM 2010. LNCS, vol. 6129, pp. 251–259 (2010)
10. Currie, J.D.: Palindrome positions in ternary square-free words. Theoretical Computer Science 396(1-3), 254–257 (2008)
11. Droubay, X.: Palindromes in the Fibonacci word. Information Processing Letters 55(4), 217–221 (1995)
12. Droubay, X., Pirillo, G.: Palindromes and Sturmian words. Theoretical Computer Science 223(1-2), 73–85 (1999)
13. Duval, J.P., Lecroq, T., Lefebvre, A.: Efficient validation and construction of border arrays and validation of string matching automata. RAIRO - Theoretical Informatics and Applications 43(2), 281–297 (2009)
14. Duval, J.P., Lecroq, T., Lefevre, A.: Border array on bounded alphabet. Journal of Automata, Languages and Combinatorics 10(1), 51–60 (2005)

15. Duval, J.P., Lefebvre, A.: Words over an ordered alphabet and suffix permutations. Theoretical Informatics and Applications 36, 249–259 (2002)
16. Franek, F., Gao, S., Lu, W., Ryan, P.J., Smyth, W.F., Sun, Y., Yang, L.: Verifying a border array in linear time. J. Comb. Math. and Comb. Comp. 42, 223–236 (2002)
17. Gasieniec, L., Karpinski, M., Plandowski, W., Rytter, W.: Efficient algorithms for Lempel-Ziv encoding. In: Proc. SWAT 1996. LNCS, vol. 1097, pp. 392–403. Springer-Verlag (1996)
18. Gawrychowski, P., Jez, A., Jez, L.: Validating the Knuth-Morris-Pratt failure function, fast and online. In: Proc. CSR 2010. LNCS, vol. 6072, pp. 132–143 (2010)
19. Glen, A.: Occurrences of palindromes in characteristic Sturmian words. Theoretical Computer Science 352(1), 31–46 (2006)
20. Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press, New York (1997)
21. Hsu, P.H., Chen, K.Y., Chao, K.M.: Finding all approximate gapped palindromes. In: Proc. ISAAC 2009. LNCS, vol. 5878, pp. 1084–1093 (2009)
22. I, T., Inenaga, S., Bannai, H., Takeda, M.: Counting parameterized border arrays for a binary alphabet. In: Proc. LATA 2009. LNCS, vol. 5457, pp. 422–433 (2009)
23. I, T., Inenaga, S., Bannai, H., Takeda, M.: Verifying a parameterized border array in $O(n^{1.5})$ time. In: Proc. CPM 2010. LNCS, vol. 6129, pp. 238–250 (2010)
24. Knuth, D.E., Morris, J.H., Pratt, V.R.: Fast pattern matching in strings. SIAM J. Comput. 6(2), 323–350 (1977)
25. Kolpakov, R., Kucherov, G.: Searching for gapped palindromes. Theoretical Computer Science 410(51), 5365–5373 (2009)
26. de Luca, A., Luca, A.D.: Palindromes in Sturmian words. In: Proc. DLT 2005. LNCS, vol. 3572, pp. 199–208 (2005)
27. Manacher, G.: A new linear-time "On-Line" algorithm for finding the smallest initial palindrome of a string. Journal of the ACM 22(3), 346–351 (1975)
28. Matsubara, W., Inenaga, S., Ishino, A., Shinohara, A., Nakamura, T., Hashimoto, K.: Efficient algorithms to compute compressed longest common substrings and compressed palindromes. Theoretical Computer Science 410(8–10), 900–913 (2009)
29. Moore, D., Smyth, W.F., Miller, D.: Counting distinct strings. Algorithmica 23(1), 1–13 (1999)
30. Porto, A.H.L., Barbosa, V.C.: Finding approximate palindromes in strings. Pattern Recognition 35(11), 2581–2591 (2002)
31. Schürmann, K.B., Stoye, J.: Counting suffix arrays and strings. Theoretical Computer Science 395(2-1), 220–234 (2008)