New and Efficient Approaches to the Quasiperiodic Characterisation of a String

Tomáš Flouri[1], Costas S. Iliopoulos[2,3], Tomasz Kociumaka[4], Solon P. Pissis[1,5⋆], Simon J. Puglisi[2⋆⋆], William F. Smyth[2,3,6], and Wojciech Tyczyński[4]

[1] Heidelberg Institute for Theoretical Studies, 35 Schloss-Wolfsbrunnenweg, Heidelberg D-69118, Germany
{tomas.flouri,solon.pissis}@h-its.org
[2] King's College London, Dept. of Informatics, The Strand, London WC2R 2LS, UK
{c.iliopoulos,simon.puglisi}@kcl.ac.uk
[3] University of Western Australia, School of Mathematics and Statistics, 35 Stirling Highway, Crawley, Perth WA 6009, Australia
[4] University of Warsaw, Faculty of Mathematics, Informatics and Mechanics, Warsaw, Poland
{kociumaka,w.tyczynski}@mimuw.edu.pl
[5] University of Florida, Florida Museum of Natural History, 1659 Museum Road, Gainesville, FL 32611, USA
[6] McMaster University, Dept. of Computing and Software, 1280 Main St. West, Hamilton, Ontario L8S 4K1, Canada
smyth@mcmaster.ca

Abstract. A factor u of a string y is a *cover* of y if every letter of y lies within some occurrence of u in y; thus every cover u is also a *border* – both prefix and suffix – of y. A string y covered by u thus generalises the idea of a *repetition*; that is, a string composed of exact concatenations of u. Even though a string is coverable somewhat more frequently than it is a repetition, still a string that can be covered by a single u is rare. As a result, seeking to find a more generally applicable and descriptive notion of cover, many articles were written on the computation of a *minimum k-cover* of y; that is, the minimum cardinality set of strings of length k that collectively cover y. Unfortunately, this computation turns out to be NP-hard. Therefore, in this article, we propose new, simple, easily-computed, and widely applicable notions of string covering that provide an intuitive and useful characterisation of a string and its prefixes: the *enhanced cover* and the *enhanced cover array*.

Keywords: periodicity, quasiperiodicity, covers

1 Introduction

The notion of periodicity in strings and its many variants have been well-studied in many fields like combinatorics on strings, pattern matching, data compression, automata theory, formal language theory, and molecular biology (cf. [18]). Periodicity is of paramount importance in many applications – for example, periodic factors in DNA are of interest to genomics researchers [16] – as well as in theoretical studies in combinatorics on words. Not long ago the term *regularity* [10] was coined to cover such variants, and a recent survey [22] provides coverage of the exact regularities so far identified and the sequential algorithms proposed to compute them. In this article, in an effort to capture a more natural characterisation of a string in terms of its factors, we introduce a new form of regularity that is both descriptive and easy to compute.

⋆ Supported by the NSF-funded iPlant Collaborative (NSF grant #DBI-0735191).
⋆⋆ Supported by a Newton Fellowship.

Tomáš Flouri, Costas S. Iliopoulos, Tomasz Kociumaka, Solon P. Pissis, Simon J. Puglisi, William F. Smyth, Wojciech Tyczyński: New and Efficient Approaches to the Quasiperiodic Characterisation of a String, pp. 75–88.
Proceedings of PSC 2012, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-05095-8

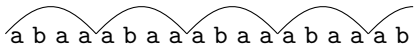

Figure 1. Periodicity in string abaaabaaabaaabaaab

A string y is a *repetition* if $y = u^k$ for some non-empty string u of length m and some integer $k \geq 2$; in this case, y has *period* m (see Fig. 1). But the notion of periodicity is too restrictive to provide a description of a string such as $x =$ abaababaaba, which is covered by copies of aba, yet not exactly periodic. To fill this gap, the idea of *quasiperiodicity* was introduced [1,2]. In a periodic string, the occurrences of the single periods do not overlap. In contrast, the quasiperiods of a quasiperiodic string may overlap. Quasiperiodicity thus enables the detection of repetitive structures that would be ignored by the classical characterisation of periods (see Fig. 2 in contrast with Fig. 3). The most well-known formalisation of quasiperiodicity is the cover of string. A factor u of length m of a string y of length n is said to be a *cover* of y if $m < n$, and every letter of y lies within some occurrence of u. Note that a cover of y must also be a *border* – both suffix and prefix – of y. Thus in the above example aba is a cover of $x =$ abaababaaba.

Figure 2. Quasiperiodicity in string abaabaabaaabaaab

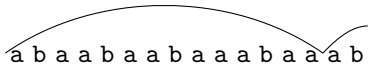

Figure 3. Periodicity in string abaabaabaaabaaab

In [3], Apostolico, Farach, and Iliopoulos described a recursive linear-time algorithm to compute the shortest cover of a string y of length n, if it has a cover; otherwise to report that no cover exists. Breslauer [4] introduced the *minimal cover array* C – an array of size n of integers such that $\mathsf{C}[i]$, for all $0 \leq i < n$, gives the shortest cover of $y[0\,..\,i]$, or zero if no cover exists. Moreover, he described an online linear-time algorithm to compute C. In [19,20], Moore and Smyth described a linear-time algorithm to compute *all* the covers of y. An $\mathcal{O}(\log(\log n))$-time parallel algorithm was given later by Iliopoulos and Park in [13]. Finally, Li and Smyth [17] introduced the *maximal cover array* C^{M} – an array of size n of integers such that $\mathsf{C}^{\mathrm{M}}[i]$ gives the longest cover of $y[0\,..\,i]$, or zero if no cover exists – and showed that, analogous to the border array [21], C^{M} actually specifies *all* the covers of every prefix of y. They then described a linear-time algorithm to compute C^{M}.

Still it remains unlikely that an arbitrary string, even on alphabet $\{a, b\}$, has a cover; for example, changing the above example x to $x' =$ abaaababaaba yields a string that not only has no cover, but whose every prefix also has no cover. Accordingly, in an effort to extend the descriptive power of quasiperiodicity, the notion of k-cover was introduced [14]: if for a given string y and a given positive integer k there

exists a set $\mathcal{C}_k$ of factors of y, each of length k, such that every letter of y lies within some occurrence of some element of $\mathcal{C}_k$, then $\mathcal{C}_k$ is said to be a *k-cover* of y; a *minimal k-cover* if no smaller set has this property. Originally it was thought, incorrectly, that a minimal k-cover of a string y could be computed in time polynomial in n [14], but then later the problem was shown to be NP-complete for every $k \geq 2$ [8], even though an approximate solution could be computed in polynomial time [11].

Our contribution. We have seen that while the notion of cover captures very well the repetitive nature of extremely repetitive strings, nevertheless most strings, and particularly those encountered in practice, will have no cover, and so this measure of repetitiveness breaks down. More strings will have a useful k-cover, but this feature is hard to compute. Therefore we introduce a new and more natural and applicable form of quasiperiodicity.

– **Enhanced cover.** A border u of a string y is an *enhanced cover* of y, if the number of letters of y which lie within some occurrence of u in y is a maximum over all borders of y (see Fig. 5).

  This gives rise to the following data structure.

– **Enhanced cover array.** An array of size n of integers is the *enhanced cover array* of y, if it stores the length of the enhanced cover for every prefix of y.

In this article, we present efficient methods for computing all enhanced covers and the enhanced cover array of a string, and, in particular, the *minimal enhanced cover* and the *minimal enhanced cover array*. These methods are based on the maintenance of a new, simple but powerful data structure, which stores the number of positions covered by some prefixes of the string. This data structure allows us to compute the minimal enhanced cover of a string of length n in time $\mathcal{O}(n)$ and the minimal enhanced cover array in time $\mathcal{O}(n \log n)$.

The rest of this article is structured as follows. In Section 2, we present basic definitions and notation used throughout this article, and we also formally define the problems solved. In Section 3, we prove several combinatorial properties of the borders of y, which may be of independent interest. In Section 4, we show how to compute a few auxiliary arrays, which will be used for designing the proposed algorithms. In Section 5, we present an algorithm for computing the minimal enhanced cover of y. In Section 6, we present an algorithm for computing the minimal enhanced cover array of y. In Section 7, we present some experimental results. Finally, we briefly conclude with some future proposals in Section 8.

2 Definitions and notation

An *alphabet* Σ is a finite non-empty set whose elements are called *letters*. A *string* on an alphabet Σ is a finite, possibly empty, sequence of elements of Σ. The zero-letter sequence is called the *empty string*, and is denoted by ε. The *length* of a string x is defined as the length of the sequence associated with the string x, and is denoted by $|x|$. We denote by $x[i]$, for all $0 \leq i < |x|$, the letter at index i of x. Each index i, for all $0 \leq i < |x|$, is a position in x when $x \neq \varepsilon$. It follows that the ith letter of x is the letter at position $i-1$ in x, and that $x = x[0\,..\,|x|-1]$.

The *concatenation* of two strings x and y is the string of the letters of x followed by the letters of y. It is denoted by xy. For every string x and every natural number n, we define the nth *power* of the string x, denoted by x^n, by $x^0 = \varepsilon$ and $x^k = x^{k-1}x$, for all $1 \leq k \leq n$. A string x is a *factor* of a string y if there exist two strings u and v, such that $y = uxv$. A factor x of a string y is *proper* if $x \neq y$. Let the strings x, y, u, and v be such that $y = uxv$. If $u = \varepsilon$, then x is a *prefix* of y. If $v = \varepsilon$, then x is a *suffix* of y.

Let x be a non-empty string. An integer p, such that $0 < p \leq |x|$, is called a *period* of x if $x[i] = x[i + p]$, for all $0 \leq i < |x| - p$. Note that the length of a non-empty string is a period of this string, so that every non-empty string has at least one period. We define thus without any ambiguity *the period* of a non-empty string x as the smallest of its periods. It is denoted by $\mathsf{per}(x)$. A *border* of a non-empty string x is a proper factor of x (including the empty string) that is both a prefix and a suffix of x. We define *the border* of a non-empty string x as the longest border of x. By $\mathsf{border}(x)$, we denote the length of the border of x. The notions of period and of border are dual. It is a known fact (cf. [9]) that, for any non-empty string x, it holds $\mathsf{per}(x) + \mathsf{border}(x) = |x|$. The *border array* B of a non-empty string y of length n is the array of size n of integers for which $\mathsf{B}[i]$, for all $0 \leq i < n$, stores the length of the border of the prefix $y[0\,..\,i]$ of y – zero if none.

A non-empty string u of length m is a *cover* of a non-empty string y if both $m < n$, and there exists a set of positions $P \subseteq \{0, \ldots, n - m\}$ that satisfies both $y[i\,..\,i + m - 1] = u$, for all $i \in P$, and $\bigcup_{i \in P}\{i, \ldots, i + m - 1\} = \{0, \ldots, n - 1\}$. In other words, u is a cover of y, if every letter of y lies within some occurrence of u in y, and $u \neq y$.

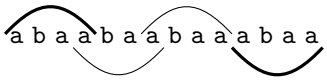

Figure 4. Cover of string abaabaabaaabaa

A string u is the *minimal cover* of string y if u is the shortest cover of y. The *minimal cover array* C of a non-empty string y of length n is the array of size n of integers for which $\mathsf{C}[i]$, for all $0 \leq i < n$, stores the length of the minimal cover of the prefix $y[0\,..\,i]$ of y – zero if none.

Definition 1. *A border u of a string y is an* enhanced cover *of y if the number of letters of y which lie within occurrences of u in y is a maximum over all borders of y.*

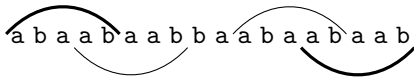

Figure 5. Enhanced cover of string abaabaabbaabaabaab

Definition 2. *We define as* minimal enhanced cover *the shortest enhanced cover of y.*

Definition 3. *The* minimal enhanced cover array *MEC of a non-empty string y of length n is the array of size n of integers for which $\mathsf{MEC}[i]$, for all $0 \leq i < n$, stores the length of the minimal enhanced cover of the prefix $y[0\,..\,i]$ of y – zero if none.*

Example 4. Consider the string $y =$ `abaaababaabaaaababaa`. The following table illustrates the border array B of y, the minimal cover array C of y, and the minimal enhanced cover array MEC of y. In this example, array C consists of only zeros, as a minimal cover does not exist for any of the prefixes of y. In contrast, array MEC is a more powerful data structure than array C, as apart from the minimal cover of every prefix, it also contains the minimal enhanced cover of every prefix in the case when a minimal cover does not exist. For instance, border `abaa` is the minimal enhanced cover of y, as 15 letters of y lie within some occurrence of `abaa` in y.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$y[i]$	a	b	a	a	a	b	a	b	a	a	b	a	a	a	a	b	a	b	a	a
$\mathsf{B}[i]$	0	0	1	1	1	2	3	2	3	4	2	3	4	5	1	2	3	2	3	4
$\mathsf{C}[i]$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\mathsf{MEC}[i]$	0	0	1	1	1	2	3	2	3	4	2	3	4	1	1	2	3	2	3	4

We consider the following problems for a non-empty string y.

Problem 5. Compute the minimal enhanced cover of y.

Problem 6. Compute the minimal enhanced cover array MEC of y.

3 Combinatorial properties of non-periodic borders

Definition 7. *A string w is called* periodic *if it is non-empty and $2\mathsf{per}(w) \leq |w|$. Otherwise it is called* non-periodic.

The efficiency of the algorithms in this article is a consequence of considering only non-periodic factors. The following fact explains why this restriction is valid.

Fact 1 *A periodic string always has a (proper) cover. As a consequence, the minimal enhanced cover is never periodic.*

Proof. Let w be a periodic string with the border u. As a factor of w, u has at least two occurrences: as a prefix and as a suffix. Since $2|u| \geq |w|$, these occurrences cover v. In particular, if w is an enhanced cover, then u is a shorter enhanced cover. Hence w cannot be the minimal enhanced cover. □

We prove two combinatorial properties of non-periodic borders, which are then used to prove the time complexities of our algorithms. The first one is simple fact, but, with corollaries, is the main reason behind the restriction to non-periodic borders.

Fact 2 *Let u and v be borders of y, such that $|v| > |u|$, and v is non-periodic. Then $|v| > 2|u|$.*

Proof. Clearly, u is a border of v, hence $|v| - |u|$ is a period of v. However v is non-periodic, so $2(|v| - |u|) > |v|$, i.e. $|v| > 2|u|$. □

Corollary 8. *The length of the kth shortest non-periodic border of y is at least $2^k - 1$. In particular, the total number of the non-periodic borders of y is at most $\log n$.*

Proof. Let b_k be the length of the kth shortest non-periodic border of y. From Fact 2, $b_{k+1} \geq 2b_k + 1$. Moreover, $b_1 \geq 1$. The first part of Corollary follows by induction. For the second part, it is enough to see that if there were $k > \log n$ non-periodic borders, then $b_k \geq 2^k - 1 > n - 1$, which is clearly a contradiction. □

Note that by Corollary 8, the total number of occurrences of the non-periodic prefixes of y is $\mathcal{O}(n \log n)$. This is because if a prefix v ends with an occurrence of u, then $u = v$ or u is a border of v. This induces a one-to-one correspondence between such occurrences – with exception of $\mathcal{O}(n)$ ones starting 0 – and the non-periodic borders of prefixes of y. It turns out that, if we consider just the occurrences of non-periodic borders of y, the number drops to $\mathcal{O}(n)$.

Before we proceed, let us introduce a notion, which we use across the proofs below. Let X be an array of size n of integers for which $\mathsf{X}[i]$, for all $0 \leq i < n$, stores the number of those non-periodic borders of the prefix $y[0\,..\,i]$ of y, which are simultaneously borders of y.

Lemma 9. *The total number of occurrences of the non-periodic borders of y is linear. More precisely*

$$\sum_{i=0}^{n-1} \mathsf{X}[i] \leq 2n.$$

Let us start with an auxiliary claim.

Claim. Let $0 \leq i, j < n$ be integers. If $\mathsf{X}[i] > k$ then $i \geq 3 \cdot 2^k - 2$. Moreover, if $i < j$, $\mathsf{X}[i] > k$ and $\mathsf{X}[j] > k$ then $j - i \geq 2^k$.

Proof. Clearly, if u and v are non-periodic borders of y, then the shorter one is a non-periodic border of the longer one. Thus $\mathsf{X}[i] > k$ if and only if the $k+1$th shortest non-periodic border of y is a border of $y[0\,..\,i]$. Let us denote this border by b. By Corollary 8, $|b| \geq 2^{k+1} - 1$. Any two occurrences of b in y must have their starting positions distant by at least $\frac{|b|+1}{2} \geq 2^k$. A pair of closer occurrences would induce a period of b no larger than $\frac{|b|}{2}$, which may not exist, since b is non-periodic.

For the first part of the Claim, consider the occurrence starting at 0 and the occurrence ending at i, i.e. starting at $i - |b| + 1$. These are different occurrences, so $i \geq |b| - 1 + 2^k \geq 3 \cdot 2^k - 2$. In the second part, there are occurrences of b ending at i and at j, which implies that $j - i \geq 2^k$. □

Proof (of Lemma 9). In the following proof, we use the Iverson bracket $[P]$. For a logical statement P, the Iverson bracket $[P]$ is by definition equal to 1 if P is satisfied, and 0 otherwise.

Clearly for a non-negative integer m we have $m = \sum_{k=0}^{\infty} [k < m]$. Hence

$$\sum_{i=0}^{n-1} \mathsf{X}[i] = \sum_{i=0}^{n-1} \sum_{k=0}^{\infty} [k < \mathsf{X}[i]] = \sum_{k=0}^{\infty} \sum_{i=0}^{n-1} [\mathsf{X}[i] > k].$$

Let us bound the single term of the outer sum. This sum counts positions i such that $\mathsf{X}[i] > k$. By Claim, the first of them is at least $3 \cdot 2^k - 2$, and the distance between any two such positions is at least 2^k. This means that if we write them all in increasing order, then the mth one is at least $(m+2) \cdot 2^k - 2$. In particular, for $m > \frac{n}{2^k}$, the

mth position would be at least $n + 2 \cdot 2^k - 2 \geq n$, which is clearly impossible. Hence, there cannot be more than $\frac{n}{2^k}$ such positions, i.e.

$$\sum_{i=0}^{n-1} [\mathsf{X}[i] > k] \leq \frac{n}{2^k}.$$

Therefore

$$\sum_{i=0}^{n-1} \mathsf{X}[i] = \sum_{k=0}^{\infty} \sum_{i=0}^{n-1} [\mathsf{X}[i] > k] \leq \sum_{k=0}^{\infty} \frac{n}{2^k} = 2n.$$

□

4 Auxiliary arrays

Our algorithms make use of a few auxiliary arrays. In this section, we show how to compute these arrays efficiently. Below we assume the availability of the border array B, which is computable in linear time (see, e.g. [15,9,21]), and an array CB of size n such that, for all $0 \leq i < n$, $\mathsf{CB}[i]$ is 1 if $i + 1$ is a border of y, and 0 otherwise. CB is trivially computed from B as the borders of y are exactly the longest border of y and its borders.

Definition 10. *Given a string y of length n, the* pruned border array *A is an array of size n of integers for which $\mathsf{A}[i]$, for all $0 \leq i < n$, stores the length of the longest non-periodic border of $y[0\,..\,i]$ – zero if none.*

The pruned border array A of a string y can be computed by Algorithm 1 in linear time. Algorithm 1 loops through the prefixes of y and in each step considers two cases. If the longest border u of a prefix v is non-periodic, then u is the border of v we are looking for. Otherwise, the longest non-periodic border of v is the longest non-periodic border of u.

Algorithm 1: PrunedBorderArray

Input : The border array B of string $y[0\,..\,n-1]$
Output: The pruned border array A

```
1 for i ← 0 to n − 1 do
2     b ← B[i]
3     if b = 0 or 2 · B[b − 1] < b then
4         A[i] ← b
5     else
6         A[i] ← A[b − 1]
```

Definition 11. *Given a string y of length n, let R be an array of size n of integers for which $\mathsf{R}[i]$, for all $0 \leq i < n$, stores the length of the longest non-periodic border of $y[0\,..\,i]$, which is also a border of y – zero if none.*

R can be computed by Algorithm 2 in linear time. For each prefix v of y we determine the longest non-periodic border u of v and consider the following cases. If u is a border of y (in particular if u is empty), then clearly $\mathsf{R}[i] = \mathsf{A}[i]$. Otherwise, the border we seek is a shorter non-periodic border of v, so the result for v is same as for u.

Algorithm 2: ARRAY R

Input : The pruned border array A and array CB of string $y[0\,..\,n-1]$
Output: Array R

```
1 for i ← 0 to n − 1 do
2     b ← A[i]
3     if b = 0 or CB[b − 1] = 1 then
4         R[i] ← b
5     else
6         R[i] ← R[b − 1]
```

Definition 12. *Given a string y of length n, PCP is an array of size n of integers for which PCP$[i]$, for all $0 \leq i < n$, stores the number of letters of y which lie within an occurrence of the non-periodic prefix of length $i+1$ having at least two occurrences in y – zero if the prefix is periodic or does not have two occurrences.*

The PCP array of string y can be computed by Algorithm 3. It takes as input the pruned border array A of y. We also maintain an array LO of size n of integers, for which LO$[i]$, for all $0 \leq i < n$, stores the ending position of the last occurrence of the non-periodic prefix of length $i+1$ in y, not taking into account the occurrence as a prefix. Fields corresponding to periodic prefixes are never read or written.

Algorithm 3: POSITIONSCOVEREDBYPREFIXESARRAY

Input : The pruned border array A$[0\,..\,n-1]$ of string $y[0\,..\,n-1]$
Output: The PCP array

```
1  PCP ← FillWithZeros
2  for i ← 0 to n − 1 do
3      b ← A[i]
4      while b > 0 do
5          if PCP[b − 1] = 0 then
6              PCP[b − 1] ← min(2b, i + 1)
7          else
8              PCP[b − 1] ← PCP[b − 1] + min(b, i − LO[b − 1])
9          LO[b − 1] ← i
10         b ← A[b − 1]
```

The algorithm consists of an outer for loop, going through the pruned border array A, and an inner while loop, iterating through the non-periodic borders of prefix $y[0\,..\,i]$. If the first occurrence of some border of length b of $y[0\,..\,i]$ is found (line 5), we take the minimum between $2b$, that is in case $y[0\,..\,b-1]$ does not overlap with $y[i-b+1\,..\,i]$, and $i+1$, that is in case they overlap (line 6). If another occurrence of the same border is found (line 7), we update PCP$[b-1]$ by adding the minimum between b, that is in case $y[i-b+1\,..\,i]$ does not overlap with the last occurrence of the border, and $i -$ LO$[b-1]$, that is in case they overlap (line 8). Hence we obtain the following result.

Theorem 13. *The PCP array of a string of length n can be computed by Algorithm 3 in time $\mathcal{O}(n \log n)$.*

Proof. The algorithm consists of an outer for loop, going through the pruned border array A, and an inner while loop, iterating through the non-periodic borders of prefix $y[0\,..\,i]$. By Corollary 8, the number of non-periodic borders of each prefix is bounded by $\log n$. Hence, in overall, the time required is $\mathcal{O}(n \log n)$. □

We define one more array, similar to PCP but restricted to borders of the whole string only.

Definition 14. *Given a string y of length n, PCB is an array of size n of integers for which PCB$[i]$, for all $0 \leq i < n$, stores the number of letters of y which lie within an occurrence of the non-periodic prefix of length $i+1$, which is a border of y – zero if the prefix is periodic or is not a border of y.*

Example 15. Consider the string $y =$ aabaabaabbaaabaabaa. The following table illustrates the border array B of y, and the auxiliary arrays A, CB, R, PCP, and PCB of y.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$y[i]$	a	a	b	a	a	b	a	a	b	b	a	a	a	b	a	a	b	a	a
B$[i]$	0	1	0	1	2	3	4	5	6	0	1	2	2	3	4	5	6	7	8
A$[i]$	0	1	0	1	1	3	4	5	3	0	1	1	1	3	4	5	3	4	5
CB$[i]$	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
R$[i]$	0	1	0	1	1	0	1	5	0	0	1	1	1	0	1	5	0	1	5
PCP$[i]$	13	0	15	14	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PCB$[i]$	13	0	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0

5 Minimal enhanced cover

In this section, we show how to compute the minimal enhanced cover of string y. The minimal enhanced cover of y can be computed by Algorithm 4. It takes as input the array R of y. The algorithm consists of an outer for loop, going through the array R, and an inner while loop, iterating through the non-periodic borders of prefix $y[0\,..\,i]$ which are also borders of y. The key idea is the on-line maintenance of the PCB array (lines 5–8). Notice that array R considers only borders of the prefixes of y which are also borders of y (line 3). The number of positions covered by the border of length b is given by PCB$[b-1]$ (line 10).

By Lemma 9, the total number of occurrences of the considered borders is bounded by $2n$. Hence we obtain the following result.

Theorem 16. *The minimal enhanced cover of a string of length n can be computed in time $\mathcal{O}(n)$.*

6 Minimal enhanced cover array

In this section, we show how to compute the minimal enhanced cover array MEC of string y. Array MEC of y can be computed by Algorithm 5. It takes as input the pruned border array A of y. Similarly as in the case of Algorithm PositionsCoveredByPrefixesArray, it goes through the pruned border array A, and iterates through the pruned set of borders of prefix $y[0\,..\,i]$. Thus we are able to maintain the PCP array on-line, and use it to compute array MEC. For each prefix $y[0\,..\,i]$, in addition to the maintenance of the PCP array, we store the maximum value of

Algorithm 4: MinimalEnhancedCover

Input : The array R of string $y[0\,..\,n-1]$
Output: The length ℓ of the minimal enhanced cover

```
1  PCB[0..n−1] ← FillWithZeros; δ ← 0; ℓ ← 0
2  for i ← 0 to n − 1 do
3      b ← R[i]
4      while b > 0 do
5          if PCB[b − 1] = 0 then
6              PCB[b − 1] ← min(2b, i + 1)
7          else
8              PCB[b − 1] ← PCB[b − 1] + min(b, i − LO[b − 1])
9          LO[b − 1] ← i
10         if PCB[b − 1] > δ then
11             δ ← PCB[b − 1]
12             ℓ ← b
13         else if PCB[b − 1] = δ and b < ℓ then
14             ℓ ← b
15         b ← R[b − 1]
```

Algorithm 5: MinimalEnhancedCoverArray

Input : The pruned border array $A[0\,..\,n-1]$ of string $y[0\,..\,n-1]$
Output: The minimal enhanced cover array MEC

```
1  PCP[0..n−1] ← FillWithZeros
2  for i ← 0 to n − 1 do
3      b ← A[i]
4      ℓ ← 0
5      δ ← 0
6      while b > 0 do
7          if PCP[i] = 0 then
8              PCP[i] ← min(2b, i + 1)
9          else
10             PCP[i] ← PCP[i] + min(b, i − LO[b − 1])
11         if PCP[b − 1] ≥ δ then
12             δ ← PCP[b − 1]
13             ℓ ← b
14         LO[b − 1] ← i
15         b ← A[b − 1]
16     MEC[i] ← ℓ
```

$\mathsf{PCP}[b-1]$, for each border of length b of that prefix, in a variable δ, and the length b in a variable ℓ (lines 11–13).

By Theorem 13, the PCP array can be computed in time $\mathcal{O}(n \log n)$. Hence we obtain the following result.

Theorem 17. *The minimal enhanced cover array of a string of length n can be computed in time $\mathcal{O}(n \log n)$.*

7 Experimental results

We were able to verify the runtime of the proposed algorithms in experiments.

Fig. 6 illustrates the maximal ratio of the total number of occurrences of the non-periodic borders of y, computed by Algorithm MinimalEnhancedCover, to the length n of string, for all strings on the binary alphabet of lengths 1 to 31. These ratios are known to be smaller than 2 by Lemma 9. However, values close to this bound are not observed for small word length.

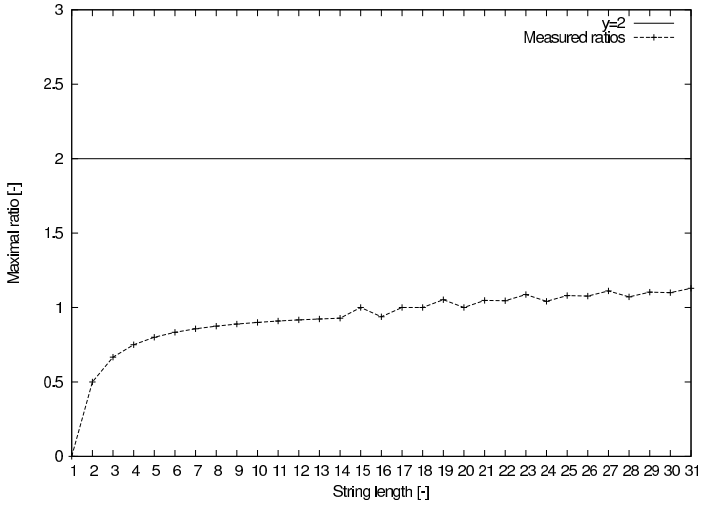

Figure 6. Maximal ratio of of the total number of occurrences of the non-periodic borders to the length n of string, for all strings on the binary alphabet

Fig. 7 and Fig. 8 illustrate the maximal ratio of the number of operations of Algorithm MinimalEnhancedCoverArray to the length n of string, for all strings on the binary alphabet of lengths 1 to 31, and the ratio of the number of operations to the length n of string, for Fibonacci strings f_3 to f_{45}, respectively. These ratios are known to be smaller than $\log n$ by Theorem 13.

The main observation from Fig. 7 is that, although the upper theoretical bound of these ratios is $\mathcal{O}(\log n)$, in practice, this is much less for strings on the binary alphabet. Fig. 8 strongly indicates that these ratios are probably constant for Fibonacci strings; something it would be interesting to show in the future.

In order to evaluate the performance of Algorithm MinimalEnhancedCover-Array with real datasets, we measured the ratio of the number of operations to the length of three DNA sequences: the single chromosome of *Escherichia coli* str. K-12 substr. MG1655; chromosome 1 of *Mus musculus* (laboratory mouse), Build 37.2; and chromosome 1 of *Homo sapiens* (human), Build 37.2. The measured ratios are 1.000006, 1.000000, and 1.001192, respectively, suggesting linear runtime of the proposed algorithms in practical terms.

The implementation of the proposed algorithms is available at a website (`http://www.exelixis-lab.org/solon/asc.html`) for further testing.

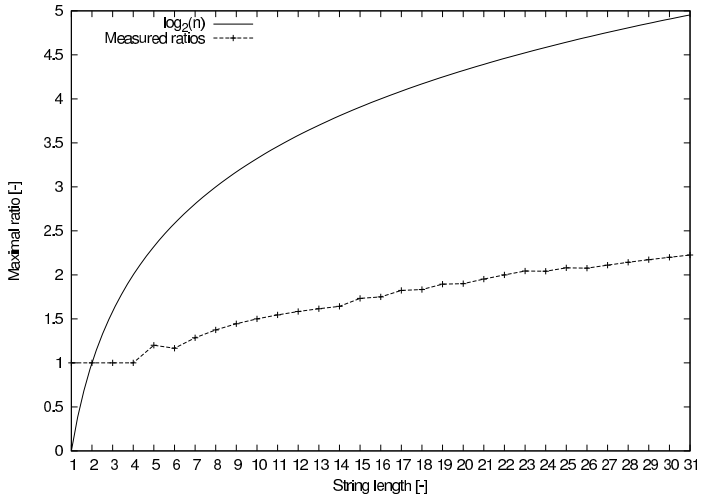

Figure 7. Maximal ratio of the number of operations of Algorithm 4 to the length n of string, for all strings on the binary alphabet

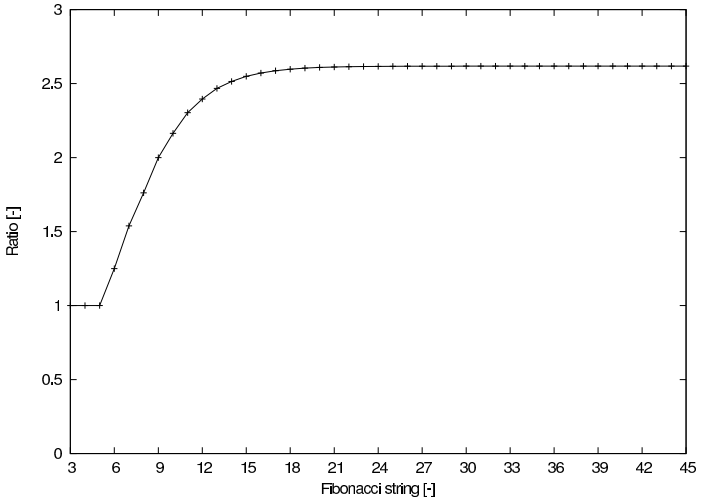

Figure 8. Ratio of the number of operations of Algorithm 4 to the length n of string, for Fibonacci strings

8 Concluding remarks

There are several directions for future work. Our immediate target is to investigate analogous data structures for other quasiperiodic notions such as the seed [12], the left seed [7], and the right seed [6] of a string. We will also consider the following problems for an array A of size n of integers.

Problem 18. Decide if A is the minimal enhanced cover array of some string.

Problem 19. When A is a valid minimal enhanced cover array, infer a string, whose minimal enhanced cover array is A.

For certain applications, the definition of the minimal enhanced cover might not be useful, since it primarily optimises the number of positions covered, while the length of the enhanced cover cannot be controlled. We can extend this notion by

introducing the d-restricted enhanced cover of string y, which is the shortest border of y of length not exceeding d which covers the largest number of positions among borders no longer than d. The algorithm computing the minimal enhanced cover, with almost no extra computations, can compute the d-restricted enhanced covers for every positive integer $d < n$. Moreover, the algorithm computing the minimal enhanced cover array can be given an additional array D of size n of integers as input, and compute the $\mathsf{D}[i]$-restricted enhanced cover of $y[0\,..\,i]$, for all $0 \leq i < n$. This also requires no additional effort.

Another interesting open problem is to allow the enhanced cover to be any factor of the string – not only a border. A similar problem, though with different constraints on the occurrences of the cover, is considered in the context of grammar compression [5], but, to the best of our knowledge, no efficient solution for either problem has been published.

References

1. A. Apostolico and A. Ehrenfeucht: *Efficient detection of quasiperiodicities in strings*, Tech. Rep. CSD-1R-I048, Purdue University, 1990.
2. A. Apostolico and A. Ehrenfeucht: *Efficient detection of quasiperiodicities in strings.* Theoretical Computer Science, 119(2) 1993, pp. 247–265.
3. A. Apostolico, M. Farach, and C. S. Iliopoulos: *Optimal superprimitivity testing for strings.* Information Processing Letters, 39(1) 1991, pp. 17–20.
4. D. Breslauer: *An on-line string superprimitivity test.* Information Processing Letters, 44(6) 1992, pp. 345–347.
5. M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat: *The smallest grammar problem.* IEEE Transactions on Information Theory, 51(7) 2005, pp. 2554–2576.
6. M. Christou, M. Crochemore, O. Guth, C. S. Iliopoulos, and S. P. Pissis: *On the right-seed array of a string*, in Proceedings of the seventeenth annual International Computing and Combinatorics Conference (COCOON 2011), B. Fu and D.-Z. Du, eds., vol. 6842 of Lecture Notes in Computer Science, USA, 2011, Springer, pp. 492–502.
7. M. Christou, M. Crochemore, C. S. Iliopoulos, M. Kubica, S. P. Pissis, J. Radoszewski, W. Rytter, B. Szreder, and T. Walen: *Efficient seed computation revisited*, in Proceedings of the twenty-second annual Symposium on Combinatorial Pattern Matching (CPM 2011), R. Giancarlo and G. Manzini, eds., vol. 6661 of Lecture Notes in Computer Science, Italy, 2011, Springer, pp. 350–363.
8. R. Cole, C. S. Iliopoulos, M. Mohamed, W. F. Smyth, and L. Yang: *The complexity of the minimum k-cover problem.* Journal of Automata, Languages and Combinatorics, 10(5/6) 2005, pp. 641–653.
9. M. Crochemore, C. Hancart, and T. Lecroq: *Algorithms on Strings*, Cambridge University Press, USA, 2007.
10. C. S. Iliopoulos, M. Mohamed, L. Mouchard, W. F. Smyth, K. G. Perdikuri, and A. K. Tsakalidis: *String regularities with don't cares.* Nordic Journal of Computing, 10(1) 2003, pp. 40–51.
11. C. S. Iliopoulos, M. Mohamed, and W. F. Smyth: *New complexity results for the k-covers problem.* Information Sciences, 181(12) 2011, pp. 2571–2575.
12. C. S. Iliopoulos, D. Moore, and K. Park: *Covering a string.* Algorithmica, 16(3) 1996, pp. 288–297.
13. C. S. Iliopoulos and K. Park: *A work-time optimal algorithm for computing all string covers.* Theoretical Computer Science, 164(1–2) 1996, pp. 299–310.
14. C. S. Iliopoulos and W. F. Smyth: *On-line algorithms for k-covering*, in Proceedings of the ninth Australasian Workshop on Combinatorial Algorithms (AWOCA 2008), Curtin University of Technology, 1998, pp. 64–73.
15. D. E. Knuth, J. H. M. Jr., and V. R. Pratt: *Fast pattern matching in strings.* SIAM Journal on Computing, 6(2) 1977, pp. 323–350.

16. R. Kolpakov, G. Bana, and G. Kucherov: *mreps: efficient and flexible detection of tandem repeats in DNA*. Nucleic Acid Research, 31(13) 2003, pp. 3672–3678.
17. Y. Li and W. F. Smyth: *Computing the cover array in linear time*. Algorithmica, 32(1) 2002, pp. 95–106.
18. M. Lothaire, ed., *Applied Combinatorics on Words*, Cambridge University Press, 2005.
19. D. Moore and W. F. Smyth: *An optimal algorithm to compute all the covers of a string*. Information Processing Letters, 50(5) 1994, pp. 239–246.
20. D. Moore and W. F. Smyth: *A correction to "An optimal algorithm to compute all the covers of a string"*. Information Processing Letters, 54(2) 1995, pp. 101–103.
21. W. F. Smyth: *Computing patterns in strings*, Addison-Wesley, 2003.
22. W. F. Smyth: *Computing regularities in strings: a survey*. European Journal of Combinatorics, 2012, (to appear).