Counting Parameterized Border Arrays for a Binary Alphabet

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,bannai,takeda}@i.kyushu-u.ac.jp
inenaga@c.csce.kyushu-u.ac.jp

Abstract. The parameterized pattern matching problem is a kind of pattern matching problem, where a pattern is considered to occur in a text when there exists a renaming bijection on the alphabet with which the pattern can be transformed into a substring of the text. A *parameterized border array* (*p-border array*) is an analogue of a border array of a standard string, which is also known as the failure function of the Morris-Pratt pattern matching algorithm. In this paper we present a linear time algorithm to verify if a given integer array is a valid p-border array for a binary alphabet. We also show a linear time algorithm to compute all binary parameterized strings sharing a given p-border array. In addition, we give an algorithm which computes all p-border arrays of length at most n, where n is a a given threshold. This algorithm runs in time linear in the number of output p-border arrays.

1 Introduction

1.1 Parameterized Matching and Parameterized Border Array

The *parameterized matching* (*p-matching*) problem [1] is a kind of string matching problem, where a pattern is considered to occur in a text when there exists a renaming bijection on the alphabet with which the pattern can be transformed into a substring of the text. A *parameterized string* (*p-string*) is formally an element of $(\Pi \cup \Sigma)^*$, where Π is the set of *parameter symbols* and Σ the set of *constant symbols*. The renaming bijections used in p-matching are the identity on Σ, that is, every constant symbol $\mathtt{X} \in \Sigma$ is mapped to $\mathtt{X}$, while symbols in Π can be interchanged. Parameterized matching has applications in software maintenance [2, 1], plagiarism detection [3], and RNA structural matching [4], thus it has been extensively studied in the last decade [5–12].

Of various efficient methods solving the p-matching problem, this paper focuses on the algorithm of Idury and Schäffer [13] that solves the p-matching problem for multiple patterns. Their algorithm modifies the Aho-Corasick automata [14], replacing the *goto* and *fail* functions with the *pgoto* and *pfail* functions, respectively. When the input is a single pattern p-string of length m, the *pfail* function can be implemented by an array of length m, and we call the array the *parameterized border array* (*p-border array*) of the pattern p-string, which is the parameterized version of the border array [15]. The p-border array of a given pattern can be computed in linear time [13].

1.2 Reverse and Enumerating Problems on Strings

The reverse problem for standard border arrays [15] was first introduced by Franěk et al. [16]. They proposed a linear time algorithm to verify if a given integer array is the border array of some string. Their algorithm works for both bounded and unbounded alphabets. Duval et al. [17] proposed a simpler algorithm to solve the same problem in linear time for bounded alphabets.

Moore et al. [18] presented an algorithm to enumerate all border arrays of length at most n, where n is a given positive integer. They proposed a notion of *b-equivalence* of strings such that two strings are b-equivalent if they have the same border array. The lexicographically smallest one of each b-equivalence class is called *b-canonical* string of the class. Their algorithm is also able to output all b-canonical strings of length up to a given integer n. Franěk et al. [16] pointed out that the time complexity analysis of [18] is incorrect, and showed a new algorithm which solves the same problem in $O(b_n)$ time using $O(b_n)$ space, where b_n denotes the number of border arrays of length at most n.

The reverse problem for some other string data structures, such as suffix arrays [19], directed acyclic word graphs [20], directed acyclic subsequence graphs [21] have been solved in linear time [22, 23]. The problem of enumerating all suffix arrays was considered in [24]. An algorithm to enumerate all p-distinct strings was proposed in [18], where two strings are said to be p-distinct if they do *not* parameterized-match.

1.3 Our Contribution

This paper considers the reversal of the problem of computing the p-border array of a given pattern p-string. That is, given an integer array α, determine if there exists a p-string whose p-border array is α. In this paper, we present a linear time algorithm which solves the above problem for a binary parameter alphabet ($|\Pi| = 2$). We then consider a more challenging problem: given a positive integer n, enumerate all p-border arrays of length at most n. We propose an algorithm that solves the enumerating problem in $O(B_n)$ time for a binary parameter alphabet, where B_n is the number of all p-border arrays of length n for a binary parameter alphabet. We also give a simple algorithm to output all strings which share the same p-border array.

A p-border is a dual concept of a *parameterized period* of a p-string. Apostolico and Giancarlo [11] showed that a complete analogy to the weak periodicity lemma [25] stands for p-strings over a binary alphabet. Our result reveals yet another similarity of p-strings over a binary alphabet and standard strings in terms of periodicity.

2 Preliminaries

2.1 Parameterized String Matching

Let Σ and Π be two disjoint finite sets of *constant symbols* and *parameter symbols*, respectively. An element of $(\Sigma \cup \Pi)^*$ is called a *p-string*. The length of any p-string s is the total number of constant and parameter symbols in s and is denoted by $|s|$. For any p-string s of length n, the i-th symbol is denoted by $s[i]$ for each $1 \leq i \leq n$, and the *substring* starting at position i and ending at position j is denoted by $s[i:j]$ for $1 \leq i \leq j \leq n$. In particular, $s[1:j]$ and $s[i:n]$ denote the *prefix* of length j and the *suffix* of length $n-i+1$ of s, respectively.

Any two p-strings s and t of the same length m are said to *parameterized match* if s can be transformed into t by applying a renaming function f from the symbols of s to the symbols of t, such that f is the identity on the constant alphabet. For example, let $\Pi = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, $\Sigma = \{\mathtt{X}, \mathtt{Y}\}$, $s = \mathtt{abcXabY}$ and $t = \mathtt{bcaXbcY}$. We then have $s \simeq t$ with the renaming function f such that $f(\mathtt{a}) = \mathtt{b}$, $f(\mathtt{b}) = \mathtt{c}$, $f(\mathtt{c}) = \mathtt{a}$, $f(\mathtt{X}) = \mathtt{X}$, and $f(\mathtt{Y}) = \mathtt{Y}$. We write $s \simeq t$ when s and t p-match.

Amir et al. [5] showed that we have only to consider p-strings over Π when considering p-matching.

Lemma 1 ([5]). *The p-matching problem on alphabet $\Sigma \cup \Pi$ is reducible in linear time to the p-matching problem on alphabet Π.*

2.2 Parameterized Border Arrays

As in the case of standard string matching, we can define the parameterized border (p-border) and the parameterized border array (p-border array).

Definition 1. *A* parameterized border *(*p-border*) of a p-string s of length n is any integer j such that $0 \leq j < n$ and $s[1:j] \simeq s[n-j+1:n]$.*

For example, the set of p-borders of p-string $\mathtt{aabbaa}$ is $\{4, 2, 1, 0\}$, since $\mathtt{aabb} \simeq \mathtt{bbaa}$, $\mathtt{aa} \simeq \mathtt{aa}$, $\mathtt{a} \simeq \mathtt{a}$, and $\varepsilon \simeq \varepsilon$.

Definition 2. *The* parameterized border array *(*p-border array*) β_s of any p-string s of length n is an array of length n such that $\beta_s[i] = j$, where j is the longest p-border of $s[1:i]$.*

For example, the p-border array of p-string $\mathtt{aabbaa}$ is $[0, 1, 1, 2, 3, 4]$.

When it is clear from the context, we abbreviate β_s as β.

The p-border array β_s of p-string s was first explicitly introduced by Idury and Schäffer [13] as the *pfail* function, where the *pfail* function is used in their Aho-Corasick [14] type algorithm that solves the p-matching problem for multiple patterns. Given a pattern p-string p of length m, the p-border array β_p can be computed in $O(m \log |\Pi|)$ time, and the p-matching problem can be solved in $O(n \log |\Pi|)$ time for any text p-string of length n.

2.3 Problems

This paper deals with the following problems.

Problem 1 (Verifying valid p-border array). Given an integer array α of length n, determine if there exists a p-string s such that $\beta_s = \alpha$.

Problem 2 (Computing all p-strings sharing the same p-border array). Given an integer array α which is a valid p-border array, compute every p-string s such that $\beta_s = \alpha$.

Problem 3 (Computing all p-border arrays). Given a positive integer n, compute all p-border arrays of length at most n.

In the following section, we give efficient solutions to the above problems for a binary alphabet, that is, $|\Pi| = 2$.

3 Algorithms

This section presents our algorithms which solve Problem 1, Problem 2 and Problem 3 for the case $|\Pi| = 2$.

We begin with the basic proposition on p-border arrays.

Proposition 1. *For any p-border array $\beta[1..i]$ of length $i \geq 2$, $\beta[1..i-1]$ is a p-border array of length $i - 1$.*

Proof. Let s be any p-string such that $\beta_s = \beta$. It is clear from Definition 2 that $\beta_s[1..i-1]$ is the p-border array of the p-string $s[1 : i-1]$. □

Due to the above proposition, given an integer array $\alpha[1..n]$, we can check if it is a p-border array of some string of length n by testing each element of α in increasing order (from 1 to n). If we find any $1 \leq i \leq n$ such that $\alpha[1..i]$ is not a p-border array of length i, then $\alpha[1..n]$ can never be a p-border of length n. In what follows, we show how to check each element of a given integer array in increasing order.

For any p-border array β of length n and any integer $1 \leq i \leq n$, let

$$\beta^k[i] = \begin{cases} \beta[i] & \text{if } k = 1, \\ \beta[\beta^{k-1}[i]] & \text{if } k > 1 \text{ and } \beta^{k-1}[i] \geq 1. \end{cases}$$

It follows from Definition 2 that the sequence $i, \beta[i], \beta^2[i], \ldots$ is monotone decreasing to zero, hence finite.

Lemma 2. *For any p-string s of length i, $\{\beta_s^1[i], \beta_s^2[i], \ldots, 0\}$ is the set of the p-borders of s.*

Proof. First we show by induction that for every k, $1 \leq k \leq k'$, $\beta_s^k[i]$ is a p-border of s, where k' is the integer such that $\beta_s^{k'}[i] = 0$. By Definition 2, $\beta_s^1[i]$ is the longest p-border of s. Suppose that for some k, $1 \leq k < k'$, $\beta_s^k[i]$ is a p-border of s. Here $\beta_s^{k+1}[i]$ is the longest p-border of $\beta_s^k[i]$. Let f and g be the bijections such that

$$f(s[1])f(s[2])\cdots f(s[\beta_s^k[i]]) = s[i - \beta_s^k[i] + 1 : i],$$
$$g(s[1])g(s[2])\cdots g(s[\beta_s^{k+1}[i]]) = s[\beta_s^k[i] - \beta_s^{k+1}[i] + 1 : \beta_s^k[i]].$$

Since

$$\begin{aligned}
&f(g(s[1]))f(g(s[2]))\cdots f(g(s[\beta_s^{k+1}[i]])) \\
&= f(s[\beta_s^k[i] - \beta_s^{k+1}[i] + 1])f(s[\beta_s^k[i] - \beta_s^{k+1}[i] + 2])\cdots f(s[\beta_s^k[i]]) \\
&= s[i - \beta_s^{k+1}[i] + 1 : i],
\end{aligned}$$

we obtain $s[1 : \beta_s^{k+1}[i]] \simeq s[i - \beta_s^{k+1}[i] + 1 : i]$. Hence $\beta_s^{k+1}[i]$ is a p-border of s.

We now show any other j is not a p-border of s. Assume for contrary that j, $\beta_s^{k+1}[i] < j < \beta_s^k[i]$, is a p-border of s. Let q be the bijection such that

$$q(s[i - j + 1])q(s[i - j + 2])\cdots q(s[i]) = s[1 : j].$$

Since

$$\begin{aligned}
&q(f(s[\beta_s^k[i] - j + 1]))q(f(s[\beta_s^k[i] - j + 2]))\cdots q(f(s[\beta_s^k[i]])) \\
&= q(s[i - j + 1])q(s[i - j + 2])\cdots q(s[i]) \\
&= s[1 : j],
\end{aligned}$$

we obtain $s[1 : j] \simeq s[\beta_s^k[i] - j + 1 : \beta_s^k[i]]$. Hence j is a p-border of $s[1 : \beta_s^k[i]]$. However this contradicts with the definition that $\beta_s^{k+1}[i]$ is the longest p-border of $s[1 : \beta_s^k[i]]$. □

Lemma 3. *For any p-string s of length $i \geq 1$ and $a \in \Pi$, every p-border of sa is an element of the set $\{\beta_s^1[i] + 1, \beta_s^2[i] + 1, \ldots, 1\}$.*

Proof. Assume for contrary that sa has a p-border $j + 1 \notin \{\beta_s^1[i] + 1, \beta_s^2[i] + 1, \ldots, 1\}$. Since $s[1 : j + 1] \simeq s[i - j + 1 : i]a$, we have $s[1 : j] \simeq s[i - j + 1 : i]$ and j is a p-border of s. It follows from Lemma 2 that $j \in \{\beta_s^1[i], \beta_s^2[i], \ldots, 0\}$. This contradicts with the assumption. □

Based on Lemma 2 and Lemma 3, we can efficiently compute the p-border array β_s of a given p-string s. Also, our algorithm to solve Problem 1 is based on these lemmas. Note that Proposition 1, Lemma 2 and Lemma 3 hold for p-strings over Π of arbitrary size.

In the sequel we show how to select $m \in \{\beta_s^1[i]+1, \beta_s^2[i]+1, \ldots, 1\}$ such that $\beta_s[1..i]m$ is a valid p-border array of length $i + 1$. The following proposition, lemmas and theorems hold for a binary parameter alphabet, $|\Pi| = 2$.

For p-border arrays of length at most 2, we have the next proposition.

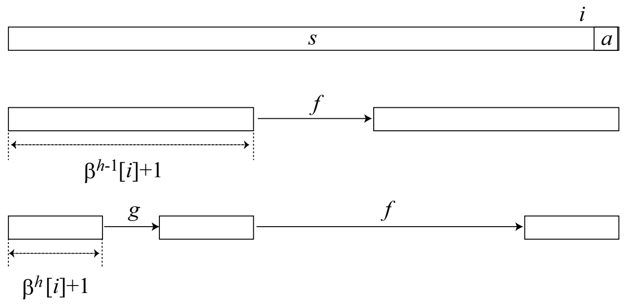

Fig. 1. Illustration for Lemma 5.

Proposition 2. *For any p-string s of length 1, $\beta_s[1] = 0$. For any p-string s' of length 2, $\beta_{s'}[2] = 1$.*

Proof. Let $\Pi = \{a, b\}$. It is clear that the longest p-border of a and b is 0. The p-strings of length 2 over Π are aa, ab, ba, and bb. Obviously the longest p-border of each of them is 1. □

For p-border arrays of length more than 2, we have the following lemmas.

Lemma 4. *For any p-string $s \in \Pi^*$, if $j \geq 2$ is a p-border of sa with $a \in \Pi$, then j is not a p-border of sb, where $b \in \Pi - \{a\}$.*

Proof. Assume for contrary that j is a p-border of sb. Then, let f and g be the bijections on Π such that

$$f(s[1])f(s[2])\cdots f(s[j]) = s[i-j+2:i]a,$$
$$g(s[1])g(s[2])\cdots g(s[j]) = s[i-j+2:i]b.$$

We get from $f(s[1])f(s[2])\cdots f(s[j-1]) = s[i-j+2:i] = g(s[1])f(s[2])\cdots g(s[j-1])$ that f and g are the same bijections. However, $f(s[j]) = a \neq b = g(s[j])$ implies that f and g are different bijections, a contradiction. Hence j is not a p-border of sb. □

Lemma 5. *For any p-string s of length i, if $\beta_s[\beta_s^{h-1}[i]+1] = \beta_s^h[i]+1$ and $\beta_s^{h-1}[i]+1$ is a p-border of sa with $a \in \Pi$, then $\beta_s^h[i]+1$ is a p-border of sa. (See also Fig. 1.)*

Proof. Let f and g be the bijections on Π such that

$$f(s[1])f(s[2])\cdots f(s[\beta_s^{h-1}[i]+1]) = s[i-\beta_s^{h-1}[i]+1:i]a,$$
$$g(s[1])g(s[2])\cdots g(s[\beta_s^h[i]+1]) = s[\beta_s^{h-1}[i]-\beta_s^h[i]+1:\beta_s^{h-1}[i]+1].$$

Since

$$\begin{aligned} & f(g(s[1]))f(g(s[2]))\cdots f(g(s[\beta_s^h[i]+1])) \\ &= f(s[\beta_s^{h-1}[i]-\beta_s^h[i]+1])f(s[\beta_s^{h-1}[i]-\beta_s^h[i]+2])\cdots f(s[\beta_s^{h-1}[i]+1]) \\ &= s[i-\beta_s^h[i]+1:i]a, \end{aligned}$$

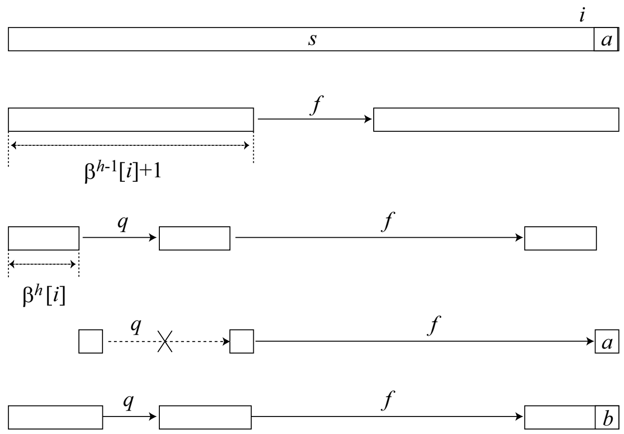

Fig. 2. Illustration for Lemma 6.

we obtain $s[1:\beta_s^h[i]+1] \simeq s[i-\beta_s^h[i]+1:i]a$. Hence $\beta_s^h[i]+1$ is a p-border of sa. □

Lemma 6. *For any p-string s of length i, if $\beta_s[\beta_s^{h-1}[i]+1] \neq \beta_s^h[i]+1$ and $\beta_s^{h-1}[i]+1$ is a p-border of sa with $a \in \Pi$, then $\beta_s^h[i]+1$ is a p-border of sb such that $b \in \Pi - \{a\}$. (See also Fig. 2.)*

Proof. Let f and g be the bijections on Π such that

$$f(s[1])f(s[2])\cdots f(s[\beta_s^{h-1}[i]+1]) = s[i-\beta_s^{h-1}[i]+1:i]a,$$
$$q(s[1])q(s[2])\cdots q(s[\beta_s^h[i]]) = s[\beta_s^{h-1}[i]-\beta_s^h[i]+1:\beta_s^{h-1}[i]].$$

Because $\beta_s[\beta_s^{h-1}[i]+1] \neq \beta_s^h[i]+1$, we know that $q(s[\beta_s^h[i]+1]) \neq s[\beta_s^{h-1}[i]+1]$. Since $f(s[\beta_s^{h-1}[i]+1]) = a$ and $\Pi = \{a,b\}$, $f(q(s[\beta_s^h[i]+1])) = b$. Hence $\beta_s^h[i]+1$ is a p-border of sb. □

The following is a key lemma to solving our problems.

Lemma 7. *For any p-border array β of length $i \geq 2$, $\beta[1..i]m_1$ and $\beta[1..i]m_2$ are the p-border arrays of length $i+1$, where $m_1 = \beta[i]+1$ and*

$$m_2 = \begin{cases} \beta^l[i]+1 & \text{if } \beta[\beta^{l-1}[i]+1] \neq \beta^l[i]+1 \text{ for some } 1 < l < k' \text{ and} \\ & \beta[\beta^{h-1}[i]+1] = \beta^h[i]+1 \text{ for any } 1 < h < l, \\ 1 & \text{otherwise,} \end{cases}$$

where k' is the integer such that $\beta^{k'}[i] = 0$.

Proof. Consider any p-string s of length i such that $\beta_s = \beta$. By definition, there exists a bijection f on Π such that $f(s[1])f(s[2])\cdots f(s[\beta[i]]) = s[i-\beta[i]+1:i]$. Let $a = f(s[\beta[i]+1])$. Then $f(s[1])f(s[2])\cdots f(s[\beta[i]])f(s[\beta[i]+1]) = s[i-\beta[i]+1:$

Algorithm 1: Algorithm to solve Problem 1

```
Input: α[1..n] : a given integer array
Output: return whether α is a valid p-border array or not
if α[1..2] ≠ [0, 1] then return invalid;
for i = 3 to n do
    if α[i] = α[i − 1] + 1 then continue;
    d′ ← α[i − 1];
    d ← α[d′];
    while d > 0 & d + 1 = α[d′ + 1] do
        d′ ← d;
        d ← α[d′];
    if α[i] = d + 1 then continue;
    return invalid;
return valid;
```

$i]a$. Note that $\beta[1..i](\beta[i]+1)$ is the p-border array of sa because sa can have no p-borders longer than $\beta[i]+1$.

It follows from Lemma 5 that $\beta^h[i]+1$ is a p-border of sa. Then, by Lemma 6, $\beta^l[i]+1$ is a p-border of sb. Since $\beta^h[i] \geq 1$, by Lemma 4, $\beta^h[i]+1$ is not a p-border of sb. Hence $\beta^l[i]+1$ is the longest p-border of sb. □

We are ready to state the following theorem.

Theorem 1. *Problem 1 can be solved in linear time for a binary parameter alphabet.*

Proof. Algorithm 1 describes the operations to solve Problem 1. Given an integer array of length n, the algorithm first checks if $\alpha[1..2] = [0,1]$ due to Proposition 2. If $\alpha[1..2] = [0,1]$, then for each $i = 3, \ldots, n$ (in increasing order) the algorithm checks whether $\alpha[i]$ satisfies one of the conditions of Lemma 7.

The time analysis is similar to that of Theorem 2.3 of [16]. In each iteration of the **for** loop, the value of d' increases by at most 1. However, each execution of the **while** loop decreases the value of d' by at least 1. Hence the total time cost of the **for** loop is $O(n)$. □

Theorem 2. *Problem 2 can be solved in linear time for a binary parameter alphabet.*

Proof. It follows from Proposition 2 that the p-border array of all p-string of length 2 (aa, ab, ba, and bb) is $[0,1]$. By Proposition 1, for any p-border array $\beta[1..n]$ with $n \geq 2$, we have $\beta[1..2] = [0,1]$. Hence each p-border array $\beta[1..n]$ with $n \geq 2$ corresponds to exactly four p-strings each of which begins with aa, ab, ba, and bb, respectively. Algorithm 2 is an algorithm to solve Problem 2. Technically x_{aa} can be computed by $s_{aa}[\beta[i]]$ xor $s_{aa}[\beta[i]+1]$ xor $s_{aa}[i]$ on binary alphabet $\Pi = \{0,1\}$. Hence this counting algorithm works in linear time. □

Algorithm 2: Algorithm to compute all p-strings sharing the same p-border array

Input: $\beta[1..n]$: a p-border array
Output: all p-strings sharing the same p-border array $\beta[1..n]$

$s_{aa} \leftarrow aa$; $s_{ab} \leftarrow ab$; $s_{bb} \leftarrow bb$; $s_{ba} \leftarrow ba$;
for $i = 3$ **to** n **do**
    Let f be the bijection on Π s.t. $f(s_{aa}[\beta[i]]) = s_{aa}[i]$;
    Let g be the bijection on Π s.t. $g(s_{ab}[\beta[i]]) = s_{ab}[i]$;
    $x_{aa} \leftarrow f(s_{aa}[\beta[i] + 1])$; $x_{ab} \leftarrow g(s_{ab}[\beta[i] + 1])$;
    $\overline{x_{aa}} \leftarrow y \in \Pi - \{x_{aa}\}$; $\overline{x_{ab}} \leftarrow z \in \Pi - \{x_{ab}\}$;
    if $\beta[i] = \beta[i-1] + 1$ **then**
        $s_{aa}[i] \leftarrow x_{aa}$; $s_{ab}[i] \leftarrow x_{ab}$;
        $s_{bb}[i] \leftarrow \overline{x_{aa}}$; $s_{ba}[i] \leftarrow \overline{x_{ab}}$;
    else
        $s_{aa}[i] \leftarrow \overline{x_{aa}}$; $s_{ab}[i] \leftarrow \overline{x_{ab}}$;
        $s_{bb}[i] \leftarrow x_{aa}$; $s_{ba}[i] \leftarrow x_{ab}$;
    end
Output: $s_{aa}[1 : n]$, $s_{ab}[1 : n]$, $s_{bb}[1 : n]$, $s_{ba}[1 : n]$

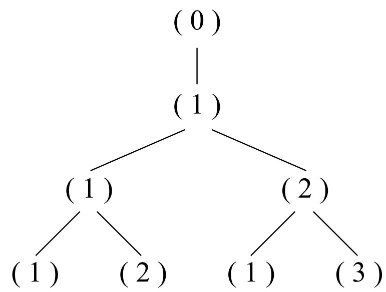

Fig. 3. The tree T_4 which represents all p-border arrays of length at most 4 for a binary alphabet.

We now consider Problem 3. By Proposition 1 and Lemma 7, computing all p-border arrays of length at most n can be accomplished using a rooted tree structure T_n of height $n - 1$. Each node of T_n of height $i - 1$ corresponds to an integer j such that j is the longest p-border of some p-string of length i over a binary alphabet, hence the path from the root to that node represents the p-border array of the p-string. Fig. 3 represents T_4.

Theorem 3. *Problem 3 can be solved in $O(B_n)$ time for a binary parameter alphabet, where B_n denotes the number of p-border arrays of length n.*

Proof. Proposition 2 and Lemma 7 imply that every internal node of T_n of height at least 1 has exactly two children. Hence the total number of nodes of T_n is $O(B_n)$. We compute T_n in a depth-first manner. Algorithm 3 shows a function that computes the children of a given node of T_n. It is not difficult to see that

Algorithm 3: Function to compute the children of a node of T_n

Input: i : length of the current p-border array, $2 \le i \le n$
Result: compute the children of the current node
// $\beta[1..n]$ `is allocated globally and` $\beta[1..i]$ `represents the current p-border array.`

function `getChildren`(i)
if $i = n$ **then** **return** ;
$\beta[i+1] \leftarrow \beta[i] + 1$;
report $\beta[i+1]$;
`getChildren`$(i+1)$;
$d' \leftarrow \beta[i]$;
$d \leftarrow \beta[d']$;
while $d > 0$ & $d + 1 = \beta[d' + 1]$ **do**
  $d' \leftarrow d$;
  $d \leftarrow \beta[d']$;
$\beta[i+1] \leftarrow d + 1$;
report $\beta[i+1]$;
`getChildren`$(i+1)$;
return ;

each child of a given node can be computed in amortized constant time. Hence Problem 3 can be solved in $O(B_n)$ time for a binary parameter alphabet. □

We remark that if each p-border array in T_n can be discarded after it is generated, then we can compute all p-border arrays of length at most n using $O(n)$ space. Since every internal node of T_n of height at least 1 has exactly two children and the root has one child, $B_n = 2^{n-2}$ for $n \ge 2$. Thus the space requirement can be reduced to $O(\log B_n)$.

4 Conclusions and Open Problems

A parameterized border array (p-border array) is a useful data structure for parameterized pattern matching. In this paper, we presented a linear time algorithm which tests if a given integer array is a valid p-border array for a binary alphabet. We also gave a linear time algorithm to compute all binary p-strings that share a given p-border array. Finally, we proposed an algorithm which computes all p-border arrays of length at most n, where n is a given threshold. This algorithm works in $O(B_n)$ time, where B_n denotes the number of p-border arrays of length n for a binary alphabet.

Problems 1,2, and 3 are open for a larger alphabet. To see one of the reasons of why, we show that Lemma 4 does not hold for a larger alphabet. Consider a p-string $s = \mathtt{abac}$ over $\Pi = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. Observe that $\beta_s = [0, 1, 2, 2]$. Although $\beta_s[4] = 2$ is a p-border of `abac`, it is also a p-border of another p-string `abab` since `ab` $\simeq$ `ab`. Hence Lemma 4 does not hold if $|\Pi| \ge 3$.

Our future work also includes the following:

- Verify if a given integer array is a *parameterized suffix array* [12].
- Compute all parameterized suffix arrays of length at most n.

In [12], a linear time algorithm which directly constructs the parameterized suffix array for a given binary string was proposed. This algorithm might be used as a basis for solving the above problems regarding parameterized suffix arrays.

References

1. Baker, B.S.: Parameterized pattern matching: Algorithms and applications. Journal of Computer and System Sciences **52**(1) (1996) 28–42
2. Baker, B.S.: A program for identifying duplicated code. Computing Science and Statistics **24** (1992) 49–57
3. Fredriksson, K., Mozgovoy, M.: Efficient parameterized string matching. Information Processing Letters **100**(3) (2006) 91–96
4. Shibuya, T.: Generalization of a suffix tree for RNA structural pattern matching. Algorithmica **39**(1) (2004) 1–19
5. Amir, A., Farach, M., Muthukrishnan, S.: Alphabet dependence in parameterized matching. Information Processing Letters **49**(3) (1994) 111–115
6. Baker, B.S.: Parameterized pattern matching by Boyer-Moore-type algorithms. In: Proc. 6th annual ACM-SIAM Symposium on Discrete Algorithms (SODA'95). (1995) 541–550
7. Kosaraju, S.: Faster algorithms for the construction of parameterized suffix trees. In: Proc. 36th Annual Symposium on Foundations of Computer Science (FOCS'95). (1995) 631–637
8. Hazay, C., Lewenstein, M., Tsur, D.: Two dimensional parameterized matching. In: Proc. 16th Annual Symposium on Combinatorial Pattern Matching (CPM'05). Volume 3537 of Lecture Notes in Computer Science. (2005) 266–279
9. Hazay, C., Lewenstein, M., Sokol, D.: Approximate parameterized matching. ACM Transactions on Algorithms **3**(3) (2007) Article No. 29.
10. Apostolico, A., Erdös, P.L., Lewenstein, M.: Parameterized matching with mismatches. Journal of Discrete Algorithms **5**(1) (2007) 135–140
11. Apostolico, A., Giancarlo, R.: Periodicity and repetitions in parameterized strings. Discrete Applied Mathematics **156**(9) (2008) 1389–1398
12. Deguchi, S., Higashijima, F., Bannai, H., Inenaga, S., , Takeda, M.: Parameterized suffix arrays for binary strings. In: Proc. The Prague Stringology Conference '08 (PSC'08). (2008) 84–94
13. Idury, R.M., Schäffer, A.A.: Multiple matching of parameterized patterns. Theoretical Computer Science **154**(2) (1996) 203–224
14. Aho, A.V., Corasick, M.J.: Efficient string matching: An aid to bibliographic search. Communications of the ACM **18**(6) (1975) 333–340
15. Morris, J.H., Pratt, V.R.: A linear pattern-matching algorithm. Technical Report Report 40, University of California, Berkeley (1970)
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. Combinatorial Math. and Combinatorial Computing **42** (2002) 223–236
17. Duval, J.P., Lecroq, T., Lefevre, A.: Border array on bounded alphabet. Journal of Automata, Languages and Combinatorics **10**(1) (2005) 51–60

18. Moore, D., Smyth, W., Miller, D.: Counting distinct strings. Algorithmica **23**(1) (1999) 1–13
19. Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. SIAM J. Computing **22**(5) (1993) 935–948
20. Blumer, A., Blumer, J., Haussler, D., Ehrenfeucht, A., Chen, M.T., Seiferas, J.: The smallest automaton recognizing the subwords of a text. Theoretical Computer Science **40** (1985) 31–55
21. Baeza-Yates, R.A.: Searching subsequences (note). Theoretical Computer Science **78**(2) (1991) 363–376
22. Duval, J.P., Lefebvre, A.: Words over an ordered alphabet and suffix permutations. Theoretical Informatics and Applications **36** (2002) 249–259
23. Bannai, H., Inenaga, S., Shinohara, A., Takeda, M.: Inferring strings from graphs and arrays. In: Proc. 28th International Symposium on Mathematical Foundations of Computer Science (MFCS 2003). Volume 2747 of Lecture Notes in Computer Science. (2003) 208–217
24. Schürmann, K.B., Stoye, J.: Counting suffix arrays and strings. Theoretical Computer Science **395**(2-1) (2008) 220–234
25. Lyndon, R.C., Schützenberger, M.P.: The equation $a^M = b^N c^P$ in a free group. Michigan Math. J. **9**(4) (1962) 289–298